



Making things easy

© MCS Electronics , 1995-2011

BASCOM-AVR user manual

Introduction

by MCS Electronics

Dear reader.

Thank you for your interest in BASCOM.

BASCOM was "invented" in 1995. It was intended for personal usage only. I decided to make it public as I found no other tool that was so simple to use. Since that time, a lot of options and extensions were added. Without the help and patience of the many users, BASCOM would not be what it is today : "the best and most affordable tool for fast proto typing".

We hope that BASCOM will contribute in making your work with microprocessors Easy and enjoyable.

Please notice that the samples in the manual are intended as simple samples. You should have a look at the sample code provided in the SAMPLES directory.

The MCS Electronics Team

BASCOM-AVR

© 2011 MCS Electronics

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: 20-9-2011

Publisher

MCS Electronics

Managing Editor M.C.Alberts

Technical Editors M.C.Alberts

Cover Designer

B.F.de Graaff

Special thanks to:

All the people who contributed to this document, all the forum members that contributed in a positive way, all beta testers , and all customers.

While there is not enough space to mention all contributors, there are a few that I feel must be mentioned

Josef Franz Vögel. He wrote the Trig libraries, the AVR-DOS file system and the DOUBLE library.

Luciano Ian and Adrian, they are very active on the user forum. They take the time to give other forum members free help and advise. They do this for free just to help other BASOM users.

Table of Contents

	Foreword	0
Part I	Index	22
1	Keyword Reference	23
2	About MCS Electronics	
	Custom Designs	
	Application Notes	
Part II	Installation	30
1	Installation of BASCOM	
2	Updates	
3	Move to new PC	45
4	Installation on multiple computers	45
Part III	BASCOM IDE	47
1	Running BASCOM-AVR	
2	File New	
3	File Open	50
4	File Close	50
5	File Save	50
6	File Save As	50
7	File Print Preview	51
8	File Print	51
9	File Exit	51
10	Edit Undo	51
11	Edit Redo	51
12	Edit Cut	51
13	Edit Copy	51
14	Edit Paste	52
15	Edit Find	52
16	Edit Find Next	52
17	Edit Replace	52
18	Edit Goto	52
	Edit Toggle Bookmark	
20	Edit Goto Bookmark	52
21	Edit Indent Block	53
	Edit Unindent Block	
	Edit Remark Block	
	Edit Encrypt Selected Code	
	View PinOut	
26	View PDF viewer	58

Contents

27	View Error Panel	60
28	View Tip	60
29	Program Compile	61
30	Program Syntax Check	61
31	Program Show Result	63
	Program Simulate	
	Program Send to Chip	
	Tools Terminal Emulator	
	Tools LCD Designer	
	Tools LIB Manager	
	Tools Graphic Converter	
38	Tools Stack Analyzer	83
39	Tools Plugin Manager	83
40	Tools Batch Compile	
41	Tools PDF Update	87
42	Tools Resource Editor	
43	Options Compiler	89
	Options Compiler Chip	90
	Options Compiler Output	
	Options Compiler Communication	
	Options Compiler I2C, SPI, 1WIRE	
	Options Compiler LCD	
	Options Communication	
45	Options Environment	
45	•	
45 46	Options Environment Options Simulator Options Programmer	96 100 101
45 46	Options Environment Options Simulator Options Programmer Supported Programmers	96 100 101 102
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer PG302 programmer	96 100 101 102 103 104
45 46	Options Environment Options Simulator Options Programmer Supported Programmers	96 100 101 102 103 104 105
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer PG302 programmer Sample Electronics cable programmer KITSRUS Programmer	96 100 101 102 103 104 105 106
45 46	Options Environment Options Simulator Options Programmer Supported Programmers	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer PG302 programmer Sample Electronics cable programmer KITSRUS Programmer MCS Universal Interface Programmer	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer PG302 programmer Sample Electronics cable programmer KITSRUS Programmer	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer PG302 programmer Sample Electronics cable programmer	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer PG302 programmer Sample Electronics cable programmer KITSRUS Programmer	
45 46	Options Environment Options Simulator Options Programmer	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer PG302 programmer Sample Electronics cable programmer KITSRUS Programmer MCS Universal Interface Programmer STK500 Programmer Lawicel BootLoader AVR ISP Programmer. USB-ISP Programmer. MCS Bootloader PROGGY FLIP	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer. PG302 programmer. Sample Electronics cable programmer. KITSRUS Programmer. MCS Universal Interface Programmer. STK500 Programmer. Lawicel BootLoader. AVR ISP Programmer. USB-ISP Programmer. MCS Bootloader. PROGGY. FLIP USBprog Programmer / AVR ISP mkll. KamProg for AVR. USBASP.	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer. PG302 programmer. Sample Electronics cable programmer. KITSRUS Programmer. MCS Universal Interface Programmer. STK500 Programmer. Lawicel BootLoader. AVR ISP Programmer. USB-ISP Programmer. MCS Bootloader. PROGGY. FLIP USBprog Programmer / AVR ISP mkll. KamProg for AVR. USBASP. STK600.	
45 46	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer. PG302 programmer. Sample Electronics cable programmer. KITSRUS Programmer. MCS Universal Interface Programmer. STK500 Programmer. Lawicel BootLoader. AVR ISP Programmer. USB-ISP Programmer. MCS Bootloader. PROGGY. FLIP USBprog Programmer / AVR ISP mkll. KamProg for AVR. USBASP. STK600 ARDUINO.	
45 46 47	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer. PG302 programmer. Sample Electronics cable programmer. KITSRUS Programmer. MCS Universal Interface Programmer. STK500 Programmer. Lawicel BootLoader. AVR ISP Programmer. USB-ISP Programmer. MCS Bootloader. PROGGY. FLIP USBprog Programmer / AVR ISP mkll. KamProg for AVR. USBASP. STK600. ARDUINO. LIBUSB	
45 46 47	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer. PG302 programmer. Sample Electronics cable programmer. KITSRUS Programmer. MCS Universal Interface Programmer. STK500 Programmer. Lawicel BootLoader. AVR ISP Programmer. USB-ISP Programmer. MCS Bootloader. PROGGY. FLIP USBprog Programmer / AVR ISP mkll. KamProg for AVR. USBASP. STK600 ARDUINO. LIBUSB Options Monitor	
45 46 47 47 48 48	Options Environment Options Simulator Options Programmer Supported Programmers ISP programmer. PG302 programmer. Sample Electronics cable programmer. KITSRUS Programmer. MCS Universal Interface Programmer. Lawicel BootLoader. AVR ISP Programmer. USB-ISP Programmer.	

BASCOM-AVR

51	Window Tile	141
52	Window Arrange Icons	141
53	Windows Maximize All	141
54	Window Minimize All	141
55	Help About	142
56	Help Index	143
57	Help MCS Forum	143
58	Help MCS Shop	144
59	Help Support	145
60	Help Knowledge Base	145
61	Help Credits	146
62	BASCOM Editor Keys	147
63	Program Development Order	148
64	Plugins	148
	Font Editor	148
Part IV	BASCOMHARDWARE	152

1	Additional Hardware	.152
	AVR Internal Hardware	-
	AVR Internal Registers	
	•	
	AVR Internal Hardware TIMER0	
-	AVR Internal Hardware TIMER1	
6	AVR Internal Hardware Watchdog timer	. 158
7	AVR Internal Hardware Port B	. 158
8	AVR Internal Hardware Port D	. 160
9	Adding XRAM	. 161
10	Attaching an LCD Display	. 162
11	Memory usage	. 163
12	Using the UART	. 165
	USING RS485	
	Using the I2C protocol	
	Using the 1 WIRE protocol	
	Using the SPI protocol	
	Power Up	
	•	
18	Chips	
	AT86RF401	-
	AT90S1200 AT90S2313	
	AT90S2313	192 193
	AT9052525	193
	AT9052343	
	AT90S4414	
	AT90S4433	196
	AT90S4434	198
	AT90S8515	199
	AT90S8535	199

Contents

	. 200
AT90PWM2-3 AT90PWM216	
AT90CAN128	
AT90USB162	
ATtiny12	
ATtiny13	
ATtiny15	
ATtiny10	
ATtiny20	
ATtiny22	
ATtiny24	
ATtiny26	
ATtiny44	
ATtiny45	
ATtiny48	
ATtiny84	
ATtiny85	
ATtiny88	
ATtiny167	
ATtiny261	
ATtiny461	
ATtiny861	
ATtiny2313	
ATtiny4313	
ATMEGA8	
ATMEGA16	. 213
ATMEGA16U4	. 214
ATMEGA32	. 215
ATMEGA16M1_32M1_64M1	. 216
ATMEGA8U2_162_32U2	. 216
ATMEGA16U4_32U4	. 218
ATMEGA48	. 219
ATMEGA48P	. 219
ATMEGA88	. 220
ATMEGA88P	. 220
ATMEGA64	. 221
ATMEGA103	. 221
ATMEGA128	. 223
	-
ATMEGA128	. 223
ATMEGA128ATMEGA1284P	. 223 . 225
ATMEGA128 ATMEGA1284P ATMEGA161	. 223 . 225 . 225
ATMEGA128	. 223 . 225 . 225 . 226
ATMEGA128	. 223 . 225 . 225 . 226 . 226 . 227
ATMEGA128	223 225 225 226 226 227 227
ATMEGA128	223 225 225 226 226 227 228 228 229
ATMEGA128	. 223 . 225 . 225 . 226 . 227 . 228 . 229 . 229
ATMEGA128	. 223 . 225 . 225 . 226 . 227 . 228 . 229 . 229 . 229 . 230
ATMEGA128 ATMEGA1284P ATMEGA161 ATMEGA162 ATMEGA163 ATMEGA164P ATMEGA165 ATMEGA168 ATMEGA168P ATMEGA169 ATMEGA323	. 223 225 225 226 227 228 229 229 229 229 230 . 230
ATMEGA128 ATMEGA1284P ATMEGA161 ATMEGA162 ATMEGA163 ATMEGA163 ATMEGA164P ATMEGA165 ATMEGA168 ATMEGA168 ATMEGA169 ATMEGA323 ATMEGA324P	. 223 225 225 226 227 228 229 229 229 229 229 230 230 231
ATMEGA128 ATMEGA1284P ATMEGA161 ATMEGA162 ATMEGA163 ATMEGA163 ATMEGA164P ATMEGA165 ATMEGA168 ATMEGA168 ATMEGA168P ATMEGA169 ATMEGA323 ATMEGA325	. 223 225 225 226 227 228 229 229 229 230 230 231 231
ATMEGA128	 223 225 225 226 227 228 229 229 230 231 232 233
ATMEGA128 ATMEGA1284P ATMEGA161 ATMEGA162 ATMEGA163 ATMEGA164P ATMEGA165 ATMEGA168 ATMEGA168 ATMEGA169 ATMEGA323 ATMEGA324P ATMEGA325 ATMEGA328	223 225 225 226 227 228 229 229 229 230 230 231 232 233 233 234
ATMEGA128 ATMEGA1284P ATMEGA161 ATMEGA162 ATMEGA163 ATMEGA164P ATMEGA165 ATMEGA168 ATMEGA168 ATMEGA168P ATMEGA169 ATMEGA323 ATMEGA324P ATMEGA325 ATMEGA328P	 223 225 225 226 227 228 229 229 230 231 232 233 234 234
ATMEGA128	 223 225 225 226 227 228 229 229 230 231 231 233 234 234 235
ATMEGA128	223 225 225 226 227 228 229 229 230 230 231 231 232 233 234 234 235 235
ATMEGA128 ATMEGA128 ATMEGA161 ATMEGA162 ATMEGA162 ATMEGA164P ATMEGA165 ATMEGA168 ATMEGA168P ATMEGA168P ATMEGA169 ATMEGA323 ATMEGA324P ATMEGA325P ATMEGA325P ATMEGA328P ATMEGA328P ATMEGA329 ATMEGA329 ATMEGA329	 223 225 225 226 227 228 229 229 230 231 231 232 233 234 235 236
ATMEGA128	 223 225 225 226 227 228 229 229 230 231 231 233 234 235 236 238

	ATMEGA645	. 240
	ATMEGA649	
	ATMEGA2560	
	ATMEGA2561	
	ATMEGA8535	
	ATXMEGA16A4 ATXMEGA16D4	
	ATXMEGA18D4	
	ATXMEGA52D4	
	ATXMEGA128A1	
	ATXMEGA128A3	254
	ATXMEGA128D3	. 254
	ATXMEGA128D4	255
	ATXMEGA192A3	256
	ATXMEGA192D3	. 257
	ATXMEGA256A3	. 258
	ATXMEGA256A3B	. 259
	ATXMEGA256D3	260
19	Reference Designs	261
	EM4095 RFID Reader	. 261
	USB162 module	. 270
Part V	BASCOM Language Fundamentals	273
1	Changes compared to BASCOM-8051	273
-		
2	Language Fundamentals	274
2 3	Language Fundamentals Mixing ASM and BASIC	274 286
2 3 4	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics	274 286 291
2 3 4	Language Fundamentals Mixing ASM and BASIC	274 286 291
2 3 4 5	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics	274 286 291 296
2 3 4 5 6	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes	274 286 291 296 297
2 3 4 5 6 7	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems	274 286 291 296 297 301
2 3 4 5 6 7 8	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems Tips and tricks	274 286 291 296 297 301 302
2 3 4 5 6 7 8	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems	274 286 291 296 297 301 302
2 3 4 5 6 7 8 9	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems Tips and tricks ASCII chart	274 286 291 296 297 301 302
2 3 4 5 6 7 8 9	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems Tips and tricks ASCII chart BASCOM Language Reference	274 286 291 296 297 301 302 303 307
2 3 4 5 6 7 8 9 Part VI 1	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems Tips and tricks ASCII chart BASCOM Language Reference #AUTOCODE	274 286 291 296 297 301 302 303 307 307
2 3 4 5 6 7 8 9 Part VI 1 2	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems Tips and tricks ASCII chart BASCOM Language Reference #AUTOCODE #IF ELSE ENDIF	274 286 291 296 301 302 303 307 307 307
2 3 4 5 6 7 8 9 Part VI 1 2	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems Tips and tricks ASCII chart BASCOM Language Reference #AUTOCODE	274 286 291 296 301 302 303 307 307 307
2 3 4 5 6 7 8 9 Part VI 1 2 3	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems Tips and tricks ASCII chart BASCOM Language Reference #AUTOCODE #IF ELSE ENDIF	274 286 291 296 297 301 302 303 307 307 307 309
2 3 4 5 6 7 8 9 Part VI 1 2 3 4	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics	274 286 291 296 297 301 302 303 307 307 307 309 309 309
2 3 4 5 6 7 8 9 Part VI 1 2 3 4 5	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics Reserved Words Error Codes Newbie problems Tips and tricks	274 286 291 296 301 302 303 307 307 307 309 310
2 3 4 5 6 7 8 9 Part VI 1 2 3 4 5	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics	274 286 291 296 301 302 303 307 307 307 309 310
2 3 4 5 6 7 8 9 Part VI 1 2 3 4 5	Language Fundamentals Mixing ASM and BASIC Assembler mnemonics	274 286 291 296 297 301 302 303 307 307 307 309 310 311
2 3 4 5 6 7 8 9 Part VI 1 2 3 4 5 6	Language Fundamentals Mixing ASM and BASIC	274 286 291 296 301 301 302 303 307 307 307 309 310 311 312
2 3 4 5 6 7 8 9 Part VI 1 2 3 4 5 6 7 8	Language Fundamentals Mixing ASM and BASIC	274 286 291 296 297 301 302 303 307 307 309 310 311 312 314
2 3 4 5 6 7 8 9 Part VI 1 2 3 4 5 6 7 8 9	Language Fundamentals Mixing ASM and BASIC	274 286 291 296 301 302 303 307 307 307 307 309 310 311 312 314 314

11	\$CRYSTAL	. 316
12	\$DATA	. 316
13	\$DBG	. 318
14	\$DEFAULT	. 320
15	\$EEPLEAVE	. 321
16	\$EEPROM	. 321
17	\$EEPROMHEX	. 322
18	\$EEPROMSIZE	. 323
19	\$EXTERNAL	. 324
20	\$FILE	. 325
21	\$FRAMESIZE	. 325
22	\$HWSTACK	. 326
23	\$HWCHECK, \$SWCHECK, \$SOFTCHECK	. 327
24	\$INC	. 330
25	\$INCLUDE	. 330
26	\$INITMICRO	. 331
27	\$LCD	. 332
28	\$LCDPUTCTRL	. 335
29	\$LCDPUTDATA	. 336
30	\$LCDRS	. 337
31	\$LCDVFO	. 340
32	\$LIB	. 340
33	\$LOADER	. 343
34	\$LOADERSIZE	. 348
35	\$MAP	. 349
36	\$NOCOMPILE	. 350
37	\$NOINIT	. 351
38	\$NORAMCLEAR	. 351
39	\$NORAMPZ	. 352
40	\$NOTRANSFORM	. 352
41	\$PROJECTTIME	. 353
42	\$PROG	. 354
43	\$PROGRAMMER	. 355
44	\$REGFILE	. 356
45	\$RESOURCE	. 357
46	\$ROMSTART	. 360
47	\$SERIALINPUT	. 360
48	\$SERIALINPUT1	. 362
49	\$SERIALINPUT2LCD	. 363
50	\$SERIALOUTPUT	. 363
51	\$SERIALOUTPUT1	364
52	\$SIM	. 365

53	\$SWSTACK	. 365
54	\$TIMEOUT	. 366
55	\$TINY	. 368
56	\$VERSION	. 369
57	\$WAITSTATE	. 370
58	\$XA	. 370
59	\$XRAMSIZE	. 371
60	\$XRAMSTART	. 371
61	1WIRECOUNT	. 372
62	1WRESET	. 375
63	1WREAD	. 377
64	1WSEARCHFIRST	. 379
65	1WSEARCHNEXT	. 382
66	1WVERIFY	. 384
67	1WWRITE	. 386
68	ABS	. 388
69	ACOS	. 389
70	ADR, ADR2	. 390
71	AESDECRYPT	. 394
72	AESENCRYPT	. 396
73	ALIAS	. 398
74	ASC	. 399
75	ASIN	. 402
76	ATN	. 403
77	ATN2	. 404
78	BASE64DEC	. 405
79	BASE64ENC	. 406
80	BAUD	. 407
81	BAUD1	. 408
	BCD	
83	BIN	. 411
	BINVAL	
85	BIN2GRAY	. 413
	BITWAIT	
	BITS	
	BLOAD	
	вох	-
	BOXFILL	
	BSAVE	
	BUFSPACE	
	BYVAL	
	CALL	
- T		

Contents

95	CHARPOS	. 424
96	CHECKFLOAT	. 425
97	CHECKSUM CHECKSUMXOR	. 427
98	CHR	. 428
99	CIRCLE	. 429
100	CLEAR	. 432
101	CLS	. 433
102	CLOCKDIVISION	. 436
103	CLOSE	. 437
104	CLOSESOCKET	. 439
105	CONFIG	. 442
106	CONFIG 1WIRE	. 444
107	CONFIG ACI	. 448
108	CONFIG ACXX	. 449
109	CONFIG ADC	. 451
110	CONFIG ADCx	. 453
111	CONFIG ATEMU	. 455
112	CONFIG BASE	. 457
113	CONFIG BCCARD	. 458
114	CONFIG CLOCK	. 461
115	CONFIG CLOCKDIV	. 465
116	CONFIG COM1	. 466
117	CONFIG COM2	. 467
118	CONFIG COMx	. 469
119	CONFIG DACx	. 471
120	CONFIG DATE	. 473
121	CONFIG DCF77	. 475
122		. 482
123	CONFIG DMA	. 483
124	CONFIG DMACHx	. 484
125	CONFIG DMXSLAVE	. 488
126	CONFIG DP	. 490
127	CONFIG EEPROM	. 491
128	CONFIG ERROR	. 491
129	CONFIG EVENT_SYSTEM	. 492
130	CONFIG EXTENDED_PORT	. 493
131	CONFIG HITAG	. 494
132	CONFIG I2CDELAY	. 498
133	CONFIG I2CSLAVE	. 500
134	CONFIG INPUT	. 503
135	CONFIG INTx	. 504
136	CONFIG GRAPHLCD	. 506

137	CONFIG KBD	. 511
138	CONFIG KEYBOARD	. 513
139	CONFIG LCD	.516
140	CONFIG LCDBUS	. 520
141	CONFIG LCDMODE	. 523
142	CONFIG LCDPIN	. 523
143	CONFIG OSC	. 527
144	CONFIG PORT	. 528
145	CONFIG POWER_REDUCTION	. 530
146	CONFIG PRIORITY	. 533
147	CONFIG PRINT	. 534
148	CONFIG PRINTBIN	. 535
149	CONFIG PS2EMU	. 535
150	CONFIG RC5	. 538
151	CONFIG SDA	. 538
152	CONFIG SCL	. 539
153		. 540
154	CONFIG SERIALOUT	. 545
155	CONFIG SINGLE	. 547
156	CONFIG SHIFTIN	. 548
157	CONFIG SPI	. 548
158	CONFIG SPIx	. 550
159	CONFIG SERVOS	. 552
160	CONFIG SUBMODE	. 555
161	CONFIG SYSCLOCK	. 556
162	CONFIG TCXX	. 557
163	CONFIG TCPIP	. 559
164	CONFIG TIMER0	. 562
165	CONFIG TIMER1	. 564
166	CONFIG TIMER2	. 567
167	CONFIG TWI	. 569
168	CONFIG TWISLAVE	. 571
169	CONFIG USB	. 575
170	CONFIG VPORT	. 582
171	CONFIG WAITSUART	. 584
172	CONFIG WATCHDOG	. 584
173	CONFIG X10	. 587
174	CONFIG XRAM	. 589
175	CONST	. 594
176	COS	. 596
177	COSH	. 597
178	COUNTER0 and COUNTER1	. 597

179	СРЕЕК	. 598
180	СРЕЕКН	. 600
181	CRC8	. 601
182	CRC16	. 603
183	CRC16UNI	. 605
184	CRC32	. 607
185	CRYSTAL	. 608
186	CURSOR	. 609
187	DATA	. 612
188	DAYOFWEEK	. 615
189	DAYOFYEAR	. 624
190	DATE\$. 625
191	DATE	. 627
192	DBG	. 637
193	DCF77TIMEZONE	. 637
194	DEBUG	. 638
195	DEBOUNCE	. 639
196	DECR	. 640
197	DECLARE FUNCTION	. 642
198	DECLARE SUB	. 644
199	DEFxxx	. 648
200	DEFLCDCHAR	. 648
201	DEG2RAD	. 649
202	DELAY	. 650
203	DELCHAR	. 651
204	DELCHARS	. 652
205	DIM	. 653
206	DIR	. 656
207	DISABLE	. 657
208	DISKFREE	. 660
209	DISKSIZE	. 661
210	DISPLAY	. 661
211	DO-LOOP	. 664
212	DriveCheck	. 666
213	DriveGetIdentity	. 666
214	Drivelnit	. 667
215	DriveReset	. 668
216	DriveReadSector	. 669
217	DriveWriteSector	. 670
218	DTMFOUT	. 671
219	ECHO	. 673

Contents

221	ENABLE	676
222	ENCODER	678
223	END	680
224	EOF	681
225	EXIT	682
226	EXP	683
227	FILEATTR	684
228	FILEDATE	685
229	FILEDATETIME	686
230	FILELEN	687
231	FILETIME	687
232	FIX	688
233	FLUSH	689
234	FORMAT	690
235	FOR-NEXT	691
236	FOURTHLINE	693
237	FRAC	694
238	FREEFILE	695
239	FUSING	696
240	GET	697
241	GETADC	700
242	GETATKBD	703
243	GETATKBDRAW	707
244	GETDSTIP	708
	GETDSTPORT	
246	GETKBD	709
247	GETRC	711
	GETRC5	
249	GETREG	716
-	GETTCPREGS	-
	GETSOCKET	
	GLCDCMD	
	GLCDDATA	
	GOSUB	
-	GOTO	-
	GRAY2BIN	-
	НЕХ	-
	HEXVAL	
	HIGH	
	HIGHW	
	НОМЕ	
202	120111	123

Contents

		-
	I2CRECEIVE	
-	I2CSEND	
	I2START,I2CSTOP, I2CRBYTE, I2CWBYTE, I2CREPSTART	
	IDLE	
267	IF-THEN-ELSE-END IF	.735
268	INCR	.736
	INITFILESYSTEM	
270	INITLCD	.738
271	INKEY	.738
272	INP	.740
273	INPUTBIN	.741
274	INPUTHEX	.742
275	INPUT	.744
276	INSERTCHAR	.746
277	INSTR	.747
278	INT	.748
279	IP2STR	.749
280	ISCHARWAITING	.750
281	KILL	.751
282	LCASE	.751
283	LCD	.752
284	LCDAT	.755
285	LCDAUTODIM	.757
286	LCDCONTRAST	.758
287	LEFT	.758
288	LEN	.759
289	LINE	.759
290	LINEINPUT	.762
291	LTRIM	.763
292	LOAD	.764
293	LOADADR	.764
294	LOADLABEL	. 765
295	LOADWORDADR	.765
296	LOC	.766
297	LOF	.767
298	LOCAL	.768
299		.771
	LOG	
	LOG10	

305	LOW	.776
306	LOWERLINE	.776
307	MACRO	.777
308	MAKEBCD	.778
309	MAKEINT	.778
310	MAKEDEC	.779
311	MAKEMODBUS	. 780
312	MAKETCP	. 782
313	MAX	. 782
314		.784
315	MIN	. 785
316	MID	. 787
317	MOD	. 787
318	NBITS	. 788
319	ON INTERRUPT	. 789
320	ON VALUE	.792
321	OPEN	.794
322	OUT	. 802
323	PEEK	. 803
324	POKE	. 804
325	POPALL	. 805
326	POWER	. 805
327	POWER MODE	. 808
328	POWERDOWN	. 809
329	POWERSAVE	. 809
330	PRINT	. 810
331	PRINTBIN	. 812
	PSET	
333	PS2MOUSEXY	. 816
	PULSEIN	
	PULSEOUT	
336	PUSHALL	. 818
337	PUT	. 819
338	QUOTE	. 821
339	RAD2DEG	. 822
340	RC5SEND	. 822
	RC5SENDEXT	-
	RC6SEND	
	READ	
	READEEPROM	
	READHITAG	. 832
346	READHITAG	

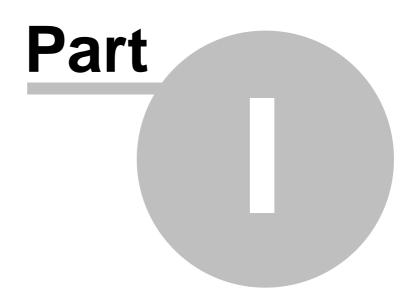
347	READSIG	837
348	REM	839
349	REPLACECHARS	840
350	RESET	840
351	RESTORE	841
352	RETURN	843
353	RIGHT	844
354	RND	844
355	ROTATE	846
356	ROUND	847
357	RTRIM	848
358	SECELAPSED	849
359	SECOFDAY	849
360	SEEK	850
361	SELECT-CASE-END SELECT	852
362	SET	853
363	SETFONT	855
364	SETREG	858
365	SETTCP	858
366	SETTCPREGS	860
367	SENDSCAN	862
368	SENDSCANKBD	864
369	SERIN	868
370	SEROUT	870
371	SETIPPROTOCOL	872
372	SGN	874
373	SHIFT	875
374	SHIFTCURSOR	877
375	SHIFTIN	877
	SHIFTOUT	
377	SHIFTLCD	882
	SHOWPIC	
	SHOWPICE	
380		
	SINH	
•••	SOCKETCONNECT	
	SOCKETLISTEN	
	SOCKETSTAT	
	SORVETSTAT	
	SORT	
500		554

Contents

389	SPC		898
390	SPIIN		899
391	SPIINIT	٢	900
392	SPIMO	VE	900
393	SPIOU	Т	901
394	SPLIT		901
395	SQR		903
396	START		904
397	STCHE	СК	906
398	STOP		910
399	STR		911
400	STR2D	IGITS	912
401	STRING	G	913
402	SUB		914
403	SYSSE	C	915
404	SYSSE	CELAPSED	916
405	SYSDA	Y	917
406	SWAP		918
407	TAN		920
408	тсрсн	IECKSUM	920
409	TCPRE	AD	924
410	TCPWF	RITE	924
411	TCPWF	RITESTR	925
412	TANH		929
413	THIRDI	LINE	930
414	TIME\$		930
415	TIME		931
416	TOGGL	_E	933
417	TRIM		935
418	UCASE		936
419	UDPRE	AD	937
420	UDPW	RITE	940
421	UDPW	RITESTR	941
422	UPPER	LINE	945
423	VAL		945
424	VARPT	R	946
425	VER		947
426	VERSIC	ON	947
427	WAIT		948
428	WAITK	EY	949
429	WAITM	IS	950
430	WAITU	S	951

	Content	:s 19
431	WHILE-WEND	
-	WRITE	
	X10DETECT	
435	X10SEND	958
Part VII	International Resellers	961
1	International Resellers	961
Part VIII	ASM Libraries and Add-Ons	963
1	EXTENDED I2C	
2	FM24C16	
3	FM24C64 256	
	FM25C256	
	I2C TWI	
	_	
	MCSBYTEINT	
8	TCPIP	967
9	LCD	968
	LCD4BUSY	
	LCD4.LIB	
	GLCD	
	GLCDSED	
	PCF8533	
	LCD-EPSON glcdR7565R	
	gicdSD1325_96x64	
10	AVR-DOS	
	AVR-DOS File System	
11	CF Card	
	Compact FlashCard Driver	
	Elektor CF-Interface	
	XRAM CF-Interface for simulation	
10	New CF-Card Drivers	
12	FP TRIG	
	PP_1RIG DOUBLE	
13	I2C SLAVE	
	I2CSLAVE	
	I2C TWI Slave	
14	SPI	
	SPISLAVE	
15	DATE TIME	
16	DATETIME PS2-AT Mouse and Keyboard Emulation	
10	AT EMULATOR	
	PS2MOUSE EMULATOR	

17	BCCARD	
	BCCARD	
	BCDEF	
	BCCALL	
	BCRESET	
18	USB	
	USB Add On	
19	MODBUS Slave/Server	
Part IX	Tools	1020
1	LCD RGB-8 Converter	
	Index	1022



1 Index

BASCOM®AVR®





Version 2.0.7.2 document build 39

MCS Electronics may update this documentation without notice. Products specification and usage may change accordingly. MCS Electronics will not be liable for any miss-information or errors found in this document.

All software provided with this product package is provided 'AS IS' without any warranty expressed or implied.

MCS Electronics will not be liable for any damages, costs or loss of profits arising from the usage of this product package.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without written permission of MCS Electronics.

Copyright MCS Electronics. All rights reserved.

1.1 Keyword Reference

1WIRE

1Wire routines allow you to communicate with Dallas 1wire chips. <u>1WRESET</u> [375], <u>1WREAD</u> [377], <u>1WWRITE</u> [386], <u>1WSEARCHFIRST</u> [378], <u>1WSEARCHNEXT</u> [382] <u>,1WVERIFY</u> [384], <u>1WIRECOUNT</u> [372]

Conditions

Conditions execute a part of the program depending on a condition being True or False

IF-THEN-ELSE-END IF [738], WHILE-WEND [552], ELSE 678], DO-LOOP 664], SELECT CASE - END SELECT [552], FOR-NEXT 699]

Configuration

Configuration commands initialize the hardware to the desired state. <u>CONFIG</u>[442], <u>CONFIG ACI</u>[448], <u>CONFIG ADC</u>[457], <u>CONFIG ADC</u>[457], <u>CONFIG CONFIG BCCARD</u> [458], <u>CONFIG CLOCK</u>[467], <u>CONFIG COM1</u>[468], <u>CONFIG COM2</u>[467], <u>CONFIG DAC</u>[477], <u>CONFIG DATE</u>[473], <u>CONFIG DMXSLAVE</u>[488], <u>CONFIG EEPROM</u>[497], <u>CONFIG I2CSLAVE</u>[500] <u>CONFIG INPUT</u>[503], <u>CONFIG GRAPHLCD</u>[506], <u>CONFIG ATEMU</u>[455], <u>CONFIG I2CSLAVE</u>[500] <u>CONFIG INPUT</u>[503], <u>CONFIG GRAPHLCD</u>[506], <u>CONFIG KEYBOARD</u>[513], <u>CONFIG</u> <u>TIMER0</u>[5627], <u>CONFIG TIMER1</u>[564], <u>CONFIG LCDBUS</u>[520], <u>CONFIG LCDMODE</u>[523], <u>CONFIG 1WIRE</u>[444], <u>CONFIG LCD</u>[516], <u>CONFIG OSC</u>[527], <u>CONFIG SERIALOUT</u>[545], <u>CONFIG SERIALIN</u>[546], <u>CONFIG SPI</u>[546], <u>CONFIG SPI</u>[5560], <u>CONFIG SYSCLOCK</u>[556], <u>CONFIG LCDPIN</u>[523], <u>CONFIG PRIORITY</u>[533], <u>CONFIG SDA</u>[538], <u>CONFIG SUBLOUT</u>[543], <u>CONFIG DEBOUNCE</u>[482], <u>CONFIG WATCHDOG</u>[584], <u>CONFIG PORT</u>, [528]COUNTER0 AND <u>COUNTER1</u>[597], <u>CONFIG TCPIP</u>[558], <u>CONFIG TWISLAVE</u>[577], <u>CONFIG SINGLE</u>[547], <u>CONFIG X10</u>[587], <u>CONFIG RRAM</u>[588], <u>CONFIG DP[490]</u>, <u>CONFIG TCXX</u> [557], <u>CONFIG VPORT</u>[582] <u>CONFIG ERROR</u>[497], <u>CONFIG DPWER REDUCTION</u>[530], <u>CONFIG EVENT</u>], <u>EVENT SYSTEM</u>[492], <u>CONFIG DMA</u>[483], <u>CONFIG DMACHx</u>[484], <u>CONFIG SUBMODE</u>[555]]

Conversion

A conversion routine is a function that converts a number or string from one form to another.

BCD [409], GRAY2BIN [720], BIN2GRAY [413], BIN [411], MAKEBCD [778], MAKEDEC [779], MAKEINT [778], FORMAT [600], FUSING [606], BINVAL [412], CRC8 [601], CRC16 [603], CRC16UNI [605], CRC32 [607], HIGH [723], HIGHW [724], LOW [776], AESENCRYPT [396], AESDECRYPT [394]

DateTime

Date Time routines can be used to calculate with date and/or times. <u>DATE</u> [627], <u>TIME</u> [937], <u>DATE</u>\$ [628], <u>TIME</u>\$ [938], <u>DAYOFWEEK</u> [615], <u>DAYOFYEAR</u> [624], <u>SECOFDAY</u> [849], <u>SECELAPSED</u> [849], <u>SYSDAY</u> [917], <u>SYSSEC</u> [916], <u>SYSSECELAPSED</u> [916]

Delay

Delay routines delay the program for the specified time. WAIT 948 , WAITMS 950 , WAITUS 951 , DELAY 650

Directives

Directives are special instructions for the compiler. They can override a setting from the IDE.

24 BASCOM-AVR

\$ASM 3001, \$BAUD 3001, \$BAUD1 3011, \$BIGSTRINGS 3141, \$BGF 3121, \$BOOT 3141, \$CRYSTAL 3161, \$DATA 3161, \$DBG 3161, \$DEFAULT 3201, \$EEPLEAVE 3211, \$EEPROM 3211, \$EEPROMHEX 3221, \$EEPROMSIZE 3231, \$EXTERNAL 3241, \$HWSTACK 3261, \$INC 3301, \$INCLUDE 3301, \$INITMICRO 3311, \$LCD 3321, \$LCDRS 3371, \$LCDPUTCTRL 3351, \$LCDPUTDATA 3361, \$LCDVFO 3401, \$LIB 3401, \$LOADER 3431, \$LOADERSIZE 3461, \$MAP 3491, \$NOCOMPILE 3301, \$NOINIT 3311, \$NORAMCLEAR 3311, \$NORAMPZ 3321, \$PROJECTTIME 3531, \$PROG 3541, \$PROGRAMMER 3551, \$REGFILE 3661, \$RESOURCE 3571, \$ROMSTART 3601 \$SERIALINPUT 3601, \$SERIALINPUT1 3621, \$SERIALINPUT2LCD 3631, \$SERIAL OUTPUT 3631, \$SERIAL OUTPUT1 3641, \$SIM 3651, \$SWSTACK 3651, \$TIMEOUT 3661, \$TINY 3661, \$WAITSTATE 3701, \$XRAMSIZE 3711, \$XRAMSTART 3771, \$XA 3701, \$CRYPT 3151, \$NOTRANSFORM 3521, \$FILE 3251, \$AESKEY 3091

File

 File commands can be used with AVR-DOS, the Disk Operating System for AVR.

 BSAVE [420], BLOAD [416], GET [697], VER [947], DISKFREE [660], DIR [656], DriveReset [668],

 DriveInit [667], LINE INPUT [762], INITFILESYSTEM [737], EOF [681], WRITE [953], FLUSH [689],

 FREEFILE [665], FILEATTR [684], FILEDATE [686], FILETIME [687], FILEDATETIME [686], FILELEN

 [687], SEEK [850], KILL [751], DriveGetIdentity [666], DriveWriteSector [670], DriveReadSector

 [689], LOC [766], LOF [767], PUT [819], OPEN [794], CLOSE [437]

Graphical LCD

Graphical LCD commands extend the normal text LCD commands. <u>GLCDCMD</u> [718], <u>GLCDDATA</u> [718], <u>SETFONT</u> [855], <u>LINE</u> [759], <u>PSET</u> [813], <u>SHOWPIC</u> [883], <u>SHOWPICE</u> [883], <u>CIRCLE</u> [429], <u>BOX</u> [417]

I2C

I2C commands allow you to communicate with I2C chips with the TWI hardware or with emulated I2C hardware.

<u>I2CINIT</u> [726], <u>I2CRECEIVE</u> [726], <u>I2CSEND</u> [727], <u>I2CSTART, I2CREPSTART, I2CSTOP,</u> <u>I2CRBYTE, I2CWBYTE</u> [728]

ΙΟ

 $I/O \ commands \ are \ related \ to \ the \ I/O \ pins \ and \ ports \ of \ the \ processor \ chip. \\ \underline{ALIAS}_{[39]}, \ \underline{BITWAIT}_{[41]}, \ \underline{TOGGLE}_{[933]}, \ \underline{RESET}_{[840]}, \ \underline{SET}_{[853]}, \ \underline{SHIFTIN}_{[877]}, \ \underline{SHIFTOUT}_{[887]}, \ \underline{SHIFTIN}_{[877]}, \ \underline{SHIFTOUT}_{[817]}$

Micro

Micro statements are specific to the micro processor chip. IDLE [734], POWER mode [808], POWERDOWN [809], POWERSAVE [809], ON INTERRUPT [789], ENABLE [676], DISABLE [657], START [904], END [680], VERSION [947], CLOCKDIVISION [436], CRYSTAL [608], STOP [910]

Memory

Memory functions set or read RAM, EEPROM or flash memory. <u>ADR</u>[390], <u>ADR2</u>[390], <u>WRITEEEPROM</u>[954], <u>CPEEK</u>[598], <u>CPEEKH</u>[600], <u>PEEK</u>[603], <u>POKE</u>[804], <u>OUT</u>[802], <u>READEEPROM</u>[830], <u>DATA</u>[612], <u>INP</u>[740], <u>READ</u>[828], <u>RESTORE</u>[841], <u>LOOKDOWN</u>[772], <u>LOOKUP</u>[774], <u>LOOKUPSTR</u>[775], <u>CPEEKH</u>[600], <u>LOAD</u>[764], <u>LOADADR</u>[764], <u>LOADLABEL</u>[765], <u>LOADWORDADR</u>[765], <u>MEMCOPY</u>[784], <u>GETREG</u>[876], <u>SETREG</u>[858]

Remote Control

Remote control statements send or receive IR commands for remote control. <u>RC5SEND</u>^[822], <u>RC6SEND</u>^[826], <u>GETRC5</u>^[712], <u>SONYSEND</u>^[897]

RS-232

 $\begin{array}{l} \text{RS-232 are serial routines that use the UART or emulate a UART.} \\ \underline{\text{BAUD}}_{[407]}, \underline{\text{BAUD1}}_{[408]}, \underline{\text{BUFSPACE}}_{[427]}, \underline{\text{CLEAR}}_{[432]}, \underline{\text{ECHO}}_{[673]}, \underline{\text{WAITKEY}}_{[949]}, \\ \underline{\text{ISCHARWAITING}}_{[750]}, \underline{\text{INKEY}}_{[738]}, \underline{\text{INPUTBIN}}_{[747]}, \underline{\text{INPUTHEX}}_{[742]}, \underline{\text{INPUT}}_{[742]}, \underline{\text{PRINT}}_{[810]}, \\ \underline{\text{PRINTBIN}}_{[812]}, \underline{\text{SERIN}}_{[868]}, \underline{\text{SEROUT}}_{[870]}, \underline{\text{SPC}}_{[898]}, \underline{\text{MAKEMODBUS}}_{[780]} \end{array}$

SPI

SPI routines communicate according to the SPI protocol with either hardware SPI or software emulated SPI. <u>SPIIN</u> (B), <u>SPIINIT</u> (C), <u>SPIMOVE</u> (C), <u>SPIOUT</u> (C), <u>SPIOU</u>

String

String routines are used to manipulate strings.

ASC [399], CHARPOS [424], UCASE [398], LCASE [757], TRIM [393], SPLIT [907], LTRIM [763], INSTR [747], SPACE [397], STRING [313], RTRIM [348], LEFT [758], LEN [758], MID [787], RIGHT [344], VAL [345], STR [317], CHR [428], CHECKSUM [427], CHECKSUMXOR [427], HEX [722], HEXVAL [722], QUOTE [827], REPLACECHARS [840], STR2DIGITS [312], DELCHAR [667], DELCHARS [662], INSERTCHAR [746]

TCP/IP

TCP/IP routines can be used with the W3100/IIM7000/IIM7010 modules. <u>BASE64DEC</u>[406], <u>BASE64ENC</u>[406], <u>IP2STR</u>[749], <u>UDPREAD</u>[937], <u>UDPWRITE</u>[940], <u>UDPWRITESTR</u>[941], <u>TCPWRITE</u>[924], <u>TCPWRITESTR</u>[925], <u>TCPREAD</u>[924], <u>GETDSTIP</u>[708], <u>GETDSTPORT</u>[708], <u>SOCKETSTAT</u>[890], <u>SOCKETCONNECT</u>[886], <u>SOCKETLISTEN</u>[889], <u>GETSOCKET</u>[717], <u>CLOSESOCKET</u>[439], <u>SETTCP</u>[858], <u>GETTCPREGS</u>[716], <u>SETTCPREGS</u>[860], <u>SETIPPROTOCOL</u>[872], <u>TCPCHECKSUM</u>[920]

Text LCD

 Text LCD routines work with normal text based LCD displays.

 HOME
 TOPERLINE

 INITLCD
 INITLCD

 INITLCD
 INITLCD

Trig & Math

Trig and Math routines work with numeric variables.

 $\begin{array}{l} \underline{ACOS}_{389}^{389}, \underline{ASIN}_{402}^{402}, \underline{ATN}_{403}^{403}, \underline{ATN2}_{404}^{404}, \underline{EXP}_{683}^{683}, \underline{RAD2DEG}_{822}^{6822}, \underline{FRAC}_{694}^{694}, \underline{TAN}_{920}, \underline{TANH}_{929}^{929}, \underline{COS}_{596}^{596}, \underline{COSH}_{597}^{597}, \underline{LOG}_{772}^{772}, \underline{LOG10}_{772}^{772}, \underline{ROUND}_{847}^{6447}, \underline{ABS}_{388}^{588}, \underline{INT}_{748}^{748}, \underline{MAX}_{782}^{782}, \underline{MIN}_{786}^{786}, \underline{SQR}_{903}^{903}, \underline{SGN}_{874}^{674}, \underline{POWER}_{806}^{806}, \underline{SIN}_{886}^{806}, \underline{SINH}_{886}^{806}, \underline{FIX}_{688}^{688}, \underline{INCR}_{736}^{736}, \underline{DECR}_{649}^{649}, \underline{DEG2RAD}_{649}^{649}, \underline{CHECKFLOAT}_{425}^{425}, \underline{MOD}_{787}^{787} \end{array}$

Various

This section contains all statements that were hard to put into another group <u>CONST</u> [594], <u>DBG</u> [637], <u>DECLARE FUNCTION</u> [642], <u>DEBUG</u> [638], <u>DECLARE SUB</u> [644], <u>DEFXXX</u> [648], <u>DIM</u> [653], <u>DTMFOUT</u> [677], <u>EXIT</u> [682], <u>ENCODER</u> [678], <u>GETADC</u> [708], <u>GETKBD</u> [709],

 GETATKBD
 GOSUB
 GOSUB
 GOTO
 Table
 LOCAL
 GOSUB
 COSUB
 <

XMEGA

READSIG 837

1.2 About MCS Electronics

About the Founder

Since I was young, I was intrigued by remote control, robots, transmitters, in short, all electronics. I created countless electronic devices. I designed a lot of PCB's by hand, using ink. When the ATARI came along, with the ST1040 and an affordable PCB design tool, I bought my first real computer.

Dot matrix printers, at that time, did not produce very good print. Design of a PCB was still time consuming. I found that a nice BASIC interpreter, which was similar to GW-BASIC, was included in the OS(TOS). For some reason, I liked this language. It was easy to master and very intuitive.

I found out, that Atmel made the 89c2051, which was a 20 pin chip with flash memory. I was excited to find out that there was a small micro processor that could be erased/reprogrammed without the need to UV erase the EPROM. Before the Atmel chip, I used the 8052AH, with a BASIC interpreter. It worked nice, but code ran too slow. The EPROM's had to be erased by UV light which took a long time.

Those days, electronic circuits consisted of numerous CMOS and TTL chips. I saw the 89C2051, as an ideal replacement, for a lot of CMOS/TTL chips. It would make PCB design much simpler.

The idea to be able to change the behaviour of an electronic circuit, just by reprogramming it, without using a solder iron, intrigued me. Today, it is common practice, to update firmware, to fix bugs or add features. In 1993, it was not so common, at least not to my knowledge.

I initially wrote a complete tool for DOS. I rewrote the tool, when I was reasonably satisfied that Windows 3.1 was stable. The tool was for my own usage. When I discovered, that it would be usable to others, I decided to add Help files and to sell it for a small fee.

In 1995, MCS started to sell BASCOM-LT, a BASIC compiler for Windows 3.1. It was the first Windows application that offered a complete and affordable solution, editor, compiler, simulator and programmer. BASCOM-LT was a 8051 BASIC compiler. The reason it became popular, was that it included a lot of functionality that was easy to use from BASIC. To use an LCD display was simple, just a configuration line to define the used pins and voila, a working application in minutes. When you needed a different LCD display, you could simply change the CONFIG line.

When a different processor was needed, you only had to change the definition file!

Another reason for its success, was that we hid much of the complexity, from the user. No ASM to deal with, simple statements. Of course free updates and support.

Small companies that used the BASIC Stamp also recognized another advantage, that there was no need for expensive modules and the code ran much quicker.

When Windows 95 became more an industry standard, users also wanted a 32 bit version. A big part of BASCOM-LT was rewritten with the additional support for arrays and floating point (single).

With the many different 8051 variants, it was impossible to support all the chips. Having device definition "DAT" files, made it easy for the user to configure the 8051 variants.

When Atmel launched the AVR chip, the 8051 compiler was rewritten, once again, to support the powerful AVR chips. The result was BASCOM-AVR.

The AVR chip has a lot of internal memory. It uses simple linear memory addressing. The best part, is that you can make the chip program itself. No wonder this chip family became so popular.

Since the AVR chip is so powerful, we could extend the compiler as well. We could add features, which are almost impossible to add for the 8051.

With more and more users, there was no way I could manage everything in my spare time. So in order to guarantee the future of BASCOM, I decided to work full time for MCS.

Today, MCS is still a small company, with only 3 employees.

We believe in free updates and support. With the number of (demo) users, it is however not possible to support everybody. You need to realize that reading and answering emails is time consuming.

Not to mention to duplicate used hardware. We are unique, in that we even support hardware!

For a long time, we are working on a more professional version of the software.

We occasionally add new features to the current BASCOM version.

An ARM version is also under development as well.

Note, that we do not provide details or time frames, for these versions, nor do we, for other features.

In order to migrate to a new version, it is important that you keep your software up to date. This will make migration more simple.

Things we find important :

- The environment. We reuse all usable packing material like foam, plastic bubbles, when we ship your order.
- That everybody can use microprocessors. They are like all other chips.
- Customer privacy: We keep your name, details and code confidential.
- Little advertisements on our web. You will only find them at our homepage.

BASCOM-AVR

28

- Free updates. They have been free since 1995 but there is no guarantee that they will remain free for ever. The intention is to keep them free.
- Free, but limited, support. Limited only, because we do not have the resources to read/answer all emails.
- Support for new chips. It is important to be able to support newly released chips.
- The customer : We simply add what is requested most. It does not matter what, as long as it is requested a lot.
- That you have fun with electronics, no matter where you live, no matter which religion you are, no matter how old you are or if you are male, female, purple or white.
- That you can use the free demo for free. That you pay for a full version, if you use it commercial. Please do not use cracked software. Only download from the mcselec.com web. Copies from other sites may contain spy ware, virus or other malware. Using cracked software, means the end of all software.

Have fun !

Mark Alberts

MCS Electronics

1.2.1 Custom Designs

MCS does produce hardware to support special options. Like the <u>EM4095 Reference</u> <u>Design</u> and <u>adapter</u> boards. We try not to use SMD parts. In some cases this is not possible however.

For a prototype or small series, through hole components are simple to use. We do this with the hobbyist in mind. So our reference designs use little SMD parts as possible.

We also do custom hard and software projects. Of course we can also produce hardware with SMD parts only. We also produce custom Windows software.

MCS knows a number of BASCOM consultants that can help you with your design. See also $\frac{About MCS}{26}$

1.2.2 Application Notes

When you want to show your application at our web as an example on what you can achieve with BASCOM, we like to publish it our web, but of course with your permission. We never publish anything without your explicit permission.

AN's are also welcome. When you developed a great AN you want to share with other BASCOM users, just send it and we will make an AN out of it. It is important that the comment in the source is in English.



2 Installation

2.1 Installation of BASCOM

After you have downloaded the ZIP file you need to UNZIP the file. On Windows XP, for the DEMO version, run the setupdemo.exe file from within the Zipped file.

The commercial version comes with a license file in the form of a DLL. This file is always on the disk where the file SETUP.EXE is located. When explorer does not show this file, you must set the option in explorer to view system files (because a DLL is a system file).

For the commercial version the setup file is named SETUP.EXE

Some resellers might distribute the DLL file in a zipped file. Or the file might have the extension of a number like "123". In this case you must rename the extension to DLL.

Make sure the DLL is in the same directory as the SETUP.EXE file.

When you are using the DEMO version you don't need to worry about the license file.

When you are installing on a NT machine like NT4 , W2000, XP or Vista, you need to have Administrator rights.

After installing BASCOM you must reboot the computer before you run BASCOM.

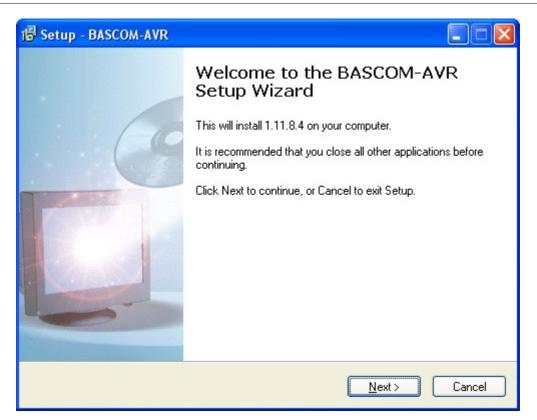
The installation example will describe how the FULL version installs. This is almost identical to the installation of the DEMO version.

Run the SETUPDEMO.EXE (or SETUP.EXE) by double clicking on it in explorer.

The following window will appear:

(screen shots may differ a bit)

31



Click on the **Next button** to continue installation.

The following license info window will appear:

😼 Setup - BASCOM-AVR	
License Agreement Please read the following important information before continuing.	L &
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.	
BASCOM-AVR	^
MCS Electronics NO-NONSENSE LICENSE STATEMENT AND LIMITED WARRANTY	_
IMPORTANT - READ CAREFULLY This license statement and limited warranty constitutes a legal agreement ("License Agreement") between you (either as an individual or a single entity) and MCS Electronics for the software product ("Software") identified above, including any software, media, and accompanying on-line or	>
I accept the agreement	
OI do not accept the agreement	
< <u>B</u> ack <u>N</u> ext >	Cancel

Read the instructions , select 'I accept the agreement' and press the Next button.

The following window will be shown :

Setup - BASCOM-AVR	
Information Please read the following important information before continuing.	8
When you are ready to continue with Setup, click Next.	
Thank you for using BASCOM-AVR	~
This file contains some additional info on BASCOM-AVR.	
When you get an Error 123 you probably don't have a printer installed Install a printer (HP Laserjet 4 for example) and run again. You do not need to connect a printer, but a printerdriver must be installed.	
When you get an OleSysError, reboot Windows and run SETUP again. After rebooting again, run BASCOM again.	
When you install the software on an NT/W2000/XP/VISTA machine you need	v
< <u>B</u> ack <u>N</u> ext >	Cancel

Read the additional information and click the **Next button** to continue.

Now the next screen will appear:

🕼 Setup - BASCOM-AVR
Select Destination Location Where should BASCOM-AVR be installed?
Setup will install BASCOM-AVR into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files\MCS Electronics\BASCOM-AVR Browse
At least 18.4 MB of free disk space is required.
< <u>B</u> ack <u>N</u> ext > Cancel

You can select the drive and path where you like BASCOM to be installed. You can also accept the default value which is :

C:\Program Files\MCS Electronics\BASCOM-AVR

When you are finished click the **Next Button** to continue. When the directory exists, because you install a newer version, you will get a warning

Folder I	Exists
2	The folder:
4	C:\Program Files\MCS Electronics\BASCOM-AVR
	already exists. Would you like to install to that folder anyway?
	Yes No

In case of this warning, select Yes.

You will now see the following window:

🕫 Setup - BASCOM-AVR
Select Start Menu Folder Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
MCS Electronics\BASCOM-AVR Browse
< <u>B</u> ack <u>N</u> ext > Cancel

You can choose to create into a new Program Group named 'BASCOM-AVR', or you can modify the name, or install into an existing Program Group. Press the **Next-button** after you have made your choice.

Now the files will be installed.

After the main files are installed, some additional files will be installed

🕼 Setup - BASCOM-AVR	
Additional Files Installing additional files	
Installing additional files	-1
	J
	J
<u>N</u> ext >	

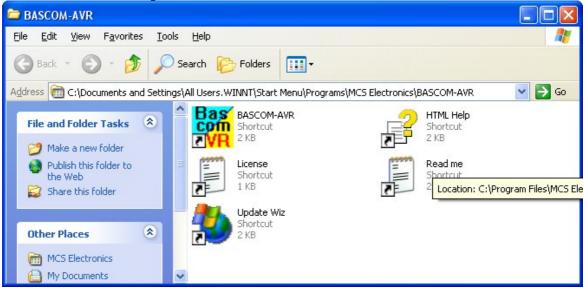
These additional files can be PDF files when the program is distributed on a CD-ROM.

When the installation is ready you will see the last screen :



You have to reboot your computer when you want to make advantage of the programmers that BASCOM supports. You can also do this at a later stage.

The BASCOM-AVR Program folder is created:



You can view the "Read me" and "License" files content and you can start BASCOM-AVR.

BASCOM supports both HTML Help and old Win help(HLP). The HLP file is not distributed in the setup. You need to use the Update Wiz to download it. But it is advised to use the HTML-Help file.

When you used to use the HLP file, and find it missing now, turn on 'Use HTML Help' in Options, Environment, IDE.

When the UpdateWiz is not installed, you can download it from the register 3.

2.2 Updates

The update process is simple.

- Go to the main MCS website at <u>http://www.mcselec.com</u>
- In the left pane under 'Main Menu' you will find a link named 'Registration/Updates'



Notice that the website uses two different accounts : one for the forum/shop and one for the registration/updates. You will see the following screen:

Product registration Login		
::Product registration Login		
User Name :		
Password :		
	Forgot your login data ?	
	Log in	
A 1997	A	
le l		
Create new account	Need Help ?	
For troubles	hooting read here	

• Click the link and select 'Create new account'

::Create new account * R	equired Information	
User Name :	agooduserName	*
Password :	xalalalalalalalalalalalalal	*
Enter Password Again :	*000000000000	*
Email :	thismustbe@∨alid.mail	*
Enter Email again :	thismustbe@∨alid.mail	*
Full name :	My Full Name	*
Company :		
Sending Email notify on updates :	~	
	Submit Registration	

You need to provide a username, password, email and full name. Company name is optional. When you want to receive notifications when updates are available, select this option.

When you filled in the information, click 'Submit Registration'.

- After you click submit, you can get various error messages. For example that a username already exists. Press the Back-button in your browser, and correct the problem, then try again
- If the registration is successful you will get a message that the registration succeeded.
- Now you can login. You will see the following screen :

	i loudot logiot attori Login	
::Product registration Login		
Your current status SLA Service access	: Registration approved : Yes	
	ave. 22 Difference	
Modify user information	Modify email address	Modify password
Product registration	Download Lic files	Logout
		2
Support Center	Add help tip	Help
	SLA Service	

Product registration Login

- You need to chose 'Product registration'.
- The following screen will be shown:

::Product registration Contract Main page		Logout
Avaiable products : Product serial number (example: CD-0000-12345)	BASCOM-AVR	

- Select a product from the list.
- Enter the serial number

It is important that you enter a **valid** serial number. Do not try to enter serial numbers from cracked versions. When you enter invalid serial numbers, you will loose support and the ability to update.

The valid serial number is shown in the Help, About box.

About		×
	BASCOM-AVR	
Compiler version Compiler build IDE version Serial number	: 1.11.8.9.001 : 1.11.8.9	
Windows OS Windows SP Explorer Company Owner Windows dir App data dir System dir	<pre>: Service Pack 2 : 7.0.5730.11 : MCS : Mark Alberts : C:\WINNT : C:\Documents and Settings\Mark</pre>	
Support	: <u>support@mcselec.com</u>	Close

When the product is selected, the serial number is entered, and you press 'Register product' you will see the following message :

The pa	ge at http://register.mcselec.com says: 🚺
⚠	Your number is now registered
	ОК

- This does mean that you registered successfully.
- MCS Electronics will validate all registrations once in a few days. When the product is validated you will receive an email. After you receive the email, you can login to the register again. When you did not received an email within 1 week, check if the email address was entered correct. If it was correct, send an email to sales.
- Now you need to select 'Download LIC files'. The following screen will be shown:

it.

lain page			Lo
lumber of registered p	roducts: 3		
Product	Serial number	S/N status	Date of registration
BASCOM-AVR		Valid	2005-11-15 06:58.36
BASCOM-8051		Valid	2006-05-12 20:01.27
BASCOM-AVR		Validation pending	2007-09-04 19:04.26
Actual version of BASCON Download full BASCOM-A	VR	2.0.13.0 Download	
Download full BASCOM-80		Download on CD (you will be redirecte	d to shop)
		-	d to shop) file. But instead of the LIC file you o

At the top you can see which products are registered, and which status they have. When you want to do a FULL SETUP, you need to download the full version.

You **do not** need to **uninstall a previous version**. You can install an update into the same directory or a new directory.

You can also order the same update on CD-ROM. You will be directed to the on line shop. Notice that the shop uses a different account/username

When you uninstall a previous version, it will remove the license file which is not part of the setup.exe

So in the event that you do run uninstall first, make a backup of the license dll ; bscavrL.DLL

The ZIP file you download contains only one setup.exe. You need to run this executable.

It is also important that you put the license DLL into the same directory as setup.exe Setup will copy this file to the BASCOM application directory. You can also manual copy this file.

The license file is on CD-ROM, diskette, or the media (email) you received it on. It is only supplied once.

Without the file, BASCOM will not run.

The file is named bsc5132L.DLL for BASCOM-8051 and bscavrL.DLL for BASCOM-AVR When you got the license by email, it was zipped and probably had a different extension. Consult the original installation instructions. The file is only provided once, we can not, and do not provide it again.

See Installing BASCOM 30 on how to do a full install.

It is also possible to do a partial update. For example to update some DAT files, or to update to a beta which is only available as an update. For partial updates, you need the Update Wiz.

BASCOM AVR 1.11.8.9 BASCOM AVR 1.11.8.8 Latest BASCOM AVR 1.11.8.5 BASCOM AVR 1.11.8.4	Download Download Download	History History
BASCOM AVR 1.11.8.5		History
	Download	
BASCOM AVR 1 11 8 4		History
DA300M AN(1.11.0.4	Download	History
BASCOM AVR 1.11.8.3	Download	History
BASCOM AVR 1.11.8.2	Download	History
BASCOM AVR 1.11.8.1	Download	History
BASCOM AVR 1.11.8.0	Download	History
BASCOM AVR 1.11.8.6	Download	History
BASCOM AVR 1.11.8.7	Download	History
BASCOM 8051 2.0.13.0 Latest	Download	History
BASCOM 8051 2.0.12.0	Download	History

When you do not have the Update Wiz, you can download it. Unzip it to the same directory as BASCOM.

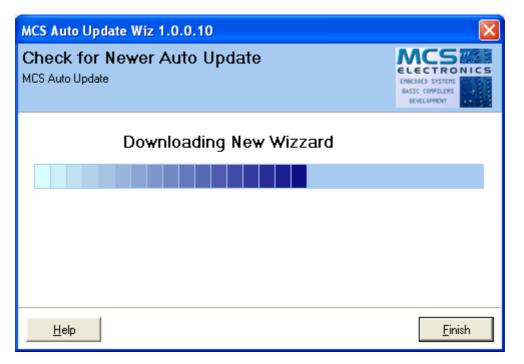
The Update Wiz uses LIC files which you can download. A LIC file is a text file, it is not the LICENSE DLL !

Store the downloaded LIC file in the same directory as the Update Wiz.

When you store the Update Wiz into the same directory as BASCOM, the license DLL already exist there.

When you put the Update Wiz and the LIC files into a separate directory, you need to copy the BASCOM license DLL to this directory too.

When you run the Update Wiz, it will check for a new version and will download this if available. It will then run again.



When the Update Wiz finds a LIC file, it will check if the update/install location is specified. For new downloaded LIC files, the update wiz does not know the update directory, and will ask for the directory you want to update. This can be any (new) directory, but usually is the BASCOM application directory.

Select Directory		
Directory <u>N</u> ame: c:\Program Files\MCS Electronics\BASCOM+	AVR	
Directories:	<u>F</u> iles: (*.*)	
C:\ Program Files MCS Electronics BASCOM-AVR BACKUP LIB pdf	1200DEF.DAT 2313DEF.dat 2323DEF.dat 2333DEF.DAT 2343DEF.dat 4414DEF.dat 4433DEF.DAT	
	C: [cdisk]	▼ Cancel

After you click Ok, the directory to update is stored in the LIC file. It will not be asked again.

MCS Auto Update Wiz 1.0.0.13	
Welcome MCS Auto Update	ELECTRONICS ENECODE SYSTEMS BASIC COMPLEXS EVELOPMENT
	Welcome to the MCS Auto Update Wiz
	Setup Rollback
Help	< Back <u>N</u> ext > Cancel

Click the Next button to start the update.

It depends on the downloaded LIC files how many products are found. You will get a similar window :

MCS Auto Update Wiz 1.0.0.14	
Product Selection MCS Auto Update	CHECKLOPICAS
Select an Installed Product you want to upda	te
BASCOM-AVR 1.11.8.3	Update PWD
Bas Com BASCOM-AVR 1.11.8.4	When you press the Next-button, the
Bas Com BASCOM-AVR 1.11.8.5	Wiz will check the Internet for updates
Bas Com AVR BASCOM-AVR 1.11.8.6 Bas	Remove
<u>H</u> elp	< <u>B</u> ack. <u>N</u> ext > Cancel

You need to select the product that you want to update. In the sample there are multiple choices.

Press the Next-button to continue.

The Wiz will compare files on the web with your local files in the specified directory.

MCS Auto Update Wiz 1.0.0.13	
Available Updates MCS Auto Update	ENECTRONICS ENECOLD SYSTEMS BASIC COMPILERS DEVELOPMENT
Components found to update:	
✓ help.zip	Info Select <u>All</u> Unselect All
Press the Next-button to download the selected components. Select at	least 1 component.
<u>H</u> elp < <u>B</u> ack <u>N</u>	ext > Cancel

When it finds packages that are newer, they will be shown in a list. By default they are all selected.

You can unselect the packages you do not want to update. Press Next to download the selected packages.

MCS Auto Update Wiz 1.0.0.13	×
Downloading MCS Auto Update	MCS ELECTRONICS Declos Vytris BASIC (OMPILERS BCVL OPPERT
Downloading help.zip 28%	
2.0.13.0 *** IMPORTANT *** Run setupio8051.exe after the Wizard has finished Setupio8051.exe is located in the BASCOM-8051 Application dir.	
- Vista and Xp-64 supported	>
Help < Back Next >	Cancel

During the download you will see the history file. When all packages are downloaded, they will be installed/unzipped. Press the Next-button to install the downloaded files.

MCS Auto Update Wiz 1.0.0.13	×
Install New Files MCS Auto Update	ELECTRONICS ENECADO SYSTEMS BASIC COMPLENS EVELOPMENT
Archive Progress	
Help < Back Finish	Cancel

During the installation you will see the progress. When installation is ready, you need to press the Finish-button.

The Wiz can also backup all files it will replace. Use the Setup button on the main screen of the UpdateWiz to change the settings. A full zipped backup will be made. The name of the backup files has the name of the license file with the ZIP extension.

You can install multiple versions in different directories.

2.3 Move to new PC

When you want to move BASCOM to a new PC. You have a number of options. The most simple is to download a full setup file from <u>http://register.mcselec.com</u> Then, after the installation, copy the license file **bscavrL.DLL** to the bascom-avr application directory of the new PC.

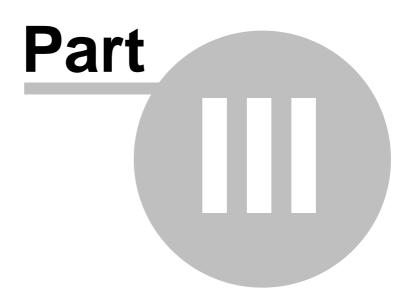
Or let setup.exe do this for you. When you put the license file in the same directory as setup.exe, setup will copy/install the file for you.

2.4 Installation on multiple computers

The following applies to the licensed version and the license key.

You may install BASCOM on multiple computers. For example on your laptop and your desk PC. There is no limit to the number of PC's you install the software. But you may only use one PC at the same time. Since you can only operate one PC at the same time, this is not a real restriction.

When you install on multiple PC's and others work on these PC's at the same time as you, you need multiple licenses!



3 BASCOM IDE

3.1 Running BASCOM-AVR

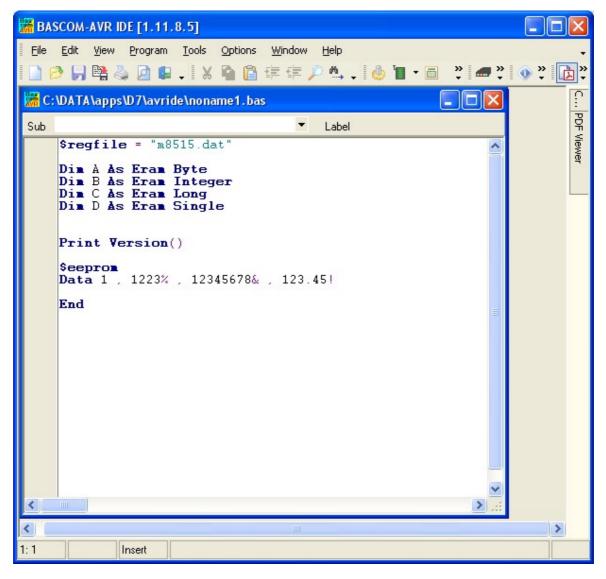
After you have installed BASCOM, you will find a program entry under MCS *Electronics\BASCOM-AVR*



Double-click the BASCOM-AVR icon

to run BASCOM.

The following window will appear. (If this is your first run, the edit window will be empty.)



The most-recently opened file will be loaded automatically. Like most Windows programs, there is a menu and a toolbar. The toolbar can be customized. To do this, place the mouse cursor right beside the 'Help' menu. Then right-click. You can turn on/off the toolbars or you can choose 'Customize'.

This will show the following window:

ustomize	
Toolbars Commands Options	
Toolb <u>a</u> rs:	
Main Menu	<u>N</u> ew
 ✓ File ✓ Edit ✓ Program 	R <u>e</u> name
	Delete
View	<u>R</u> eset
	Close

You have the option to create new Toolbars or the reset the toolbars to the default. To place a new button on a menu bar, select the 'Commands' TAB.

ustomize	í.			
Toolbars Categories Default Edit Program Tools Options File Window Help Menus View Description		Options Comm	an <u>d</u> s: Syntax check Compile Show result Simulate Send to chip Reset Chip	
				Close

In the example above, the Program Category has been selected and at the right pane, all buttons that belong to the Program-category are shown.

You can now select a button and drag & drop it to the Toolbar. To remove a button from the Toolbar, you drag it out of the Toolbar and release the left mouse button.

On the Options-TAB you can further customize the Toolbar:

Customize	
Toolbars Commands Options Personalized Menus and Toolbars Menus show recently used commands first Show full menus after a short delay Reset my usage data	
Other Large icons Show Tool <u>T</u> ips on toolbars	
Menu animations: (None)	
(Close

To preserve screen space there are no large icons available.

Option	Description
Menus show recent used commands first	With this option the IDE will learn the menu options you use. It will show only the most used menu options. The idea is that you can find your option quicker this way.
Show full menus after a short delay	This option will show the remaining menu options after short delay so you do not need to click another menu option to show all menu options.
Reset my usage data	This option will reset the data the IDE has collected about your menu choices.
Show Tool tips on toolbars	This option is on by default and it will show a tool tip when you hold the mouse cursor above a toolbar button
Show shortcut keys in Tool tips	This option is on by default and it will show the shortcut in the tool tip. For example CTRL+C for the Copy button.

3.2 File New

This option creates a new window in which you will write your program. The focus is set to the new window.

You can have multiple windows open at the same time.

Only one window can have the focus. When you execute other functions such as $\underline{Simulate}$ or $\underline{Program Chip}$ (75), BASCOM will use the files that belong to the current active program. This is in most cases the program which has the focus.

File new shortcut: 📗	, CTRL	+	Ν
----------------------	--------	---	---

3.3 File Open

50

With this option you can load an existing program from disk.

BASCOM saves files in standard ASCII format. Therefore, if you want to load a file that was made with another editor be sure that it is saved as an ASCII file. Most programs allow you to export the file as a DOS or ASCII file.

Note that you can specify that BASCOM must reformat the file when it opens it with the Options Environment option. This should only be necessary when loading files made with another editor.

File open shortcut : 🏓, CTRL+O

3.4 File Close

Close the current program.

The current editor window will be closed. When you have made changes to the program, you will be asked to save the program first. You can then decide to save, cancel, or not to save the changes you have made.

File close shortcut :

3.5 File Save

With this option, you save your current program to disk under the same file name. The file name is visible in the Windows caption of the edit window.

If the program was created with the <u>File New</u> option, you will be asked to name the file first. Use the <u>File Save As</u> option to give the file another name.

Note that the file is saved as an ASCII file.

File save shortcut : 🗐, CTRL+S

3.6 File Save As

With this option, you can save your current program to disk under a different file name.

When you want to make some changes to your program, but you do not want to make changes to the current version you can use the "Save As" option. It will leave your program as it was saved, and will create a new file with a new name so you end up with two copies. You then make changes to the new created file.

Note that the file is saved as an ASCII file.

File save as shortcut : 🖺

© 2011 MCS Electronics

3.7 File Print Preview

With this option, you can preview the current program before it is printed. Note that the current program is the program that has the focus.

File print preview shortcut : 🞑

3.8 File Print

With this option, you can print the current program. Note that the current program is the program that has the focus.

File print shortcut : 🍓, CTRL+P

3.9 File Exit

With this option, you can leave BASCOM. If you have made changes to your program, you can save them upon leaving BASCOM.

All of the files you have open, at the moment you choose exit, will be remembered. The next time you run BASCOM, they will be opened automatically.

File exit shortcut : 歸

3.10 Edit Undo

With this option, you can undo the last text manipulation.

Edit Undo shortcut : 🧐, CTRL+Z

3.11 Edit Redo

With this option, you can redo the last undo.

Edit Redo shortcut : 💜, CTRL+SHIFT+Z

3.12 Edit Cut

With this option, you can cut selected text into the clipboard.

Edit cut shortcut : 😹, CTRL+X

3.13 Edit Copy

With this option, you can copy selected text into the clipboard.

Edit copy shortcut : ៉, CTRL+C

3.14 Edit Paste

52

With this option, you can paste text from the clipboard starting at the current cursor position.

Edit paste shortcut : 🛅, CTRL+V

3.15 Edit Find

With this option, you can search for text in your program.

Text at the current cursor position will automatically be placed in the find dialog box. All text you search for is saved so the next time you search, you can retrieve the search phrase from a list.

To clear the history, right click the mouse above the 'Text to Find' label and select 'Clear History' from the popup menu.

Edit Find shortcut : \swarrow , CTRL+F

3.16 Edit Find Next

With this option, you can search again for the last specified search item. Edit Find Next shortcut : \mathbb{A} , F3

3.17 Edit Replace

With this option, you can replace selected text in your program.

Edit Replace shortcut : 🏭, CTRL+R

3.18 Edit Goto

With this option, you can immediately go to a specified line number.

Edit go to line shortcut : 12,CTRL+G

3.19 Edit Toggle Bookmark

With this option, you can set/reset a bookmark, so you can jump in your code with the Edit Go to Bookmark option. Shortcut : CTRL+K + x where x can be 1-8

Bookmarks are stored in a file named <project>.BM

3.20 Edit Goto Bookmark

With this option, you can jump to a bookmark.

There can be up to 8 bookmarks. Shortcut : CTRL+Q+ x where x can be 1-8

Bookmarks are stored in a file named <project>.BM

3.21 Edit Indent Block

With this option, you can indent a selected block of text.

Edit Indent Block shortcut : 💷, CTRL+SHIFT+I

3.22 Edit Unindent Block

With this option, you can unindent a block.

Edit Unindent Block shortcut : 💷, CTRL+SHIFT+U

3.23 Edit Remark Block

With this option, you can Remark or Unremark a selected block of text. While you can use '(and ') to remark a block of code, you might prefer the old BASIC way using just one '.

When a remark is found, it will be removed. When there is no remark, it will insert a remark.

3.24 Edit Encrypt Selected Code

This option allows you to encrypt portions of your code.

Because the encryption can not be undone, you will get this warning:

Warning	
	Encryption can NOT be undone, are you sure?
	Yes <u>N</u> o

If you chose YES, the selected code will be encrypted and will result in lines like :

\$CRYPT 6288E522B4A1429A6F16D639BFB7405B \$CRYPT 7ABCF89E7F817EB166E03AFF2EB64C4B \$CRYPT 645C88E996A87BF94D34726AA1B1BCCC \$CRYPT 9405555D91FA3B51DEEC4C2186F09ED1 \$CRYPT 6D4790DA2ADFF09DE0DA97C594C1B074

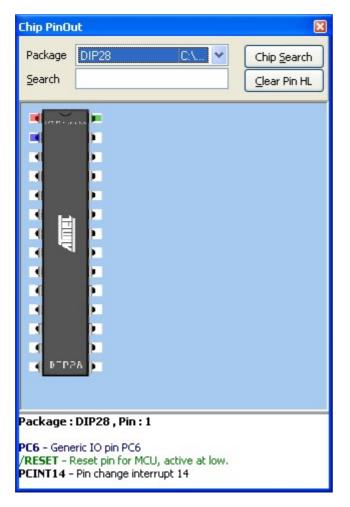
Only the compiler can decrypt and process these lines. There is no way you can change the \$CRYPT lines back into source code !

So make a backup of your code before you use this option. Typically, it will only be used on finished projects.

If the encrypted code contains errors, you will get error messages pointing to the $\frac{CRYPT}{315}$ lines.

3.25 View PinOut

The Pin Out viewer is a dock able window that shows the case of the active chip. The active chip is determined by the value of $\frac{REGFILE}{SO}$.



When you move the mouse cursor over a pin, you will see that the pin will be colored red. At the bottom of the window, a pin description is show. In the sample above you will see that each line has a different color. This means that the pin has multiple alternative functions.

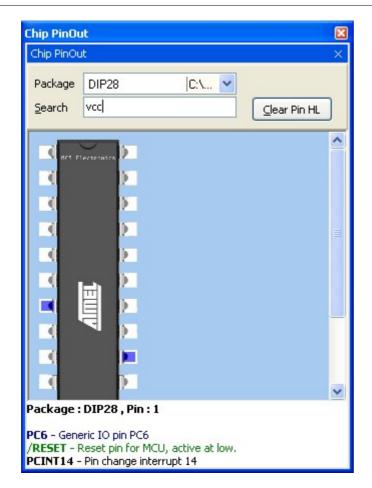
The first blue colored function is as generic IO pin.

The second green colored function is RESET pin.

The third black colored function is PIN change interrupt.

A pin can have one or more functions. Some functions can be used together. When you move the mouse cursor away, the pin will be colored blue to indicate that you viewed this pin. For example, when you need to look at it again.

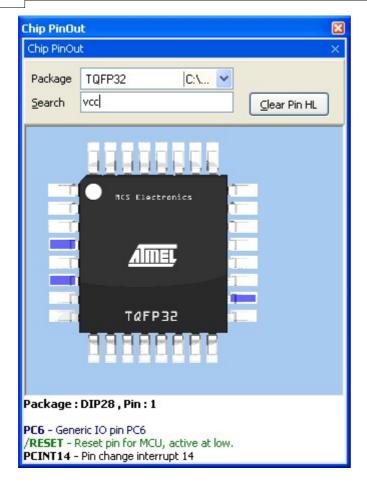
You can also search for a pin description. Enter some text and return. Here is an example when you search the VCC pin :



When pins are found that have the search phrase in the description, the pin will be colored blue.

By clicking 'Clear Pin HL' you can clear all colored pins.

Some chips might have multiple cases. You can select the case from the package list.



When you change from package, all pin colors will be cleared.

When you double click a pin, the pin will be colored green. Another double click will color it red/blue.

When a pin is green, it will not be colored red/blue. The green color serves as a kind of bookmark.

The only exception is the search function. It will make bookmarked green pins, blue too.

Use the right mouse to access a popup menu. This menu allows you to zoom the image to a bigger or smaller size.

Double click the chip to show the chip data.

BASCO	M-AVR	
	Chip : ATmega88 FLASH : 8 kbytes EEPROM : 0.5 kbytes SRAM : 1024 bytes I/O Pins : 23 Fmax : 20 MHz Vcc : 1.8-5.5 V Timer 16bit : 1 Timer 8bit : 2 PWM channels : 6 RTC : 1 SPI : 1 USART : 1 UART : 1 TWI : 1 ISP : 1 A/D channels : 8 Analog comparator : 1 BrownOut : 1 Watchdog : 1	
	Interrupts : 26 Ext.interrupts : 26	
		ОК

When you want to search for a chip, click the 'Chip Search' button. It will show the following window:

🚟 Chip Search							
	ICD			A			
Flash	ISP			Analog comp.			
SRAM	RTC	2		PWM channel	s		
EEPROM	TW			A/D channels			
1/O pins	SPI			Ext. Interrupts			
8-bit timers	Brov	vn Out		Fmax			
16-bit timers Interrupts			Vcc			Search	
UARTS 2	Wat	chdog		USART			Clear
	Package	Flash	SRAM	EEPROM	1/0	Fmax	16 bit 📩
ATmega128	TQFP64	128	4096	4	53	16	2
ATmega1280	TQFP100	128	8192	4	86	16	4
ATmega1281	TQFP64	128	8192	4	54	16	4
ATmega2560	TQFP100	256	8192	4	86	16	4
ATmega2561	TQFP64	256	8192	4	54	16	4
ATmega324	DIP40	32	2048	1	32	20	1
ATmega324	TQFP44	32	2048	1	32	20	1
ATmogo224D	Iniown	22	00.00	1	22	20	1
							>

You can provide criteria such as 2 UARTS. All criteria are OR-ed together. This means that when one of the criteria is met, the chip will be included in the list.

Only chips supported by BASCOM will be listed. When a chip has SRAM, and is not supported yet, it will be in the near future since the goal is to support all chips.

When you find an error in the pin description, please send an email to support so it can be corrected.

3.26 View PDF viewer

The PDF viewer is dock able panel which is located by default on the right side of the IDE.

	ער בער בער בער בער בער בער בער בער בער ב
Sub Sub Sub Sub Sub Sub Sub Sregfile = "m8515.dat" Dim A As Eram Byte Dim B As Eram Integer Dim C As Eram Iong Dim D As Eram Single Sff Print Version() Seeprom Data 1 , 1223% , 12345678& , End	PDF Viewer PX Label Image: Second
14: 25 Modified Insert Errors : 1 found	Analog Com Analog-to-D Analog-to-D Com S

The viewer itself contains a tree with the topics and the actual PDF viewer. The tree topics can be searched by right clicking on the tree. Choose 'Search' and enter a search text.

When a topic has sub topics, the topic is **bold**.

When you have enabled 'Auto open Processor PDF' in Options, Environment, PDF, the data sheet will be automatically loaded when you change the \$REGFILE value. It can be shown in a new sheet or it can replace the current PDF.

2	Open a PDF.
	Copy selected text to the clipboard. You can not copy from protected PDF documents.
	First page.

0	Previous page.
Y ∕+	Current page indicator. You can enter a page number to jump to a different page.
0	Next page.
	Last page.
	Find text in PDF.
	Zoom in.
	Zoom out.
	Rotate page to the left and right.
	Print page(s).

When you right click in the PDF, a pop up menu with the most common options will appear.

In <u>Options, Environment, PDF</u> you can specify how data sheets must be downloaded.

Data sheets can be downloaded automatic. When the \$REGFILE is changed and the PDF is not present, you will be asked if the PDF must be downloaded. If you choose to download, it will be downloaded from the Atmel website.

BASCOM-AVR			
?	Download PDF		
~	Do you want to download PDF atmega32.pdf?		
	Yes No		
Do not show this message again			

When you click 'Do not show this message again' , you will not be asked anymore if you want to download the Mega32.PDF. You will be asked to download other PDF documents when they do not exist.

During the download you will see a similar window:

PDF Download	
Downloading PDF	
Downloading atmega32.pdf	
(*********	
	Cancel

You can also download all newer PDF's from the Atmel website with the option : <u>Tools, PDF Update</u>

When PDF's are downloaded with the UpdateWiz, they are downloaded from the MCS Electronics website.

3.27 View Error Panel

This option will show the Error panel.



When there are no errors, the list will be empty. You will also be able to close the window.

When there are errors :

Errors			×
Error:1	Line :	8	Unknown statement [SFF] , in File : C:\DATA\AP

You will not be able to close the window until the error is solved and the program is checked/compiled.

The panel is dockable and by default docked to the bottom of the IDE.

3.28 View Tip

Action

Shows the Tip of the day Window

[12] Tip of the day : HTML Help				
	Did you know			
	That besides the HLP files, BASCOM supports CHM files? In O ptions, Enviroment, IDE, you can choose to enable the HTML help. A restart of BASCOM is needed after you change the setting. Because VISTA will not support the HLP format, we advise to switch to the CHM/HTML help.			
Show tips at startup Next Close				

You can click the Next-button to show another tip. Or you can close the window. When you do not want to see the tips when BASCOM is started, you can unselect the 'Show tips at startup' option.

You can submit your own tips at the register : <u>http://register.mcselec.com</u>

3.29 Program Compile

With this option, you compile your current program.

Your program will be saved automatically before being compiled.

The following files will be created depending on the Option Compiler Settings.

File	Description		
xxx.BIN	Binary file which can be programmed into the microprocessor		
xxx.DBG	Debug file that is needed by the simulator.		
xxx.OBJ	Object file for simulating using AVR Studio. Also needed by the internal simulator.		
xxx.HEX	Intel hexadecimal file, which is needed by some programmers.		
xxx.ERR	Error file. Only created when errors are found.		
xxx.RPT	Report file.		
xxx.EEP EEPROM image file			

If a serious error occurs, you will receive an error message in a dialog box and the compilation will end.

All other errors will be displayed at the bottom of the edit window, just above the status bar.

When you click on the line with the error info, you will jump to the line that contains the error. The margin will also display the \bigcirc sign.

At the next compilation, the error window will disappear or reappear if there are still errors.

See also <u>'Syntax Check'</u> for further explanation of the Error window.

Program compile shortcut:

3.30 Program Syntax Check

With this option, your program is checked for syntax errors. No file will be created except for an error file, if an error is found.

Program syntax check shortcut \mathbf{M} , CTRL + F7

When there is an error, an error window will be made visible at the bottom of the screen.

🞆 BAS	SCOM-AVR IDE [1.11.8.5]				
<u>Eile</u>	Eile Edit View Program Tools Options Window Help				
I 🗋 🖻	▶ 🕞 😫 🌭 🖻 🗣 + 1 🗴 🛍 📋 津 津 🔎 🖧 + 1 🧄 🛅 × 🗐 义 🛲 夹 💽 📀 夹	▶ *			
🚟 C:\	\DATA\apps\D7\avride\noname1.bas	^			
Sub	▼ Label	PDF Viewer			
	\$regfile = "m8515.dat"	liewe			
	Din A As Eran Byte Din B As Eran Integer Din C As Eran Long Din D As Eran Single	-			
	ads Print Version()				
	Seepron Data 1 , 1223% , 12345678& , 123.45!				
	End				
8: 15		×			
Errors					
Error : 1	Error : 1 Line : 8 Unknown statement [ADS] , in File : C:\DATA\APPS\D7\AVRIDE\NONAME1.BAS				

You can double click the error line to go to the place where the errors is found. Some errors point to a line zero that does not exist. These errors are caused by references to the assembler library and are the result of other errors.

The error window is a dockable window that is docked by default to the bottom of the screen. You can drag it outside this position or double click the caption(Errors) to make it undock :

Errors		×
Error:1	Line: 8	Unknown statement [ADS] , in File : C:\DATA\APP

Here the panel is undocked. Like most windows you can close it. But the error must be resolved (corrected and syntax checked/recompiled) for this window can be closed !

By double clicking the caption (top space where the name of the window is show) you can dock it back to it's original position.

When you have closed the window and want to view it again, you can choose the

View, Error Panel option from the main menu.

3.31 Program Show Result

Use this option to view information concerning the result of the compilation.

See the Options Compiler Output [91] for specifying which files will be created.

The files that can be viewed are "report" and "error".

File show result shortcut : 🖾,CTRL+W

Info	Description		
Report	Name of the program		
Date and time	The compilation date and time.		
Compiler The version of the compiler.			
Processor	The selected target processor.		
SRAM	Size of microprocessor SRAM (internal RAM).		
EEPROM	Size of microprocessor EEPROM (internal EEPROM).		
ROMSIZE	Size of the microprocessor FLASH ROM.		
ROMIMAGE	Size of the compiled program.		
BAUD	Selected baud rate.		
XTAL	Selected XTAL or frequency.		
BAUD error	The error percentage of the baud rate.		
XRAM	Size of external RAM if available.		
Stack start	The location in memory, where the hardware stack points to. The HW-stack pointer grows downward.		
S-Stacksize	The size of the software stack.		
S-Stackstart	The location in memory where the software stack pointer points to. The software stack pointer grows downward.		
Framesize	The size of the frame. The frame is used for storing local variables.		
Framestart	The location in memory where the frame starts.		
LCD address	The address that must be placed on the bus to enable the LCD display E-line.		
LCD RS	The address that must be placed on the bus to enable the LCD RS-line		
LCD mode	The mode the LCD display is used with. 4 bit mode or 8 bit mode.		
LCD DB7-DB4	The port pins used for controlling the LCD in pin mode.		
LCD E	The port pin used to control the LCD enable line.		
LCD RS	The port pin used to control the LCD RS line.		
Variable	The variable name and address in memory		
Constant	Constants name and value		
	Some internal constants are :		
	_CHIP : number that identifies the selected chip _RAMSIZE : size of SRAM		

Information provided in the report:

	_ERAMSIZE : size of EEPROM _XTAL : value of crystal _BUILD : number that identifies the version of the compiler _COMPILER : number that identifies the platform of the compiler
Warnings	This is a list with variables that are dimensioned but not used. Some of them
EEPROM binary image map	This is a list of all ERAM variables with their value. It is only shown when $DATA$ and $IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII$

3.32 Program Simulate

With this option, you can simulate your program.

You can simulate your programs with AVR Studio or any other Simulator available or you can use the built in Simulator.

The simulator that will be used when you press F2, depends on the selection you made in the Options Simulator TAB. The default is the built in Simulator.

Program Simulate shortcut : 🎒, F2

To use the built in Simulator the files DBG and OBJ must be selected from the Options Compiler Output TAB.

The OBJ file is the same file that is used with the AVR Studio simulator.

The DBG file contains info about variables and many other info needed to simulate a program.

AVR Simulator Variable		Simulator				
Variable Locals W Watch W P Interrupts /ariable Value Hex Bin /ariable Value						
Variables Locals & Watch & uP Interrupts Avridue Value Hex Bin Avridue 25 19 00011001 Avridue 25 19 00011001 Avridue 25 19 00011001 Avridue 25 19 0001000 Avridue 25 19 00000000 UARTO UARTO UARTO UARTO UARTO UARTO UARTO UARTO 33 34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Coml = Durmay, Synchrone = 0 , Parity = None , Stopbits = 1 , Datal 37 Config Coml = Durmay , Synchrone = 0 , Parity = None , Stopbits = 1 , Datal 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Unput 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UARTI"		n = 93 (3 - 5	🕽 🌭 R IO M 🗌	🎇 🗌 Terminal	🗌 Sim Timers 🗧	
Value Hex Bin ArnOl 25 19 00011001 AdmOn 0 0 00000000 AdmOn 0 0 00000000 0 AdmOn 0 0 00000000 0 AdmOn 0 0 00000000 0 AdmOn 0 0 0 0 AdmOn				<u></u>		
25 19 00011001 dr0h 0 0 00000000 dr1h 25 19 00011001 dr1h 0 0 00000000 dr1h 0 0 00000000 dr1h 0 0 00000000 dr1h 0 0 00000000 uART0 0 0 00000000 uART1 14 14 14 10 0 0 0 0 33 34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 So 'config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Datab 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Datab 37 Config Com2 : " For Binary As #1 1 1 1 1 1 42 'try to access an extended register 43 Config Portf = Input 45 6 6 Print "Hello" 4 41 'Config Portf = Input 45 0 9 Print "UARTO" 5 0 Print #1 , "UARTI" 5 <td></td> <td></td> <td></td> <td>1</td> <td></td> <td></td>				1		
demOh 0 0 00000000 demTh 25 19 00011001 demTh 0 0 00000000 demTh 0 0 00000000 demTh 0 0 00000000 UARTO UARTI fello 0 0 00000000 Jartit 0 0 0 0 Jartit 1 0 0 0 Jartit 0 0 0 0 Jartit 0 0 0 0 0 Jartit 0 0 0 0 0 Jartit 0 0 0 0 0 0 Jartit 0 0 0 0 0						_
abril 25 19 00011001 abril 0 0 00000000 abril 0 0 0 00000000 abril 0 0 0 00000000 abril 0 0 0 0 0 abril 0 0 0 0 0 0 abrit 0 0						
dwnhh 0 0 00000000 uUARTO UARTO Hello 0 00000000 33 34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Datab 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Datab 38 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 'config Portf = Output 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 A 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print #1 , "UARTO" 50 Print #1 , "UART1"						
UARTO UARTO UARTO ARTO UARTO 33 34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy, Synchrone = 0, Parity = None, Stopbits = 1, Data 37 Config Com2 = Dummy, Synchrone = 0, Parity = None, Stopbits = 1, Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary &s #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 9 Af Print "Hello" 47 Dim B &s Byte 48 Do 49 Print "UARTO" 50 Print #1, "UARTI"						
<pre>33 34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Output 44 'Config Portf = Input 45 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1, "UARTI" </pre>						~
<pre>33 34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1, "UARTI" </pre>						
<pre>33 34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary &s #1 41 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Output 45 46 Print "Hello" 47 Dim B &s Byte 48 Do 49 Print "UARTO" 50 Print #1, "UART1" 54 The Second Secon</pre>						
<pre>34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1"</pre>						<u>-</u>
<pre>34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1"</pre>						
<pre>34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1"</pre>						
<pre>34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary &s #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B &s Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1"</pre>						N
<pre>34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary &s #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B &s Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UARTI"</pre>	<					>
<pre>34 'The M128 has an extended UART. 35 'when CONFIG COMx is not used, the default N,8,1 will be used 36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1"</pre>	-	33				
<pre>36 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary &s #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Output 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1" 50 Print #1 , "UART1"</pre>		34 'The M128 has an	extended UART.			
<pre>37 Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Data 38 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UARTI"</pre>						
<pre>38 39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UARTI"</pre>		35 'when CONFIG COM	n ze mee ween,	the default N,8,:	l will be used	
<pre>39 'try the second hardware UART 40 Open "com2:" For Binary As #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UARTI"</pre>	- I					= 1 , Datał
<pre>40 Open "com2:" For Binary &s #1 41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B &s Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1" </pre>		36 Config Com1 = Du	mmy , Synchrone	= 0 , Parity $= 1$	None , Stopbits =	
<pre>41 42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1" 50 Print #1 , "UART</pre>	•	36 Config Com1 = Du 37 Config Com2 = Du 38	mmy , Synchrone mmy , Synchrone	= 0 , Parity $= 1$	None , Stopbits =	
<pre>42 'try to access an extended register 43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1" 50 Print #1 , "UART1"</pre>)	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second	mmy , Synchrone mmy , Synchrone hardware UART	= 0 , Parity $= 1$	None , Stopbits =	
<pre>43 Config Portf = Output 44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1" 50 Print #1 , "UART1"</pre>	•	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For	mmy , Synchrone mmy , Synchrone hardware UART	= 0 , Parity $= 1$	None , Stopbits =	
44 'Config Portf = Input 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1, "UART1"		36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1, "UART1" 50 Print #1, "UART1"	D	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
<pre>46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "UART1" 51 P Tober() 53 Print #1 / **********************************</pre>		36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a 43 Config Portf = O	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi utput	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1, "UART1" 51 D T TOTON))	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a 43 Config Portf = O 44 'Config Portf =	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi utput	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
48 Do 49 Print "UARTO" 50 Print #1, "UART1" 51 D T TOTON	5	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a 43 Config Portf = O 44 'Config Portf =	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi utput	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
49 Print "UARTO" 50 Print #1 , "UART1" 51 D	D :	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a 43 Config Portf = O 44 'Config Portf = 45 46 Print "Hello"	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi utput	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
50 Print #1 , "UART1" 51 D 52 D))	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a 43 Config Portf = O 44 'Config Portf = 45 46 Print "Hello" 47 Dim B As Byte	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi utput	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
	0	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a 43 Config Portf = O 44 'Config Portf = 45 46 Print "Hello" 47 Dim B As Byte 48 Do	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi utput Input	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
)))	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a 43 Config Portf = O 44 'Config Portf = O 44 'Config Portf = 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO"	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi utput Input	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	
= 0085 Cycl. = 25030 : 3.12875 mS Pause Move mouse over variable to show value	•	36 Config Com1 = Du 37 Config Com2 = Du 38 39 'try the second 40 Open "com2:" For 41 42 'try to access a 43 Config Portf = O 44 'Config Portf = 45 46 Print "Hello" 47 Dim B As Byte 48 Do 49 Print "UARTO" 50 Print #1 , "U	mmy , Synchrone mmy , Synchrone hardware UART Binary As #1 n extended regi utput Input	= 0 , Parity = M = 0 , Parity = M	None , Stopbits =	= 1 , Data

The Simulator window is divided into a few sections:

The Toolbar

The toolbar contains the buttons you can press to start an action.

This is the RUN button, it starts a simulation. You can also press F5. The simulation will pause when you press the pause button. It is advised, that you step through your code at the first debug session. When you press F8, you step through the code line by line which is a clearer way to see what is happening.

This is the PAUSE button. Pressing this button will pause the simulation.

This is the STOP button. Pressing this button will stop the simulation. You can't continue from this point, because all of the variables are reset. You need to press the RUN button when you want to simulate your program again.

This is the STEP button. Pressing this button (or F8) will simulate one code line of your BASIC program. The simulator will go to the RUN state. After the line is

66

executed the simulator will be in the PAUSE state. If you press F8 again, and it takes a long time too simulate the code, press F8 again, and the simulator will go to the pause state.

This is the STEP OVER button or SHIFT+F8). It has the same effect as the STEP button, but sub programs are executed completely, and th simulator does not step into the SUB program.

This is the RUN TO button. The simulator will RUN until it gets to the current line. The line must contain executable code. Move the cursor to the desired line before pressing the button.

This button will show the processor registers window.

Regis	ters	×		
Reg	Val	^		
R21	00			
R22	08			
R23	00			
R24	01			
R25	00			
R26	29			
R27	01			
R28	80			
R29	04			
R30	50	-		
R31	07			
Regi	sters IC			

The values are shown in hexadecimal format. To change a value, click the cell in the VAL column, and type the new value. When you right click the mouse, you can choose between the Decimal, Hexadecimal and Binary formats.

The register window will show the values by default in **black**. When a register value has been changed, the color will change into **red**. Each time you step through the code, all changed registers are marked **blue**. This way, the red colored value indicate the registers that were changed since you last pressed F8(step code). A register that has not been changed at all, will remain black.

This is the IO button and will show processor Input and Output registers.

10		X
REG	Val	^
PORTB	00	
PINC	00	_
DDRC	00	
PORTC	00	
PIND	00	
DDRD	00	
PORTD	04	
TIFRO	00	
TIFR1	02	
TIFR2	00	
PCIFR	00	
CICD	00	
Registers	ю	

The IO window works similar to the Register window.

A right click of the mouse will show a popup menu so you can choose the format of the values.

And the colors also work the same as for the registers : black, value has not been changed since last step(F8). Red : the value was changed the last time your pressed F8. Blue : the value was changed since the begin of simulation. When you press the STOP-button, all colors will be reset to black.

 $^{\rm M}$ Pressing this button shows the Memory window.

Memory	Y																	X
SRAM	EE	PRC	M															
	00	01	02	03	04	05	06	07	08	09	0A	OB	0C	OD	0E	OF		^
0100	00	00	00	00	00	00	00	00	00	00	00	00	00	08	00	00		
0110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0120	00	00	00	00	00	00	00	00	01	00	63	63	00	00	00	00	cc	
0130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
0160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		(38)
0170	lon	nn	nn	nn	nn	00	00	00	00	nn	00	nn	00	00	nn	00)	
Occup	ied	by	:1															

The values can be changed the same way as in the Register window.

When you move from cell to cell you can view in the status bar which variable is stored at that address.

The SRAM TAB will show internal memory and XRAM memory.

The EEPROM TAB will show the memory content of the EEPROM.

The colors work exactly the same as for the register and IO windows. Since internal ram is cleared by the compiler at startup, you will see all values will be colored blue. You can clear the colors by right clicking the mouse and choosing 'Clear Colors'.

The refresh variables button will refresh all variables during a run (F5). When you use the hardware simulator, the LEDS will only update their state when you have

enabled this option. Note that using this option will slow down simulation. That is why it is an option. When you use F8 to step through your code you do not need to turn this option on as the variables are refreshed after each step.

Sim Timers When you want to simulate the processors internal timers you need to turn this option on. Simulating the timers uses a lot of processor time, so you might not want this option on in most cases. When you are debugging timer code it is helpful to simulate the timers.

The simulator supports the basic timer modes. As there are many new chips with new timer modes it is possible that the simulator does not support all modes. When you need to simulate a timer the best option may be to use the latest version of AVR Studio and load the BASCOM Object file.

Even AVR Studio may have some flaws, so the best option remains to test the code in a real chip.

Terminal This option allows you to use a real terminal emulator for the serial communication simulation.

Normally the simulator prints serial output to the blue window, and you can also enter data that needs to be sent to the serial port.

When you enable the terminal option, the data is sent to the actual serial port, and when serial data is received by the serial port, it will be shown.

Under the toolbar section there is a TAB with a number of pages:

🕌 AVR Simulate	Dr			
)) =	۴ 🛄 🖣	RIOM	🕍 🗌 Sim Ti	mers 🗌 Terminal 🗧
Variables Local	s 😚 Watch 🦠 uP	Interrupts		
Variable	Value	Hex	Bin	
portb	1	1	0000001	
<u></u>				
T.				
	stack = 10			' defa
15 \$fra	amesize = 40			' defa

VARIABLES

This section allows you to see the value of program variables. You can add variables by double clicking in the Variable-column. A list will pop up from which you can select the variable.

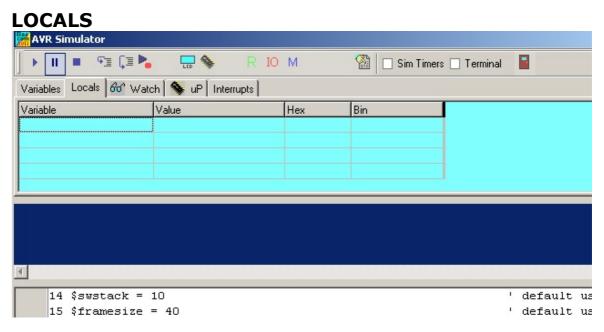
To watch an array variable, type the name of the variable with the index.

During simulation you can change the values of the variables in the Value-column, Hex-column or Bin-column. You must press ENTER to store the changes.

To delete a variable, you can press CTRL+DEL.

To enter more variables, press the DOWN-key so a new row will become visible.

It is also possible to watch a variable by selecting it in the code window, and then pressing enter. It will be added to the variable list automatically. Notice that it takes time to refresh the variables. So remove variables that do not need to be watched anymore for faster simulation speed.



The LOCALS window shows the variables found in a SUB or FUNCTION. Only local variables are shown. You can not add variables in the LOCALS section. Changing the value of local variables works the same as in the Variables TAB.

WATCH

M 🔛	AVR Simulator							- 🗆 🗙
	• 🔳 • 🕫 (I 🍋 🛛 🖶 🦠	RIOM		Sim Timers	Terminal			
Varia	iables Locals 😚 Watch 💊 uP Inte	errupts						
	ortb>10		Add	Modify	J			
			Remove					
			portb>10		1			
								<u>^</u>
								-
₹.								Þ
	14 \$swstack = 10				1	default	use	10 fc 🔺
	15 \$framesize = 40					default		
	16							
	17							
0	18 Config Portb = Output 19							
	20							
	21 Do							
0	22 Portb = Portb + 1							
->	23 Loop Until Inkey() = 2	7						-1
								· •
PC =	003D Cycl. = 40654 : 10.1635 mS P	Pause						1.

The Watch-TAB can be used to enter an expression that will be evaluated during simulation. When the expression is true the simulation is paused.

To enter a new expression, type the expression in the text-field below the Remove button, and press the Add-button.

When you press the Modify-button, the current selected expression from the list will be replaced with the current typed value in the text field.

To delete an expression, select the desired expression from the list, and press the Remove-button.

During simulation when an expression becomes true, the expression that matches will be selected and the Watch-TAB will be shown.

uP

🚟 AYR Simul	ator						
) =	با 🕄 🗈		RI	O M		🗌 🗌 Sim Timers 🔲 Termi	nal 冒
Variables Loc	cals 60° Watch	🦠 uP 🛛 Inte	rrupts				
Flags							
	✓ H □ S □	V 🗌 N 🗌 Z	✓ C	🗌 Fran	ne Overflow	Stack Overflow	
Soft Stack	02E0 Soft	Stack Min 028	0	HW Stack	< 02FF	HW Min 02FF	
Frame pointer	02C6 Fram	ie Max 🛛 🛛 🛛 🖓	26		Snapshot]	

This TAB shows the value of the microprocessor status register (SREG).

The flags can be changed by clicking on the check boxes.

The software stack, hardware stack, and frame pointer values are shown. The minimum or maximum value that occurred during simulation is also shown. When

one of these data areas enter or overlap another one, a stack or frame overflow occurs.

This will be signaled with a pause and a check box.

Pressing the snapshot-button will save a snapshot of the current register values and create a copy of the memory.

You will notice that the Snapshot-button will change to 'Stop'

Now execute some code by pressing F8 and press the Snapshot-button again.

A window will pop up that will show all modified address locations. This can help to determine which registers or memory a statement uses.

Snapshot
Changed memory positions:
Address : \$18, old value :99 , new value :0 Address : \$25, old value :99 , new value :101 Address : \$5F, old value :33 , new value :53

When you write an ISR (Interrupt Service Routine) with the NOSAVE option, you can use this to determine which registers are used and then save only the modified registers.

INTERRUPTS

► II	• 93	Ç 🛛 🍋	LED	•	RIO	M		🗌 Sim T	imers 🗌 Te	erminal		
ariables	Locals	for Watch	uP	Interrupt	s							
INTO	INT1	PCINTO	PCINT1	PCINT2	WDT	OC2A	OC28		Pulse	1	1/4	
OVF2	ICP1	OC1A	OC1B	OVF1	0C0A	O COB	0VF0		Delay	1	1/4	
SPI	URXC	UDRE	UTXC	ADCC	ERDY	ACI	TWI	SPM	Pin	PORT	D.4 -	

This TAB shows the interrupt sources. When no ISR's are programmed all buttons will be disabled.

When you have written an ISR (using ON INT...), the button for that interrupt will be enabled. Only the interrupts that are used will be enabled.

By clicking an interrupt button the corresponding ISR is executed. This is how you simulate the interrupts. When you have enabled 'Sim Timers' it can also trigger the event.

The pulse generator can be used to supply pulses to the timer when it is used in

counter mode.

First select the desired pin from the pull down box. Depending on the chip one or more pins are available. Most chips have 2 counters so there will usually be 2 input pins.

Next, select the number of pulses and the desired delay time between the pulses, then press the Pulse-button to generate the pulses.

The delay time is needed since other tasks must be processed as well.

The option 'Sim timers' must be selected when you want to simulate timers/counters.

TERMINAL Section

Under the window with the TABS you will find the terminal emulator window. It is the dark blue area.

In your program when you use PRINT, the output will be shown in this window.

When you use INPUT in your program, you must set the focus to the terminal window and type in the desired value.

You can also make the print output go directly to the COM port. Check the Terminal option to enable this feature.

The terminal emulator settings will be used for the baud rate and COM port. Any data received by the COM port will also be shown in the terminal emulator window.

Notice that most microprocessors have only 1 UART. The UART0-TAB is used to communicate with tis UART. The UART1-TAB need to be selected in order to view the UART1 output, or to send data to UART1.

Software UARTS are not supported by the simulator. They can not be simulated.

SOURCE Section

Under the Terminal section you find the Source Window.

It contains the source code of the program you are simulating. All lines that contain executable code have a yellow point in the left margin.

You can set a breakpoint on these lines by selecting the line and pressing F9.

By holding the mouse cursor over a variable name, the value of the variable is shown in the status bar.

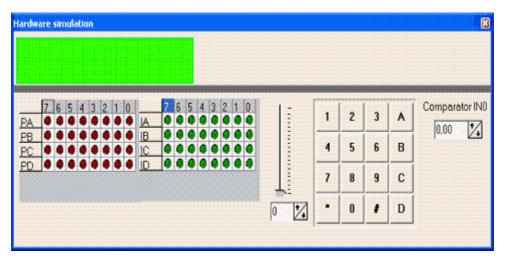
If you select a variable, and press ENTER, it will be added to the Variable window.

In order to use the function keys (F8 for stepping for example), the focus must be set to the Source Window.

A blue arrow will show the line that will be executed next..

The hardware simulator.

By pressing the hardware simulation button the windows shown below will be displayed.



The top section is a virtual LCD display. It works to display code in PIN mode, and bus mode. For bus mode, only the 8-bit bus mode is supported by the simulator.

Below the LCD display area are LED bars which give a visual indication of the ports.

By clicking an LED it will toggle.

PA means PORTA, PB means PORTB, etc.

IA means PINA, IB means PINB etc. (Shows the value of the Input pins) It depends on the kind of microprocessor you have selected, as to which ports will be

shown.

Right beside the PIN led's, there is a track bar. This bar can be used to simulate the input voltage applied the ADC converter. Note that not all chips have an AD converter. You can set a value for each channel by selecting the desired channel below the track bar.

Next to the track bar is a numeric keypad. This keypad can be used to simulate the GETKBD() function.

When you simulate the Keyboard, it is important that you press/click the keyboard button **before** simulating the getkbd() line !!!

To simulate the Comparator, specify the comparator input voltage level using Comparator INO.

Enable Real Hardware Simulation

By clicking the button you can simulate the actual processor ports in-circuit! The processor chip used must have a serial port.

In order simulate real hardware you must compile the basmon.bas file.

To do this, follow this example: Lets say you have the DT006 simmstick, and you are using a 2313 AVR chip.

Open the basmon.bas file and change the line REGFILE = xxx to REGFILE = 2313def.dat

Now compile the program and program the chip.

It is best to set the lock bits so the monitor does not get overwritten if you accidentally press F4.

The real hardware simulation only works when the target micro system has a serial

port. Most have and so does the DT006.

Connect a cable between the COM port of your PC and the DT006. You probably already have one connected. Normally it is used to send data to the terminal emulator with the PRINT statement.

The monitor program is compiled for 19200 baud. The Options Communication settings must be set to the same baud rate! The same settings for the monitor program are used for the Terminal emulator, so select the COM port, and the baud rate of 19200.

Power up or reset the DT006. It probably already is powered since you just previously compiled the basmon.bas program and stored it in the 2313. When you press the real hardware simulation button now the simulator will send and receive data when a port, pin or DDR register is changed.

This allows you to simulate an attached hardware LCD display for example, or something simpler, like an LED. In the SAMPLES dir, you will find the program DT006. You can compile the program and press F2.

When you step through the program the LED's will change!

All statements can be simulated this way but they have to be able to use static timing. Which means that 1-wire will not work because it depends on timing. I2C has a static bus and thus will work.

NOTE: It is important that when you finish your simulation sessions that you click the button again to disable the Real hardware simulation.

When the program hangs it probably means that something went wrong with the serial communication. The only way to escape is to press the Real hardware

Simulation Multiple Simulation again.

The Real Hardware Simulation is a cost effective way to test attached hardware.

The refresh variables button will refresh all variables during a run(F5). When you use the hardware simulator, the LEDS will only update their state when you have enabled this option. Note that using this option will slow down the simulation.

Watchdog Simulation

Most AVR chips have an internal Watchdog. This Watchdog timer is clocked from an internal oscillator. The frequency is approximately 1 MHz. Voltage and temperature variations can have an impact on the WD timer. It is not a very precise timer. So some tolerance is needed when you refresh/reset the WD-timer. The Simulator will warn you when a WD overflow will occur. But only when you have enabled the WD timer.

The status bar

PC = 003D Cycl. = 40677 : 10.16925 mS Pause

The status bar shows the PC (program counter) and the number of cycles. You can reset the cycles by positioning the mouse cursor on the status bar and then right click. You will then get a pop up menu with the option to reset the cycles. You can use this to determine how much time a program statement takes. Do not jump to a conclusion too quick, the time shown might also depend on the value of a variable.

For example, with WAITMS var this might be obvious, but with the division of a value the time might vary too.

3.33 Program Send to Chip

Program send to chip shortcut 🔳, F4

This option will bring up the selected programmer window, or will program the chip directly if the 'Auto Flash' option is selected in the <u>Programmer options</u> [10[†]] section.

The following section applies to the Programmer window (program chip directly NOT selected) otherwise this is not shown to the user.

"Buffer" below refers to the buffer memory that holds data to be programmed to, or read from the chip.

Menu item	Description				
File Exit	Return to editor				
File, Test	With this option you can set the logic level to the LPT pins. This is only intended for the Sample Electronics programmer.				
Buffer Clear	Clears buffer				
Buffer Load from file	Loads a file into the buffer				
Buffer Save to file	Saves the buffer content to a file				
Chip Identify	Identifies the chip				
Write buffer into chip	Programs the buffer into the chip ROM or EEPROM				
Read chip code into buffer	Reads the code or data from the chips code memory or data memory				
Chip blank check	Checks if the chip is blank or erased				
Chip erase	Erase the content of both the program memory and the data memory				
Chip verify	Verifies if the buffer is the same as the chip program or data memory				
Chip Set lock bits	Writes the selected lock bits LB1 and/or LB2. Only an erase will reset the lock bits				
Chip auto program	Erases the chip and programs the chip. After the programming is completed, verification is performed.				

The following window will be shown for most programmers:

BASCOM-AVR

76

😹 AVI	r isi	p s'	IK p	orog	gran	nme	er.											
<u>File</u>	uffer	⊆	hip															
03		1			-		5	.	۳				Ch	ip	\ T9	0S2	313 🚽 🛃 🖉	
Manu Chip	ManufactorUnknown ATmega8515Flash ROM EEPROM2KB 128SizeProgrammed:20																	
Flash	RO	М	EE	PRO	DM	Lo	ick i	and	Fus	se E	Bits							
	00	01	02	03	04	05	06	07	08	09	0A	OB	OC	OD	OE	OF		~
0000	CO	10	95	18	95	18	95	18	95	18	95	18	95	18	95	18	Аннн	
0010	95	18	95	18	95	18	95	18	95	18	95	18	95	18	95	18	1111111	
0020	95	18	E5	8F	BF	8D	Ε4	CO	E3	E8	2E	4E	ΕO	82	BF	8E	I.ål¿läÀãè.Nàl¿l	
0030	ΕO	D2	ΕO	F2	2E	5F	EF	EE	ΕO	F1	E6	Α0	ΕO	BO	27	88	àÒàòïîàñæ à*'	
0040	93	8D	97	31	F7	E9	E1	89	B9	89	ΕO	80	BD	80	E1	88	III1÷éá[¹]à€½€á]	
0050	B9	84	24	66	E2	A0	EO	B2	ΕA	E8	EO	FO	DO	1C	DO	03	¹ \$fâ à²êèàðĐ.Đ.	
0060	DO	0A	94	F8	CF	FF	E2	A0	EO	B2	91	8D	23	88	FO	11	Ð, þølijia ærij#jð.	
0070	DO	05	CF	FB	95	08	EO	8D	DO	01	ΕO	8A	9B	5D	CF	FE	Ð.Ïúl.àlÐ.àll]Ïþ	
0080	B9	8C	95	08	97	31	F7	F1	95	08	94	68	F8	62	95	08	1 . 1÷ñ . høb .	
0090	94	E8	F8	62	95	08	DO	04	92	0D	20	00	F7	E1	95	08	jèøb∥.Ð.1÷áj.	
00A0	95	C8	96	31	20	00	95	08	32	30	31	2D	2D	33	30	32	iči1 .j.201302	
00B0	37	30	30	20	ЗA	30	33	31	34	ЗA	00	34	FF	FF	FF	FF	700 :0314:.4ÿÿÿÿÿ	
0000	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	<u> </u>	
00D0	IFF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	000000000000000000000000000000000000000	<u> </u>
AVR IS	SP S	тк	pro	gra	mm	er												
188 ROM	188 ROM 0 EPROM NONAME1.BIN																	

Note that a chip must be ERASED before it can be programmed.

By default the Flash ROM TAB is shown and the binary data is displayed. When you have an EEPROM in your project, the EEPROM TAB will show this data too.

The most important TAB is in many cases the Lock & Fuse Bits TAB. When you select it , the lock and fuse bits will be read.

<u>File B</u> uffer <u>⊂</u> hip						
	8 . 8 .		0 ATmega88			
	mel Tmega88	Flash ROM EEPROM	8 KB 512	Size III Programmed	:20	
FlashROM	PROM Lock an	d Fuse Bits				
🗖 Chip					Refresh	
Name	MEGA	88			Write LB	
Calibrati	on 0 89				vvnte Lb	
Lockbits					Write FS	
Lockbit 8	65 11:No	restrictions for SPM or LP	M accessing the b	oot loader section	- THICLO	
Lockbit 4	13 11:No	restrictions for SPM or LP	M accessing the a	pplication section	Write FSH	
Lockbit 2	21 11:No	memory lock features ena	bled for parallel ar	nd serial programmir 🔜		
Fusebits					Write FSE	
Fusebit (D 1:Divio	le Clock by 8 Disabled				
Fusebit E	B 1:CLO	CK Output disabled			Write PRG	
Fusebit ł		D:Int. RC Osc. 8 MHz; Sta	irt-up time PWRDV	VN/RESET: 6 CK/		
Fusebits High	jh			_		
AVR ISP STK programmer						

These Lock and Fuse bits are different in almost every chip !

You can select new settings and write them to the chip. But be careful ! When you select a wrong oscillator option , you can not program the chip anymore without applying an external clock signal.

This is also the solution to communicate with the chip again : connect a clock pulse to the oscillator input. You could use an output from a working micro, or a clock generator or simple 555 chip circuit.

When you found the right settings, you can use $\frac{PROG}{354}$ to write the proper settings to new, un-programmed chips. To get this setting you press the 'Write PRG' button.

After a new chip is programmed with \$PROG, you should remark the line for safety and quicker programming.

The 'Write PRG' will write the settings, read from the Microprocessor, it will NOT insert the unsaved settings you have made manual. Thus, you must first use the 'Write XXX' buttons to write the changed fuse bits settings to the chip, then you can use the 'Write PRG'.

Notice that the Write xxx buttons are disabled by default. Only after you have changed a lock or fuse bit value, the corresponding button will be enabled. You must click this button in order to apply the new Lock or Fuse bit settings.

Many new chips have an internal oscillator. The default value is in most cases 8 MHz. But since in most cases the 'Divide by 8' option is also enabled, the oscillator value will be 1 MHz. We suggest to change the 'Divide by 8' fuse bit so you will have a speed of 8 MHz.

In your program you can use $\frac{\text{scrystal}}{316} = 8000000$ then.

\$crystal will only inform the compiler which oscillator speed you have selected.

78 BASCOM-AVR

This is needed for a number of statements. \$crystal will NOT set the speed of the oscillator itself.

Do not change the fuse bit that will change the RESET to a port pin. Some chips have this option so you can use the reset pin as a normal port pin. While this is a great option it also means you can not program the chip anymore using the ISP.

3.34 Tools Terminal Emulator

With this option you can communicate via the RS-232 interface to the microcomputer. The following window will appear:

BASCOM-AVR Terminal emulator	
<u>File</u> <u>T</u> erminal	
	<u>^</u>
	>
COM1:38400,N,8,1	

Information you type and information that the computer board sends are displayed in the same window.

Note that you must use the same baud rate on both sides of the transmission. If you compiled your program with the Compiler Settings at 4800 baud, you must also set the Communication Settings to 4800 baud.

The setting for the baud rate is also reported in the report file.

NOTE: The focus MUST be on this window in order to see any data (text, etc) sent from the processor. You will NOT see any data sent by the processor right after a reset. You must use an external hardware reset AFTER the terminal Emulator window

is given focus in order to see the data. Using the Reset Shortcut, you will not be able to see any data because pressing the shortcut causes the Terminal emulator to lose focus. This is different than "Hyper Terminal" which always receives data even when the Hyper terminal window does not have focus. Use Hyper terminal if you need to see the program output immediately after programming or reset.

File Upload

Uploads the current program from the processor chip in HEX format. This option is meant for loading the program into a monitor program for example. It will send the current compiled program HEX file to the serial port.

File Escape

Aborts the upload to the monitor program.

File Exit

Closes terminal emulator.

Terminal Clear

Clears the terminal window.

Terminal Open Log

Opens or closes the LOG file. When there is no LOG file selected you will be asked to enter a filename or to select a filename. All info that is printed to the terminal window is captured into the log file. The menu caption will change into 'Close Log' and when you choose this option the file will be closed.

Terminal Send ASCII

This option allows you to send any ASCII character you need to send. Values from 000 to 255 may be entered.

Send BYTE			×
Enter ASCI	value		
000			
	OK	Cancel	

Terminal Send Magic number

This option will send 4 bytes to the terminal emulator. The intention is to use it together with the boot loader examples. Some of the boot loader samples check for a number of characters when the chip resets. When they receive 4 'magic' characters after each other, they will start the boot load procedure. This menu options send these 4 magic characters.

Terminal Setting

This options will show the terminal settings so you can change them quickly. It is the same as <u>Options</u>, <u>Communication</u> [95].

3.35 Tools LCD Designer

With this option you can design special characters for LCD-text displays.

The following window will appear:

BASCOM-AVR	
LCD designer	×
	Clear <u>a</u> ll
	<u>S</u> et all
	X Cancel

The LCD-matrix has 7x5 points. The bottom row is reserved for the cursor but can be used.

You can select a point by clicking the left mouse button. If a cell was selected it will be unselected.

Clicking the Set All button will set all points. Clicking the Clear All button will clear all points.

When you are finished you can press the Ok button : a statement will be inserted in your active program-editor window at the current cursor position. The statement looks like this :

Deflcdchar ?,1,2,3,4,5,6,7,8

You must replace the ?-sign with a character number ranging from 0-7. The eight bytes define how the character will appear. So they will be different depending on the character you have drawn.

See Also

Font Editor 148

3.36 Tools LIB Manager

With this option the following window will appear:

LIB Manager		
at_emulator.lib at_emulator.lib AVR-DOS_lib AVR-DOS_V4.lib bccard.lib bccard.lib bcd.lib bug.lib CF_EmbedSysXRAM.LIB cremsglcdSED.lib datetime.lib Double_Iib Double_Trig4.lib ds1307clock.lib EDTP_XRAMDrive0.lib epson.lib eurotimedate.lib	AVRDOSVER CLEARFILESYSTEM CHECKFILESYSTEMTYPE CHECKBOOTSIGN GETFILESYSTEM SAVEA2LONGY LONGY2A INITDIRENTRY MKDIR CHECK_ROOT CHDIR RMDIR FILERENAME FILEOPEN_INPUT FILEOPEN_APPEND FILEOPEN_BINARY	<u>A</u> dd Delete
History = disks without a m History = - added Check for	0 for _GetFileSystem is ignoring t aster boot record (MBR) valid Sectors per Cluster in _Getf ding in binary mode; file-pointer w	Compile

The Libraries are shown in the left pane. When you select a library, the routines that are in the library will be shown in the right pane.

After selecting a routine in the left pane, you can DELETE it with the DELETE button..

Clicking the ADD button allows you to add an ASM routine to the library.

The COMPILE button will compile the lib into an LBX file. When an error occurs you will get an error. By watching the content of the generated lbx file you can determine the error.

A compiled LBX file does not contain comments and a huge amount of mnemonics are compiled into object code. This object code is inserted at compile time of the main BASIC program. This results in faster compilation time.

The DEMO version comes with the compiled MCS.LIB file which is named MCS.LBX. The ASM source (MCS.LIB) is included only with the commercial edition.

With the ability to create LBX files you can create add on packages for BASCOM and sell them. For example, the LBX files could be distributed for free, and the ASM source could be sold.

Some library examples :

- MODBUS crc routine for the modbus slave program.
- Glcd.lib contains the graphical LCD asm code

Commercial packages available from MCS:

- I2CSLAVE library
- BCCARD for communication with www.basiccard.com chipcards

See Also

<u>\$LIB</u> [340] for writing your own libraries

3.37 Tools Graphic Converter

The Graphic converter is intended to convert BMP files into BASCOM Graphic Files (. BGF) that can be used with Graphic LCD displays.

Graphic converter						
MCS Electronic		Load Save				
64	240	<u>•</u> <u>0</u> K				
LCD type 128*64 240*128 240*240 240*64 128*128	<u>F</u> ont ○ 6*8 ⊙ 8*8	SED series				

The following dialog box will be shown:

To load a picture click the Load button. The picture can be maximum 128 pixels high and 240 pixels width.

When the picture is larger it will be adjusted.

You can use your favorite graphic tool to create the bitmaps and use the Graphic converter to convert them into black and white images.

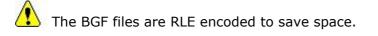
When you click the Save-button the picture will be converted into black and white. Any non-white color will be converted into black.

The resulting file will have the BGF extension.

You can also paste a picture from the clipboard by clicking the Paste button.

Press the Ok-button to return to the editor.

The picture can be shown with the <u>ShowPic</u> statement or the ShowpicE statement.



When you use your own drawing routine you can also save the pictures uncompressed by setting the Uncompressed check box. The resulting BGF files can not be shown with the showpic or showpicE statements anymore in that case!

The BGF format is made up as following:

- first byte is the height of the picture
- second byte is the width of the picture
- for each row, all pixels are scanned from left to right in steps of 6 or 8 depending on the font size. The resulting byte in stored with RLE compression

The RLE method used is : byte value, AA(hex), repeats. So a sequence of 5, AA, 10 means that a byte with the value of 5 must

So a sequence of 5, AA, 10 means that a byte with the value of 5 must be repeated 16 times (hex notation used)

Option	Description
Height	The height in pixels of the image.
Width	The width in pixels of the image.
	The T6963 supports 6x8 and 8x8 fonts. This is the font select that must match the CONFIG statement. For other displays, use 8*8.
	The size of the display. When the size is not listed, use one with the same width.
SED Series	If your display is a SEDxxxx chip, select this option.
	Images are RLE encoded. Select this option when you do not want to compress the image.

3.38 Tools Stack Analyzer

The Stack analyzer helps to determine the proper stack size.

See $\frac{\text{SDBG}}{\text{SIB}}$ for the proper usage of this option.

3.39 Tools Plugin Manager

The Plug in Manager allows you to specify which Plug-in's needs to be loaded the next time you start BASCOM.

Plugin Manager 🛛 🗙	J
PlugIns to Load	
✓ fonteditor.bpl	

Just select the plug in's you want to load/use by setting the check box. The plug in's menu's will be loaded under the Tools Menu.

To add a button to the toolbar, right click the mouse on the menu bar, and choose customize.

When you want to write your own plug in's, contact support@mcselec.com

3.40 Tools Batch Compile

The Batch Compiler is intended to compile multiple files. Shortcut : CTRL+B

The Batch compile option was added for internal test usage. It is used by MCS to test the provided test samples. The following window is shown :

Batch Compile		
File Batch		•
 C:\DATA\apps\D7\BASAVR\SAMPLE 	S\\$serialinput.bas S\123.BAS S\adc.bas S\adc_int.bas S\ALIAS.bas S\Array.bas S\asc.bas S\asc.bas S\barcode.bas S\basmon_test.bas S\bccard.bas S\bcc.ard.bas S\bcc.bas	
Batch Compiler		

There are a number of menu options.

File Load Batch

Load an earlier created and saved batch file list from disk.

File Save Batch

Save a created list of files to disk When you have composed a list with various files it is a good idea to save it for later re usage.

File Save Result

Save the batch compile log file to disk. A file named batchresult.txt will be saved in the BASCOM application directory.

File Exit

Close window

Batch Compile

Compile the checked files. By default all files you added are checked. During compilation all files that were compiled without errors are unchecked.

🚟 Batch Compile	
File Batch	+
 C:\DATA\apps\D7\BASAVR\SAMPLES\\$inc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\\$serialinput.bas C:\DATA\apps\D7\BASAVR\SAMPLES\123.BAS C:\DATA\apps\D7\BASAVR\SAMPLES\adc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\adc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\adc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\adc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\AtlAS.bas C:\DATA\apps\D7\BASAVR\SAMPLES\atc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\atc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\atc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\atc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\atc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\baccode.bas C:\DATA\apps\D7\BASAVR\SAMPLES\basmon_test.bas C:\DATA\apps\D7\BASAVR\SAMPLES\bccard.bas C:\DATA\apps\D7\BASAVR\SAMPLES\bccard.bas C:\DATA\apps\D7\BASAVR\SAMPLES\bcch_bas C:\DATA\apps\D7\BASAVR\SAMPLES\bcch_bas C:\DATA\apps\D7\BASAVR\SAMPLES\bcch_bas C:\DATA\apps\D7\BASAVR\SAMPLES\bcch_bas C:\DATA\apps\D7\BASAVR\SAMPLES\bcch_bas C:\DATA\apps\D7\BASAVR\SAMPLES\bcch_bas C:\DATA\apps\D7\BASAVR\SAMPLES\bcch_fnc_zeh.bas 	
 31-3-2007-12:51:21 Starting batch compile C:\DATA\apps\D7\BASAVR\SAMPLES\\$inc.bas C:\DATA\apps\D7\BASAVR\SAMPLES\\$serialinput.bas C:\DATA\apps\D7\BASAVR\SAMPLES\123.BAS 	
Array.bas	

This screen print shows that \$inc.bas could not be compiled. And that array.bas was not yet compiled.

Batch Add Files

Add files to the list. You can select multiple *.BAS files that will be added to the list.

Batch Add Dir

Add a directory to the list. All sub directories will be added too. The entire directory and the sub directories are searched for *.BAS files. They are all added to the list.

Batch Clear List

Clear the list of files.

Batch Clear Good

Remove the files that were compiled without error. You will keep a list with files that compiled with an error.

All results are shown in an error list at the bottom of the screen. When you double click an item, the file will be opened by the editor.

See Also

© 2011 MCS Electronics

3.41 Tools PDF Update

Use this option to update all Atmel PDF files. The Atmel data sheets are stored in the **\PDF** subdirectory. The following window will be shown :

Download and	Update PDF Fi	iles	X
PDF Files			
Check	Select All	Unselect All	Download
			<u>^</u>
<			>

There is only one option available : Check. When you click the Check-button, the Atmel server will be checked for newer versions of the PDF documents. You need to make sure that BASCOM is allowed to contact the internet.

The check will read all available DAT files and check if there is a reference to the PDF. When an item is disabled(grayed) then it means there is no link to the PDF in the DAT file.

During the check the window will look like this :

Download and Update PDF Files	
PDF Files	
atmega162.pdf atmega163.pdf atmega164p_324p_644p.pdf atmega165.pdf atmega48_88_168.pdf atmega169_revatoe.pdf ✓ atmega323.pdf atmega325_3250_645_6450.pdf atmega329_3290_649_6490.pdf ✓ atmega329_3290_649_6490.pdf ✓ atmega32.pdf atmega64.pdf atmega8515.pdf atmega8535.pdf ✓ atmega8.pdf ↓ atmega8.pdf ↓ atmega8.pdf ↓ atmega8.pdf ↓ atmega8.pdf ↓ atmega8.pdf	
Check Select All Unselect All Download]
Checking : http://www.atmel.com/dyn/resources/prod_documents/doc255 Checking : http://www.atmel.com/dyn/resources/prod_documents/doc245 Checking : http://www.atmel.com/dyn/resources/prod_documents/25125. Checking : http://www.atmel.com/dyn/resources/prod_documents/doc250 Checking : http://www.atmel.com/dyn/resources/prod_documents/doc246 Checking : http://www.atmel.com/dyn/resources/prod_documents/doc755 Done	

All PDF's that are newer will have a check mark.

You can manual unselect or select the PDF's.

In the log window at the bottom of the window you can view which files will be downloaded.

When you want to download the selected files, press the Download-button. This will close all PDF documents in the PDF viewer. A backup of each PDF file downloaded will be made before it is downloaded. You need to restore it when something goes wrong during the download(server drops the connection for example).

When a document is downloaded, the check mark will be removed.

After all documents are downloaded, they documents are opened again in the PDF viewer.

3.42 Tools Resource Editor

The resource editor can be used to edit the resource strings of your application. The resource editor will create a <project>.BCR file. The resource editor is part of the Resource Add On, and is only available when you have this add on installed.

The simplest way to get the resources from your application is to create a BCS file using the DUMP option.

Then import them with the resource editor.

English	Dutch	German	Italian
This	Dit	Dies	Questo
Name	Naam	Name	Nome
Multi language test	Meertalen test		Test multilingua
Hello	Hallo	Hallo	Ciao
is a test	is een test	ist ein test	è un test
mark	Mark	Marcus	Marco

The following options are available when you right click with the mouse in the resource editor.

Option	Description
Search	Search for a string.
Find Next	Find next occurrence.
Delete Row	Delete the current row.
Add Row	Add a new row for a new string.
Import	This option will import the BCS file which you can create with the \$RESOURCE DUMP option.
Set Language Name	Change the language name of the current language/column.
Add Language	Add a new column for a new language.
Delete Language	Delete the current column (language).

The resource editor is pretty simple. The only task is allow you to edit the various strings. You can also use notepad or Excel to create the BCR file which is explained in the $\frac{\text{SRESOURCE}}{357}$ topic.

3.43 Options Compiler

With this option, you can modify the compiler options.

The following TAB pages are available:

Options Compiler Chip Options Compiler Output Options Compiler Communication Options Compiler I2C, SPI, 1WIRE Options Compiler LCD Path

3.43.1 Options Compiler Chip

BAS	COM-AVR C	ptions			×
Co	mpiler Co <u>m</u> r	munication [<u>E</u> r	nvironment <u>S</u> imulato	r <u>P</u> rogrammer M <u>o</u> ni	tor Printer
	Chip Ou	tput Commun	iication I2C, SPI, 1V	/IRE LCD	
	Chip	9058515	•	FlashROM	8 KB
	XRAM	None	▼	SRAM	512
	HW Stack	38		EEPROM	512
	Soft Stack	32		🗖 XRAM wa	itstate
	Framesize	32		🔲 External A	ccess Enable
	Default		<u>k</u>	X <u>C</u> ancel	

The following options are available:

Options Compiler Chip

Item	Description			
Chip	Selects the target chip. Each chip has a corresponding x.DAT file with specifications of the chip. Note that some DAT files are not available yet.			
XRAM	Selects the size of the external RAM. KB means Kilo Bytes.			
HW Stack	 For 32 KB you need a 62256 STATIC RAM chip. The amount of bytes available for the hardware stack. When you use GOSUB or CALL, you are using 2 bytes of HW stack space. When you nest 2 GOSUB's you are using 4 bytes (2*2). Most statements need HW stack too. An interrupt needs 32 bytes. 			
Soft Stack	Specifies the size of the software stack. Each local variable uses 2 bytes. Each variable that is passed to a sub program uses 2 bytes too. So when you have used 10 locals in a SUB and the SUB passes 3 parameters, you need 13 * 2 = 26 bytes.			
Frame size	 Specifies the size of the frame. Each local variable is stored in a space that is named the frame space. When you have 2 local integers and a string with a length of 10, you need a frame size of (2*2) + 11 = 15 bytes. The internal conversion routines used when you use INPUT num, or STR(), or VAL(), etc, also use the frame. They need a maximum of 			

	16 bytes. So for this example $15+16 = 31$ would be a good value.
XRAM wait state	Select to insert a wait state for the external RAM.
enable	Select this option to allow external access of the micro. The 8515 for example can use port A and C to control a RAM chip. This is almost always selected if XRAM is used
	Press or click this button to use the current Compiler Chip settings as default for all new projects.

3.43.2 Options Compiler Output

BASCOM	-AVR Options					
<u>C</u> ompiler	Communication	<u>Environment</u>	Simulator	Programmer	Monitor	Printer
Chip	Output Comn	nunication 12C	, SPI, 1WIR	ELCD		
💌 В	inary file	AVR Studio	Object file			
D	ebug File	📃 Size warning	g			
ШH	IEX file	Swap words	s			
R	eport file	🗹 Optimize co	de			
E	rror file	Show intern	al variables			
C. D. C.			k	X Can	cel	
Defau						

Options Compiler Output

Item	Description
Binary file	Select to generate a binary file. (xxx.bin)
Debug file	Select to generate a debug file (xxx.dbg)
Hex file	Select to generate an Intel HEX file (xxx.hex)
Report file	Select to generate a report file (xxx.rpt)
Error file	Select to generate an error file (xxx.err)
AVR Studio object file	Select to generate an AVR Studio object file (xxx.obj)
Size warning	Select to generate a warning when the code size exceeds the Flash ROM size.
Swap words	This option will swap the bytes of the object code words. Useful for some programmers. Should be disabled for most programmers. Don't use it with the internal supported programmers.
Optimize code	This options does additional optimization of the generated code. Since it takes more compile time it is an option.
Show internal variables	Internal variables are used. Most of them refer to a register. LikeTEMP1 = R24. This option shows these variables in the report.

3.43.3 Options Compiler Communication

BASCOM	-AVR Opti	ons					
<u>C</u> ompiler	Co <u>m</u> munica	tion <u>E</u> nv	vironment	Simulator	Programmer	M <u>o</u> nitor	Printer
Chip	Output C	ommunica	ition 12C,	SPI, 1WIR	ELCD		
Baud	drate	9600		*			
Freq	uency	4000000		► F	z		
Error			0,16%				
Defau	.lt		<u> </u>	k	X <u>C</u> an	cel	

Options Compiler Communication

Item	Description
Baud rate	Selects the baud rate for the serial communication statements. You can also type in a new baud rate. It is advised to use <u>\$BAUD</u> and in the source code which overrides this setting.
Frequency	Select the frequency of the used crystal. You can also type in a new frequency. It is advised to use <u>\$CRYSTAL</u> [316] in the source code which overrides this setting. Settings in source code are preferred since it is more clear.

The settings for the internal hardware UART are:

No parity , 8 data bits , 1 stop bit

Some AVR chips have the option to specify different data bits and different stop bits and parity.

Note that these settings must match the settings of the terminal emulator. In the simulator the output is always shown correct since the baud rate is not taken in consideration during simulation. With real hardware when you print data at 9600 baud, the terminal emulator will show weird characters when not set to the same baud rate, in this example, to 9600 baud.

3.43.4 Options Compiler I2C, SPI, 1WIRE

ompiler	Commur	nication <u>E</u> nvironment	Simulator Proc	grammer Monitor Printer
Chip	Output	Communication 12C,	SPI, 1WIRE	CD
-120-			SPI	
SCL	port	PORTA.0	Clock	PORTB.5
SDA	port	PORTA.0	MOSI	PORTB.6
-1 Wi	re		MISO	PORTB.7
1 wire	•	PORTB.0	SS	PORTB.5
				Use Hardware SPI
				I2C/SPI/1WIRE
		v 0		X <u>C</u> ancel

Options Compiler I2C, SPI, 1WIRE

Item	Description
SCL port	Select the port pin that serves as the SCL-line for the I2C related statements.
SDA port	Select the port pin that serves as the SDA-line for the I2C related statements.
1WIRE	Select the port pin that serves as the 1WIRE-line for the 1Wire related statements.
Clock	Select the port pin that serves as the clock-line for the SPI related statements.
MOSI	Select the port pin that serves as the MOSI-line for the SPI related statements.
MISO	Select the port pin that serves as the MISO-line for the SPI related statements.
SS	Select the port pin that serves as the SS-line for the SPI related statements.
Use hardware SPI	Select to use built-in hardware for SPI, otherwise software emulation of SPI will be used. The 2313 does not have internal HW SPI so it can only be used with software SPI mode. When you do use hardware SPI, the above settings are not used anymore since the SPI pins are dedicated pins and can not be chosen by the user.

It is advised to use the various \underline{CONFIG} and \underline{CONFIG} commands in your source code. It make more clear in the source code which pins are used.

3.43.5 Options Compiler LCD

BASCOM-AVR Options				
Compiler Communication Environment Simulato	r <u>P</u> rogra	mmer M <u>o</u> nitor Printer		
Chip Output Communication I2C, SPI, 1WI	RE LCD			
LCD type 16 * 1a 🗸 🗸	Enable	PORTB.3 💌		
BUS mode Data mode	RS	PORTB.2 💌		
● 4-bit ● 8-bit ● bus	DB7	PORTB.7 💌		
	DB6	PORTB.6 💌		
LCD-address C000	DB5	PORTB.5 💌		
RS-address 8000	DB4	PORTB.4 🔽		
Make upper 3 bits '1' in LCD designer				
		Compiler Options		
Defaultk	×			

Options Compiler LCD

Item	Description
LCD type	The LCD display used.
Bus mode	The LCD can be operated in BUS mode or in PIN mode. In PIN mode, the data lines of the LCD are connected to the processor port pins. In BUS mode the data lines of the LCD are connected to the data lines of the BUS.
	Select 4 when you have only connect DB4-DB7. When the data mode is 'pin', you should select 4.
Data mode	Select the mode in which the LCD is operating. In PIN mode, individual processor pins can be used to drive the LCD. In BUS mode, the external data bus is used to drive the LCD.
LCD address	In BUS mode you must specify which address will select the enable line of the LCD display. For the STK200, this is C000 = $A14 + A15$.
RS address	In BUS mode you must specify which address will select the RS line of the LCD display. For the STK200, this is $8000 = A15$
Enable	For PIN mode, you must select the processor pin that is connected to the enable line of the LCD display.
RS	For PIN mode, you must select the processor pin that is connected to the RS line of the LCD display.
DB7-DB4	For PIN mode, you must select the processor pins that are connected to the upper four data lines of the LCD display.
Make upper 3 bits high	Some displays require that for setting custom characters,

in LCD designer the upper 3 bits must be 1. Should not be used by default.

It is advised to use the CONFIG LCD command. This way the settings are stored in your source code and not in the separate CFG file.

3.44 Options Communication

With this option, you can modify the communication settings for the terminal emulator.

BASCOM-A	VR Options								
<u>C</u> ompiler	Communication	<u>E</u> nvironment	<u>S</u> imulator	Program	mer	M <u>o</u> nitor	Printer		
COM por	t COM1	-	Handsh	iake (None			•	
Baudrate	19200	•	Emulatio	on (NONE	-		•	
Parity	None	•		[BT:	S			
Databits	8	•	Font	[F	ont			
Stopbits	1	•	Backco	lor		Navy			
🥅 Keep Terminal emulator open									
🔽 Use E	Existing COM port	ts							
Default	t	✓ <u>0</u>	k	X !	<u>C</u> ance	el			

Item	Description
Comport	The communication port of your PC that you use for the terminal emulator.
Baud rate	The baud rate to use.
Parity	Parity, default None.
Data bits	Number of data bits, default 8.
Stop bits	Number of stop bits, default 1.
Handshake	The handshake used, default is none.
Emulation	Emulation used, default BBS ANSI.
Font	Font type and color used by the emulator.
Back color	Background color of the terminal emulator.
Keep TE open	This option will keep the terminal emulator COM port open when you close the window or move the focus away. Some serial programmers which close the COM port when they need to program, will not work in this mode when they use the same COM port.
Use Existing	When you select this option, you will get a list with the available COM

96 BASCOM-AVR

COM ports	ports only at places you can select a COM port. When you insert an USB virtual COM port, it will be added to list automatically. Removing virtual COM ports will also update the available COM port list. When you do not select this option you get a list with COM1-COM255.
-----------	---

Note that the baud rate of the terminal emulator and the baud rate setting of the <u>compiler options</u> [92], must be the same in order to work correctly.

The reason why you can specify them both to be different is that you can use the terminal emulator for other purposes too.

3.45 **Options Environment**

BASCOM-AVR Options		
Compiler Communication	Environment Simulator Pro	ogrammer Monitor Printer
Editor Font IDE	PDF	
Autoindent	Comment position	060
Don't change case Reformat BAS files	TAB-size	3
🔽 Reformat code	Keymapping	Default 🔽
 Reformat code Smart TAB Syntax highlight No reformat ex Show margin Line numbers 		DAT Size of new editor window Normal Maximized
Default		vironment Options

OPTION	DESCRIPTION
Auto Indent	When you press return, the cursor is set to the next line at the current column position.
Don't change case	When set, the reformat won't change the case of the line after you have edited it.
	Default is that the text is reformatted so every word begins with upper case.
Reformat BAS files	Reformat files when loading them into the editor. All lines are reformatted so that multiple spaces are removed.
	This is only necessary when you are loading files that where created with another editor. Normally you won't need to set this option.
Reformat code	Reformat code when entered in the editor. The reformat option will change the modified line.

	For example $a = a + 1$ will be changed into : $a = a + 1$. When you forget a string end marker ", one will be added, and endif will be changed into End IF.
Smart TAB	When set, a TAB will place the cursor to the column where text starts on the previous line.
Syntax highlighting	This options highlights BASCOM statements in the editor.
Show margin	Shows a margin on the right side of the editor.
Comment	The position of the comment. Comment is positioned to the right of your source code. Exception if comment is first character of a line.
TAB-size	Number of spaces that are generated for a TAB.
Key mapping	Choose default, Classic, Brief or Epsilon.
No reformat extension	File extensions separated by a space that will not be reformatted when loaded. For example when DAT is entered, opening a DAT file can be done without that it is reformatted.
Size of new editor window	When a new editor window is created you can select how it will be made. Normal or Maximized (full window)
Line Numbers	Show line numbers in the margin.

BASCOM-AVR Options	
Compiler Communication Environment S	imulator Programmer Monitor Printer
Editor Font IDE PDF	
Background color White	EditorFont Font
Keyword color Navy	Bold
Comment color Green	💽 🗹 Italic
ASM color Purple	Environment Options
HW Register color Maroon	
String color Teal	
Variable color Black	
Default 🖉 🔍	<u>X</u> <u>C</u> ancel

OPTION	DESCRIPTION
Background color	The background color of the editor window.
Keyword color	The color of the reserved words. Default Navy.
	The keywords can be displayed in bold too.
Comment color	The color of comment. Default green.

	Comment can be shown in <i>Italic</i> too.
ASM color	Color to use for ASM statements. Default purple.
-	The color to use for the hardware registers/ports. Default maroon.
String color	The color to use for string constants : "test"
Variable color	The color to use for variables.
Editor font	Click on this label to select another font for the editor window.

BASCOM-AVR Options					
Compiler Communication	<u>E</u> nvironment	<u>S</u> imulator	Programmer	Monitor	Printer
Editor Font IDE	PDF			5 340	
 Tooltips Use HTML help Code Hints Allow multiple instance Autosave on compile Auto Backup Auto Load last file Auto Load All Files Clear 'Do not Ask' 	^{is} Hin	t time 50 t Color	InfoBk	-]	
Default	 ✓ 0 	k	X <u>C</u> an	cel	Environment Op

OPTION	DESCRIPTION
Tool tips	Show tool tips.
File location	Click to select a directory where your program files are stored. By default Windows will use the My Documents path.
Use HTML Help	HTML help or CHM Help is the preferred help file. Since HLP is not supported under Vista, it is advised to switch to CHM/ HTML Help. With the UpdateWiz you can still download the HLP file.
Code hints	Select this option to enable code hints. You can get code hints after you have type a statement that is recognized as a valid statement or function.
Hint Time	The delay time in mS before a code hint will be shown.
Hint Color	The background color of the hints.
Allow multiple Instances	Select this option when you want to run multiple instances of BASCOM. When not enabled, running a second copy will terminate the first one.
	The code is always saved when you compile. When you select this option, the code is saved under the same name. When this option is not selected, you will be prompted for a new

	filename.
Auto backup	Check this option to make periodic backups. When checked you can specify the backup time in minutes. The file will also be saved when you press the compiler button.
Auto load last file	When enabled, this option will load the last file that was open into the editor, when you start BASCOM.
Auto load all files	When enabled, this option will load all files that were open when you closed BASCOM.
Reset docking	This will reset the dockable windows to the default position.
Language	This will set the language in the main menu to the selected language. Not all listed languages are supported/translated yet.

BASCOM-AVR Options
Compiler Communication Environment Simulator Programmer Monitor Printer
Editor Font IDE PDF
 Auto open processor PDF Open PDF in new sheet Auto save/load project PDF
Environment Options
Default

OPTION	DESCRIPTION
Auto open processor PDF	This option will automatic load the PDF of the selected micro processor in the PDF viewer. The \$REGFILE value determines which data sheet is loaded. The PDF must exist otherwise it can not be loaded.
	Every time you change the value of the \$REGFILE the processor PDF can be shown in the same sheet, or a new sheet can be shown with the PDF. A good option in case your project uses multiple processors.
Auto save/load project PDF	Load all PDF's when the project is opened that were loaded when the project was closed.

3.46 **Options Simulator**

With this option you can modify the simulator settings.

BASCOM-AVR Options						
Compiler	Communication	<u>E</u> nvironment	<u>S</u> imulator	Programmer	Monitor	Printer
Program Paramete		ted Simulator tor after compil	ation			Simulator Options
Defau	it	 <u>0</u> 	k	X <u>C</u> an	cel	

OPTION	DESCRIPTION
Use integrated simulator	Set this option to use BASCOM's simulator. You can also use AVR Studio by clearing this option.
Run simulator after compilation	Run the selected simulator after a successful compilation.
Program	The path with the program name of the external simulator.
Parameter	The parameter to pass to the program. {FILE}.OBJ will supply the name of the current program with the extension . OBJ to the simulator.

3.47 Options Programmer

With this option you can modify the programmer settings.

BASCOM-AVR Options							
<u>Compiler</u> Communica	tion <u>Environment</u> Simulator <u>Programmer</u> Monitor Printer						
Programmer STK200/STK300 Programmer							
Play sound							
	Erase warning Auto Flash AutoVerify Upload Code and Data Program after compile Set focus to terminal emulator after programming						
Parallel							
LPT-address	378 +						
Port delay	0						
Default	<u>✓ Ok</u> <u>✓ Cancel</u>						

OPTION	DESCRIPTION			
Programmer	Select one from the list.			
Play sound	Name of a WAV file to be played when programming is finished.			
	Press the directory button to select a file.			
Erase Warning	Set this option when you want a confirmation when the chip is erased.			
Auto flash	Some programmers support auto flash. Pressing F4 will program the chip without showing the programmer window.			
Auto verify	Some programmers support verifying. The chip content will be verified after programming.			
Upload code and data	Set this option to program both the FLASH memory and the EEPROM memory			
Program after compile	When compilation is successful, the chip will be programmed			
Set focus to terminal emulator	When the chip is programmed, the terminal emulator will be shown			
	Parallel printer port programmers			
LPT address	Port address of the LPT that is connected to the programmer.			
Port delay An optional delay in uS. It should be 0. But on some sy delay might be needed.				

	Serial port programmer			
COM port	The com port the programmer is connected to.			
STK500 EXE	The path of stk500.exe. This is the full file location to the files stk500.exe that comes with the STK500.			
USB	For mkII and other Atmel USB programmers you can enter the serial number here. Or you can look it up from the list.			
	Other			
Use HEX	Select when a HEX file must be sent instead of the bin file.			
Program	The program to execute. This is your programmer software.			
Parameter	The optional parameter that the program might need.			
	Use {FILE} to insert the binary filename(file.bin) and {EEPROM} to insert the filename of the generated EEP file.			
	When `Use Hex' is checked the filename (file.hex) will be inserted for {FILE}. In all cases a binary file will be inserted for {EEPROM} with the extension .EEP			

See Also

Supported programmers 102

3.47.1 Supported Programmers

BASCOM supports the following programmers

AVR ICP910 based on the AVR910.ASM application note

STK200 ISP programmer from Atmel

The PG302 programmer from Iguana Labs

The simple cable programmer from Sample Electronics.

KITSRUS KIT122 Programmer 106

MCS Universal Interface Programmer

The MCS Universal Interface supports a number of programmers as well. In fact it is possible to support most parallel printer port programmers.

STK500 programmer and Extended STK500 programmer.

Lawicel BootLoader 112

USB-ISP Programmer 113

MCS Bootloader 122

PROGGY 124

FLIP 125

USBprog Programmer / AVR ISP mkII (AVRISP)

KamProg for AVR 127

USBASP 128

STK600 128

ARDUINO 131

3.47.1.1 ISP programmer

BASCOM supports the STK200 and STK200+ and STK300 ISP programmer from Atmel.

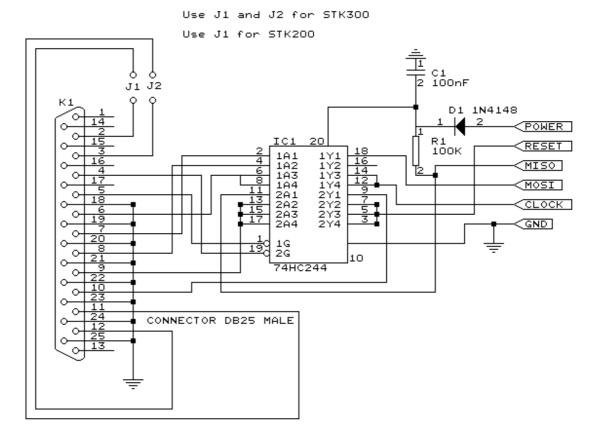
This is a very reliable parallel printer port programmer. The STK200 ISP programmer is included in the STK200 starter kit. Most programs were tested with the STK200.

For those who don't have this kit and the programmer the following schematic shows how to make your own programmer:

The dongle has a chip with no identification but since the schematic is all over the web, it is included. MCS also sells a STK200 compatible programmer.

Here is a tip received from a user :

If the parallel port is disconnected from the interface and left floating, the '244 latch outputs will waver, causing your micro controller to randomly reset during operation. The simple addition of a 100K pull-up resistor between pin 1 and 20 of the latch, and another between pin 19 and 20, will eliminate this problem. You'll then have HIGH-Z on the latch outputs when the cable is disconnected (as well as when it's connected and you aren't programming), so you can use the MOSI etc. pins for I/O.



3.47.1.2 PG302 programmer

The PG302 is a serial programmer. It works and looks exactly as the original PG302 software.

PG302 programmer		×
<u>D</u> evice		
AT90S2313	-	
Memory O Flash O EE	PROM	Lockbits IZ LB1 🗖 LB2 🗖 LB3
<u>B</u> lank check	Erase	Auto Erase
⊻erify	<u>P</u> rogram	Auto Verify
<u>R</u> ead	<u>S</u> et lockbit	X Close

Select the programmer from The Option Programmer menu or right click on the **button** to show the **Option Programmer here**

3.47.1.3 Sample Electronics cable programmer

Sample Electronics submitted the simple cable programmer.

They produce professional programmers too. This simple programmer you can make yourself within 10 minutes.

What you need is a DB25 centronics male connector, a flat cable and a connector that can be connected to the target MCU board.

DB25 pin		Target MCU M103/M128	Target MCU pin 8515	DT104
2, D0	MOSI, pin 6	PE.0, 2	MOSI, 6	J5, pin 4
4, D2	RESET, pin 9	RESET, 20	RESET, 9	J5, pin 8
5, D3	CLOCK, pin 8	PB.1,11	CLOCK, 8	J5, pin 6
11, BUSY	MISO, pin 7	PE.1, 3	MISO, 7	J5, pin 5
18-25,GND	GROUND	GROUND	GND,20	J5, pin 1

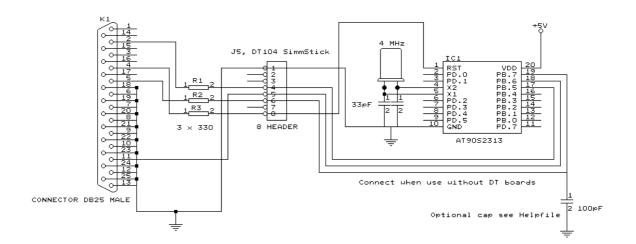
The connections to make are as following:

The MCU pin numbers are shown for an 8535! And 8515 Note that 18-25 means pins 18,19,20,21,22,23,24 and 25

You can use a small resistor of 100-220 ohm in series with the D0, D2 and D3 line in order not to short circuit your LPT port in the event the MCU pins are high. It was tested without these resistors and no problems occurred.

Tip : when testing programmers etc. on the LPT it is best to buy an I/O card for your PC that has a LPT port. This way you don't destroy your LPT port that is on the motherboard in the event you make a mistake!

The following picture shows the connections to make. Both a setup for the DT104 and stand-alone PCB are shown.



I received the following useful information:

106 BASCOM-AVR

I have been having spurious success with the simple cable programmer from Sample Electronics for the AVR series.

After resorting to hooking up the CRO I have figured it out (I think). When trying to identify the chip, no response on the MISO pin indicates that the Programming Enable command has not been correctly received by the target.

The SCK line Mark/Space times were okay but it looked a bit sad with a slow rise time but a rapid fall time. So I initially tried to improve the rise time with a pull-up. No change ie still could not identify chip. I was about to add some buffers when I came across an Atmel app note for their serial programmer "During this first phase of the programming cycle, keeping the SCK line free from pulses is critical, as pulses will cause the target AVR to loose synchronization with the programmer. When synchronization is lost, the only means of regaining synchronization is to release the RESET line for more than 100ms."

I have added a 100pF cap from SCK to GND and works first time every time now. The SCK rise time is still sad but there must have been enough noise to corrupt the initial command despite using a 600mm shielded cable.

3.47.1.4 KITSRUS Programmer

The K122 is a KIT from KITSRUS. (www.kitsrus.com)

The programmer supports the most popular 20 and 40 pins AVR chips.

On the Programmer Options tab you must select this programmer and the COM port it is connected to.

On the Monitor Options tab you must specify the upload speed of 9600, Monitor delay of 1 and Prefix delay 1.

When you press the Program button the Terminal Emulator screen will pop up:

BASCOM-AVR Terminal emulator			_ 🗆 ×
<u>F</u> ile <u>T</u> erminal			
	Flash	O EEPROM	
			-
			▼ ►
COM1:19200,N,8,1			

A special toolbar is now visible.

You must press the Program enable button to enable the programmer. When you enable the programmer the right baud rate will be set. When you are finished you must press the Enable button again to disable it. This way you can have a micro connected to your COM port that works with a different BAUD rate.

There is an option to select between FLASH and EEPROM. The prompt will show the current mode which is set to FLASH by default.

The buttons on the toolbar allow you to : ERASE, PROGRAM, VERIFY, DUMP and set the LOCK BITS. When DUMP is selected you will be asked for a file name. When the DUMP is ready you must CLOSE the LOGFILE where the data is stored. This can be done to select the CLOSE LOGFILE option form the menu.

3.47.1.5 MCS Universal Interface Programmer

The MCS Universal Interface programmer allows you to customize the pins that are used for the ISP interface. The file prog.settings stores the various interfaces.

The content :

; how to use this file to add support for other programmers

;first create a section like [newprog]

; then enter the entries:

- ; BASE= \$hexaddress
- ; MOSI= address in form of BASE[+offset] , bit [,inverted]
- ; CLOCK= same as MOSI
- ; RESET=same as MOSI
- ; MISO=same as MOSI
- ; The bit is a numer that must be written to set the bit
- ; for example 128 to set bit 7
- ; Optional is ,INVERTED to specify that inverse logic is used
- ; When 128 is specified for the bit, NOT 128 will be written(127)

[FUTURELEC]

;tested and ok BASE=\$378 MOSI=BASE+2,1,inverted CLOCK=BASE,1 RESET=BASE,2 MISO=BASE+1,64

[sample] ;tested and ok BASE=\$378 MOSI=BASE,1 CLOCK=BASE,8 RESET=BASE,4 MISO=BASE+1,128,INVERTED

[stk200] ;tested and ok BASE=\$378 MOSI=BASE,32 CLOCK=BASE,16 RESET=BASE,128 MISO=BASE+1,64

Four programmers are supported : Futurelec, Sample and STK200/STK300 and WinAVR/ SP12.

To add your own programmer open the file with notepad and add a new section name. For the example I will use stk200 that is already in the file.

[stk200]

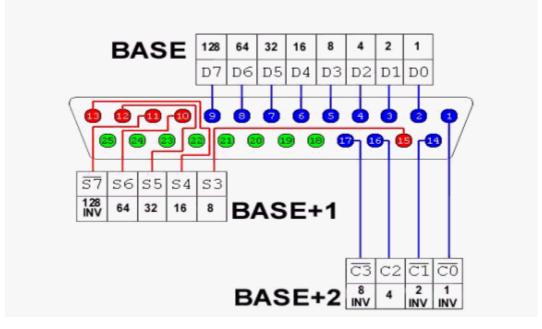
The LPT base address must be specified. For LPT1 this is in most cases \$378. \$ means hexadecimal.

The pins that are needed are MOSI, CLOCK, RESET and MISO. Add the pin name MOSI =

After the pin name add the address of the register. For the STK200 the data lines are used so BASE must be specified. After the address of the register, specify the bit number value to set the pin high. Pin 0 will be 1, pin 1 would be 2, pin 2 would be 4 etc. D5 is used for the stk so we specify 32.

When the value is set by writing a logic 0, also specify, INVERTED. After you have specified all pins, save the file and restart BASCOM. Select the Universal Programmer Interface and select the entry you created. After you have selected an entry save your settings and exit BASCOM. At the next startup of BASCOM, the settings will be used.

The following picture shows the LPT connector and the relation of the pins to the LPT registers.



Always add your entry to the bottom of the file and email the settings to support@mcselec.com so it can be added to BASCOM.

3.47.1.6 STK500 Programmer

When you select the STK500 programmer, BASCOM will run the file named stk500. exe that is installed with AVR Studio.

That is why you have to specify the file location of the stk500.exe The normal STK500 support will erase, and program the flash. The STK500.EXE supports a number of Atmel programmers which all use t

The STK500.EXE supports a number of Atmel programmers which all use the STK500 V1 or V2 protocol.

For the AVR ISP mkII, you need to supply the serial number of the USB programmer. The USB port will be used then instead of the serial port.

You can also use the **native** driver which does not use/need the stk500.exe If you select this programmer, you will see the following window when you launch the programmer with F4(manual program)

	Bu			·		_						ogi	-														 	 		 	10-0
	2		Iz			.	3	F		T _E															 		 	 	 	 	
-				£	₽	0	•	-	0	-																					
hip	?	??							FI	lash	ı siz	в	0	KB							EEP	ROM	size		O By	es					
Flash	E	EPF	10M		_oc	k ar	nd F	us	еb	its																		 	 	 	
	00 0	01 C)2 I	33	04	05	06	s 0	17	08	09	QΑ	OB	00	00	0	: 0	F						Γ							
																			8 11 .	. I											0
)10	95 1	18 (00	00	95	18	00) ()	10 !	95	18	00	00	95	18	0) ()	D I	II.	I											
)20	95 1	18 (00 1	00	95	18	00) ()	10	95	18	00	00	95	18	0) ()	D	II.	I											
_			_	_			_	_	_						_	_	_		II.												
_							_	_	_						-	_	_		II.												
_							-	_						-	-	_	_	-	II.	-											
_							_	_	_							_	_		II.												
_		_					_	_	_						-	_	_	_	íÆéî		-										
_		_	_	_			_	_	_					_	-	_	_)_ïîà		±1										
							_	_	_							_	_		á½ð												
																			÷éá ľ												
																			21ál*												
_							_	_	_							_	_		Sel 21												
_							-	_							-	_	_		»[Âå ∋r ≪s												
_		_	_	_			_	_	_					_	-	_	_	-	j∥…ãà àeàñ												
rul	CUIC	54 6	-01		EU		134			51	эг	54	UE	TOC		12	+ UI		aean	q.,	e l .										
SI S C C	TK50 :\dat)0 c a\ap	pps\	D7	\BA	SA	VR'	\SA	٨MF	PLE										es in	ito FL	ASH	buff	er							
	BIN							FLA									ом														

When the source code is compiled and the BIN file exists, it is loaded automatic into the buffer.

When an EEPROM image file exists (EEP), it is loaded too into the EEPROM buffer. When it does not exist you will see a warning which you can ignore.

When the target device is not read yet, the CHIP will be unidentified which is marked as ???.

In the status bar you can see the loaded file, and the size of the file. Notice that 16000 will be shown as 16 KB.

You can select the EEPROM-TAB to view the EEPROM image. Memory locations can be altered. Select a cell, and type a new value. Then press ENTER to confirm. You can immediately see the new value.

When you select the Lock and Fusebits-TAB the lock and fuse bits will be read.

111

ATmega162	Flash size 16 KB EEPROM size 512 Bytes		
	and Fuse bits		
Chip Name	M162]
Name Calibration 0	56		Refresh
	FF		
Lockbit 54	11:No restrictions for SPM or LPM accessing the boot loader section		Vrite LB
Lockbit 32	11:No restrictions for SPM or LPM accessing the application section		YING ED
Lockbit 10	11:No memory lock features enabled		Vrite FS
Fusebits	ED		
Fusebit 7	1:Divide clock by 16 disabled	W	/rite FSH
Fusebit 6	1:Clock output disabled		
Fusebit 98	10:SUT=10 Start-up time		/rite FSE
Fusebit DCBA	1101:CKSEL=110X External Crystal/Resonator Medium Frequency		
Fusebits High	DO		
Fusebit High E	1:Disable OCD		
Fusebit High F	1:Disable JTAG	W	rite PRG
Fusebit High G	0:Enable serial downloading		
Fusebit High H	1:Watchdogtimer always on disabled	-	
Started STK500 compatible pr	-		
	SAVR\SAMPLES\CHIPS\M162.BIN loaded 826 bytes into FLASH buffer SAVR\SAMPLES\CHIPS\M162.EEP not found.		
U: \data\apps\U7\BA;	DAVIN (SAMPLES (CHIPS (M 162.EEP not round.		

As you can see that as soon as the target chip is determined, the chip name is shown under the tool bar.

The FLASH size and EEPROM size are shown too.

As soon as you alter a lock or fuse bit, the corresponding Write-button will be enabled. You need to click it to write the new value. The lock and fuse bits are read again so you can see if it worked out. The lock and fuse bits shown will depend on the used chip. Every chip has different fuse bits. Some fuse bits can not be altered via the serial programming method. The native stk500 driver uses the serial programming method. Some fuse bits require the parallel or high voltage programming method. For example the fuse bit 'enable serial downloading' can not be changed with the serial programming method.

Fuse bits of interest are : the clock divider and the oscillator fuse bits. When you select a wrong oscillator fuse bit (for example you select an external oscillator) the chip will not work anymore till you connect such an external oscillator! Of course a simple 555 chip can generate a clock signal you can use to 'wake' a locked chip.

Once you have all settings right, you can press the 'Write PRG' button which will insert some code into your program at the current cursor position. This is the \$PROG directive.

For example : \$prog &HFF , &HED , &HD0 , &HFF

When you compile your program with the $\underline{\$PROG}$ birective it will generate a PRG file with the lock and fuse bit settings.

If you then auto program(see later) a chip, it will use these settings.

\$PROG is great to load the right lock and fuse bits into a new chip. But be careful : do not enable \$PROG till you are done with development. Otherwise programming will be slow because of the extra reading and writing steps.

The following menu options are available:

Option	Description
File	
Exit	Close programmer.
Buffer	
Clear	Clear buffer. Will put a value of 255 (FF hex) into each memory location. When the FLASH-TAB has the focus, the FLASH buffer will be cleared. When the EEPROM-TAB has the focus, the EEPROM buffer will be cleared. 255 is the value of an empty memory location.
Load from File	This will shown an open file dialog so you can select a binary file (BIN)
	The file is loaded into the buffer.
Save to File	Will save the current buffer to a file.
Reload	Reloads the buffer from the file image.
Chip	
Identify	Will attempt to read the signature of the chip. When the signature is unknown(no DAT file available) or there is no chip or other error, you will get an error. Otherwise the chip name will be shown.
Write buffer to chip	This will write the active buffer(FLASH or EEPROM) into the chip.
Read chipcode	When the chip lock bit is not set you can read the FLASH or EEPROM into the buffer.
Blank check	Check if the chip FLASH or EEPROM is empty.
Erase	Erases the chip FLASH. It depends on the fusebits if the EEPROM is erased too. Normally the EEPROM is erased too but some chip have a fuse bit to preserve EEPROM when erasing the chip. A chip MUST be erased before it can be programmed.
Verify	Checks if the buffer matches the chip FLASH or EEPROM.
Auto program	This will eraser, and program the FLASH and EEPROM and if \$PROG is used, it will set the lock and fusebits too.

Under Options, you can find a setting to change the clock frequency. The clock frequency should not be higher then a quarter of the oscillator frequency.

3.47.1.7 Lawicel BootLoader

The Lawicel Boot loader must be used with the StAVeR. The StAVeR contains a boot loader so you only need a serial interface, no parallel programmer or other programmers.

You can also use Hyper terminal.

When you have selected the Lawicel Boot loader from the Options, Programmer, the following window will appear when you press F4.

🖁 Lawicel BootLoader 🛛 🗐 🖂
Reset Staver
Cancel

As the window suggests, press the reset button on the activity board or StAVeR, and the chip will be programmed. This is visible by a second wind that will be shown during programming.

When the programming succeeds, both windows will be closed.

When an error occurs, you will get an error message and you can clock the Cancel button in order to return to the Editor.

3.47.1.8 AVR ISP Programmer

The AVRISP programmer is AVR ICP910 based on the AVR910.ASM application note.

The old ICP910 does not support Mega chips. Only a modified version of the AVR910. ASM supports Universal commands so all chips can be programmed.

The new AVRISP from Atmel that can be used with AVR Studio, is not compatible! You need to select <u>STK500 programmer</u> we because the new AVRISP programmer from Atmel, uses the STK500 protocol.

When you do not want to use the default baud rate that AVR910 is using, you can edit the file bascavr.ini from the Windows directory. Add the section [AVRISP] Then add: COM=19200,n,8,1

This is the default. When you made your own dongle, you can increase the baud rate

You need to save the file and restart BASCOM before the settings will be in effect.

3.47.1.9 USB-ISP Programmer

directory named USB.

The USB-ISP Programmer is a special USB programmer that is fully compatible with BASCOM's advanced programmer options.

Since many new PC's and especial Laptop's do not have a parallel programmer anymore, MCS selected the USB-ISP programmer from EMBUD.

The drivers can be downloaded from the MCS Electronics website. Please download from http://www.mcselec.com/index.php?option=com_docman&task=doc_download&gid=204&Ite

mid=54 After downloading, unzip the files in the BASCOM-AVR application directory in a sub



When you connect the programmer, Windows (98, ME, 2000, XP) will recognize the new device automatically.



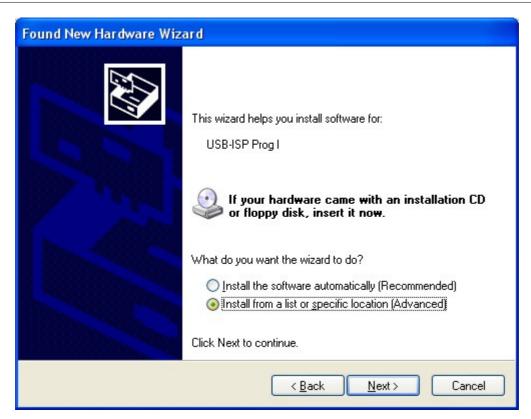
Then the Hardware wizard will be started :

Found New Hardware Wizard							
	Welcome to the Found New Hardware Wizard Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission). Read our privacy policy						
	Can Windows connect to Windows Update to search for software? Yes, this time only Yes, now and <u>e</u> very time I connect a device No, not this time						
	Click Next to continue.						
	< <u>Back</u> <u>N</u> ext > Cancel						

Select 'No, not this time' and click Next, as there is no driver at Microsoft's web.

The Wiz will show :

115



You need to select 'Install from a list or specific location' and click Next.

Found New Hardware Wizard
Please choose your search and installation options.
Search for the best driver in these locations.
Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.
Search removable <u>m</u> edia (floppy, CD-ROM)
✓ Include this location in the search:
Program Files\MCS Electronics\BASCOM-AVR\USB 🗸 🛛 Browse
O Don't search. I will choose the driver to install.
Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.
< <u>B</u> ack <u>N</u> ext > Cancel

You can specify the path of the USB driver. This is by default :

C:\Program Files\MCS Electronics\BASCOM-AVR\USB

Use the Browse-button to select it, or a different location, depending on your installation.

As the driver is not certified by Micros ft, you will see the following window:

Hardwa	re Installation
1	The software you are installing for this hardware: USB-ISP Programmer has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.) Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.
	Continue Anyway

You need to select 'Continue Anyway'. A restore point will be made if your OS supports this and the driver will be installed. After installation you must see the following window :

Found New Hardware Wiz	ard
	Completing the Found New Hardware Wizard The wizard has finished installing the software for: USB-ISP Programmer
	K <u>B</u> ack Finish Cancel

After you press Finish you will see Windows can use the programmer :

ſ	4		ardware	×	
l	Your new ha	ardware	is installed and ready	/ to use.	
٢	2 Firefox	•	Document		🌐 🧐 🖉 10:57

In BASCOM , Options, Programmer you can select the new programmer now.

BASCOM-AVR Optic	ions
Compiler Communicat	ation Environment Simulator Programmer Monitor Printer
Programmer	USB-ISP Programmer
Play sound	
Erase warning Program after cor USB-ISP	
USB Speed 0	
Default	<mark>✓ O</mark> k ∑ ancel

New models of the USB programmer allow to set the speed.

The USB-ISP programmer is very quick and supports all options that the Sample Electronics and STK200 programmers support. It is good replacement for the STK200.

When you use other USB devices that use the FTDI drivers, there might occur a problem. Manual install the drivers of these other devices, then install the USB-ISP driver.

USB-ISP on VISTA

For Vista and Vista 64, please follow the this installation description.



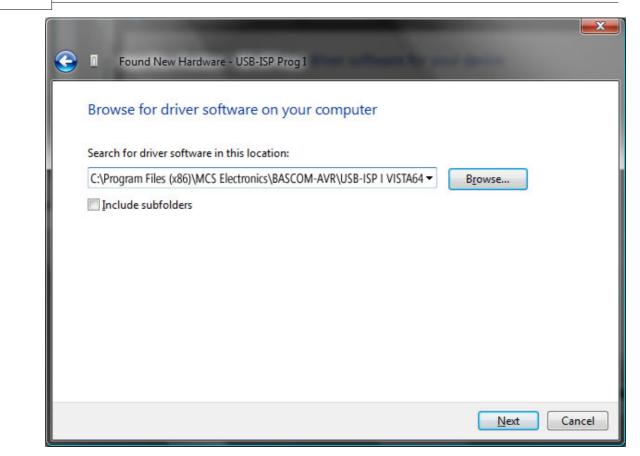
When connection the ISP-PROG I to your PC the following window will show up. Here I have to select the top selection: *Locate and Install driver software (recommended)* Vista starts it search for the driver and will come finally with the question to Insert the driver disk.

٢	Found New Hardware - USB-ISP Prog I	×)
	Insert the disc that came with your USB-ISP Prog I If you have the disc that came with your device, insert it now. Windows will automatically search the disc for driver software.	
	➔ I don't have the disc. Show me other options.	
	<u>N</u> ext Cancel	

As we have no driver CD, you have to select: I don't have the disc. Show me other options

<u></u>	Found New Hardware - USB-ISP Prog I	
Wi	ndows couldn't find driver software for your device	
•	Check for a solution Windows will check to see if there are steps you can take to get your device working.	
•	Browse my computer for driver software (advanced) Locate and install driver software manually.	
		Cancel

Now we select the Browse selection and locate the driver folder.



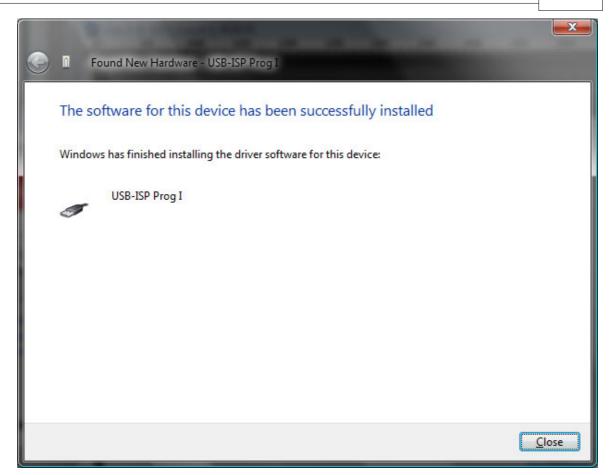
And select Next button.

As Vista 64 only allows certified drivers the following message will pop-up.

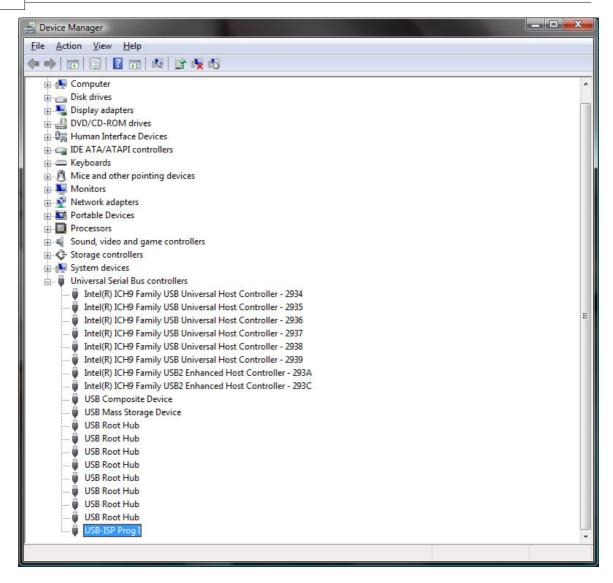


Just select *Install this driver software anyway* and Vista 64 will now start with installing the driver. Be patient as it depends on your system configuration how long it will take.

121



Finally Vista 64 will tell you that the driver is installed. To check your configuration you can go to your device manager to see if it is there.



3.47.1.10 MCS Bootloader

The MCS Boot loader is intended to be used with the <u>\$LOADER</u> and sample. It uses the X-modem Checksum protocol to upload the binary file. It works very quick.

The Boot loader sample can upload both normal flash programs and EEPROM images. The Boot loader sends a byte with value of 123 to the AVR Boot loader. This boot loader program then enter the boot loader or will jump to the reset vector (0000) to execute the normal flash program.

When it receives 124 instead of 123, it will upload the EEPROM. When you select a BIN file the flash will be uploaded. When you select an EEP file, the EEPROM will be uploaded.

The Boot loader has some specific options.

BASCOM-	AVR Options						
<u>C</u> ompiler	Communication	<u>Environment</u>	<u>S</u> imulator	<u>P</u> rogram	nmer	Monitor Printer	
Program	nmer MCS	i Bootloader				•	
Play so	und						
	gram after compile	uto Flash 📃	AutoVerif Set focus			d Code and Data nulator after programming	
Boot size 1024 Close programmer window when ready Ask to send EEP if it exists Reset O NONE O DTR O RTS O Soft abc{123}							
Default							

BOOTSIZE

You can choose the boot size which is 1024 for the BASCOM \$LOADER example. Since this space is used from the normal flash memory, it means your application has 1024 less words for the main application. (A word is 2 byte, so 2KB less) The XMEGA has a separate boot space so for Xmega you can set the value to 0.

RESET

The boot loader is started when the chip is reset. Thus you need to reset the chip after you have pressed F4(program). But when you have connected the DTR line to the chip reset (with a MAX232 buffer) you can reset the chip automatically. You do need to set the 'Reset via DTR' option then. You can also chose to use the RTS line. When your program does not use the boot vector or needs a special sequence to activate the loader, you can chose the soft reset. To send ASCII characters you can embed them between brackets {}. For example {065} will be sent as the character A or byte with value 65.

CLOSE

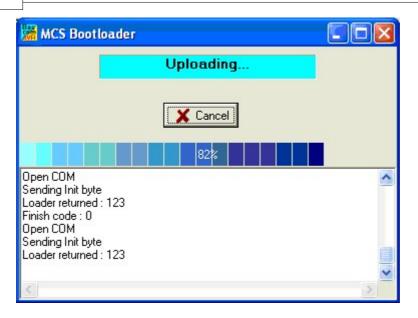
By choosing 'Close programmer window when ready' the window will be closed when the loader returns 0.

In all other cases it will remain opened so you can look at a possible cause.

EEP

If an EEP (EEPROM image file) exists, the loader can send this file instead of the flash binary file. If you enable this option, you will be asked if you want to send the EEP instead of the BIN file.

After you have pressed F4 to following window will appear :



As you can see the loader sends a byte with value of 123.

You need to reset the chip, and then you will see that the loader returned 123 which means it received the value.

It will start the upload and you see a progress bar. After the loader is ready, you see a finish code of 0.

A finish code of 0 means that all wend well.

Other finish codes will not close the window even if this option is enabled. You need to manual close the window then.

ERROR CODES

- -6001 Bad format in file name
- -6002 file not found
- -6003 file not found in folder
- -6004 folder not found
- -6005 canceled
- -6006 time out
- -6007 protocol error
- -6008 too many errors
- -6009 block sequence error
- -6016 session aborted

The most likely error is -6006 when the bootloader is not present or does not respond timely after the initial handshake. Increase the \$timeout in the boot loader in that case.

3.47.1.11 PROGGY

PROGGY is a popular USB programmer written by Red_Mamba.

You need to install it and make sure that the registry key : HKEY_CURRENT_USER\Software\Red_Mamba\Atmel programator exists with the parameter : InstallPath

InstallPath should point to the executable which name is atme.exe When you install PROGGY, it will be handled for you. When you have an older version, you need to update.

BASCOM will call the programmer with the following options : -p -s -e

The -e will cause the programmer to exit after the programming.

3.47.1.12 FLIP

FLIP is a free programmer from Atmel. BASCOM supports FLIP for the USB chips/ interface.

The USB chips are programmed with a boot loader. This is very convenient since you do not need any hardware to program the chip. FLIP can be downloaded from the Atmel site.

URL : <u>http://www.atmel.com/dyn/resources/prod_documents/Flip%20Installer%20-%203.3.1.exe</u> The FLIP website you can find at :

http://www.atmel.com/dyn/products/tools_card.asp?family_id=604&family_name=8051+ Architecture&tool_id=3886

FLIP is a Java application. The BASCOM-IDE can use the FLIP software to program the chip too. But in order to use the FLIP programmer, you need to install FLIP first.

When FLIP is working, you can select FLIP from Options, Programmer, in order to program quickly without the FLIP executable.

On Vista there is a problem with loading some of the FLIP DLL's. In case you get an error, copy the FLIP DLL's to the BASCOM application directory.

You need to copy the following files :

- atjniisp.dll
- AtLibUsbDfu.dll
- msvcp60.dll
- msvcrt.dll

You can run the **flipDLLcopy.cmd** file from the BASCOM application directory to copy these files. The content of the command file :

copy "c:\program files\atmel\flip 3.3.1\bin\atjniisp.dll" . copy "c:\program files\atmel\flip 3.3.1\bin\AtLibUsbDfu.dll" . copy "c:\program files\atmel\flip 3.3.1\bin\msvcp60.dll" . copy "c:\program files\atmel\flip 3.3.1\bin\msvcrt.dll" . pause

The last line pauses so you can view the result. Notice the . (dot) that will copy the file to the current directory, which is the reason that you need to run this file from the BASCOM application directory.

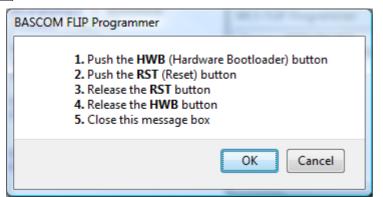
As with other programmers, you press F4 to program the HEX file into the chip. A small window will become visible.

A number of dialogs are possible:

BASCON	1-AVR	x
Device	not supported : ATmega88	
		ОК

In this case, you try to program a chip which is not supported by FLIP. The Mega88 is not an USB chip so the error makes sense.

The next dialog informs you about a missing DFU device.



In this case, the boot loader is not found. You can run the boot loader by following the sequence from the dialog box.

In order to make this work, the HWB and RST input both need a small switch to ground. When HWB is pressed(low) during a reset, the boot loader will be executed.

TIPS & Tricks

FLIP is only supported by Atmel. Here you will find some tips about FLIP. In order to use BASCOM's FLIP support, you must have running FLIP successfully first ! Here is a good tip from a user : IMO he Flip 3.3.1 Installer is a little bit stupid. The dll's are located in the Path ...\Atmel\Flip 3.3.1\bin . The Installer has set a correct Path-Variable in Windows for this path. But, the libusb0.dll isn't in that location. It is in ...\Atmel\Flip 3.3.1\USB ! So I moved the libusb0.dll into the \bin dir and Flip runs without the errors. (GRRR) In the ...\Atmel\Flip 3.3.1\USB dir I have also detected the missing .inf File. After installing this, Windows detects the AT90USB162 and Flip can connect the device.

3.47.1.13 USBprog Programmer / AVR ISP mkll

The USBprog programmer is a neat small USB programmer which is fully compatible with the AVR ISP mkII programmer.

When you select this programmer, you will get the same interface as for the <u>STK500</u> <u>native</u> 109 programmer.

F4 will launch the programmer. For more details read the help section for the STK500 programmer.

When programming XMEGA chips the interface for the fuse bits will be different. See STK600 programmer for a description.

The default clock is 125 KHz. This because most/all chips ship with a clock frequency of 1 MHz. And since the clock frequency maximum is a quarter of the oscillator frequency, the default is 125 KHz, low enough to be able to program all chips. Once your chip runs at say 8 MHz, you can select 2 MHz as the maximum.

You must have the <u>LIBSUSB</u> drivers installed on your PC. Without it, it will not work.

Options

In the Configuration options you can adjust the clock speed and the timeout of the USB.

When you are using USB 1.1 and a lot of devices that generate a lot of USB traffic,

you might need to increase the default timeout of 100 (msec).

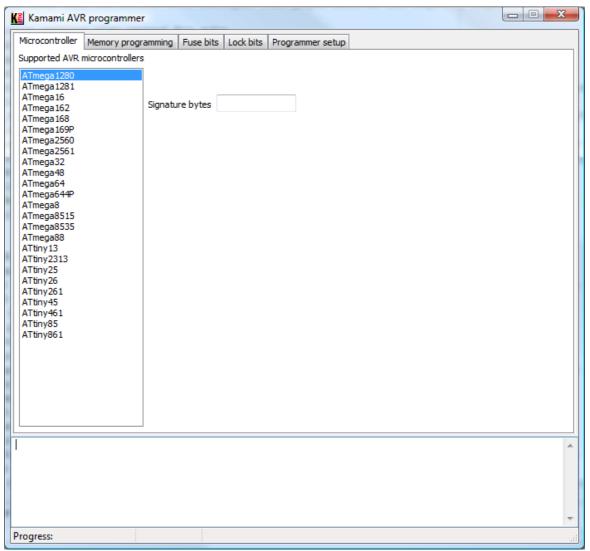
3.47.1.14 KamProg for AVR

KamProg for AVR is an USB programmer from Kamami. You need to install the software that comes with the KamProg. KamProg can be used with BASCOM but also with AVR Studio.

BASCOM will use the KamProg software to either automatic or manual program the chip.

The Kamprog programmer works on Vista32 and Vista64 and requires no special drivers.

The KamProg programmer is available from MCS Electronics webshop.



All new Mega processors are supported. Support for other processor is on going. The screen above is from the first release and outdated but gives a good impression about the interface.

3.47.1.15 USBASP

The USBASP is a popular USB programmer created by Thomas Fischl The programmer uses a Mega8 or other AVR chip as an USB device. You can find the programmer at Thomas website : <u>http://www.fischl.de/usbasp</u>

Make sure when programming the fuse and lock bits that the selected clock frequency is not too high. The clock frequency of the ISP programmer should be less then one quarter of the oscillator frequency. When your micro is running at 8 MHz, you can select up to 2 MHz. On the safe size, 125 KHz is always ok.

By default most AVR processors run at 8 MHz with an 8-divider resulting in 1 MHz clock frequency. So 250 KHz is a safe value for most processors.

BASCOM-AVR Options
Compiler Communication Environment Simulator Programmer Monitor Printer
Programmer USBASP -
Play sound
 Erase warning Auto Flash AutoVerify Upload Code and Data Program after compile Set focus to terminal emulator after programming USBASP
Clock Frequency 750 KHz
Default Qk X Cancel

You can select various clock frequencies.

See also LIBUSE for installation of LIBUSE

3.47.1.16 STK600

The STK600 is a development board from Atmel. It uses a similar protocol as the STK500 and has an integrated USB programmer on board. The programmer can be connected with a cable to the STK600 board itself, but also to an external board.

The STK600 replaces the STK500 and is advised for XMEGA development. For regular AVR chips we would recommend the STK500.

The STK600 has actual 3 different programmers on board : ISP, PDI and JTAG. the ISP/PDI protocols are combined and placed on one connector.

When programming XMEGA chips, the BASCOM programmers will automatic switch to the PDI protocol. The ISP protocol can not be used with XMEGA chips. For other chips, (non-xmega), the ISP protocol will be used.

There are affordable PDI programmers available.

The following description is also true for the AVRISP/mkII programmer which also supports the PDI protocol.

In order to use the STK600 protocol you need to have <u>LIBSUSB</u> is installed.

Identification

The BASCOM programmers always try to identify the chip before an action is performed. This is needed to check the size and to check if your program is intended for the selected chip.

It would not be a good idea for example to program an attiny13 with xmega128a1 code.

When you chose manual programming, you will get the following window:

属 STK6	00 Pro	gram	mer		i.						_	_							_	_	_	_	_	-			х
File E	Buffer	Chip																									
0 🖻		<u>L</u> 2 _2	. 🖷	-	2	—	Ъ																				
Chip	???						Flash		_	O KB	2	-		EEPI	ROM size	•	O Bj	tes									
Flash	EEPR	ом L	.ock a	nd Fu	ise bi	its																					
	00 01	10210	3104	051	0610	7 08	3109	0A1	овТ	oclo	DIDE	OF				1						 	 			 	
			_		_	_	_	_	_		_	-	<u>,,,,,,,,,,,,</u>	,,,,,,,,,,,,,	ÿÿÿ												
00010	FF FF	FFF	FF	FF	FF F	F FF	FF	FF	FF	FF FI	FF	FF	<u> </u>	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	yyy												
													YYYYYYYY														
													YYYYYYYY														
													ууууууу														
													ууууууу														
					_	_	_				_	-	ууууууу														
													yyyyyyy														
		_					_				_	_	ууууууу														
													<u>yyyyyyy</u>														
													ууууууу														
													, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,														
													99999999 99999999														
													99999999 99999999														
													yyyyyyy yyyyyyy														-
 D:\d D:\d 	lata\ap lata\ap .600 pro	ps\D7\ ps\D7\	BASA BASA	VR\S	AMF	LES	\B00	DT/B	оот	LOAI	DER>	(ME)	GA128.E	EEP no		512 bj	ytes int	o FLAS	6H bu	uffer							
OOTLO.	ADER>	KMEG.4	128.E	III FL	.ASH	: 12	9 K B			EEPF	ROM																

As you can see, the binary image is loaded and if an EEPROM EEP binary image was available it would have been loaded too.

When you click the Identify button, the programmer will read the device id. The same will happen for any other action you chose.

🔣 STK	00 Programmer	
File	Buffer Chip	
] = .		
Chip	ATXMega128A1 Flash size 136 KB EEPROM size 2 KB	
Flash	EEPROM Lock and Fuse bits	
	00 01 02 03 04 05 06 07 08 09 04 08 0C 00 0E 0F	
00000	FF	
00010	FF	
00020	FF	
00030	FF	
00040	FF	
00050	FF	
00060	FF	
00070	FF	
08000	FF	
00090	FF	
000A0	FF	
000B0	FF	
00000	FF	
000D0	FF	
	FF	
000F0	FF	-
AT:	Mega128A1 , FLASH : 139264 , EPROM : 2048	
	600 detected	
🕒 C:\	lata\apps\D7\BASAVR\SAMPLES\BOOT\BOOTLOADERXMEGA128.EEP not found.	
	lata\apps\D7\BASAVR\SAMPLES\B00T\B00TL0ADERXMEGA128.BIN loaded 132512 bytes into FLASH buffer	
	600 programmer set	
🕘 Sta	ted	
BOOTLI	ADERXMEGA128.BIT FLASH : 136 KB EEPROM : 2 KB	

The Device ID is now read and you can see the ATXMEGA128A1 is detected.

The programmer has the same options as the STK500 programmer. Only the lock and fuse byte differ for the Xmega.

When you select the Lock and Fuse bits, you will get a similar screen:

STK600 Programmer						
File Buffer Chip		•				
🥒 🤌 🔐 🛓 🖀 📅 🛔 🚟 1	🗣 🗸					
Chip ATXMega128A1 Fla	ash size 136 KB EEPROM size 2 KB					
Flash EEPROM Lock and Fuse bits						
🗆 Chip		<u> </u>				
Name	ATXMega128A1	Refresh				
🖃 lock0	FF					
Boot Section Lock Bits	Nolocks					
Application Section Lock Bits	Nolocks	Write				
Application Table Lock Bits	Nolocks	E				
Lock Bits	No locks					
🖃 fuse0	FF					
JTAG User ID						
🗆 fuse1						
Watchdog Window Configuration	16 cycles (16ms @ 3.3V)					
Watchdog Configuration	8 cycles (8ms @ 3.3V)					
∃ fuse2	BF					
BOD Operation in Power-Down Mode	BOD Disabled					
BOD Operation in Active Mode	BOD Disabled	Write PRG				
Spike Detected Enable	Disabled					
Boot Loader Section Reset Vector	Boot Loader Reset	-				
STK600 detected						
ATXMega128A1 , FLASH : 139264 , EPF	IOM : 2048					
STK600 detected		E				
	OOT\BOOTLOADERXMEGA128.EEP not found.	=				
	OOT\BOOTLOADERXMEGA128.BIN loaded 132512 bytes into FLASH buffer					
STK600 programmer set						
A second for the second		*				
BOOTLOADERXMEGA128.BIT FLASH : 136	KB EEPROM: 2 KB	1				

The XMEGA has one lock byte and 6 fuse byes named FUSE0-FUSE5. Not all fuse bytes are used. The options depend on the XMEGA chip you use. In the screen shot from above you can see that under the FUSE1 section, the 'Watchdog Window Configuration' is colored red.

When you change an option and move focus or enter, a change will result in the option to be shown in red.

When you have selected all values you can select the WRITE button to write the lock and fuse bytes.

After this the values will be read again and updated.

The WRITE PRG button will insert a \$PROG directive into your code with all lock and fuse bytes.

A description of the fuse bytes you can find in the PDF of the processor.

3.47.1.17 ARDUINO

The ARDUINO is a hardware platform based on AVR processors. ARDUINO boards/ chips are programmed with a bootloader. This bootloader is the old STK500 protocol, not longer supported by Atmel in Studio. There are various programmers for ARDUINO, AVRDUDE is probably the most versatile.

BASCOM also supports the ARDUINO/STK500 v1 protocol. the DTR/RTS lines are used to reset the board.

You can program/read flash/EEPROM but you can not read/write fuse/lock bytes. The STK500 bootloader for ARDUINO does not support this.

Under options you only need to select the programmer, and the COM port. Since an FTDI chip is used on most ARDUINO boards, this is a virtual COM port. Only present when the USB cable is connected to your PC.

Select **57600** baud for the baud rate. Older ARDUINO boards work with **19200** baud.

3.47.2 LIBUSB

Using USB programmers in BASCOM-AVR

Please read this document completely before starting to install software.

Like every other USB device, an USB programmer requires a windows driver. Some programmers use drivers that are provided (built into) by windows. For example the KamProg uses the HID class and does not require an additional third party driver.

A programmer like the AVRISP mkII does need an additional driver. This device driver is installed when you install AVR Studio.

Studio is using device drivers from JUNGO.

When you plug in the programmer and Windows informs you that it requires a driver you know that you need to install a third party driver.

When Windows does not complain it will use a driver already available on your PC.

Most USB devices need software installed before you plug them in for the first time. In many cases there is a warning sticker that you should first install the software.

BASCOM uses LIBUSB to access USB devices. LIBUSB is available as a device driver

or as a filter driver.

When your device is using a device driver you must access the device with a filter driver.

Some devices do not have a vendor supplied driver (USBASP programmer) and those require a device driver.

Scenario one : you have a 32 bit or 64 bit OS and have a product that uses a device driver.

In this example we use the AVRISP mkII that is supported by AVR Studio. When you do not have AVR Studio installed you can download it from Atmels website for free. The original programmer comes with a CD-ROM too. But many imitation/self build devices exist that do not come with a CD. For those you need to download and install AVR Studio.

The next step is to plug your programmer, and see if it works with AVR Studio. Windows will recognize it, and install the device driver. When windows is ready, press the connect button in Studio.

Select AVR Programmer	
Platform: Port: AVR ONE! Image: Constraint of the second	Connect Cancel Baud rate: 115200 Baud rate changes are
Tip: To auto-connect to the programmer used last time, press the 'Programmer' button on the toolbar. Note that a tool cannot be used for programming as long as it is connected in a debugging session. In that case, select 'Stop Debugging' first. Disconnected Mode	active immediately.

If you open Studio, and press the CON(nection) button, the window shown above will open.

Now select your programmer, in this sample AVRISP mkII and press Connect

When it functions, a new window will open

AVRISP mkII in ISP mode with AT	tiny88	
Main Program Fuses LockBits / Device and Signature Bytes ATtiny88 Signature not read	Advanced HW	Settings HW Info Auto Erase Device Read Signature
Programming Mode and Target Settin	ngs	Settings ISP Frequency: 125.0 kHz
The selected device does not support ISF Getting isp parameter SD=0x06 OK	^o programming	~

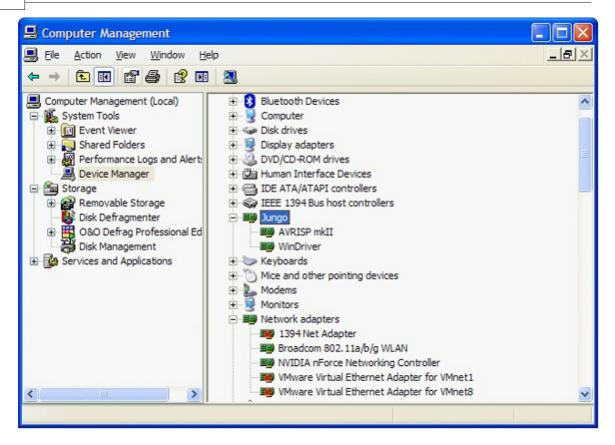
You can select the device, the programming mode and ISP frequency. This frequency should be 125 KHz (or better said, should not exceed a quarter of the chip oscillator frequency).

When you do not get this window but you return to the connection window, it means your programmer is not working.

You have to solve this first before you can continue.

The programmer will only work in BASCOM when it functions with the original software!

In the windows device manager, you can find this info: (right click Computer, select manage, and chose device manager)



The screen above shows the JUNGO usb driver which Atmel AVR Studio uses and the AVRISP mkII driver for the AVRISP mkII.

If you install AVR Studio with the USB drivers, it will install JUNGO and the WinDriver. The AVRISP mkII entry you only get when you plug the programmer.

To make it work with BASCOM, you need to install LIBUSB. LIBUSB is used by many different programs. Atmels FLIP is using it too. So there is a big change that it is available on your system already.

You can install **LIBUSB** as a **FILTER** driver or a **DEVICE** driver.

We install the FILTER driver, so we can use the programmer with Studio AND bascom.

Before you install LIBUSB it is a good idea to make a restore point.

When installing the USB driver, disconnect ALL USB devices. Obvious, you can not install from an USB flash drive since this is an USB device as well.

You can read about LIBUSB and download it from : http://sourceforge.net/apps/trac/libusb-win32/wiki

The last version is :

http://sourceforge.net/projects/libusb-win32/files/libusb-win32-releases/1.2.4.0/libus b-win32-devel-filter-1.2.4.0.exe/download

Notice that this an executable you can install. You MUST have ADMIN rights when you install this executable.

After LIBUSB has been installed you can test if it is functional.

- Look in the Program Files\LibUSB-Win32 folder (also named on Windows-7 64 bit !!!)

You will find a sub folder named **bin** which contains a number of executables.

- Run the **testlibusb-win.exe** application. When LIBUSB is functional you will see a screen with all USB devices.

When it does not work, try to install again with compatibility mode set to XP SP2. Do this by selecting the the setup exe file properties, and select 'Compatibility'.

🐺 libusb-win32-filter-bin-0.1.12.1 Properties 🛛 💌						
General Compatibility Security Details Previous Versions						
If you have problems with this program and it worked correctly on an earlier version of Windows, select the compatibility mode that matches that earlier version.						
Compatibility mode						
Run this program in compatibility mode for:						
Windows XP (Service Pack 2)						
Settings						
Run in 256 colors						
Run in 640 x 480 screen resolution						
Disable visual themes						
Disable desktop composition						
Disable display scaling on high DPI settings						
Privilege Level						
Run this program as an administrator						
Show settings for all users						
OK Cancel Apply						

Click Apply and/or OK. And run setup again.

On Windows 7 - 64 bit, this was NOT required.

Once the testlibusb-win.exe works, you can continue to the next step.

Install the filter driver for the device

You need to install a filter driver for your programmer. Each different programmer requires it's own filter driver. So you must repeat these steps if you have different programmers.

- Plug in your programmer if it was not plugged in yet
- Run the **install-filter-win.exe** application from the BIN folder.
- You will see this window:

This program adds/removes libusb-v installation.	in32 as a driv	ver to an exist	ing device	
The libusb-win32 filter driver allows while maintaining compatibility with s				
Install/Remove Device Filter(s)				
Install a device filter				
🔘 Remove a device filter				
Remove all device filters				

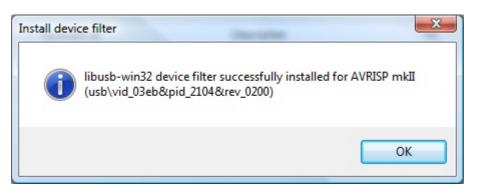
Select 'Install a device filter' and press Next.

	rom the list of unfiltered devices below. If you e filtered, be in a "driverless" state, or ilter driver.
Hardware ID	Description
vid:058f pid:6377 rev:0100	USB Mass Storage Device
vid:051d pid:0002 rev:0006	American Power Conversion USB UPS
vid:03eb pid:2104 rev:0200	AVRISP mkII
٠	

Select the programmer and press Install.

After some moments, you will get a confirmation:

137



Now the programmer will work in BASCOM. Just select the proper programmer, and timeout of 100 ms. You can try lower time outs too to make it quicker. When you get errors, increase the time out. 100 ms should do for all programmers.

Scenario two: you do not have a device driver.

In this case you can follow scenario one till the filter driver installation. Instead of running **install-filter-win.exe**, you will run **inf-wizard.exe**.

100	libusb-win32 Inf-Wizard
	Information This program will create an .inf file for your device. Before clicking "Next" make sure that your device is connected to the system.
	Next > Cancel

Press Next. And the following window will be shown.

		ist of detected devices below. If your device isn't liste "Next" and enter your device description manually.
Vendor ID	Product ID	Description
0x16C0	0x05DC	USBasp
0x058F	0x6377	USB Mass Storage Device
0x051D	0x0002	American Power Conversion USB UPS
•		III

As you can see, the USBASP was inserted in this sample. Select it (or your programmer) and press Next.

Vendor ID (hex format)	0x16C0
	0x05DC
Product ID (hex format)	
MI (hex format)	
Manufacturer Name	VOTI
Device Name	USBasp

Press Next again and select a folder to store the device driver files. These files are required to install the device.

		BASCOM IDE	139
🐻 Save As			<u> </u>
	nark 🕨 Documents 🕨	Search	٩
File <u>n</u> ame	USBasp.inf		-
Save as <u>t</u> ype	inf files (*.inf)		•
Browse Folders		Save	cel

After you have saved the files, you have the option to install the driver. Press Install Now.. button to do so.

libusb-win32 Inf-Wizard	
Information A windows driver instal following device:	lation package has been created for the
Vendor ID:	0x16C0
Product ID:	0x05DC
Device description:	USBasp
Manufacturer:	ITOV
This package contains libust platforms: x86, x64, ia64.	o-win32 v1.2.3.0 drivers and support for the following
Install Now	er package now. Done

When ready :

Driver Install Complete 🔀
Installation successful.
OK

Final note

The USB-ISP programmer form EMBUD, uses drivers from FTDI. It does not require LIBUSB.

The Kamprog programmer from KAMAMI uses a HID class and does not require LIBUSB.

Some devices gave a problem in 1.2.3.0. This problem is solved in 1.2.4.0. http://sourceforge.net/projects/libusb-win32/files/libusb-win32-releases /1.2.4.0/libusb-win32-devel-filter-1.2.4.0.exe/download

3.48 Options Monitor

With this option you can modify the monitor settings.

OPTION	DESCRIPTION
Upload speed	Selects the baud rate used for uploading
Monitor prefix	String that will be send to the monitor before the upload starts
Monitor suffix	String that us sent to the monitor after the download is completed.
Monitor delay	Time in milliseconds to wait after a line has been sent to the monitor.
Prefix delay	Time in milliseconds to wait after a prefix has been sent to the monitor.

3.49 Options Printer

With this option you can modify the printer settings.

BASCOM-AVR Options						
Compiler	Communication	Environment	Simulator	Programmer	M <u>o</u> nitor	Printer
🔽 Prin	ip Lines t Header : Numbers	=	Top Righ Bott	Margin Margin It Margin om Margin ter Options	15 15 15	
Defau	ılt	 ✓ 0 	k	X <u>C</u> ar	ncel	

OPTION	DESCRIPTION
Font	Printer font to use when printing
Setup	Click to change the printer setup
Color	Will print in color. Use this only for color printers.
Wrap lines	Wrap long lines. When not enabled, long lines will be partial shown.
Print	Print a header with the filename.

header	
Line numbers	Will be the line number before each line.
Syntax	Enable this to use the same syntax highlighting as the editor
Left margin	The left margin of the paper.
Right margin	The right margin of the paper.
Top margin	The top margin of the paper.
Bottom margin	The bottom margin of the paper.

3.50 Window Cascade

Cascade all open editor windows.

3.51 Window Tile

Tile all open editor windows.

3.52 Window Arrange Icons

Arrange the icons of the minimized editor windows.

3.53 Windows Maximize All

Maximize all open editor windows.

3.54 Window Minimize All

Minimize all open editor windows.

3.55 Help About

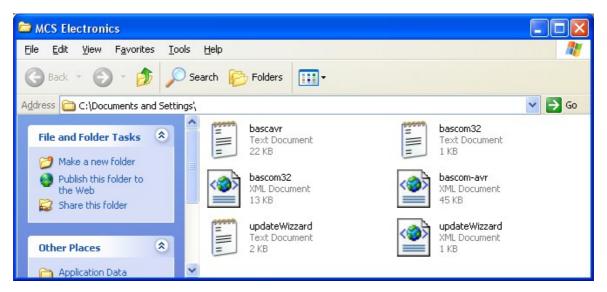
This option shows an about box as shown below.

About		×
	BASCOM-AVR	
Compiler version IDE version Serial number	: 1.11.8.5	
Windows OS Windows SP Explorer Company Owner Windows dir App data dir System dir	: Service Pack 2 : 7.0.5730.11 : MCS : Mark Alberts : C:\WINNT	
Support © :	: <u>support@mcselec.com</u> 1995-2007, MCS Electronics	se

Your serial number is shown on the third line of the about box. You will need this when you have questions about the product.

The compiler and IDE version numbers are also shown.

When you click the App data dir link, the folder which contains the BASCOM settings will be opened:



It contains the bascom-avr.xml file with all settings and the bascavr.log file. When you need support, you might be asked to email these files.

When you need support, also click the Copy-button. It will copy the following info to the clipboard, which you can paste in your email :

Dont forget that Serial numbers should not be sent to the user list. Make sure you sent your email to support and not a public list !

Compiler version :1.11.8.3 IDE version :1.11.8.5 Serial number :XX-XXXX-XXXXX Windows OS :Microsoft Windows XP Windows SP :Service Pack 2 :7.0.5730.11 Explorer :MCS Company Owner :Mark Alberts Windows dir :C:\WINNT App data dir :C:\Documents and Settings System dir :C:\WINNT\system32

When you click the support link, your email client will be started and an email to support@mcselec.com will be created.

Click on Ok to return to the editor.

3.56 Help Index

Shows the BASCOM help file.

When you are in the editor window, the current word selected or by the cursor will be used as a keyword.

Notice that when the help window is small, you might need to make the help window bigger to show the whole content.

The help contains complete sample code and partial sample code. In all cases the samples are shown to give you an idea of the operation. When trying a program you should always use the samples from the SAMPLES directory. These are updated and tested when new versions are published. The (partial) samples are not all updates, only when they contain errors. So the samples from the help might need some small adjustments while the samples form the SAMPLES dir will work at least on the used chip.

3.57 Help MCS Forum

This option will start your default Web browser and direct it to <u>http://www.mcselec.</u> com/index2.php?option=com_forum&Itemid=59

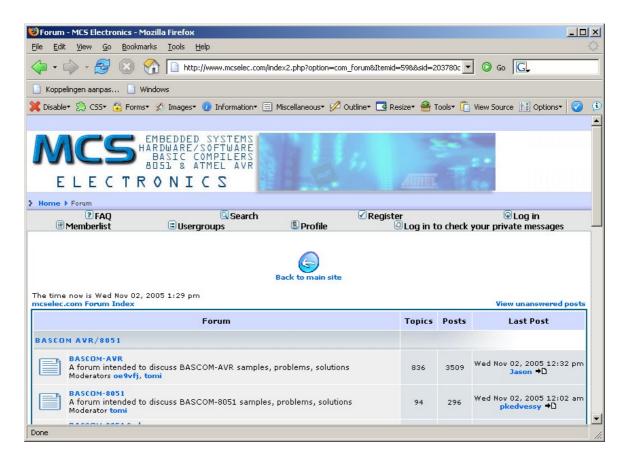
This forum is hosted by MCS Electronics. There are various forums available. You can post your questions there. Do not cross post your questions on multiple forums and to support.

The forum is available for all users : demo or commercial users. Note that everything you write might be on line for ever. So mind your language.

Users of the commercial version can email MCS support.

The forum allows uploads for code examples, circuits etc.

If you try to abuse the forum or any other part of the MCS web, you will be banned from the site.



3.58 Help MCS Shop

This option will start your default web browser and direct it to :<u>http://www.mcselec.</u> <u>com/index.php?option=com_phpshop&Itemid=1</u>

You can order items and pay with PayPal. PayPal will accept most credit cards.

Before you order, it is best to check the <u>resellers</u> and page to find a reseller near you. Resellers can help you in your own language, have all MCS items on stock, and are in the same time zone.

BASCOM IDE

145

🍪 MCS Electronics - Mozilla Firefox					_	
<u>Eile E</u> dit <u>V</u> iew <u>G</u> o <u>B</u> ookmarks <u>T</u> ools	Help					0
🗘 • 🖒 • 🎅 😣 🟠 🔤	tp://www.mcselec.com/index.php?option=com_p	hpshop&Itemid=1	•	O Go G		
📄 Koppelingen aanpas 📄 Windows						
💥 Disable* 🍰 CSS* 🙆 Forms* 💋 Image:	s* 🕡 Information* 📰 Miscellaneous* 💋 Ou	utline+ 📑 Resize+ 🚔 Tool:	🕶 📋 View Source 🔢 O	ptions•	Ø 🤅	0
						-
ELECTRONJ	D SYSTEMS //SOFTWARE COMPILERS ATMEL AVR	и Ант.				
> Home				Wednesday, 02 No	vember 20	005
 Latest News 12 october 2005, Support system offline 12 october 2005, New Forum added 11 October 2005, RSS Freeds support added 11 October 2005, New site released New account creating - Demo Main Menu Home News Products Application Notes Publications Support Center Downloads Forum Resellers Conduct Bosider 	Shopping zone Categories MCS Shop MCS Shop Growthardware (59) Growthardware (14) Growthard (14)	Software (14)	Books (2)	Service (2)		

Before you can order items, you need to create an account. Read the following about the new website : <u>http://www.mcselec.com/index.php?</u> <u>option=com_content&task=view&id=133&Itemid=1</u>

3.59 Help Support

This option will start your default browser with the following URL :

http://www.mcselec.com/support-center/

It depends from your browser settings if a new window or TAB will be created. At the support site you can browse articles. You can also search on keywords.

3.60 Help Knowledge Base

This option will ask you to enter a search string.

Knowledge Base Quer	у	×
Question		
fusebit		
ОК	Cancel	

This search string will be passed to the MCS support site. The above example that searches for "FUSEBIT" will result in the following :

146 BASCO	M-AVR
-----------	-------

	odesk :: Support Center - Mozilla Firefox		Z
<u>File E</u> dit <u>V</u> iew <u>G</u> o	Bookmarks Tools Help		0
🔷 • 🔶 • 🛃 (🔄 😭 🔟 http://www.mcselec.com/support-center/index.php?mod_id=2&ł 💌 🔘 Go	G.	_
🥐 Getting Started 🔂 La	atest Headlines		
			1
	EMBEDDED SYSTEMS HARDWARE/SOFTWARE BASIC COMPILERS BOSL & ATMEL AVR TRONICS		
Home > Support center			
Main Menu	Keyword Search Results:		
A Home	Article Summary	Match	
🚉 Browse Knowledgebase	How to set the FUSEBIT to external oscillator	100.00%	
Cards	How to set the FUSEBIT to external oscillator Didn't Find the Answer you were Looking for?	100.00%	ļ
Knowledgebase Search		100.00%	
Knowledgebase	Didn't Find the Answer you were Looking for?	100.00%	
Knowledgebase Search	Didn't Find the Answer you were Looking for? Try a Keyword Search:	100.00%	
Knowledgebase Search	Didn't Find the Answer you were Looking for? Try a Keyword Search: fusebit Search	100.00%	

You can click one of the found articles to read it.

3.61 Help Credits

BASCOM was invented in 1995. Many users gave feedback and helped with tips, code, suggestions, support, a user list, and of course with buying the software. The software improved a lot during the last 10 years and will so during the next decade.

While it is impossible to thank everybody there are a few people that deserve credits :

- Josef Franz Vögel. He wrote a significant part of the libraries in BASCOM-AVR. He is also author of AVR-DOS.
- Dr.-Ing. Claus Kuehnel for his book 'AVR RISC' , that helped me a lot when I began to study the AVR chips. Check his website at http://www.ckuehnel.ch
- Atmel, who gave permission to use the AVR picture in the start up screen. And for the great tech support. Check their website at http://www.atmel.com
- Brian Dickens, who did most of the Beta testing. He also checked the documentation on grammar and spelling errors. (he is not responsible for the spelling errors i added later :-))
- Jack Tidwell. I used his FP unit for singles. It is the best one available.

3.62 BASCOM Editor Keys

Кеу	Action		
LEFT ARROW	One character to the left		
RIGHT ARROW	One character to the right		
UP ARROW	One line up		
DOWN ARROW	One line down		
HOME	To the beginning of a line		
END	To the beginning of a line To the end of a line		
PAGE UP	Up one window		
PAGE DOWN	Down one window		
CTRL+LEFT	One word to the left		
CTRL+RIGHT	One word to the right		
CTRL+HOME	To the start of the text		
CTRL+END	To the end of the text		
CTRL+ Y	Delete current line		
INS	Toggles insert/over strike mode		
F1	Help (context sensitive)		
F2	Run simulator		
F3	Find next text		
F4	Send to chip (run flash programmer)		
F5	Run		
F7	Compile File		
F8	Step		
F9	Set breakpoint		
F10	Run to		
CTRL+F7	Syntax Check		
CTRL+F	Find text		
CTRL+G	Go to line		
CTRL+K+x	Toggle bookmark. X can be 1-8		
CTRL+L	LCD Designer		
CTRL+M	File Simulation		
CTRL+N	New File		
CTRL+O	Load File		
CTRL+P	Print File		
CTRL+Q+x	Go to Bookmark. X can be 1-8		
CTRL+R	Replace text		
CTRL+S	Save File		
CTRL+T	Terminal emulator		
CTRL+P	Compiler Options		
CTRL+W	Show result of compilation		
CTRL+X	Cut selected text to clipboard		
CTRL+Z	Undo last modification		
SHIFT+CTRL+Z	Redo last undo		
CTRL+INS	Copy selected text to clipboard		
SHIFT+INS	Copy text from clipboard to editor		
CTRL+SHIFT+J	Indent Block		
CTRL+SHIFT+U	Unindent Block		

Hold the SHIFT key down and use the cursor keys to select text.
or keep the left mouse key pressed and drag the cursor over the text to select.

3.63 **Program Development Order**

- Start BASCOM
- Open a file or create a new one
- ! Important ! Check the chip settings, baud rate and frequency settings for the target system
- Save the file
- Compile the file (this will also save the file !!!)
- If an error occurs fix it and recompile (F7)
- Run the simulator(F2)
- Program the chip(F4)

3.64 PlugIns

3.64.1 Font Editor

The Font Editor is a Plug in that is intended to create Fonts that can be used with Graphical display such as SED1521, KS108, color displays, etc.

When you have installed the Font Editor , a menu option becomes available under the Tools menu : Font Editor.

When you choose this option the following window will appear:

149

🖧 MCS For	nt Editor [font	3x8.font]	
Eile			
Fontname	Font8x8	Width 8 Height 8	From 32 to 254
ASCII 32	Char 🚺		
33	!		
34	"		
35	#		
36	\$		
37	z		
38	&		
39	·		
40	(
41)	Preview Clear all Clipbrd	+
42	*	# Set all Suggest	+ +
43	+		
44	·		_
45	- 🖌		

You can open an existing Font file, or Save a modified file.

The supplied font files are installed in the Samples directory. You can copy an image from the clipboard, and you can then move the image up , down, left and right.

When you select a new character, the current character is saved. The suggest button will draw an image of the current selected character.

When you keep the left mouse button pressed, you can set the pixels in the grid. When you keep the right mouse button pressed, you can clear the pixels in the grid.

When you choose the option to create a new Font, you must provide the name of the font, the height of the font in pixels and the width of the font in pixels.

The Max ASCII is the last ASCII character value you want to use. Each character will occupy space. So it is important that you do not choose a value that is too high and will not be used.

When you display normal text, the maximum number is 127 so it does not make sense to specify a value of 255.

A font file is a plain text file. Lets have a look at the first few lines of the 8x8 font:

Font8x8: \$asm .db 1,8,8,0 .db 0,0,0,0,0,0,0,0; .db 0,0,6,95,6,0,0,0; !

The first line contains the name of the font. With the <u>SETFONT</u> statement you can select the font. Essential, this sets a data pointer to the location of the font data.

The second line (\$ASM) is a directive for the internal assembler that asm code will follow.

All other lines are data lines.

The third line contains 4 bytes: 1 (height in bytes of the font), 8 (width in pixels of the font), 8 (block size of the font) and a 0 which was not used before the 'truetype' support, but used for aligning the data in memory. This because AVR object code is a word long.

This last position is **0** by default. Except for 'TrueType' fonts. In BASCOM a TrueType font is a font where every character can have it's own width. The letter 'i' for example takes less space then the letter 'w'. The EADOG128 library demonstrates the TrueType option.

In order to display TT, the code need to determine the space at the left and right of the character. This space is then skipped and a fixed space is used between the characters. You can replace the 0 by the width you want to use. The value 2 seems a good one for small fonts.

All other lines are bytes that represent the character.



4 BASCOM HARDWARE

4.1 Additional Hardware

talk about the chip.

Of course just running a program on the chip is not enough. You will probably attach many types of electronic devices to the processor ports. BASCOM supports a lot of hardware and so it has lots of hardware related statements. Before explaining about programming the additional hardware, it might be better to

The AVR internal hardware

Attaching an LCD display 162

Using the I2C protocol 173

Using the 1WIRE protocol

Using the SPI protocol 183

You can attach additional hardware to the ports of the microprocessor. The following statements will then be able to be used:

I2CSEND 727 and I2CRECEIVE 726 and other I2C related statements.

CLS, 433 LCD, 752 DISPLAY 661 and other related LCD-statements.

1WRESET 375), 1WWRITE 386) and 1WREAD 377)

4.2 AVR Internal Hardware

The AVR chips all have internal hardware that can be used.

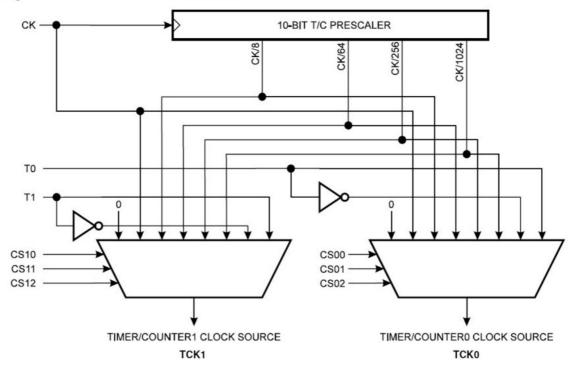
For this description of the hardware the 90S8515 was used. Newer chips like the Mega8515 may differ and have more or less internal hardware.

You will need to read the manufacturers data sheet for the processor you are using to learn about the special internal hardware available.

Timer / Counters

The AT90S8515 provides two general purpose Timer/Counters - one 8-bit T/C and one 16-bit T/C. The Timer/Counters have individual pre-scaling selection from the same 10-bit pre-scaling timer. Both Timer/Counters can either be used as a timer with an internal clock time base or as a counter with an external pin connection which triggers the counting.





More about TIMERO

More about TIMER1

The WATCHDOG Timer 158

Almost all AVR chips have the ports B and D. The 40 or more pin devices also have ports A and C that also can be used for addressing an external RAM chip (\underline{XRAM} (16)). Since all ports are similar except that PORT B and PORT D have alternative functions, only these ports are described.

PORT B 158 PORT D 160

4.3 AVR Internal Registers

You can manipulate the internal register values directly from BASCOM. They are also reserved words. Each register acts like a memory location or program variable, except that the bits of each byte have a special meaning. The bits control how the internal hardware functions, or report the status of internal hardware functions. Read the data sheet to determine what each bit function is for.

The internal registers for the AVR90S8515 are : (other processors are similar, but vary)

Addr.	Register
\$3F	SREG I T H S V N Z C
\$3E	SPH SP15 SP14 SP13 SP12 SP11 SP10 SP9 SP8
\$3D	SPL SP7 SP6 SP5 SP4 SP3 SP2 SP1 SP0
\$3C	Reserved

\$39	TIMSK TOIE1 OCIE1A OCIE1B - TICIE1 - TOIE0 -		
\$38	TIFR TOV1 OCF1A OCF1B -ICF1 -TOV0 -		
\$37	Reserved		
\$36	Reserved		
\$35	MCUCR SRE SRW SE SM ISC11 ISC10 ISC01 ISC00		
\$34	Reserved		
\$33	TCCR0 CS02 CS01 CS00		
\$32	TCNT0 Timer/Counter0 (8 Bit)		
\$31	Reserved		
\$30	Reserved		
\$2F	TCCR1A COM1A1 COM1A0 COM1B1 COM1B0PWM11 PWM10		
\$2E	TCCR1B ICNC1 ICES1 CTC1 CS12 CS11 CS10		
\$2D	TCNT1H Timer/Counter1 - Counter Register High Byte		
\$2C	TCNT1L Timer/Counter1 - Counter Register Low Byte		
\$2B	OCR1AH Timer/Counter1 - Output Compare Register A High Byte		
\$2A	OCR1AL Timer/Counter1 - Output Compare Register A Low Byte		
\$29	OCR1BH Timer/Counter1 - Output Compare Register B High Byte		
\$28	OCR1BL Timer/Counter1 - Output Compare Register B Low Byte		
\$27 #26	Reserved		
\$26 ¢25	Reserved		
\$25 ¢24	ICR1H Timer/Counter1 - Input Capture Register High Byte		
\$24 ¢22	ICR1L Timer/Counter1 - Input Capture Register Low Byte Reserved		
\$23 \$22	Reserved		
\$21	WDTCR WDTOE WDE WDP2 WDP1 WDP0		
\$20	Reserved		
\$1F	Reserved EEAR8		
\$1E	EEARL EEPROM Address Register Low Byte		
\$1D	EEDR EEPROM Data Register		
\$1C	EECR EEMWE EEWE EERE		
\$1B	PORTA PORTA7 PORTA6 PORTA5 PORTA4 PORTA3 PORTA2 PORTA1 PORTA0		
\$1A	DDRA DDA7 DDA6 DDA5 DDA4 DDA3 DDA2 DDA1 DDA0		
\$19	PINA PINA7 PINA6 PINA5 PINA4 PINA3 PINA2 PINA1 PINA0		
\$18	PORTB PORTB7 PORTB6 PORTB5 PORTB4 PORTB3 PORTB2 PORTB1 PORTB0		
\$17	DDRB DDB7 DDB6 DDB5 DDB4 DDB3 DDB2 DDB1 DDB0		
\$16	PINB PINB7 PINB6 PINB5 PINB4 PINB3 PINB2 PINB1 PINB0		
\$15	PORTC PORTC7 PORTC6 PORTC5 PORTC4 PORTC3 PORTC2 PORTC1 PORTC0		
\$14	DDRC DDC7 DDC6 DDC5 DDC4 DDC3 DDC2 DDC1 DDC0		
\$13	PINC PINC7 PINC6 PINC5 PINC4 PINC3 PINC2 PINC1 PINC0		
\$12	PORTD PORTD7 PORTD6 PORTD5 PORTD4 PORTD3 PORTD2 PORTD1 PORTD0		
\$11	DDRD DDD7 DDD6 DDD5 DDD4 DDD3 DDD2 DDD1 DDD0		
\$10	PIND PIND7 PIND6 PIND5 PIND4 PIND3 PIND2 PIND1 PIND0		
\$0F	SPDR SPI Data Register		

	SPSR SPIF WCOL
	SPCR SPIE SPE DORD MSTR CPOL CPHA SPR1 SPR0
	UDR UART I/O Data Register
	USR RXC TXC UDRE FE OR
	UCR RXCIE TXCIE UDRIE RXEN TXEN CHR9 RXB8 TXB8
	UBRR UART Baud Rate Register
	ACSR ACD - ACO ACI ACIE ACIC ACIS1 ACIS0
\$00	Reserved

The registers and their addresses are defined in the xxx.DAT files which are placed in the BASCOM-AVR application directory.

The registers can be used as normal byte variables.

PORTB = 40 will place a value of 40 into port B.

Note that internal registers are reserved words. This means that they can't be dimensioned as BASCOM variables!

So you can't use the statement DIM SREG As Byte because SREG is an internal register.

You can however manipulate the register with the SREG = value statement, or var = SREG statement.

4.4 AVR Internal Hardware TIMER0

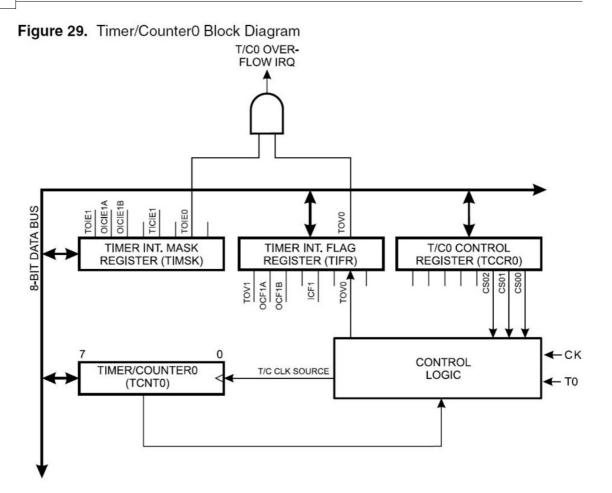
The 8-Bit Timer/Counter0

The 90S8515 was used for this example. Other chips might have a somewhat different timer.

The 8-bit Timer/Counter0 can select its clock source from CK, pre-scaled CK, or an external pin. In addition it can be stopped (no clock).

The overflow status flag is found in the Timer/Counter Interrupt Flag Register - TIFR. Control signals are found in the Timer/Counter0 Control Register - TCCR0. The interrupt enable/disable settings for Timer/Counter0 are found in the Timer/Counter Interrupt Mask Register - TIMSK.

When Timer/Counter0 is externally clocked, the external signal is synchronized with the oscillator frequency of the CPU. To assure proper sampling of the external clock, the minimum time between two external clock transitions must be at least one internal CPU clock period. The external clock signal is sampled on the rising edge of the internal CPU clock.



The 8-bit Timer/Counter0 features both a high resolution and a high accuracy mode with lower pre-scaling values. Similarly, high pre-scaling values make the Timer/ Counter0 useful for lower speed functions or exact timing functions with infrequent actions.

4.5 AVR Internal Hardware TIMER1

The 16-Bit Timer/Counter1

The 90S8515 was used for the documentation. Other chips might have a somewhat different timer.

The 16-bit Timer/Counter1 can select its clock source from CK, pre-scaled CK, or an external pin. In addition it can be stopped (no clock).

The different status flags (overflow, compare match and capture event) and control signals are found in the Timer/Counter1 Control Registers - TCCR1A and TCCR1B.

The interrupt enable/disable settings for Timer/Counter1 are found in the Timer/ Counter Interrupt Mask Register - TIMSK.

When Timer/Counter1 is externally clocked, the external signal is synchronized with the oscillator frequency of the CPU. To assure proper sampling of the external clock, the minimum time between two external clock transitions must be at least one

internal CPU clock period.

The external clock signal is sampled on the rising edge of the internal CPU clock.

The 16-bit Timer/Counter1 features both a high resolution and a high accuracy usage with lower pre-scaling values.

Similarly, high pre-scaling values make the Timer/Counter1 useful for lower speed functions or exact timing functions with infrequent actions.

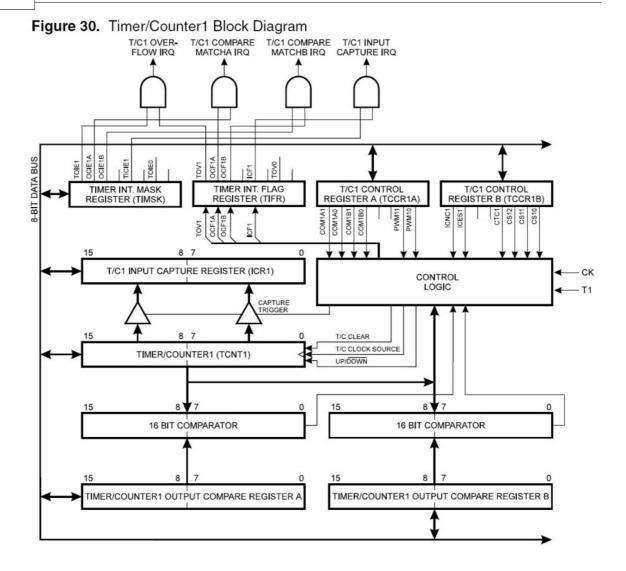
The Timer/Counter1 supports two Output Compare functions using the Output Compare Register 1 A and B -OCR1A and OCR1B as the data values to be compared to the Timer/Counter1 contents.

The Output Compare functions include optional clearing of the counter on compareA match, and can change the logic levels on the Output Compare pins on both compare matches.

Timer/Counter1 can also be used as a 8, 9 or 10-bit Pulse Width Modulator (PWM). In this mode the counter and the OCR1A/OCR1B registers serve as a dual glitch-free stand-alone PWM with centered pulses.

The Input Capture function of Timer/Counter1 provides a capture of the Timer/ Counter1 value to the Input Capture Register - ICR1, triggered by an external event on the Input Capture Pin - ICP. The actual capture event settings are defined by the Timer/Counter1 Control Register -TCCR1B.

In addition, the Analog Comparator can be set to trigger the Capture.



4.6 AVR Internal Hardware Watchdog timer

The Watchdog Timer

The Watchdog Timer is clocked from a separate on-chip oscillator which runs at approximately 1MHz. This is the typical value at VCC = 5V.

By controlling the Watchdog Timer pre-scaler, the Watchdog reset interval can be adjusted from 16K to 2,048K cycles (nominally 16 - 2048 ms). The BASCOM RESET WATCHDOG - instruction resets the Watchdog Timer.

Eight different clock cycle periods can be selected to determine the reset period.

If the reset period expires without another Watchdog reset, the AT90Sxxxx resets and program execution starts at the reset vector address.

4.7 AVR Internal Hardware Port B

Port B

Port B is an 8-bit bi-directional I/O port. Three data memory address locations are allocated for the Port B, one each for the Data Register - PORTB, \$18(\$38), Data

Direction Register - DDRB, \$17(\$37) and the Port B Input Pins - PINB, \$16(\$36). The Port B Input Pins address is read only, while the Data Register and the Data Direction Register are read/write.

All port pins have individually selectable pull-up resistors. The Port B output buffers can sink 20mA and thus drive LED displays directly. When pins PB0 to PB7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated.

The Port B pins with alternate functions are shown in the following table:

When the pins are used for the alternate function the DDRB and PORTB register has to be set according to the alternate function description.

Port	Pin	Alternate Functions
PORTB.0	то	(Timer/Counter 0 external counter input)
PORTB.1	Τ1	(Timer/Counter 1 external counter input)
PORTB.2	AIN0	(Analog comparator positive input)
PORTB.3	AIN1	(Analog comparator negative input)
PORTB.4	SS	(SPI Slave Select input)
PORTB.5	MOSI	(SPI Bus Master Output/Slave Input)
PORTB.6	MISO	(SPI Bus Master Input/Slave Output)
PORTB.7	SCK	(SPI Bus Serial Clock)

Port B Pins Alternate Functions

The Port B Input Pins address - PINB - is not a register, and this address enables access to the physical value on each Port B pin. When reading PORTB, the PORTB Data Latch is read, and when reading PINB, the logical values present on the pins are read.

PortB As General Digital I/O

All 8 bits in port B are equal when used as digital I/O pins. PORTB.X, General I/O pin: The DDBn bit in the DDRB register selects the direction of this pin, if DDBn is set (one), PBn is configured as an output pin. If DDBn is cleared (zero), PBn is configured as an input pin. If PORTBn is set (one) when the pin configured as an input pin, the MOS pull up resistor is activated.

To switch the pull up resistor off, the PORTBn has to be cleared (zero) or the pin has to be configured as an output pin.

DDBn	PORTBn	I/O	Pull up	Comment
0	0	Input	No	Tri-state (Hi-Z)

DDBn Effects on Port B Pins

0	1	Input	PBn will source current if ext. pulled low.
1	0	Output	Push-Pull Zero Output
1	1	Output	Push-Pull One Output

By default, the DDR and PORT registers are 0. CONFIG PORTx=OUTPUT will set the entire DDR register. CONFIG PINX.Y will also set the DDR register for a single bit/pin. When you need the pull up to be activated, you have to write to the PORT register.

4.8 AVR Internal Hardware Port D

Port D

Port D Pins Alternate Functions				
Port	Pin	Alternate Function		
PORTD.0	RDX	(UART Input line)		
PORTD.1	TDX	(UART Output line)		
PORTD.2	INTO	(External interrupt 0 input)		
PORTD.3	INT1	(External interrupt 1 input)		
PORTD.5	OC1A	(Timer/Counter1 Output compareA match output)		
PORTD.6	WR	(Write strobe to external memory)		
PORTD.7	RD	(Read strobe to external memory)		

Port D Pins Alternate Functions

RD - PORTD, Bit 7

RD is the external data memory read control strobe.

WR - PORTD, Bit 6

WR is the external data memory write control strobe.

OC1- PORTD, Bit 5

Output compare match output: The PD5 pin can serve as an external output when the Timer/Counter1 com-pare matches.

The PD5 pin has to be configured as an out-put (DDD5 set (one)) to serve this f unction. See the Timer/Counter1 description for further details, and how to enable the output. The OC1 pin is also the output pin for the PWM mode timer function.

INT1 - PORTD, Bit 3

External Interrupt source 1: The PD3 pin can serve as an external interrupt source to the MCU. See the interrupt description for further details, and how to enable the source

INTO - PORTD, Bit 2 INTO, External Interrupt source 0: The PD2 pin can serve as an external interrupt source to the MCU. See the interrupt description for further details, and how to enable the source.

TXD - PORTD, Bit 1

Transmit Data (Data output pin for the UART). When the UART transmitter is enabled, this pin is configured as an output regardless of the value of DDRD1.

RXD - PORTD, Bit 0

Receive Data (Data input pin for the UART). When the UART receiver is enabled this pin is configured as an output regardless of the value of DDRD0. When the UART forces this pin to be an input, a logical one in PORTD0 will turn on the internal pull-up.

When pins TXD and RXD are not used for RS-232 they can be used as an input or output pin.

No PRINT, INPUT or other RS-232 statement may be used in that case.

The UCR register will by default not set bits 3 and 4 that enable the TXD and RXD pins for RS-232 communication. It is however reported that this not works for all chips. In this case you must clear the bits in the UCR register with the following statements:

RESET UCR.3 RESET UCR.4 or as an alernative : UCR=0

4.9 Adding XRAM

Some AVR chips like the 90S8515 for example can be extended with external RAM (SRAM) memory.

On these chips Port A serves as a Multiplexed Address (A0 - A7)/Data (D0 - D7) bus. Port C also serves as the upper Address bits (A8 - A15) output when using external SRAM.

The maximum size of XRAM can be 64 Kbytes.

Example: The STK200 has a 62256 ram chip (32K x 8 bit).

Here is some info from the BASCOM user list :

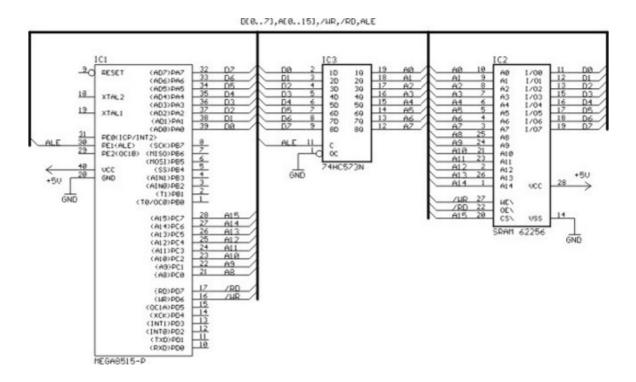
If you do go with the external ram , be careful of the clock speed. Using a 4 MHz crystal , will require a SRAM with 70 nS access time or less. Also the data latch (74HC573) will have to be from a faster family such as a 74FHC573 if you go beyond 4 MHz.

You can also program an extra wait state, to use slower memory.

Here you will find a pdf file showing the STK200 schematics: http://www.avr-forum.com/Stk200_schematic.pdf

If you use a 32 KB SRAM, then connect the /CS signal to A15 which give to the range of &H0000 to &H7FFF, if you use a 64 KB SRAM, then

tie /CS to GND, so the RAM is selected all the time.



4.10 Attaching an LCD Display

A LCD display can be connected with two methods.

- By wiring the LCD-pins to the processor port pins. This is the pin mode. The advantage is that you can choose the pins and that they don't have to be on the same port. This can make your PCB design simple. The disadvantage is that more code is needed.
- By attaching the LCD-data pins to the data bus. This is convenient when you have an external RAM chip and will add only a little extra code.

LCD DISPLAY	PORT	PIN
DB7	PORTB.7	14
DB6	PORTB.6	13
DB5	PORTB.5	12
DB4	PORTB.4	11
E	PORTB.3	6
RS	PORTB.2	4
RW	Ground	5
Vss	Ground	1
Vdd	+5 Volt	2

The LCD-display can be connected in PIN mode as follows:

Vo 0-5 Volt 3

This leaves PORTB.1 and PORTB.0 and PORTD for other purposes.

You can change these pin settings from the Options LCD and menu.

BASCOM supports many statements to control the LCD-display.

For those who want to have more control of the example below shows how to use the internal BASCOM routines.

\$ASMLdi _temp1, 5Rcall _Lcd_controlLdi _temp1,65'load register R24 with value'load register with new value (letter A)Rcall _Write_lcd'write it to the LCD-display\$END ASM

Note that _lcd_control and _write_lcd are assembler subroutines which can be called from BASCOM.

See the manufacturer's details from your LCD display for the correct pin assignment.

4.11 Memory usage

SRAM

Every variable uses memory. Variables are stored in memory. This memory is also called SRAM. (static ram).

The available memory depends on the chip. When you double click on the chip pinout, you can view the parameters of the used chip.

A special kind of memory are the registers in the AVR. Registers 0-31 have addresses 0-31.

Almost all registers are used by the compiler or might be used in the future. Which registers are used depends on the program statements you use.

This brings us back to the SRAM.

No SRAM is used by the compiler other than the space needed for the software stack and frame.

Some statements might use some SRAM. When this is the case it is mentioned in the help topic of that statement.

For example, CONFIG CLOCK in user mode requires variables to hold the time.

Each 8 bits used occupy one byte. When you dimension 1 bit, you will also use 1 byte.

Each byte variable occupies one byte.

Each integer/word variable occupies two bytes.

Each Long, Dword or Single variable occupies four bytes.

Each double variable occupies 8 bytes.

Each string variable occupies at least 2 bytes.

A string with a length of 10 occupies 11 bytes. The extra byte is needed to indicate

164 BASCOM-AVR

the end of the string.

Use bits or byte variables wherever you can to save memory. (not allowed for negative values)

The software stack is used to store the addresses of LOCAL variables and for variables that are passed to SUB routines.

Each LOCAL variable and passed variable to a SUB/FUNCTION, requires two bytes to store the address. So when you have a SUB routine in your program that passes 10 variables, you need 10 * 2 = 20 bytes. When you use 2 LOCAL variables in the SUB program that receives the 10 variables, you need additional 2 * 2 = 4 bytes.

The software stack size can be calculated by taking the maximum number of parameters in a SUB routine, adding the number of LOCAL variables and multiplying the result by 2. To be safe, add 4 more bytes for internally used LOCAL variables.

LOCAL variables are stored in a place that is named the Frame.

When you have a LOCAL STRING with a size of 40 bytes, and a LOCAL LONG, you need 41 + 4 bytes = 45 bytes of frame space.

When you use conversion routines such as STR(), VAL(), HEX(), INPUT etc. that convert from numeric to string and vice versa, you also need a frame. In fact, the compiler creates a buffer of 24 bytes that serves as scratchpad for temporare;y variables, and conversion buffer space. So the frame space should be 24 at minimum.

Note that the use of the INPUT statement with a numeric variable, or the use of the PRINT or LCD statement with a numeric variable, will also force you to reserve 24 bytes of frame space. This because these routines use the internal numeric<>string conversion routines.

XRAM

Some processors have an external memory interface. For example the atmegs128 has such an interface.

The additional memory is named XRAM memory(extended or external memory) When you add 32 KB RAM, the first address will be 0.

But because the XRAM can only start after the internal SRAM, the lower memory locations of the XRAM will not be available for use. The processor will automatically use the SRAM if an address is accessed that is in range of the SRAM memory. Thus adding 32KB of XRAM, will result in a total of 32 KB RAM.

ERAM

Most AVR chips have internal EEPROM on board. This EEPROM can be used to store and retrieve data. In BASCOM, this data space is called ERAM.

An important difference is that an ERAM variable can only be written to a maximum of 100.000 times. So only assign an ERAM variable when it is required, and **never** use it in a loop or the ERAM will become unusable.

Always use the Brown out detection of the processor to prevent EEPROM corruption.

Constant code usage

Constants are stored in a constant table. Each used constant in your program will end up in the constant table.

For example:

Print "ABCD" Print "ABCD"

This example will only store one constant (ABCD).

Print "ABCD" Print "ABC"

In this example, two constants will be stored because the strings differ.

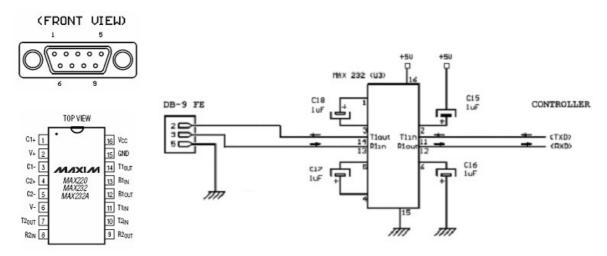
4.12 Using the UART

UART

A Universal Asynchronous Receiver and Transmitter (UART) can be used to send and receive data between two devices. More specific these devices can be PC-to-PC, PC-to-micro controller and micro controller-to-micro controller. The UART communicates using TTL voltages +5V and 0V or LVTTL depending on your micro controllers VCC voltage.

If you wish to connect to a PC you need to use RS232 protocol specifications. This means that the hardware communication is done with specific voltage levels. (+15V and -15V) This can be achieved by using a MAX232 level shifter.

The hardware is explained in this schematic:



The DB-9 connector has 9 pins but you only need to use 3 of them. Notice that the drawing above shows the FRONT VIEW thus remember that you are soldering on the other side. On most connectors the pin outs can also be found on the connector itself.

If your controller has no UART you can use a software UART see below. If your controller has one UART you connect controller pins TxD and RxD to TxD and RxD in the schematic above. If your controller has more than one UART you connect controller pins TxD0 and RxD0 to TxD and RxD in the schematic above. You now need to initialize the program in your micro controller, open a new .bas file

166 BASCOM-AVR

and add the following code in the beginning of your program.

```
$regfile = "your micro here def.dat"
$crystal = 8000000
$baud = 19200
```

Make sure to define your micro controller after \$regfile for example if you use the ATMega32 regfile = "m32def.dat"

Some new chips can use an internal oscillator, also some chips are configured to use the internal oscillator by default. Using an internal oscillator means you do not need an external crystal.

Perform this step only if you have an internal oscillator.

Open the BASCOM-AVR programmer like this:



- Select the "Lock and Fuse Bits" tab and maximize the programmer window.
- Check if you see the following in the "Fusebit" section:
 - "1:Divide Clock by 8 Disabled" and

"Int. RC Osc. 8 MHz; Start-up time: X CK + X ms; [CKSEL=XXXX SUT=XX]"

AVR ISP STK progra	immer and a second s	
	👗 📅 🚡 📲 📓 Chip ATmega88 🔹 🔮 🕐	
Manufactor Atmel		
FlashROM EEPRO	DM Lock and Fuse Bits	
Chip		Betresh
Name	MEGA88	- Tieneon
Calibration 0	9A	Write LB
Lockbits		
Lockbit 65	11:No restrictions for SPM or LPM accessing the boot loader section	Write FS
Lockbit 43	11:No restrictions for SPM or LPM accessing the application section	
Lockbit 21	11:No memory lock features enabled for parallel and serial programming	Write FSH
Fusebits		
Fusebit C	1:Divide Clock by 8 Disabled	Write FSE
Fusebit B	1:CLOCK Output disabled	
Fusebit KLA987	100010.int. RC Osc. 8 MHz: Start-up time PWRDWN/RESET: 6 CK/14 CK + 65 ms: [CKSEL=0	1010 SUT=1 Write PRG
Fusebits High		
FusebitK	1:PIN PC6 is RESET	
Fusebit J	1:debugWIRE Disabled	
Fusebitl	0:SPI enabled	
Fusebit H	1.WDT enabled by WDTCR	
Fusebit G	1:Erase EEPROM when chip erase	
Fusebit DEF	111:Brown Out Disabled	
Fusebits Exten	ded Set and read Lock and F	use bits
Fusebit RS	00:Bootsize 1024 words	
Fusebit Q	1:Select RESET vector (0000)	
/R ISP STK progra	mmer	
46 ROM	0 EPROM EDBEXPERIMENT68.80N	

These options are not available for all AVR's, if you don't have the option do not change any fuse bits.

If these options are available, but in a wrong setting. Change the setting in the drop down box and click another Fuse section. Finally click the "Program FS" button. Click

"Refresh" to see the actual setting.

Now connect a straight cable between the DB-9 connector, micro controller side and the PC side.

Program a test program into your micro controller, it should look like this:

End

Now open the BASCOM-AVR Terminal and set your connection settings by clicking "Terminal" -> "Settings" Select your computers COM port and select baud 19200, Parity none, Data bits 8, Stop bits 1, Handshake none, emulation none.

mpeer coll	munication	Environment	Simulator Progr	ammer Monitor Pr	inter
COM port	COM7		Handshake	None	•
Baudrate	19200		Emulation	NONE	-
Parity	None			🗆 RTS	
Databits	8		Font	Font	
Stopbits	1		Backcolor	Navy	-
Default]		Ok 🗙	Cancel	
Default]	_	<u>0</u> k X	Cancel	
] t Terminal em		Ok X	Cancel	=O ×
BASCOM-AVR	Terminal em			Cancel	

If you see the Hello World displayed in the BASCOM-AVR Terminal emulator window, your configuration is OK. Congratulations.

Example

You can also try this example with the BASCOM Terminal emulator, it shows you how to send and receive with various commands.

```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
```

168 BASCOM-AVR

```
Dim Akey As Byte 'Here we declare a byte variable
Print
Print "Hello, hit any alphanumerical key..."
Akey = Waitkey() 'Waitkey waits untill a char is received from the UART
Print Akey
Wait 1
Print
Print "Thanks!, as you could see the controller prints a number"
Print "but not the key you pressed."
Wait 1
Print
Print "Now try the enter key..."
Akey = Waitkey()
Akey = Waitkey()
Print Akey
Print
Print "The number you see is the ASCII value of the key you pressed."
Print "We need to convert the number back to the key..."
       'Notice what this line does
Print
Print "Please try an alphanumerical key again..."
Akey = Waitkey()
Print Chr(akey) 'Notice what this does
Print "That's fine!"
Wait 1
Print
Print "For a lot of functions, just one key is not enough..."
Print "Now type your name and hit enter to confirm"
Dim Inputstring As String * 12
                                       'Declare a string variable here
Do
Akey = Waitkey()
If Akey = 13 Then Goto Thanks
                                       'On enter key goto thanks
   Inputstring = Inputstring + Chr(akey) 'Assign the string
Loop
Thanks:
Print "Thank you " ; Inputstring ; " !"
                                                   'Notice what ; does
Wait 1
Print
Print "Take a look at the program code and try to understand"
Print "how this program works. Also press F1 at the statements"
Print
Print "If you understand everything continue to the next experiment"
End
```

ASCII

As you could have seen in the previous example we use the PRINT statement to send something to the UART. Actually we do not send just text. We send ASCII characters. ASCII means American Standard Code for Information Interchange. Basically ASCII is a list of 127 characters.

```
ASCII Table (Incomplete)
```

Decimal	Hex	Binary	Value	
000	000	00000000	NUL	(Null char.)
008	800	00001000	BS	(Backspace)
009	009	00001001	HT	(Horizontal Tab)
010	00A	00001010	LF	(Line Feed)
012	00C	00001100	FF	(Form Feed)
013	00D	00001101	CR	(Carriage Return)
048	030	00110000	0	
049	031	00110001	1	
052	034	00110100	4	
065	041	01000001	A	
066	042	01000010	В	
067	043	01000011	С	

You can find a complete ASCII table here 303

CARRIAGE RETURN (CR) AND LINE FEED (LF)

In the previous example you can also see that a second print statement always prints the printed text to the following line. This is caused by the fact that the print statement always adds the CR and LF characters.

Basically if we state: **Print** "ABC" We send 65 66 67 13 10 to the UART. (In binary format)

The carriage return character (13) returns the cursor back to column position 0 of the current line. The line feed (10) moves the cursor to the next line.

Print "ABC" ;

When we type a semicolon (;) at the end of the line... Bascom does not send a carriage return/line feed, so you can print another text after the ABC on the same line.

Print "ABC" ; Chr(13) ;

This would send only ABC CR. The next print would overwrite the ABC.

OVERVIEW

Here are some other commands that you can use for UART communications:

Waitkey()

Waitkey will until a character is received in the serial buffer.

lscharwaiting()

Returns 1 when a character is waiting in the hardware UART buffer.

Inkey()

Inkey returns the ASCII value of the first character in the serial input buffer.

Print

Sends a variable or non-variable string to the UART

ANOTHER EXAMPLE

This example shows how to use Ischarwaiting to test if there is a key pressed. And if there is, react to a variable.

'Print "Press B key to start" Dim Serialcharwaiting As Byte, Serialchar As Byte

```
170 BASCOM-AVR
```

```
Serialcharwaiting = Ischarwaiting() 'Check if B or b pressed then goto
If Serialcharwaiting = 1 Then
Serialchar = Inkey()
If Serialchar = 66 Or Serialchar = 98 Then
Goto MyRoutine
End If
Goto Main
Myroutine:
'Statements
Main:
'Statements
End
```

BUFFERING SERIAL DATA

If you wish to send and receive data at high speed, you need to use serial input and serial output buffers. This buffering is implemented in BASCOM-AVR and can only be used for hardware UART's.

To configure a UART to use buffers, you need to use the Config statement.

Config Serialout = Buffered , Size = 20 and/or **Config** Serialin = Buffered , Size = 20

More information can be found in BASCOM-Help. Search topic = "config serialin" [54 \hat{o}]. There is also a sample program "RS232BUFFER.BAS" in the samples folder if you wish a demonstration of the buffering.

SOFTWARE UART

The previous examples used the hardware UART. That means the compiler uses the internal UART registers and internal hardware (RxD(0) and TxD(0)) of the AVR. If you don't have a hardware UART you can also use a software UART.

The Bascom compiler makes it easy to "create" additional UART's. Bascom creates software UART's on virtually every port pin.

Remember that a software UART is not as robust as a hardware UART, thus you can get timing problems if you have lots of interrupts in your program.

For this example we use micro controller pins portc.1 and portc.2. Connect portc.1 to TxD and portc.2 to RxD see the schematic above.

Change the \$regfile and program this example:

```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
Dim B As Byte
Waitms 100
'Open a TRANSMIT channel for output
Open "comc.1:19200,8,n,1" For Output As #1
```

```
Print #1 , "serial output"
'Now open a RECEIVE channel for input
Open "comc.2:19200,8,n,1" For Input As #2
'Since there is no relation between the input and output pin
'there is NO ECHO while keys are typed
Print #1 , "Press any alpha numerical key"
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
   'Store in byte
  B = Inkey(#2)
   'When the value > 0 we got something
  If B > 0 Then
     Print #1 , Chr(b)
                                                 'Print the character
  End If
Loop
Close #2
                                                  'Close the channels
Close #1
End
```

After you have programmed the controller and you connected the serial cable, open

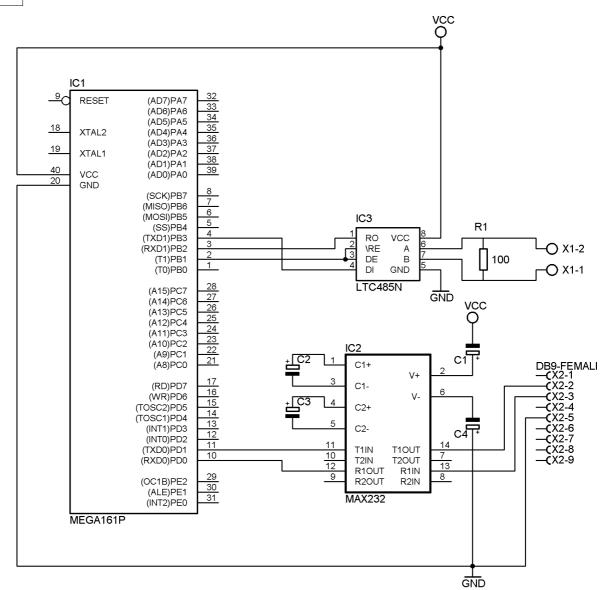
the terminal emulator by clicking on in Bascom. You should see the program asking for an alphanumerical input, and it should print the input back to the terminal.

4.13 USING RS485

RS485

RS485 is used for serial communication and well suited for transmission over large distances.

Similar to RS232 we need a level shifter.



The sample above uses a MEGA161 or MEGA162 which has 2 UARTS. This way you can have both a RS232 and RS485 interface.

The RS232 is used for debugging.

In order to test you need 2 or more similar circuits. One circuit would be the master. The other(s) would be a slave.

The same hardware is used to test the MODBUS protocol. The bus need to be terminated at both ends with a resistor. 100 ohm is a typical used value. The GND of both circuits may not be connected ! Only connect point A and B from both circuits. For industrial usage it is best to use an optical isolated level shifter.

Simple MASTER sample

\$regfile = "m162def.dat"
\$crystal = 8000000
\$baud = 19200
\$hwstack = 42
\$swstack = 40
\$framesize = 40

\$lib "modbus.lbx"
Config Print1 = Portb.1 , Mode = Set

' specify the used micro

' use baud rate

- ' default use 32 for the hardware stack
- ' default use 10 for the SW stack
- ' default use 40 for the frame space

' use portb.1 for the direction

Rs485dir Alias Portb.1 Config Rs485dir = Output Rs485dir = 0 ' go to receive mode Portc.0 = 1 ' a switch is connected to pinc.0 so activate pull up resistor TX RX 'COM0 PD.1 PD.0 monitor 'COM1 PB.3 PB.2 rs485 PB.1 data direction rs485 Config Com1 = Dummy, Synchrone = 0, Parity = None, Stopbits = 1, Databits = 8, Clockpol = 0Config Com2 = 9600, Synchrone = 0, Parity = Even, Stopbits = 1, Databits = 8, Clockpol = 0 ' MUST MATCH THE SLAVE 'use OPEN/CLOSE for using the second UART Open "COM2:" For Binary As #1 'dimension some variables Dim B As Byte Dim W As Word Dim L As Long W = &H4567 ' set some values L = &H12345678Print "RS-485 MODBUS master" Do If Pinc.0 = 0 Then ' test button ' delay since we want to send just 1 Waitms 500 frame Print "send request to slave/server" ' to debug terminal ' Print #1 , Makemodbus(2 , 3 , 8 , 2); 'slave 2, function 3, start address 8, 2 bytes ' Print #1 , Makemodbus(2 , 6 , 8 , W); 'slave 2, function 6, address 8, value of w Print #1, Makemodbus(b, 16, 8, L); 'send a long End If If Ischarwaiting(#1) <> 0 Then 'did we got something back? B = Waitkey(#1) ' yes so get it Print Hex(b) ; ","; ' print it End If Loop

A slave would simply listen to data, and once enough data received, send it back. The MODBUS slave code is available as a commercial add on.

4.14 Using the I2C protocol

I²C bus

 $\rm I^2C$ bus is an abbreviation for Inter Integrated Circuit bus. It is also known as IIC and I2C.

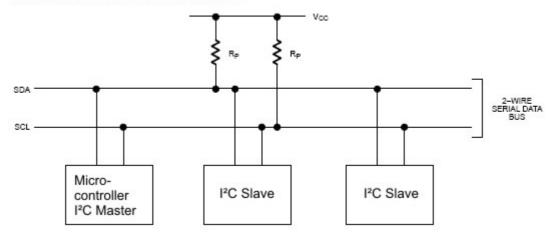
I²C is a serial and synchronous bus protocol. In standard applications hardware and

174 BASCOM-AVR

timing are often the same. The way data is treated on the I²C bus is to be defined by the manufacturer of the I²C master and slave chips.

In a simple I²C system there can only be one master, but multiple slaves. The difference between master and slave is that the master generates the clock pulse. The master also defines when communication should occur. For bus timing it is important that the slowest slave should still be able to follow the master's clock. In other words the bus is as fast as the slowest slave.

A typical hardware configuration is shown in the figure below:

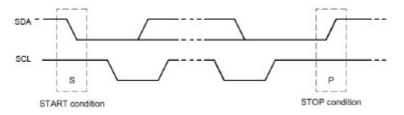


TYPICAL 2-WIRE BUS CONFIGURATION

Note that more slave chips can be connected to the SDA and SCL lines, normally Rp has a value of 1kOHM. The clock generated by the master is called Serial Clock (SCL) and the data is called Serial Data (SDA).

In most applications the micro controller is the I²C Master. Slave chips can be Real Time Clocks and Temperature sensors. For example the DS1307 and the DS1624 from <u>www.maxim-ic.com</u>. Of coarse you can also create your own slaves. In that case there is micro controller to micro controller communication.

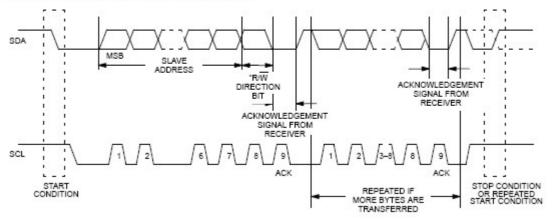
LOGIC BUS LEVELS AND CONDITIONS



Data can only occur after the master generates a **start condition.** A start condition is a high-to-low transition of

the SDA line while SCL remains high. After each data transfer a **stop condition** is generated. A stop condition is a low-to-high transition of the SDA line while SCL remains high.





As said a data transfer can occur after a **start condition** of the master. The length of data sent over I²C is always 8 bit this includes a read/write direction bit, so you can effectively send 7 bits every time.

The most significant bit MSB is always passed first on the bus.

If the master writes to the bus the R/W bit = 0 and if the master reads the R/W bit = 1.

After the R/W bit the master should generate one clock period for an acknowledgement ACK.

Each receiving chip that is addressed is obliged to generate an acknowledge after the reception of each byte. A chip that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse.

After an acknowledge there can be a stop condition, if the master wishes to leave the bus idle. Or a repeated start condition. A repeated start is the same as a start condition.

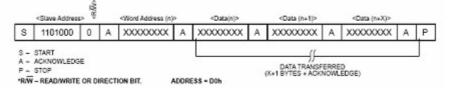
When the master reads from a slave it should acknowledge after each byte received. There are two reasons for the master not to acknowledge. The master sends a not acknowledge if data was not received correctly or if the master wishes the stop receiving.

In other words if the master wishes to stop receiving, it sends a not acknowledge after the last received byte.

The master can stop any communication on the bus **at any time** by sending a stop condition.

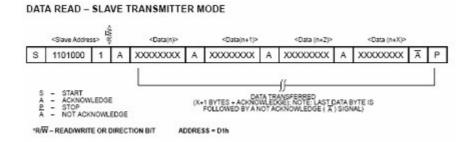
BUS ADRESSING

Let's say we have a slave chip with the address "1101000" and that the master wishes to write to that slave, the slave would then be in receiver mode, like this: DATA WRITE - SLAVE RECEIVER MODE



You can see here that the master always generates the start condition, then the master sends the address of the slave and a "0" for R/W. After that the master sends a command or word address. The function of that command or word address can be found in the data sheet of the slave addressed.

After that the master can send the data desired and stop the transfer with a stop condition.

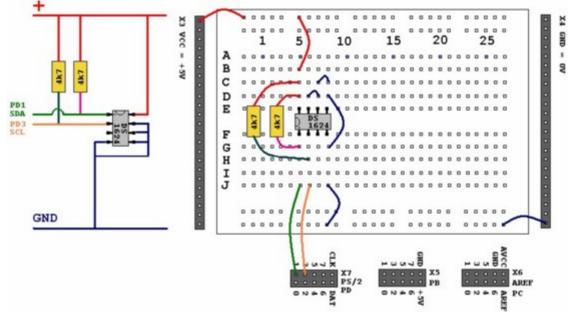


Again the start condition and the slave address, only this time the master sends "1" for the R/W bit. The slave can then begin to send after the acknowledge. If the master wishes to stop receiving it should send a not acknowledge.

EXAMPLE

This example shows you how to setup and read the temperature from a DS1624 temperature sensor.

Connect the DS1624 like this:



Then program this sample into your micro controller and connect your micro controller to the serial port of your PC.

<pre>\$regfile = "m88def.dat" \$crystal = 8000000 \$baud = 19200</pre>	'Define the chip you use 'Define speed 'Define UART BAUD rate
'Declare RAM for temperature storage Dim I2ctemp As Byte	'Storage for the temperature

```
'Configure pins we want to use for the I<sup>2</sup>C bus
Config Scl = Portd.1
                                      'Is serial clock SCL
Config Sda = Portd.3
                                      'Is serial data SDA
'Declare constants - I2C chip addresses
Const Ds1624wr = &B10010000 'DS1624 Sensor write
Const Ds1624rd = &B10010001
                                      'DS1624 Sensor read
'This section initializes the DS1624
                                       'Sends start condition
   I2cstart
                                       'Sends the address
   I2cwbyte Ds1624wr
'byte with r/w 0
'Access the CONFIG register (&HAC address byte)
  I2cwbyte & HAC
'Set continuous conversion (&H00 command byte)
  I2cwbyte &H00
                                      'Sends stop condition
  I2cstop
                    'We have to wait some time after a stop
  Waitms 25
  I2cstart
   I2cwbyte Ds1624wr
'Start conversion (&HEE command byte)
  I2cwbyte & HEE
   I2cstop
  Waitms 25
'End of initialization
Print
                                  'Print empty line
Do
   'Get the current temperature
   I2cstart
   I2cwbyte Ds1624wr
   I2cwbyte &HAA
                     'Read temperature (&HAA command byte)
   I2cstart
   I2cwbyte Ds1624rd 'The chip will give register contents
'Temperature is stored as 12,5 but the ,5 first
   I2crbyte I2ctemp
'So you'll have to read twice... first the ,5
   I2crbyte I2ctemp , Nack
'And then the 12... we don't store the ,5
   I2cstop
                                                                 'That's
why we read twice.
'We give NACK if the last byte is read
   'Finally we print
Print "Temperature: " ; Str(i2ctemp) ; " degrees" ; Chr(13);
  Waitms 25
Loop
End
```

178 BASCOM-AVR

You should be able to read the temperature in your terminal emulator. Note that the used command bytes in this example can be found in DS1624 temperature sensor data sheet.

OVERVIEW

Config Sda = Portx.x Configures a port pin for use as serial data SDA.

Config Scl = Portx.x Configures a port pin for use as serial clock SCL.

I2cstart Sends the start condition.

I2cstop Sends the stop condition.

I2cwbyte Writes one byte to an I²Cslave.

I2crbyte Reads one byte from an I²Cslave. I2csend Writes a number of bytes to an I²Cslave.

I2creceive

Reads a number of bytes from an I²Cslave.

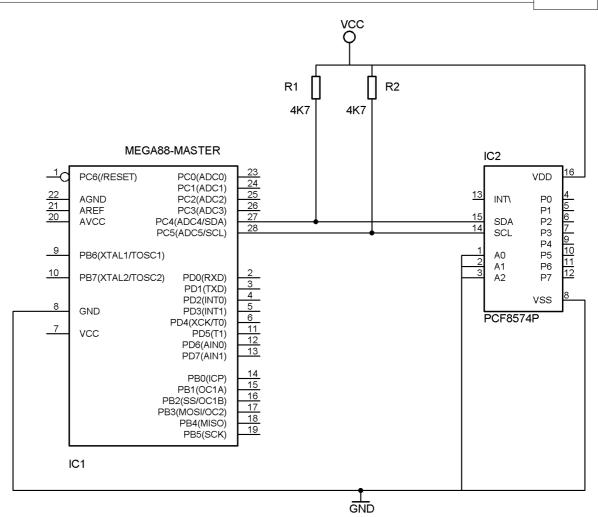
Practice

The design below shows how to implement an I2C-bus. The circuit is using a Mega88 as a master.

The TWI bus is used. While you can use any pin for software mode I2C, when a micro has TWI hardware build in, it is advised to use the TWI hardware.

R1 and R2 are 4K7 pull up resistors.

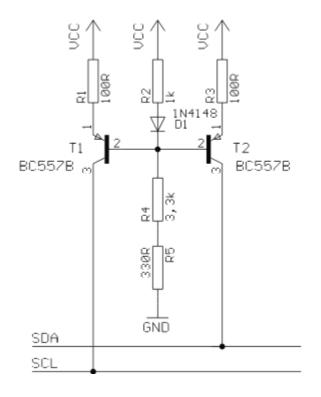
There are many I2C slave chips available. The example shows the PCF8574. With the additional TWI slave library you can make your own slave chips.



The following information was submitted by Detlef Queck.

Many people have problems over and over with I2C(TWI) Termination. Use 4,7k or 10 k pull up? How long can the SCL, SDA line be when used with pull ups etc, etc.

You can simplify this confusing problem. Here is a Schematic for an active Termination of I2C and TWI. We have used this Schematic for over 10 years, and have had no problems with it. The I2C (TWI) lines can be up to 80cm (400KHz) without any problem when the Terminator is at the end of the lines.



4.15 Using the 1 WIRE protocol

The 1-wire protocol was invented by Dallas Semiconductors and needs only 1 wire for two-way communication. You also need power and ground of course.

This topic is written by Göte Haluza. He tested the new 1-wire search routines and is building a weather station.

Dallas Semiconductor (DS) 1-wire. This is a brief description of DS 1-wire bus when used in combination with BASCOM. For more detailed explanations about the 1-wire bus, please go to http://www.maxim-ic.com. Using BASCOM makes the world a lot easier. This paper will approach the subject from a "BASCOM-user-point-of-view".

1-wire-net is a serial communication protocol, used by DS devices. The bus could be implemented in two basic ways :

With 2 wires, then DQ and ground is used on the device. Power is supplied on the DQ line, which is +5V, and used to charge a capacitor in the DS device. This power is used by the device for its internal needs during communication, which makes DQ go low for short periods of time. This bus is called the 1-wire bus.

With 3 wires, when +5V is supplied to the VDD line of the device, and DQ + ground as above. This bus is called the 2-wire bus.

So, the ground line is "not counted" by DS. But hereafter we use DS naming conventions.

How it works. (1-wire)

The normal state of the bus is DQ=high. Through DQ the device gets its power, and

performs the tasks it is designed for.

When the host (your micro controller (uC)) wants something to happen with the 1wire bus, it issues a reset-command. That is a very simple electric function that happens then; the DQ goes active low for a time (480uS on original DS 1-wire bus). This put the DS-devices in reset mode; then (they) send a presence pulse, and then (they) listen to the host.

The presence pulse is simply an active low, this time issued by the device(s).

Now, the host cannot know what is on the bus, it is only aware of that at least 1 DS device is attached on the bus.

All communication on the 1-wire bus is initialized by the host, and issued by timeslots of active-low on a normally high line (DQ), issued by the device, which is sending at the moment. The devices(s) internal capacitor supplies its power needs during the low-time.

How do you work with 1-wire-bus

Thereafter, you can read a device, and write to it. If you know you only have 1 sensor attached, or if you want to address all sensors, you can start with a "Skip Rom" - command. This means; take no notice about the IDs of the sensors - skip that part of the communication.

When you made a 1-wire-reset, all devices of the bus are listening. If you chose to address only one of them, the rest of them will not listen again before you have made a new 1-wire-reset on the bus.

I do not describe BASCOM commands in this text - they are pretty much selfexplanatory. But the uC has to write the commands to the bus - and thereafter read the answer. What you have to write as a command depends on devices you are using - and what you want to do with it. Every DS chip has a data sheet, which you can find at http://www.dalsemi.com/datasheets/pdfindex.html. There you can find out all about the actual devices command structure.

There are some things to have in mind when deciding which of the bus-types to use.

The commands, from BASCOM, are the same in both cases. So this is not a problem.

The +5V power-supply on the VDD when using a 2-wire bus has to be from a separate power supply, according to DS. But it still works with taking the power from the same source as for the processor, directly on the stabilizing transistor. I have not got it to work taking power directly from the processor pin.

Some devices consume some more power during special operations. The DS1820 consumes a lot of power during the operation "Convert Temperature". Because the sensors knows how they are powered (it is also possible to get this information from the devices) some operations, as "Convert T" takes different amount of time for the sensor to execute. The command "Convert T" as example, takes ~200mS on 2-wire, but ~700mS on 1-wire. This has to be considered during programming.

And that power also has to be supplied somehow.

If you use 2-wire, you don't have to read further in this part. You can do simultaneously "Convert T" on all the devices you attach on the bus. And save time. This command is the most power-consuming command, possible to execute on several devices, I am aware of.

If you use 1-wire, there are things to think about. It is about not consuming more power than you feed. And how to feed power? That depends on the devices (their consumption) and what you are doing with them (their consumption in a specific operation).

Short, not-so-accurate description of power needs, not reflecting on cable lengths.

Only the processor pin as power supplier, will work < 5 sensors. (AVR, 1-wire-functions use an internal pull-up. 8051 not yet tested). Don't even think of simultaneous commands on multiple sensors.

With +5V through a 4K7 resistor, to the DQ-line, 70 sensors are tested. But, take care, cause issuing "Convert T" simultaneously, would cause that to give false readings. About ~15 sensors is the maximum amount of usable devices, which simultaneously performs some action. This approach DS refers to as "pull-up resistor".

With this in mind, a bus with up to 70 devices has been successfully powered this way.

The resistor mentioned, 4K7, could be of smaller value. DS says minimum 1K5, I have tested down to 500 ohm - below that the bus is not usable any more. (AVR). Lowering the resistor feeds more power - and makes the bus more noise resistant. But, the resistor minimum value is naturally also depending on the uC-pin electric capabilities. Stay at 4K7 - which is standard recommendation.

DS recommends yet another approach, called "strong pull-up" which (short) works via a MOS-FET transistor, feeding the DQ lines with enough power, still on 1-wire, during power-consuming tasks. This is not tested, but should naturally work. Because this functionality is really a limited one; BASCOM has no special support for that. But anyway, we tell you about it, just in case you wonder. Strong pull-up has to use one uC pin extra - to drive the MOS-FET.

Cable lengths (this section is only for some limitation understanding) For short runs up to 30 meters, cable selection for use on the 1-Wire bus is less critical. Even flat modular phone cable works with limited numbers of 1-Wire devices. However, the longer the 1-Wire bus, the more pronounced cable effects become, and therefore greater importance is placed on cable selection.

For longer distances, DS recommends twisted-pair-cable (CAT5).

DS standard examples show 100 meters cable lengths, so they say, that's no problem. They also show examples with 300m cabling, and I think I have seen something with 600-meter bus (but I cant find it again).

Noise and CRC

The longer cable and the noisier environment, the more false readings will be made. The devices are equipped with a CRC-generator - the LSByte of the sending is always a checksum. Look in program examples to learn how to re-calculate this checksum in your uC. AND, if you notice that there are false readings - do something about your cables. (Shield, lower resistor)

Transfer speed

On the original 1-wire bus, DS says the transfer speed is about 14Kbits /second. And, if that was not enough, some devices has an overdrive option. That multiplies the speed by 10. This is issued by making the communication-time-slots smaller (from 60 uS to 6uS) which naturally will make the devices more sensitive, and CRC-error will probably occur more often. But, if that is not an issue, ~140Kbit is a reachable speed to the devices. So, whatever you thought before, it is FAST.

The BASCOM scanning of the bus is finds about 50 devices / second , and reading a specific sensors value to a uC should be about 13 devices / second.

Topology

Of the 1w-net - that is an issue we will not cover so much. Star-net, bus-net? It seems like you can mix that. It is a bus-net, but not so sensitive about that.

The benefit of the 1-wire bus

Each device is individual - and you can communicate with it over the media of 2 wires. Still, you can address one individual device, if you like. Get its value. There are $64 \land 2$ unique identifications-numbers.

Naturally, if lot of cables are unwanted, this is a big benefit. And you only occupy 1 processor pin.

DS supplies with different types of devices, which all are made for interfacing an uC directly. No extra hardware. There are sensors, so you can get knowledge about the real world, and there are also potentiometers and relays, so you can do something about it. On the very same bus.

And the Ibutton approach from DS (ever heard of it?) is based on 1wire technology. Maybe something to pick up.

BASCOM let you use an uC with 1wire-devices so easy, that (since now) that also has to count as a benefit - maybe one of the largest. ;-)

The disadvantages of the 1-wire bus

So far as I know, DS is the only manufacturer of sensors for the bus. Some people think their devices are expensive. And, until now, it was really difficult to communicate with the devices. Particularly when using the benefit of several devices on one bus. Still some people say that the 1w-bus is slow - but I don't think so.

Göte Haluza System engineer

4.16 Using the SPI protocol

General description of the SPI

The SPI allows high-speed synchronous data transfer between the AVR and peripheral devices or between several AVR devices. On most parts the SPI has a second purpose where it is used for In System Programming (ISP).

The interconnection between two SPI devices always happens between a master device and a slave device. Compared to some peripheral devices like sensors which can only run in slave mode, the SPI of the AVR can be configured for both master and

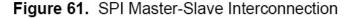
slave mode.

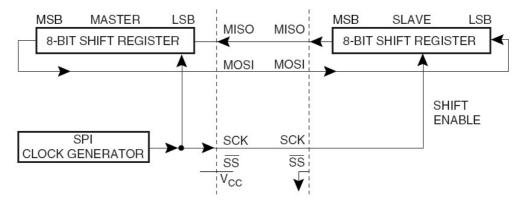
The mode the AVR is running in is specified by the settings of the master bit (MSTR) in the SPI control register (SPCR).

Special considerations about the /SS pin have to be taken into account. This will be described later in the section "Multi Slave Systems - /SS pin Functionality".

The master is the active part in this system and has to provide the clock signal a serial data transmission is based on. The slave is not capable of generating the clock signal and thus can not get active on its own.

The slave just sends and receives data if the master generates the necessary clock signal. The master however generates the clock signal only while sending data. That means that the master has to send data to the slave to read data from the slave.





Data transmission between Master and Slave

The interaction between a master and a slave AVR is shown in Figure 1. Two identical SPI units are displayed. The left unit is configured as master while the right unit is configured as slave. The MISO, MOSI and SCK lines are connected with the corresponding lines of the other part.

The mode in which a part is running determines if they are input or output signal lines. Because a bit is shifted from the master to the slave and from the slave to the master simultaneously in one clock cycle both 8-bit shift registers can be considered as one 16-bit circular shift register. This means that after eight SCK clock pulses the data between master and slave will be exchanged.

The system is single buffered in the transmit direction and double buffered in the receive direction. This influences the data handling in the following ways:

1. New bytes to be sent can not be written to the data register (SPDR) / shift register before the entire shift cycle is completed.

2. Received bytes are written to the Receive Buffer immediately after the transmission is completed.

3. The Receive Buffer has to be read before the next transmission is completed or data will be lost.

4. Reading the SPDR will return the data of the Receive Buffer.

After a transfer is completed the SPI Interrupt Flag (SPIF) will be set in the SPI Status Register (SPSR). This will cause the corresponding interrupt to be executed if this interrupt and the global interrupts are enabled. Setting the SPI Interrupt Enable

(SPIE) bit in the SPCR enables the interrupt of the SPI while setting the I bit in the SREG enables the global interrupts.

Pins of the SPI

The SPI consists of four different signal lines. These lines are the shift clock (SCK), the Master Out Slave In line (MOSI), the Master In Slave Out line (MISO) and the active low Slave Select line (/SS). When the SPI is enabled, the data direction of the MOSI, MISO, SCK and /SS pins are overridden according to the following table.

Pin Direction Overrides	Master SPI Mode Direction Overrides	Slave SPI Modes			
MOSI	User Defined	Input			
MISO	Input	User Defined			
SCK	User Defined	Input			
SS	User Defined	Input			

Table 1. SPI Pin Overrides

This table shows that just the input pins are automatically configured. The output pins have to be initialized manually by software. The reason for this is to avoid damages e.g. through driver contention.

Multi Slave Systems - /SS pin Functionality

The Slave Select (/SS) pin plays a central role in the SPI configuration. Depending on the mode the part is running in and the configuration of this pin, it can be used to activate or deactivate the devices. The /SS pin can be compared with a chip select pin which has some extra features. In master mode, the /SS pin must be held high to ensure master SPI operation if this pin is configured as an input pin. A low level will switch the SPI into slave mode and the hardware of the SPI will perform the following actions:

1. The master bit (MSTR) in the SPI Control Register (SPCR) is cleared and the SPI system becomes a slave. The direction of the pins will be switched according to Table 1.

2. The SPI Interrupt Flag (SPIF) in the SPI Status Register (SPSR) will be set. If the SPI interrupt and the global interrupts are enabled the interrupt routine will be executed. This can be useful in systems with more than one master to avoid that two masters are accessing the SPI bus at the same time. If the /SS pin is configured as output pin it can be used as a general purpose output pin which does not affect the SPI system.

Note: In cases where the AVR is configured for master mode and it can not be ensured that the /SS pin will stay high between two transmissions, the status of the MSTR bit has to be checked before a new byte is written. Once the MSTR bit has been cleared by a low level on the /SS line, it must be set by the application to re-enable SPI master mode.

In slave mode the /SS pin is always an input. When /SS is held low, the SPI is activated and MISO becomes output if configured so by the user. All other pins are inputs. When /SS is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data.

Table 2 shows an overview of the /SS Pin Functionality.

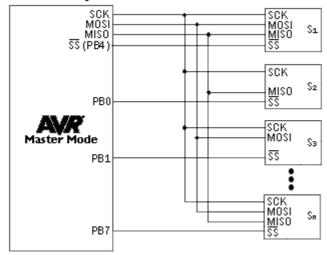
Note: In slave mode, the SPI logic will be reset once the /SS pin is brought high. If the /SS pin is brought high during a transmission, the SPI will stop sending and receiving immediately and both data received and data sent must be considered as lost.

Mode	/SS Config	/SS Pin level	Description
Slave	Always input	High	Slave deactivated
		Low	Slave activated
Master	Input	High	Master activated
		Low	Master deactivated
	Output	High	Master activated
		Low	

TABLE 2. Overview of SS pin.

As shown in Table 2, the /SS pin in slave mode is always an input pin. A low level activates the SPI of the device while a high level causes its deactivation. A Single Master Multiple Slave System with an AVR configured in master mode and /SS configured as output pin is shown in Figure 2. The amount of slaves, which can be connected to this AVR is only limited by the number of I/O pins to generate the slave select signals.

Multi Slave system



The ability to connect several devices to the same SPI-bus is based on the fact that only one master and only one slave is active at the same time. The MISO, MOSI and SCK lines of all the other slaves are tri stated (configured as input pins of a high impedance with no pull up resistors enabled). A false implementation (e.g. if two slaves are activated at the same time) can cause a driver contention which can lead to a CMOS latch up state and must be avoided. Resistances of 1 to 10 k ohms in series with the pins of the SPI can be used to prevent the system from latching up. However this affects the maximum usable data rate, depending on the loading capacitance on the SPI pins. Unidirectional SPI devices require just the clock line and one of the data lines. If the device is using the MISO line or the MOSI line depends on its purpose. Simple sensors for instance are just sending data (see S2 in Figure 2), while an external DAC usually just receives data (see S3 in Figure 2).

SPI Timing

The SPI has four modes of operation, 0 through 3. These modes essentially control the way data is clocked in or out of an SPI device. The configuration is done by two bits in the SPI control register (SPCR). The clock polarity is specified by the CPOL control bit, which selects an active high or active low clock. The clock phase (CPHA) control bit selects one of the two fundamentally different transfer formats. To ensure a proper communication between master and slave both devices have to run in the same mode. This can require a reconfiguration of the master to match the requirements of different peripheral slaves.

The settings of CPOL and CPHA specify the different SPI modes, shown in Table 3. Because this is no standard and specified different in other literature, the configuration of the SPI has to be done carefully.

SPI Mode	CPOL	СРНА	Shift SCK edge	Capture SCK edge
0	0	0	Falling	Rising
1	0	1	Rising	Falling
2	1	0	Rising	Falling
3	1	1	Falling	Rising

The clock polarity has no significant effect on the transfer format. Switching this bit causes the clock signal to be inverted (active high becomes active low and idle low becomes idle high). The settings of the clock phase, how-ever, selects one of the two different transfer timings, which are described closer in the next two chapters. Since the MOSI and MISO lines of the master and the slave are directly connected to each other, the diagrams show the timing of both devices, master and slave. The /SS line is

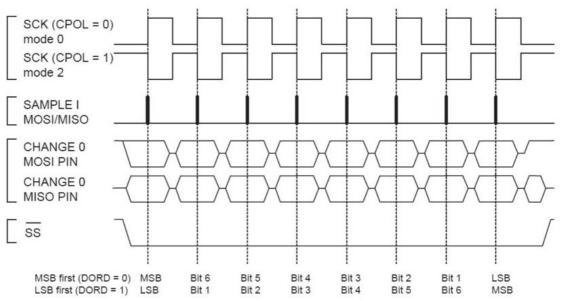
the slave select input of the slave. The /SS pin of the master is not shown in the diagrams. It has to be inactive by a high level on this pin (if configured as input pin) or by configuring it as an output pin.

A.) CPHA = 0 and CPOL = 0 (Mode 0) and CPHA = 0 and CPOL = 1 (Mode 1)

The timing of a SPI transfer where CPHA is zero is shown in Figure 3. Two wave forms are shown for the SCK signal -one for CPOL equals zero and another for CPOL equals one.

188





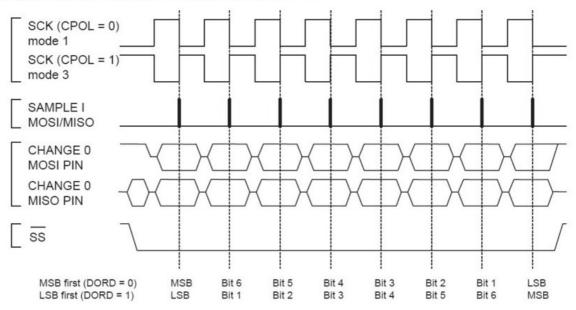
When the SPI is configured as a slave, the transmission starts with the falling edge of the /SS line. This activates the SPI of the slave and the MSB of the byte stored in its data register (SPDR) is output on the MISO line. The actual transfer is started by a software write to the SPDR of the master. This causes the clock signal to be generated. In cases where the CPHA equals zero, the SCK signal remains zero for the first half of the first SCK cycle. This ensures that the data is stable on the input lines of both the master and the slave. The data on the input lines is read with the edge of the SCK line from its inactive to its active state (rising edge if CPOL equals zero and falling edge if CPOL equals one). The edge of the SCK line from its active to its inactive state (falling edge if CPOL equals zero and rising edge if CPOL equals one) causes the data to be shifted one bit further so that the next bit is output on the MOSI and MISO lines.

After eight clock pulses the transmission is completed. In both the master and the slave device the SPI interrupt flag (SPIF) is set and the received byte is transferred to the receive buffer.

B.) CPHA = 1 and CPOL = 0 (Mode 2) and CPHA = 1 and CPOL = 1 (Mode 3)

The timing of a SPI transfer where CPHA is one is shown in Figure 4. Two wave forms are shown for the SCK signal -one for CPOL equals zero and another for CPOL equals one.





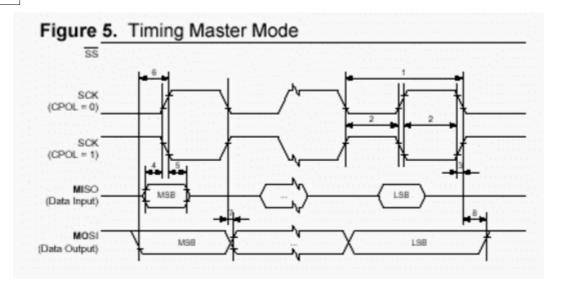
Like in the previous cases the falling edge of the /SS lines selects and activates the slave. Compared to the previous cases, where CPHA equals zero, the transmission is not started and the MSB is not output by the slave at this stage. The actual transfer is started by a software write to the SPDR of the master what causes the clock signal to be generated. The first edge of the SCK signal from its inactive to its active state (rising edge if CPOL equals zero and falling edge if CPOL equals one) causes both the master and the slave to output the MSB of the byte in the SPDR.

As shown in Figure 4, there is no delay of half a SCK-cycle like in Mode 0 and 1. The SCK line changes its level immediately at the beginning of the first SCK-cycle. The data on the input lines is read with the edge of the SCK line from its active to its inactive state (falling edge if CPOL equals zero and rising edge if CPOL equals one).

After eight clock pulses the transmission is completed. In both the master and the slave device the SPI interrupt flag (SPIF) is set and the received byte is transferred to the receive buffer.

Considerations for high speed transmissions

Parts which run at higher system clock frequencies and SPI modules capable of running at speed grades up to half the system clock require a more specific timing to match the needs of both the sender and receiver. The following two diagrams show the timing of the AVR in master and in slave mode for the SPI Modes 0 and 1. The exact values of the displayed times vary between the different pars and are not an issue in this application note. However the functionality of all parts is in principle the same so that the following considerations apply to all parts.



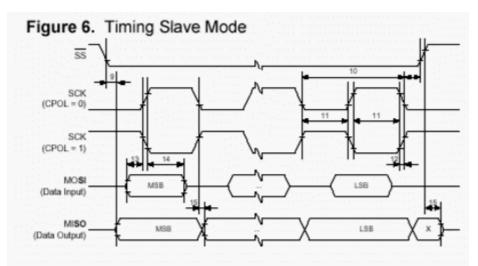
The minimum timing of the clock signal is given by the times "1" and "2". The value "1" specifies the SCK period while the value "2" specifies the high / low times of the clock signal. The maximum rise and fall time of the SCK signal is specified by the time "3". These are the first timings of the AVR to check if they match the requirements of the slave.

The Setup time "4" and Hold time "5" are important times because they specify the requirements the AVR has on the interface of the slave. These times determine how long before the clock edge the slave has to have valid output data ready and how long after the clock edge this data has to be valid.

If the Setup and Hold time are long enough the slave suits to the requirements of the AVR but does the AVR suit to the requirements of the slave?

The time "6" (Out to SCK) specifies the minimum time the AVR has valid output data ready before the clock edge occurs. This time can be compared to the Setup time "4" of the slave.

The time "7" (SCK to Out) specifies the maximum time after which the AVR outputs the next data bit while the time "8" (SCK to Out high) the minimum time specifies during which the last data bit is valid on the MOSI line after the SCK was set back to its idle state.



In principle the timings are the same in slave mode like previously described in master mode. Because of the switching of the roles between master and slave the requirements on the timing are inverted as well. The minimum times of the master mode are now maximum times and vice versa.

SPI Transmission Conflicts

A write collision occurs if the SPDR is written while a transfer is in progress. Since this register is just single buffered in the transmit direction, writing to SPDR causes data to be written directly into the SPI shift register. Because this write operation would corrupt the data of the current transfer, a write-collision error in generated by setting the WCOL bit in the SPSR. The write operation will not be executed in this case and the transfer continues undisturbed. A write collision is generally a slave error because a slave has no control over when a master will initiate a transfer. A master, however, knows when a transfer is in progress. Thus a master should not generate write collision errors, although the SPI logic can detect these errors in a master as well as in a slave mode.

When you set the SPI option from the Options, Compiler, SPI menu SPCR will be set to 01010100 which means ; enable SPI, master mode, CPOL = 1

When you want to control the various options with the hardware SPI you can use the $\underline{\text{CONFIG SPI}}$ statement.

4.17 Power Up

At power up all ports are in Tri-state and can serve as input pins.

When you want to use the ports (pins) as output, you must set the data direction first with the statement : CONFIG PORTB = OUTPUT

Individual bits can also be set to be used as input or output.

For example : DDRB = &B00001111, will set a value of 15 to the data direction register of PORTB.

PORTB.0 to PORTB.3 (the lower 4 bits) can be used as outputs because they are set high. The upper four bits (PORTB.4 to PORTB.7), can be used for input because they are set low.

You can also set the direction of a port pin with the statement :

CONFIG PINB.0 = OUTPUT | INPUT

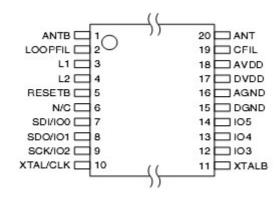
The internal RAM is cleared at power up or when a reset occurs. Use \$NORAMCLEAR to disable this feature.

You may use <u>\$INITMICRO</u> (331) to set a port level and direction immediately on startup.

4.18 Chips

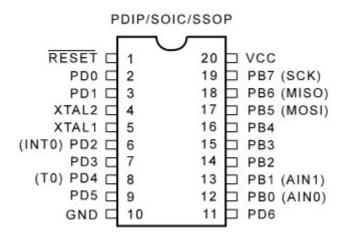
4.18.1 AT86RF401

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

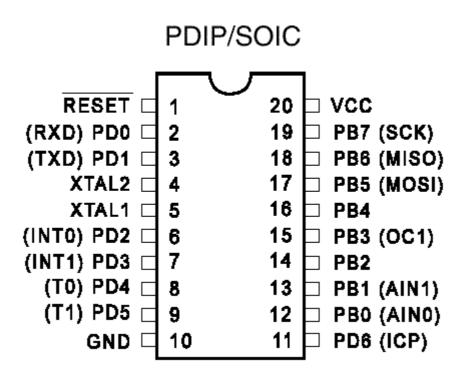


4.18.2 AT90S1200

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

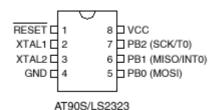


4.18.3 AT90S2313



The ATTiny2313 should be used for new designs.

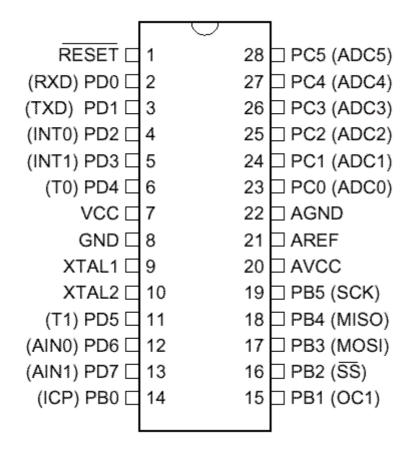
4.18.4 AT90S2323



4.18.5 AT90S2333

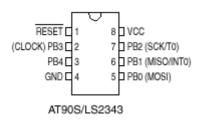
194

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.6 AT90S2343

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



[tip from Martin Verschuren]

When using the AT90S2343 with BASCOM-AVR 1.11.6.4 and the STK200. Programming must be done with jumper ext-clk.

The BASCOM build in programmer will detect a Tiny22, which seems to have the same ID string as the 2343 (Atmel source) so no wonder.

By using the internal clock RCEN=0, then the jumper of the STK200 must be on int.

clk after programming.

Don't leave this away, some AT90S2343 will not correctly startup.

In your own project notice that you have to pull up the clk pin(2) at power up else it won't work. (I just looked for it for a day to get this problem solved:-)

Note : the at90s2343 and tiny22 have the same chip ID. In BASCOM you need to choose the tiny22 even if you use the 2343.

I note from MCS : only the AT23LS43-1 has the internal oscillator programmed by default! All other 2343 chips need an external clock signal. Tip: use a AT90S2313 and connect X2 to the clock input of the 2343.

[tip from David Chambers]

Using the AT90S2343 with BASCOM 1.11.7.3 the DT006 hardware there are no problems with programming the chip ie no special jumper conditions to enable programming. However it is best to remove links connecting ports to the DT006 LED's before programming. If access to PB3 and PB4 is desired then jumpers J11 & J12 must be installed with pins 2 and 3 linked in both cases. Note that PB3 and PB4 are each connected to a momentary pushbutton on the DT006 board. These can be used to check contact closure functions, so bear this in mind when writing code for contact monitoring.

The current ATMEL data sheet specifies that all versions -1, -4 and -10 are supplied with a fuse bit set for the internal clock that operates at approximately 1Mhz. If using the internal clock make sure to enter 1000000 under Options\Compiler\Communication\frequency.

A great little chip with minimal external components. Only the resistor and capacitor required for RESET during power up.

Note that the LED's on the DT006 are not connected to the same programmed port pins when changing the chip type. This is because the special functions assigned ports varies between the 8pin, 20 pin and 28 pin products eg the MOSI, MISI and SCK functions are assigned to PB0, PB1 and PB2 for an 8 pin processor and PB5, PB6 and PB7 for a 20 pin processor. The result is that for a given program the LED's that respond are different.

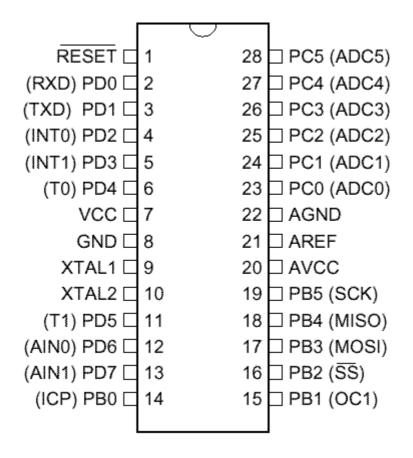
4.18.7 AT90S4414

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

PDIP

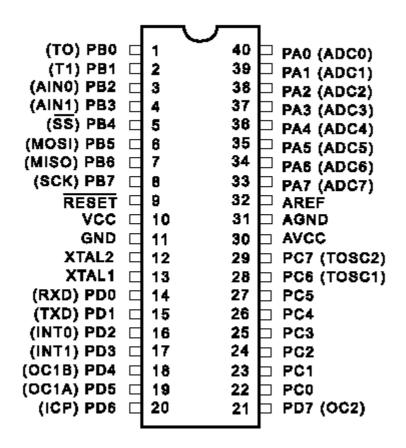
	ſ	\neg			
(TO) PB0		1	40	Þ	VCC
(T1) PB1		2	39	Þ.	PA0 (AD0)
(AIN0) PB2		3	38	ь	PA1 (AD1)
(AIN1) PB3		4	37	Б	PA2 (AD2)
(SS) PB4		5	36	Б	PA3 (AD3)
(MOSI) PB5	Н	6	35	Б	PA4 (AD4)
(MISO) PB6	F	7	34	F.	PA5 (AD5)
(SCK) PB7	Н	8	33	F.	PA6 (AD6)
RESET	Н	9	32	F.	PA7 (AD7)
(RXD) PD0	H	10	31	F.	ICP
1	H	0.002		Ľ.	
(TXD) PD1	Ч	11	30	Ч	ALE
(INT0) PD2		12	29	P	OC1B
(INT1) PD3		13	28	Þ	PC7 (A15)
PD4		14	27	Ь	PC6 (A14)
(OC1A) PD5		15	26	Ь	PC5 (A13)
(WR) PD6		16	25	Ь	PC4 (A12)
(RD) PD7		17	24	Б	PC3 (A11)
XTAL2	F	18	23	F.	PC2 (A10)
XTAL1	F	19	22	F	PC1 (A9)
GND	H	20	21	F.	PC0 (A8)
OND	Ч		21	Г	100 (A0)

4.18.8 AT90S4433



4.18.9 AT90S4434

198



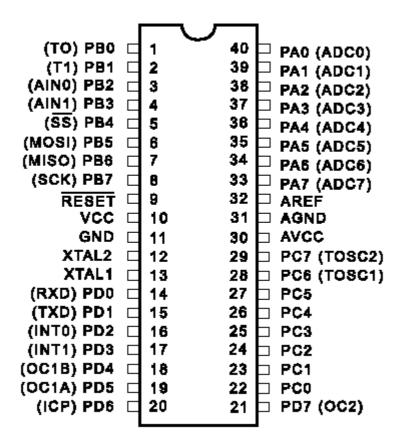
4.18.10 AT90S8515

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

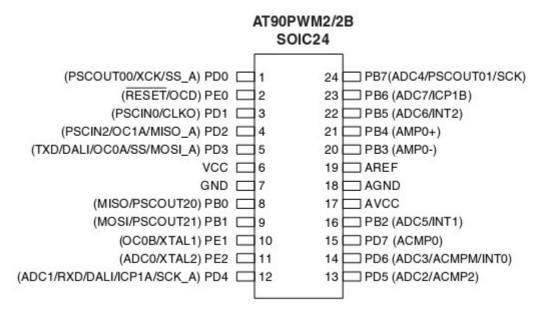
PDIP

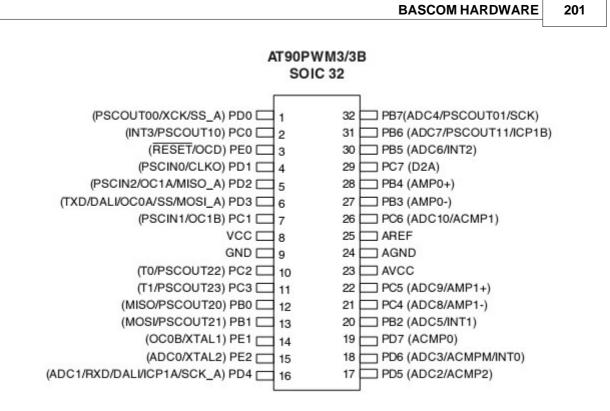
	03	22	a	
			\neg	
(TO) PB0		1	40 口	VCC
(T1) PB1		2	39 🗖	PA0 (AD0)
(AIN0) PB2		3	38 🗖	PA1 (AD1)
(AIN1) PB3		4	37 占	PA2 (AD2)
(SS) PB4		5	36 🗖	PA3 (AD3)
(MOSI) PB5		6	35 🗖	PA4 (AD4)
(MISO) PB6		7	34 🗖	PA5 (AD5)
(SCK) PB7		8	33 🗄	PA6 (AD6)
RESET		9	32 🗖	PA7 (AD7)
(RXD) PD0		10	31 🗖	ICP
(TXD) PD1		11	30 🗖	ALE
(INTO) PD2		12	29 🗖	OC1B
(INT1) PD3		13	28 🗖	PC7 (A15)
PD4		14	27 🗖	PC6 (A14)
(OC1A) PD5		15	26 🗖	PC5 (A13)
(WR) PD6		16	25 占	PC4 (A12)
(RD) PD7		17	24 6	PC3 (A11)
XTAL2	F	18	23 6	PC2 (A10)
XTAL1	H	19	22 E	PC1 (A9)
GND	H			
GND	Ч	20	21	PC0 (A8)

4.18.11 AT90S8535



4.18.12 AT90PWM2-3

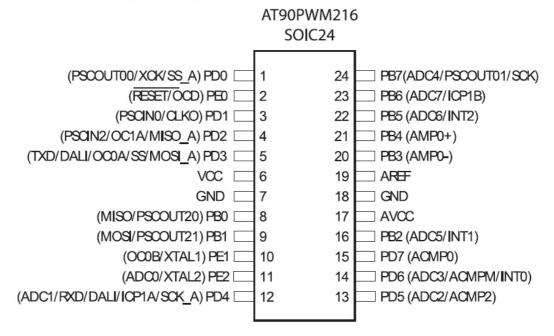




4.18.13 AT90PWM216

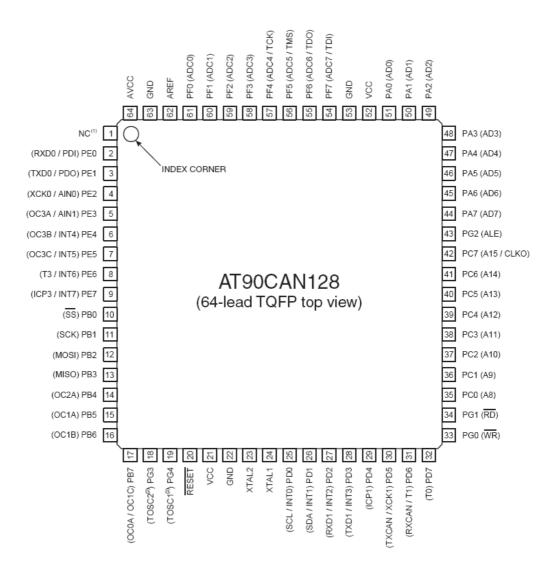
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Figure 3-1. SOIC 24-pin Package



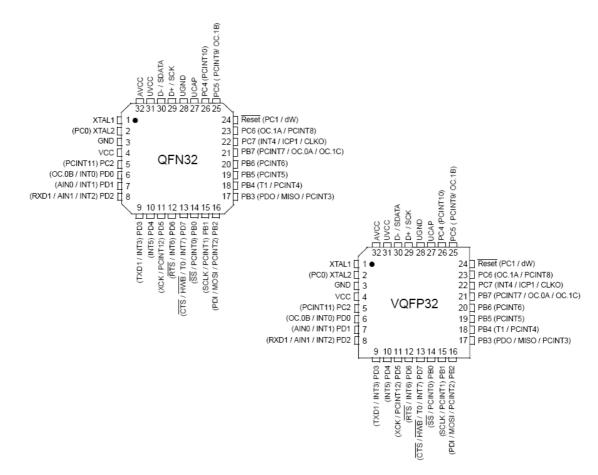
4.18.14 AT90CAN128

202

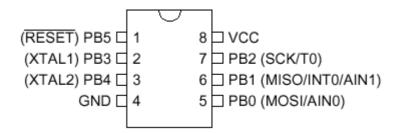


4.18.15 AT90USB162

See also the USB162 [270] module for easy soldering of proto types.



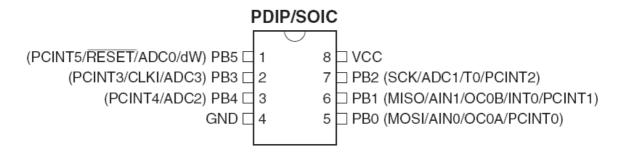
4.18.16 ATtiny12



4.18.17 ATtiny13

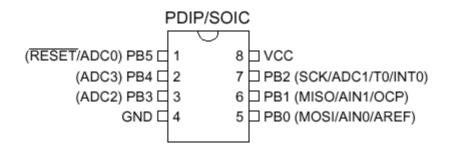
204

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.18 ATtiny15

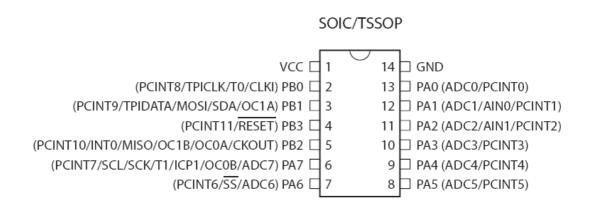
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.19 ATtiny20

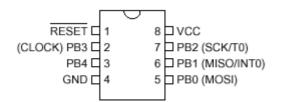
The ATTINY20 is a 14 pins AVR chip. It has NO EEPROM. It also does not have a UART.

The TWI slave interface is not compatible with TWI found in other AVR chips. The chip has a PDI programming interface and does not support ISP or JTAG. The watchdog is also different compared to other AVR chips. It is using a CCP register which is similar as the Xmega.



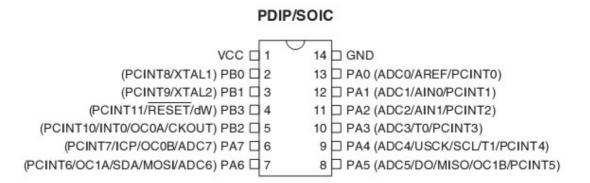
4.18.20 ATtiny22

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.21 ATtiny24

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

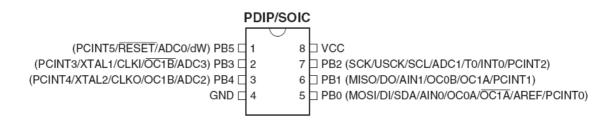


The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

HWMUL=0 ; this chip does not have hardware multiplication

Some users reported that the HWMUL did work. Some batches might support the HW MUL, but since we found chips that did not, the value is set to 0. You can change it at your own risk.

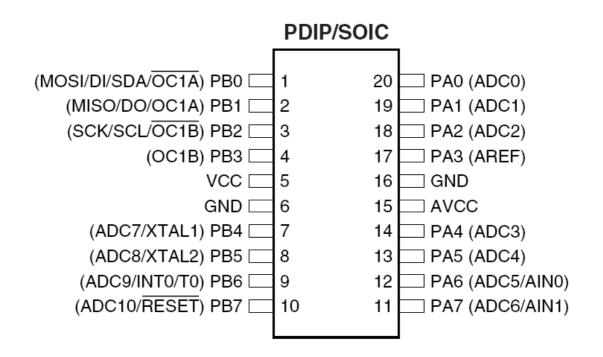
4.18.22 ATtiny25



4.18.23 ATtiny26

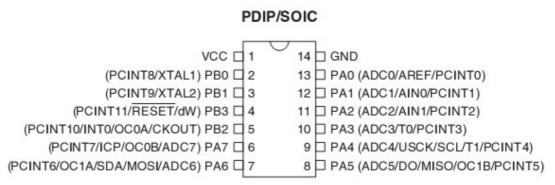
206

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.24 ATtiny44

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



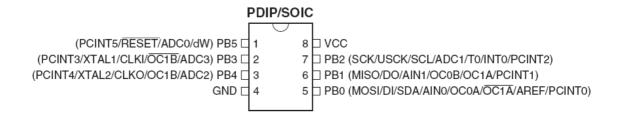
The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

HWMUL=0 ; this chip does not have hardware multiplication

Some users reported that the HWMUL did work. Some batches might support the HW MUL, but since we found chips that did not, the value is set to 0. You can change it at your own risk.

4.18.25 ATtiny45

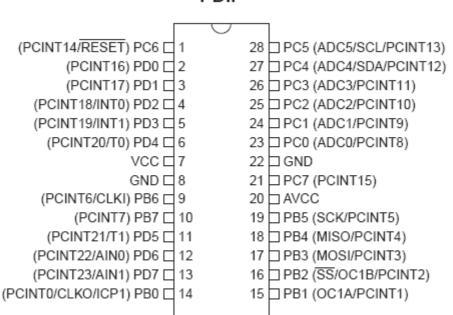
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.26 ATtiny48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Notice that the TINY48 is NOT the same as the MEGA48. The TINY48 does not have a UART.

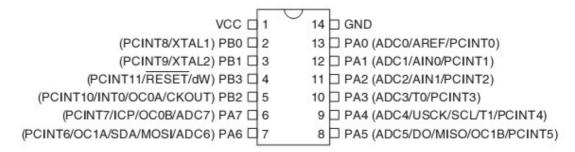


PDIP

4.18.27 ATtiny84

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

PDIP/SOIC



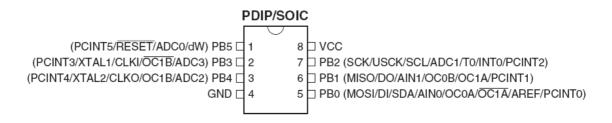
The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

HWMUL=0 ; this chip does not have hardware multiplication

Some users reported that the HWMUL did work. Some batches might support the HW MUL, but since we found chips that did not, the value is set to 0. You can change it at your own risk.

4.18.28 ATtiny85

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



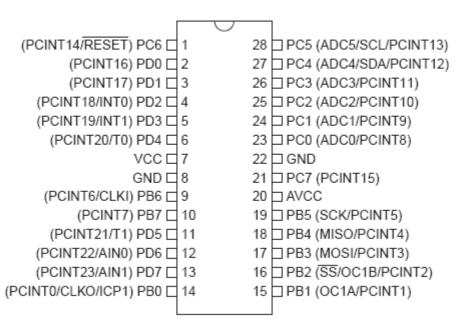
4.18.29 ATtiny88

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

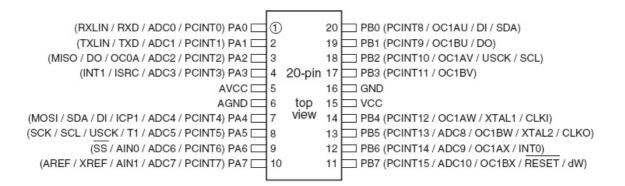
Notice that the TINY88 is NOT the same as the MEGA88. The TINY88 does not have a UART.





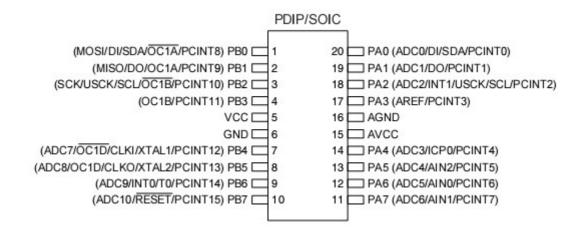


4.18.30 ATtiny167

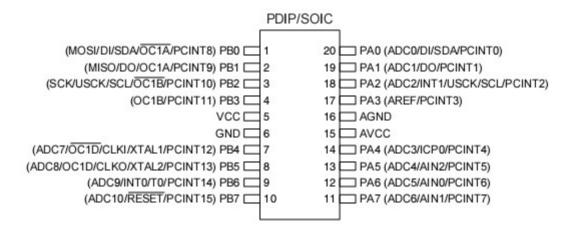


4.18.31 ATtiny261

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

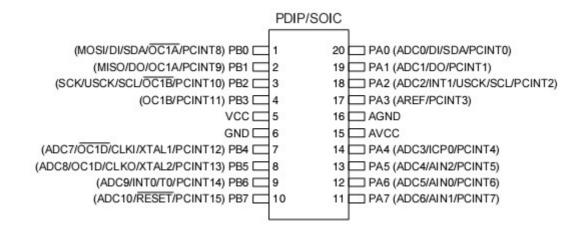


4.18.32 ATtiny461



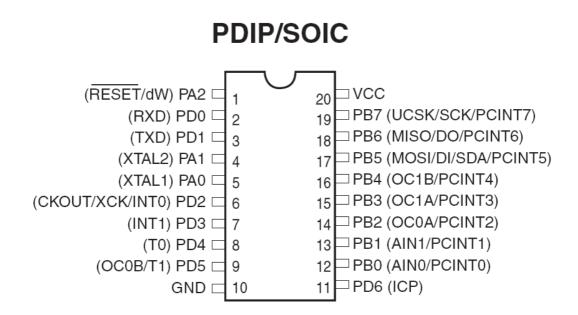
4.18.33 ATtiny861

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.34 ATtiny2313

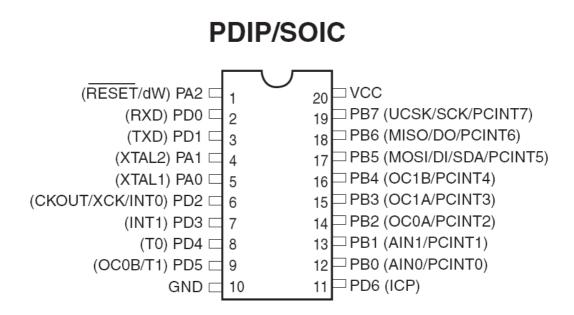
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The tiny2313 has an internal oscillator that can run at various frequencies. The 4 MHz seems not to work precise. when using the UART for serial communication you can get wrong output. You can best use the 8 MHz internal oscillator , or tweak the UBRR register. For example, UBRR=UBRR+1 That worked for 4 Mhz, at 19200 baud.

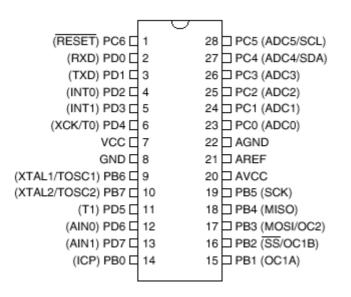
4.18.35 ATtiny4313

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

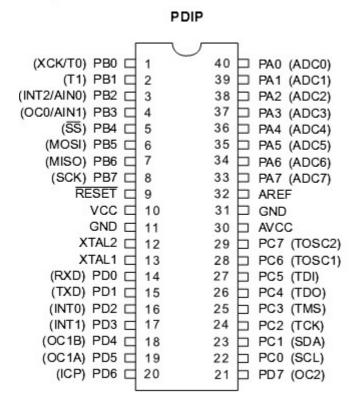


The tiny4313 has an internal oscillator that can run at various frequencies. The 4 MHz seems not to work precise. when using the UART for serial communication you can get wrong output. You can best use the 8 MHz internal oscillator , or tweak the UBRR register. For example, UBRR=UBRR+1 That worked for 4 Mhz, at 19200 baud.

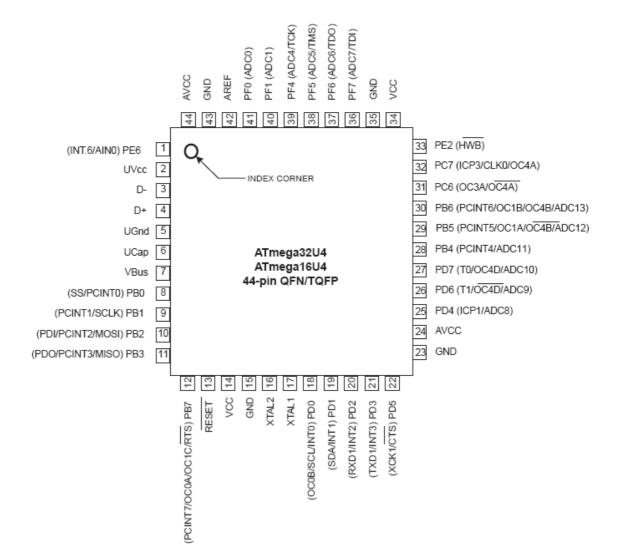
4.18.36 ATMEGA8



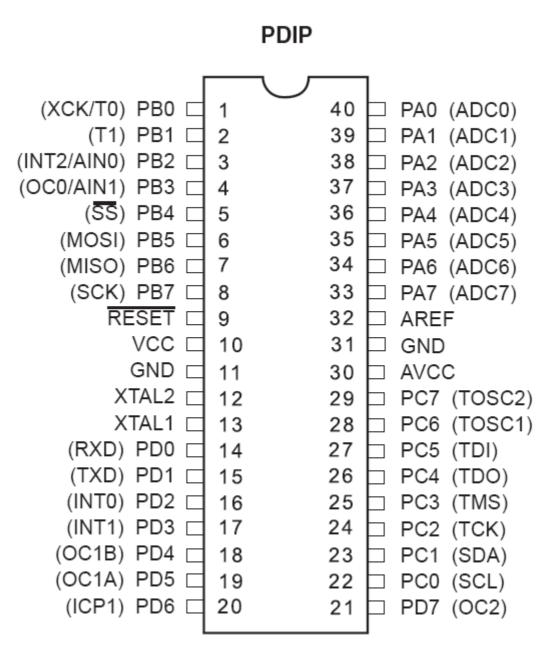
4.18.37 ATMEGA16



4.18.38 ATMEGA16U4



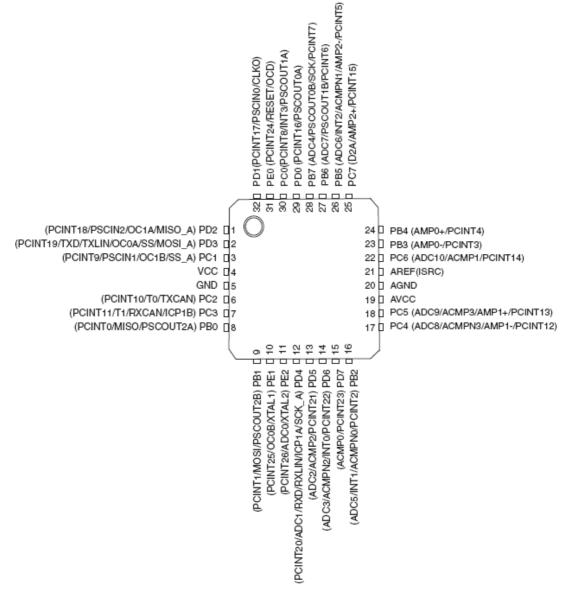
4.18.39 ATMEGA32



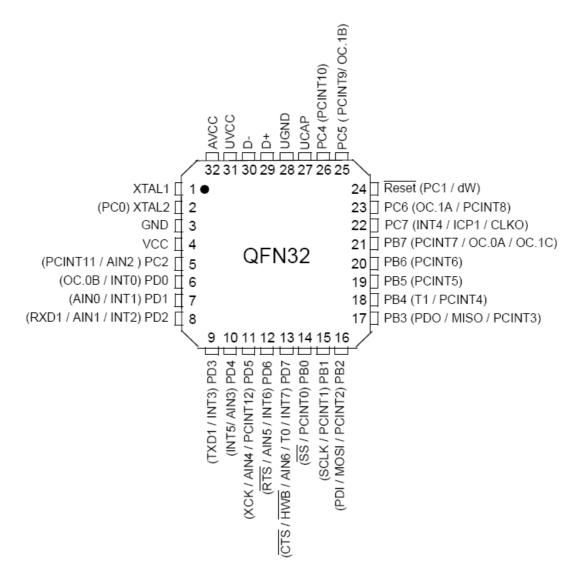
216

4.18.40 ATMEGA16M1_32M1_64M1

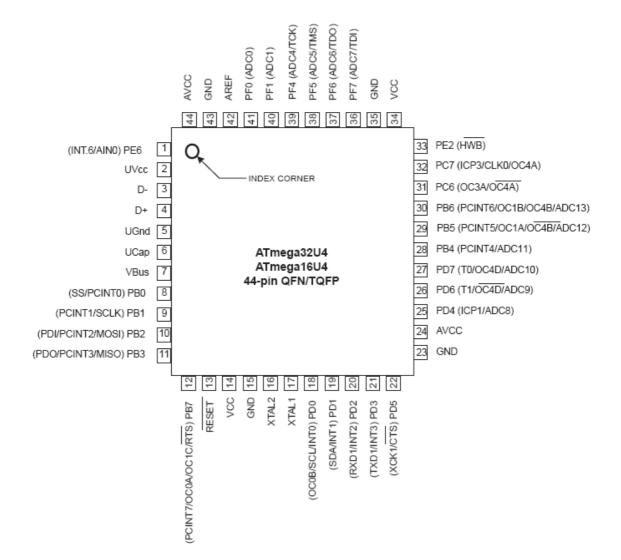
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.41 ATMEGA8U2_162_32U2

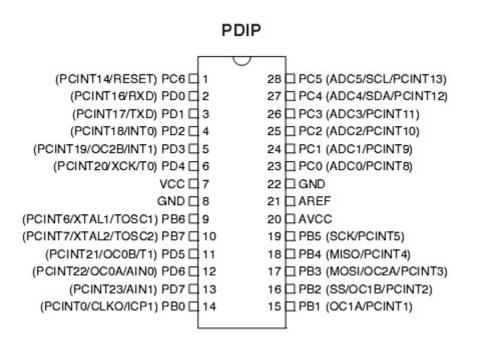


4.18.42 ATMEGA16U4_32U4

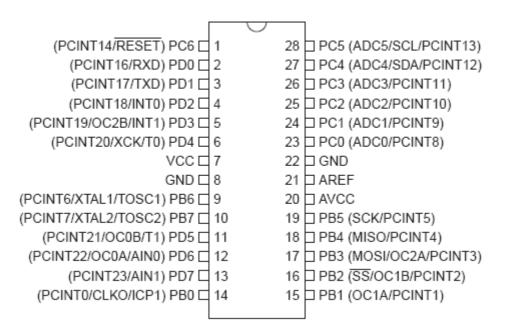


4.18.43 ATMEGA48

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.44 ATMEGA48P

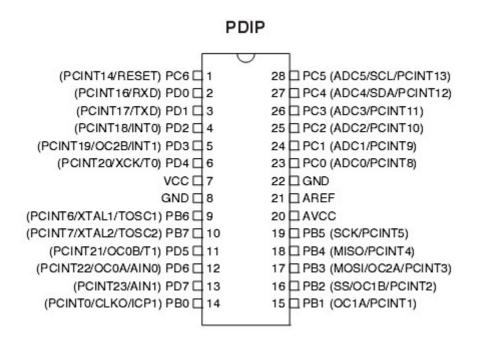


BASCOM-AVR

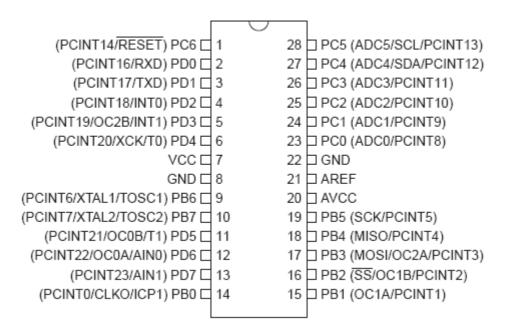
4.18.45 ATMEGA88

220

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



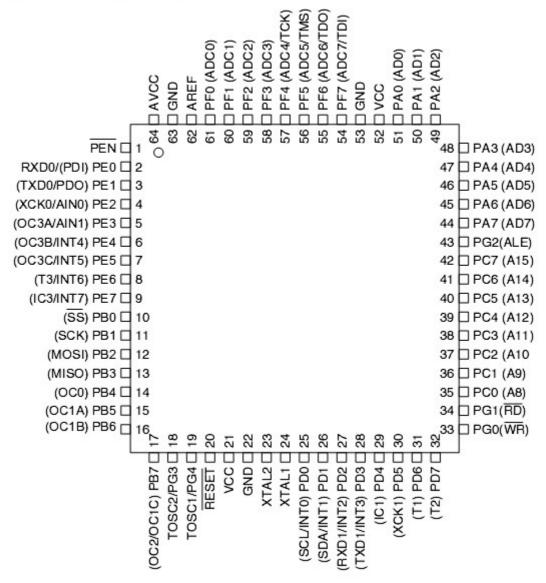
4.18.46 ATMEGA88P



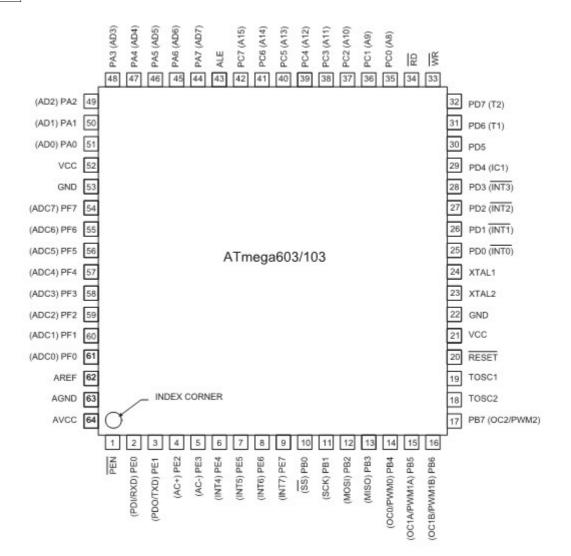
4.18.47 ATMEGA64

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Figure 1. Pinout ATmega64

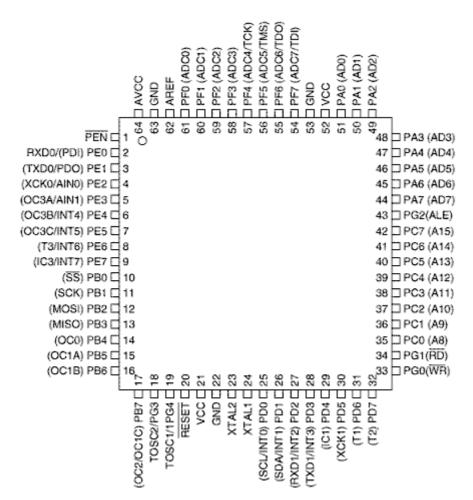


4.18.48 ATMEGA103



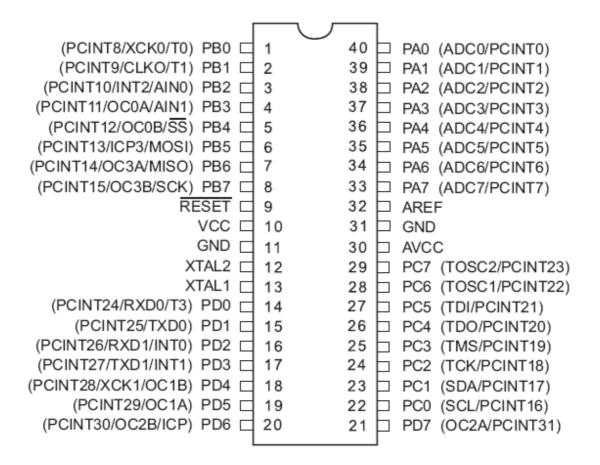
4.18.49 ATMEGA128

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



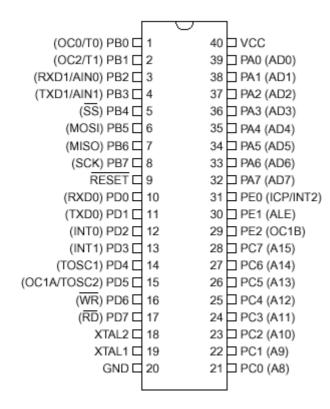
When using XRAM and IDLE, the micro need the CONFIG XRAM after returing from the power down mode.

4.18.50 ATMEGA1284P



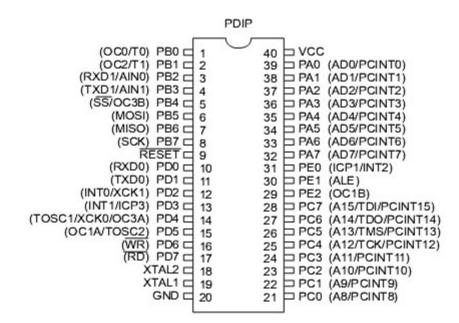
4.18.51 ATMEGA161

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.52 ATMEGA162

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The M162 has a clock-16 divider enabled by default. See the M162.bas sample file

BASCOM-AVR

4.18.53 ATMEGA163

226

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

(TO) PB0 (T1) PB1 (AIN0) PB2 (AIN1) PB3 (SS) PB4 (MOSI) PB5 (MISO) PB6 (SCK) PB7 RESET VCC GND XTAL2 XTAL1 (RXD) PD0 (TXD) PD1	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 5 6	PDIP	40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25		PA0 (ADC0) PA1 (ADC1) PA2 (ADC2) PA3 (ADC3) PA4 (ADC4) PA5 (ADC5) PA6 (ADC6) PA7 (ADC7) AREF AGND AVCC PC7 (TOSC2) PC6 (TOSC1) PC5 PC4 PC3
				B	
· · · · · · · · · · · · · · · · · · ·				E	PC5
(INT0) PD2	16		25	Ë	PC3
(INT1) PD3 (OC1B) PD4	17 18		24 23	E	PC2 PC1 (SDA)
(OC1A) PD5 (ICP) PD6	19 20		22 21	F	PC0 (SCL) PD7 (OC2)

The M163 by default uses the internal clock running at 1 MHz

When you have problems with timing set the right fuse bit A987= 0101. This will solve this problem.

I have just found a small difference in PortB when using the Mega163 in place of a 8535. The difference is in regard to PortB.4 - PortB.7 when not used as a SPI

interface. The four upper bits of PortB are shared with the hardware SPI unit.

If the SPI is configured in SLAVE mode (DEFAULT) the MOSI , SCK , /SS

Are configured as inputs, Regardless of the DDRB setting !

The /SS (slave select) pin also has restrictions on it when using it as a general input.see data sheet ATmega163 - p57.

This sample allows you to use the upper nibble of PortB as outputs.

 $Portb = \&B0000_{000}$

DDRB = &B1111_0000 'set upper bits for output.

Spcr = &B0001_0000 ' set SPI to Master and Disable.

If The SPCR register is not set for Master, you cannot set the pins for

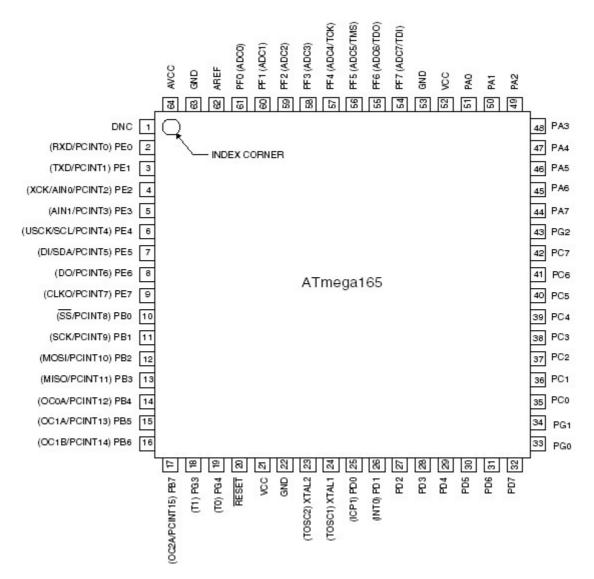
Output.

4.18.54 ATMEGA164P

]		¬¬	-	1			
(PCINT8/XCK0/T0) PB0	1	\smile	40	Þ	PA0 (ADC0/PCINT0)		
(PCINT9/CLKO/T1) PB1	2		39	Þ	PA1 (ADC1/PCINT1)		
(PCINT10/INT2/AIN0) PB2	3		38	Þ	PA2 (ADC2/PCINT2)		
(PCINT11/OC0A/AIN1) PB3	4		37	Þ	PA3 (ADC3/PCINT3)		
(PCINT12/OC0B/SS) PB4	5		36	Þ	PA4 (ADC4/PCINT4)		
(PCINT13/MOSI) PB5	6		35	Þ	PA5 (ADC5/PCINT5)		
(PCINT14/MISO) PB6	7		34	Þ	PA6 (ADC6/PCINT6)		
(PCINT15/SCK) PB7	8		33	Þ	PA7 (ADC7/PCINT7)		
RESET C	9		32	Þ	AREF		
VCC 🗖	10		31	Þ	GND		
GND 🗖			30	Þ	AVCC		
XTAL2	12		29	Þ	PC7 (TOSC2/PCINT23)		
XTAL1	13		28	Þ	PC6 (TOSC1/PCINT22)		
(PCINT24/RXD0) PD0	14		27	Þ	PC5 (TDI/PCINT21)		
(PCINT25/TXD0) PD1	15		26	Þ	PC4 (TDO/PCINT20)		
(PCINT26/RXD1/INT0) PD2	16		25	Þ	PC3 (TMS/PCINT19)		
(PCINT27/TXD1/INT1) PD3	17		24	Þ	PC2 (TCK/PCINT18)		
(PCINT28/XCK1/OC1B) PD4	18		23	Þ	PC1 (SDA/PCINT17)		
(PCINT29/OC1A) PD5	19		22	Þ	PC0 (SCL/PCINT16)		
(PCINT30/OC2B/ICP) PD6	20		21	Þ	PD7 (OC2A/PCINT31)		
l				1			

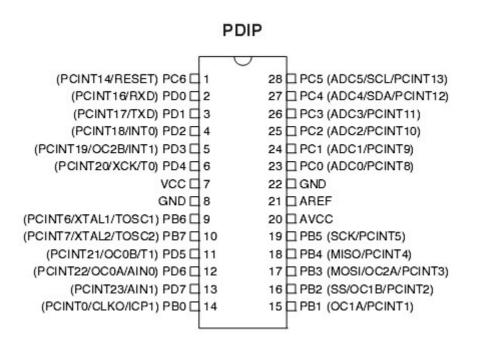
4.18.55 ATMEGA165

228

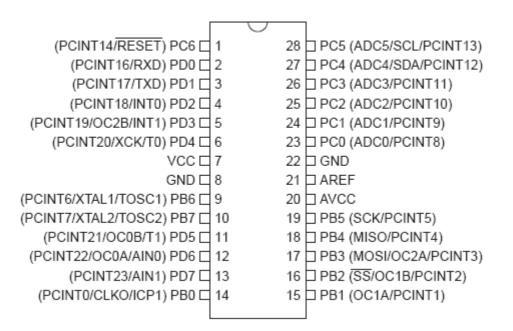


4.18.56 ATMEGA168

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



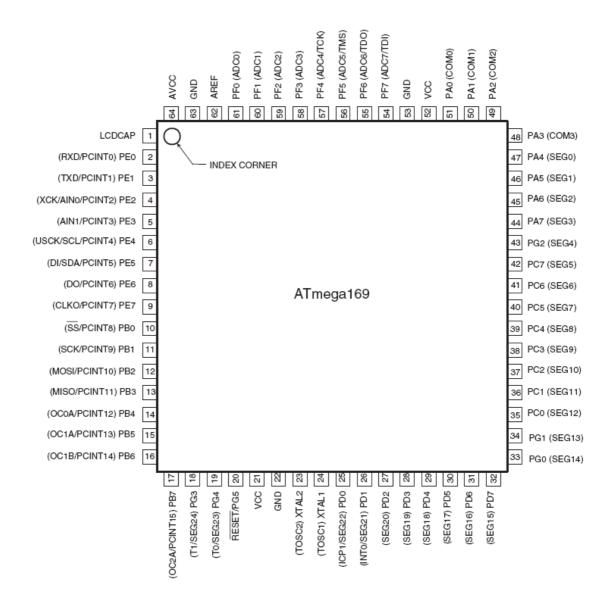
4.18.57 ATMEGA168P



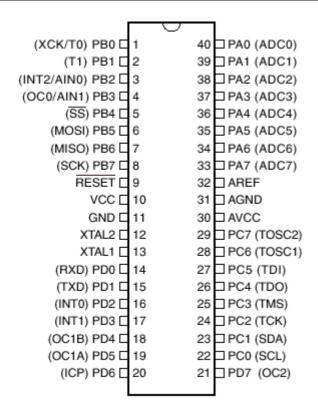
4.18.58 ATMEGA169

230

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

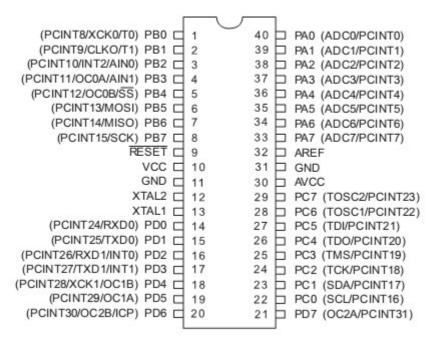


4.18.59 ATMEGA323



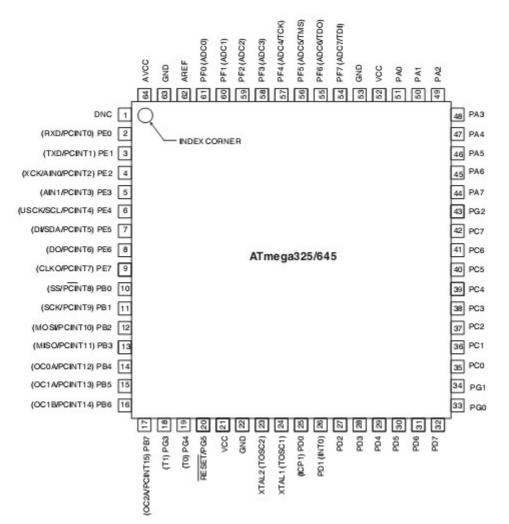
The JTAG interface is enabled by default. This means that portC.2-portC.5 pins can not be used. Program the JTAG fuse bit to disable the JTAG interface.

4.18.60 ATMEGA324P

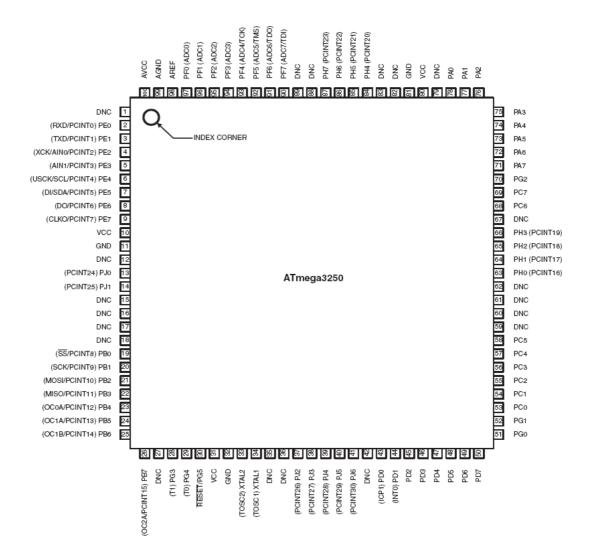


4.18.61 ATMEGA325

232



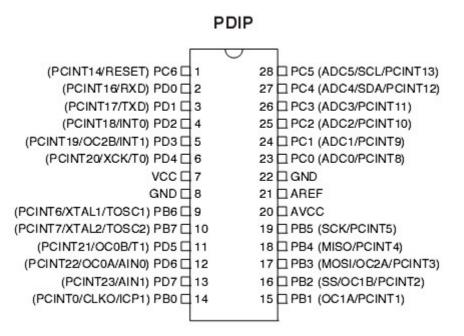
4.18.62 ATMEGA3250P



BASCOM-AVR

4.18.63 ATMEGA328

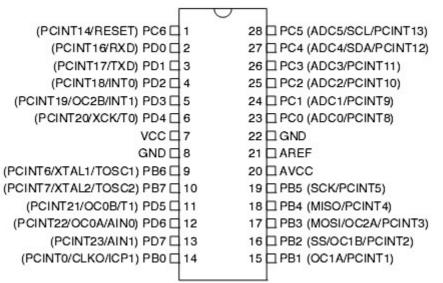
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.64 ATMEGA328P

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

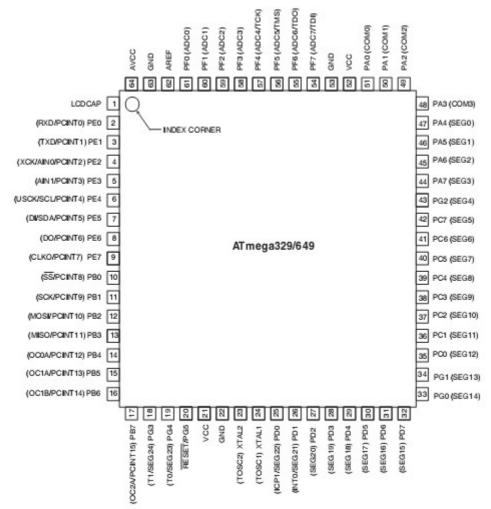




234

4.18.65 ATMEGA329

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

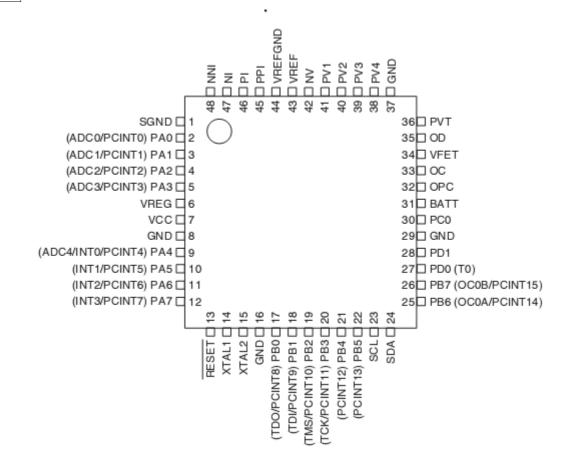


4.18.66 ATMEGA406

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

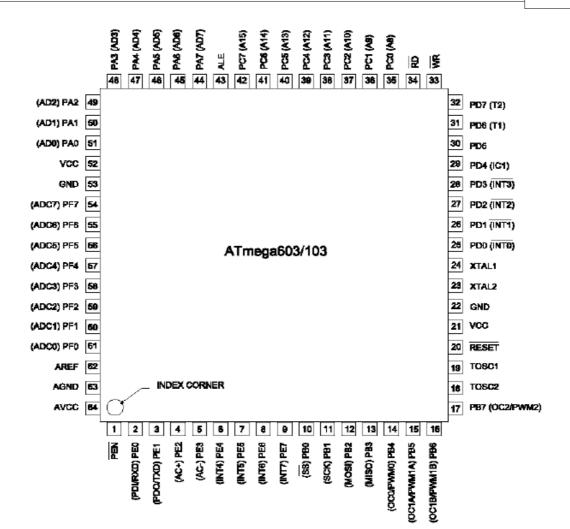
The image is from a preliminary data sheet. It is not clear yet if SCL and SDA have pin names too.

This chip can only programmed parallel and with JTAG. Normal (serial) ISP programming is not available.



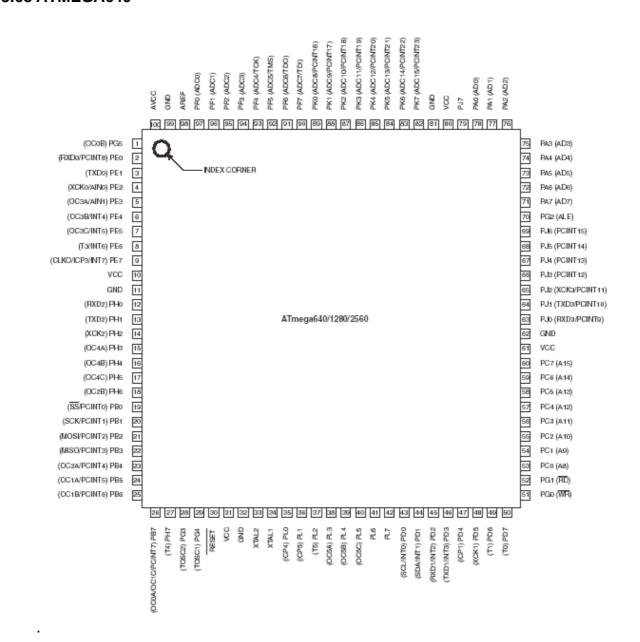
4.18.67 ATMEGA603

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet. When you have a better image available, please send it to support@mcselec.com





238



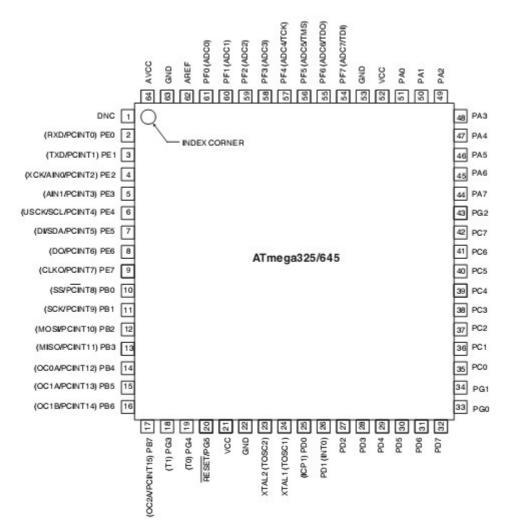
4.18.69 ATMEGA644P

Notice that there are Mega644 and Mega644P chips. P stand for PICO power. You should use the P-version for new designs. These Pico version usual add some functionality such as a second UART.

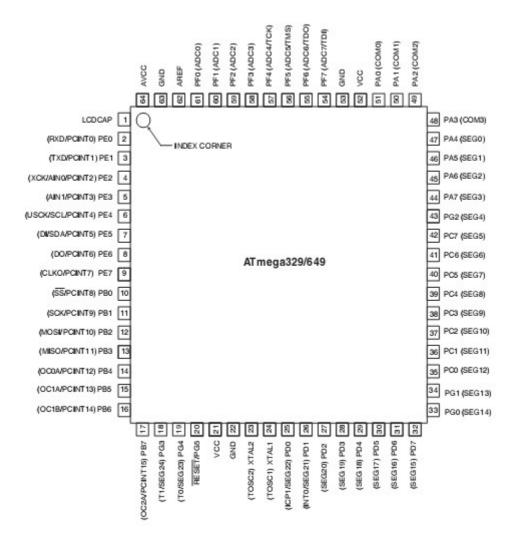
ſ			
(PCINT8/XCK0/T0) PB0	1	40 占	PA0 (ADC0/PCINT0)
(PCINT9/CLKO/T1) PB1	2	39 🗖	PA1 (ADC1/PCINT1)
(PCINT10/INT2/AIN0) PB2	3	38 🗖	PA2 (ADC2/PCINT2)
(PCINT11/OC0A/AIN1) PB3	4	37 🛓	PA3 (ADC3/PCINT3)
(PCINT12/OC0B/SS) PB4	5	36 🗖	PA4 (ADC4/PCINT4)
(PCINT13/MOSI) PB5	6	35 🗖	PA5 (ADC5/PCINT5)
(PCINT14/MISO) PB6	7	34 白	PA6 (ADC6/PCINT6)
(PCINT15/SCK) PB7	8	33 🗖	PA7 (ADC7/PCINT7)
RESET C	9	32 🗅	AREF
VCC 🗖	10	31 뉟	GND
GND 🗖	11	30 🗖	AVCC
XTAL2	12	29 🗖	PC7 (TOSC2/PCINT23)
XTAL1	13	28 🗖	PC6 (TOSC1/PCINT22)
(PCINT24/RXD0) PD0	14	27 白	PC5 (TDI/PCINT21)
(PCINT25/TXD0) PD1	15	26 🗖	PC4 (TDO/PCINT20)
(PCINT26/RXD1/INT0) PD2	16	25 🗅	PC3 (TMS/PCINT19)
(PCINT27/TXD1/INT1) PD3	17	24 🗅	PC2 (TCK/PCINT18)
(PCINT28/XCK1/OC1B) PD4	18	23 🗅	PC1 (SDA/PCINT17)
(PCINT29/OC1A) PD5	19	22 뉟	PC0 (SCL/PCINT16)
(PCINT30/OC2B/ICP) PD6	20	21 🗅	PD7 (OC2A/PCINT31)

4.18.70 ATMEGA645

240

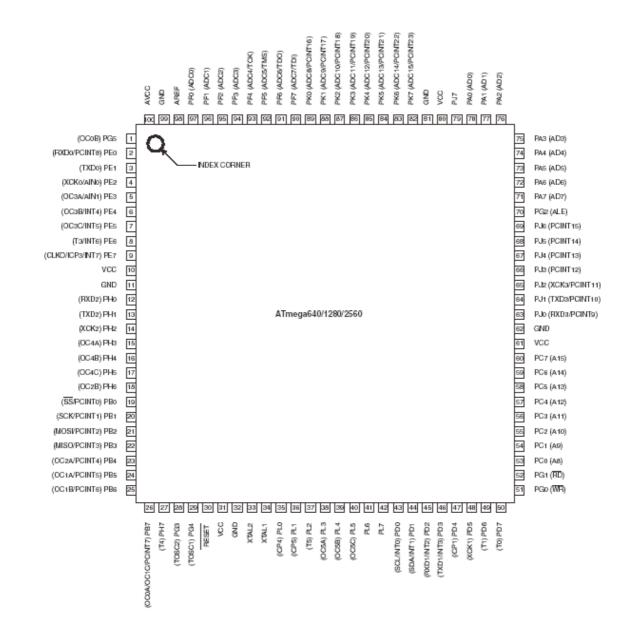


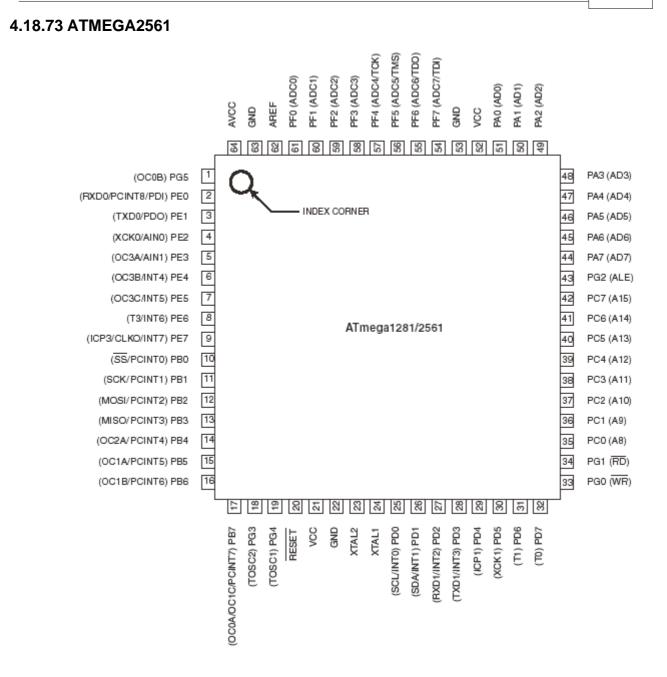
4.18.71 ATMEGA649





242





BASCOM HARDWARE 243

4.18.74 ATMEGA8515

244

	PDIP	
2		
(OC0/T0) PB0 🗆	1	40 🗆 VCC
(T1) PB1 🗆	2	39 🗖 PA0 (AD0)
(AIN0) PB2	3	38 🗆 PA1 (AD1)
(AIN1) PB3 🗆	4	37 🗖 PA2 (AD2)
(SS) PB4 🗆	5	36 🗖 PA3 (AD3)
(MOSI) PB5 🗆	6	35 🗖 PA4 (AD4)
(MISO) PB6	7	34 🗆 PA5 (AD5)
(SCK) PB7	8	33 🗖 PA6 (AD6)
RESET	9	32 🗆 PA7 (AD7)
(RXD) PD0 🗆	10	31 D PE0 (ICP/INT2)
(TDX) PD1 🗆	11	30 🗆 PE1 (ALE)
(INT0) PD2	12	29 🗆 PE2 (OC1B)
(INT1) PD3	13	28 🗆 PC7 (A15)
(XCK) PD4	14	27 🗖 PC6 (A14)
(OC1A) PD5	15	26 🗆 PC5 (A13)
(WR) PD6	16	25 🗆 PC4 (A12)
(RD) PD7	17	24 🗆 PC3 (A11)
XTAL2	18	23 🗖 PC2 (A10)
XTAL1	19	22 🗆 PC1 (A9)
GND 🗆	20	21 🗆 PC0 (A8)
3		

4.18.75 ATMEGA8535

8		~~		1		
(XCK/T0) PB0 🗆	1	\mathbf{U}	40	Þ	PA0	(ADC0)
(T1) PB1 🗆	2		39	Þ	PA1	(ADC1)
(INT2/AIN0) PB2	3		38	Þ	PA2	(ADC2)
(OC0/AIN1) PB3	4		37	Þ	PA3	(ADC3)
(SS) PB4 🗆	5		36	Þ	PA4	(ADC4)
(MOSI) PB5	6		35	Þ	PA5	(ADC5)
(MISO) PB6	7		34	Þ	PA6	(ADC6)
(SCK) PB7 🗆	8		33	Þ	PA7	(ADC7)
RESET	9		32	Þ	ARE	F
VCC 🗆	10		31	Þ	GND)
GND	11		30	þ.	AVC	С
XTAL2	12		29	Þ	PC7	(TOSC2)
XTAL1	13		28	Þ	PC6	(TOSC1)
(RXD) PD0	14		27	Þ	PC5	
(TXD) PD1	15		26	Þ	PC4	
(INT0) PD2	16		25	Þ	PC3	
(INT1) PD3 🗆	17		24	Þ	PC2	
(OC1B) PD4	18		23	Þ	PC1	(SDA)
(OC1A) PD5	19		22	Þ	PC0	(SCL)
(ICP1) PD6 🗆	20		21	Þ	PD7	(OC2)

4.18.76 ATXMEGA

The Xmega is a great new chip. It has a lot of hardware on board and a huge amount of hardware registers.

Some changes in the architecture are however breaking compatibility with normal AVR processors.

The power of the AVR is/was the the linear memory architecture. In the Xmega this has been changed : the registers are placed into a separate address space. This makes code like this fail:

Clr r31 Ldi r30,10 ; point to register R10 Ld r24,z+ ; load value from R10 and inc pointer

Of course LDS/STS will not work either on the registers. If your ASM code contains such code you need to rewrite it.

Another change which has less impact is that when writing a word register, you need to start with the LSB first.

In non-Xmega you always had to write the MSB first. But the compiler handles this for you automatic.

There is no ISP programming support. Only JTAG and PDI is supported. Of course the MCS Bootloader can be used but you need to program the chip first.

Another change with impact is that the port registers do not have an offset of one. Almost , but not all, AVR chips have the DDR register at address PIN+1, and the PORT register at PIN+2. But Xmega uses something different : DDR = PIN-8 and PORT = PIN-4. So if you used code to pass a pointer to the PIN register you must now pass DDR and use a different offset as well.

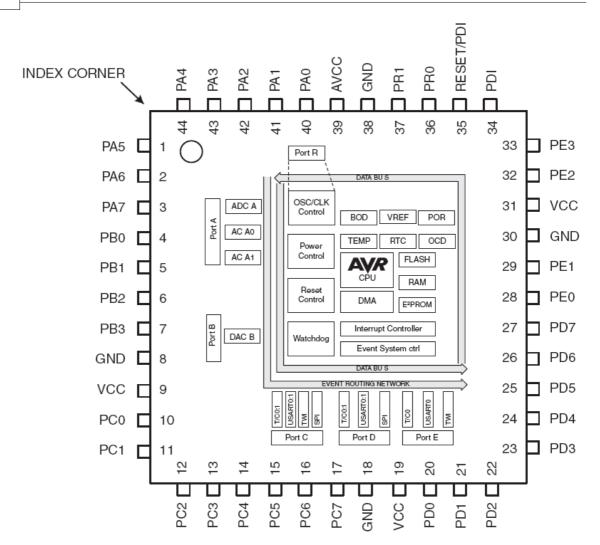
Despite these differences there is a big advantage as well : all Xmega have their registers at the same address. Some chips might not have all registers because the hardware is not inside the chip, but all DAT files are similar. And all hardware has a fixed offset. This allows to use dynamic code. For example bascom can now use a variable for the UART and the code is only needed once because all hardware has a fixed offset.

There might be other changes. Despite these changes, the Xmega is great for new designs. It has up to 8 USARTS, internal oscillators, ADC, DAC, timers, multiple SPI and TWI.

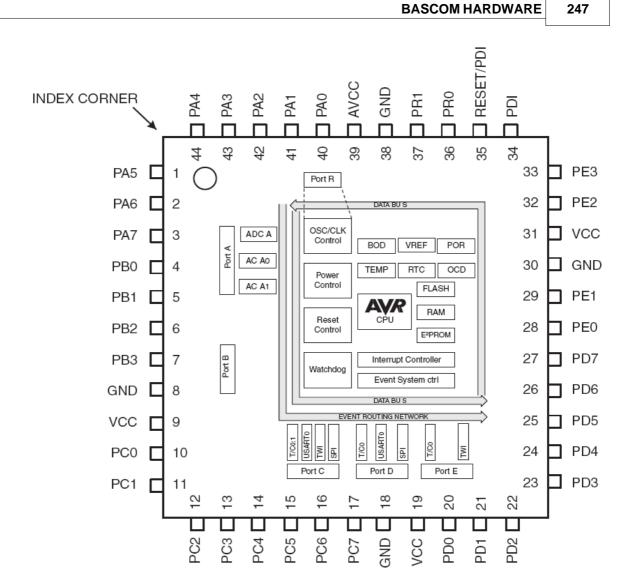
ATXMEGA is not available in PDIP. This means that the XMEGA is not really suited for hobby projects.

As a result, the DEMO version does not support the XMEGA.

4.18.77 ATXMEGA16A4



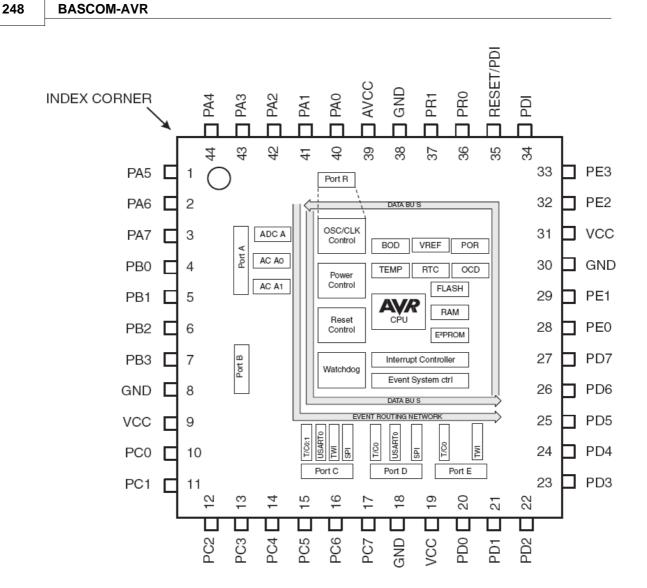
4.18.78 ATXMEGA16D4



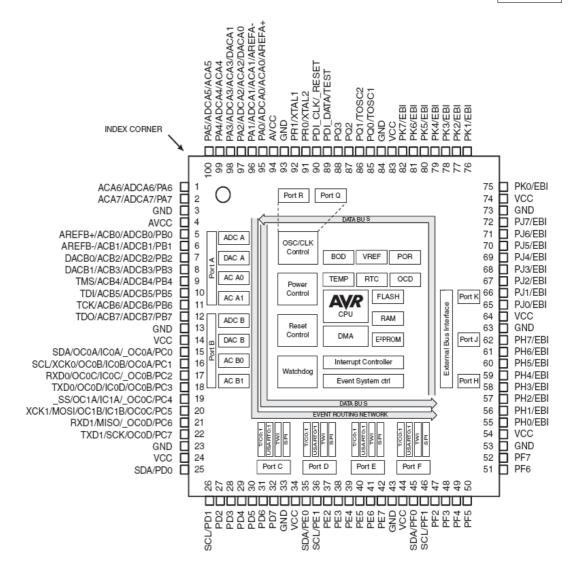
4.18.79 ATXMEGA32A4

Enter topic text here.

4.18.80 ATXMEGA32D4



4.18.81 ATXMEGA64A1



Here a note about the spike detector :

>The calibration byte in the production signature row show 0xFF and 0x00 >for the ADC Calibration byte. Are these really the calibration values ? >And I'm not able to set the HIGH Byte of the calibration register. >Errata of Rev H don't show something from calibration bytes.

Reply from Atmel :

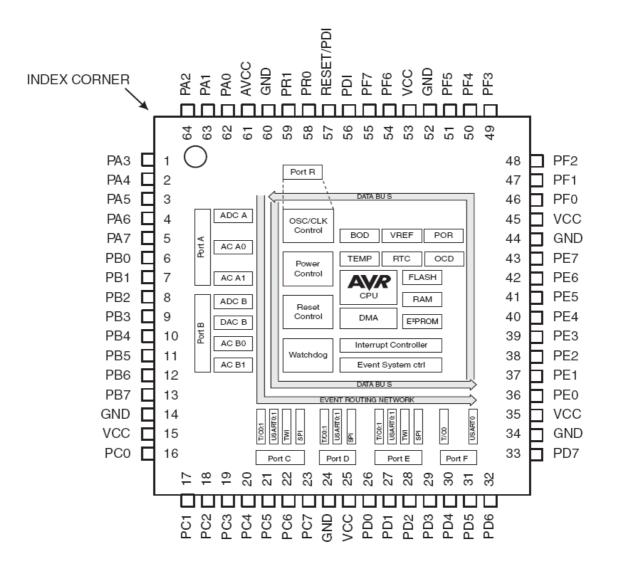
The voltage spike detector has been removed from the latest revision of the XMEGA A manual.

This is because we have, unfortunately, not been able to validate the spike detector fully. The module is disabled in currently available parts to avoid unforeseen problems for any customers.

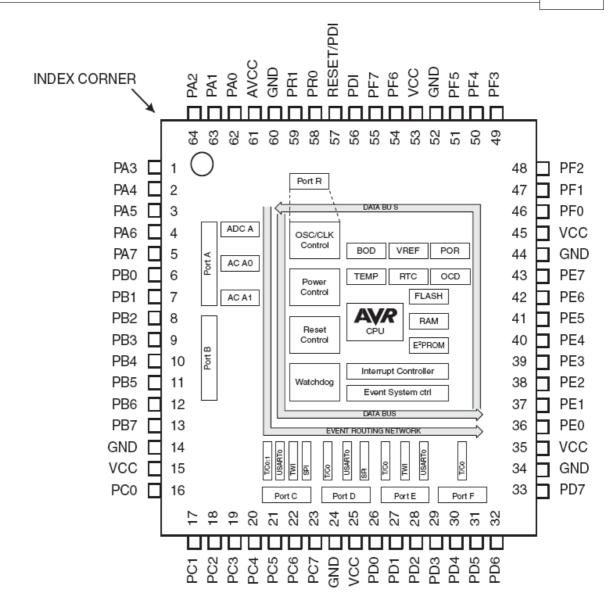
4.18.82 ATXMEGA64A3

250

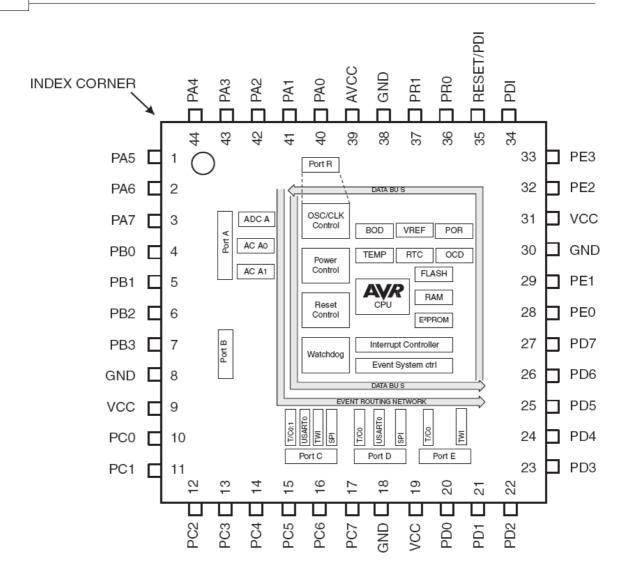
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.83 ATXMEGA64D3



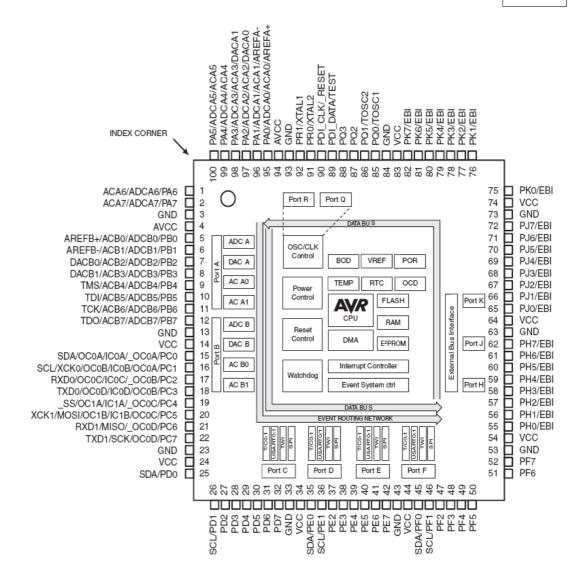
4.18.84 ATXMEGA64D4



4.18.85 ATXMEGA128A1

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

© 2011 MCS Electronics



Question: The DVDSON FUSE BIT the ATxmega A MANUAL says that for characterization data on VDROP and tSD consult the device data sheet. (Device: ATXMEGA128A1 RevH). But I can't find this Information in the datasheet ?

Answer: The voltage spike detector has been removed from the latest revision of the XMEGA A manual.

This is because we have, unfortunately, not been able to validate the spike detector fully.

Question: The calibration byte in the production signature row show 0xFF and 0x00 for the ADC Calibration byte. Are these really the calibration values ? Errata of Rev H don't show something from calibration bytes. (Device: ATXMEGA128A1 RevH)

Answer: Yes this is a known issue with ATXMEGA128A1 RevH. We will be fixing up this

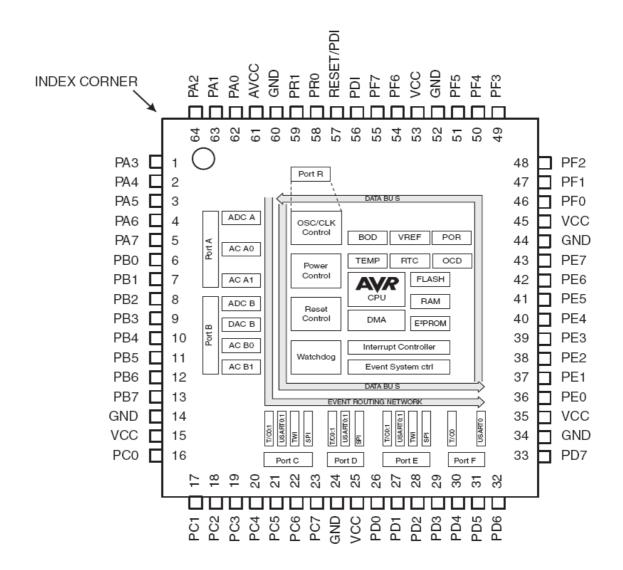
issue in the later version of the device.

You should write the code for loading the calibration registers in the firmware so that when we fix it in the later version you do not have to fix

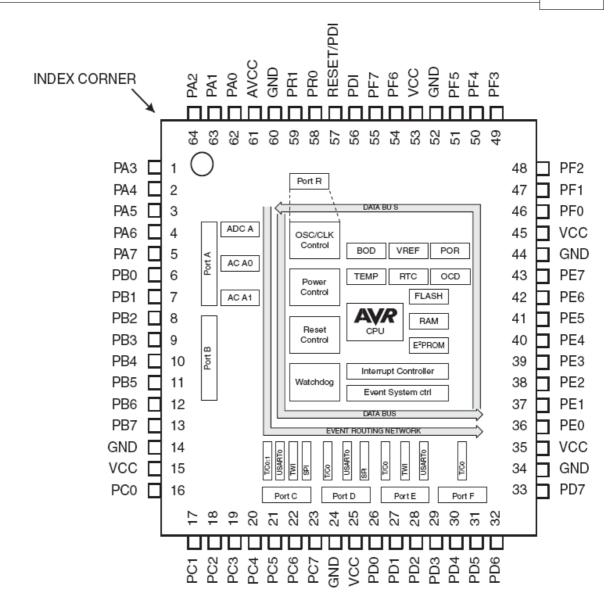
the code. Also loading it now will not cause any problem in the ADC operation.

4.18.86 ATXMEGA128A3

This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

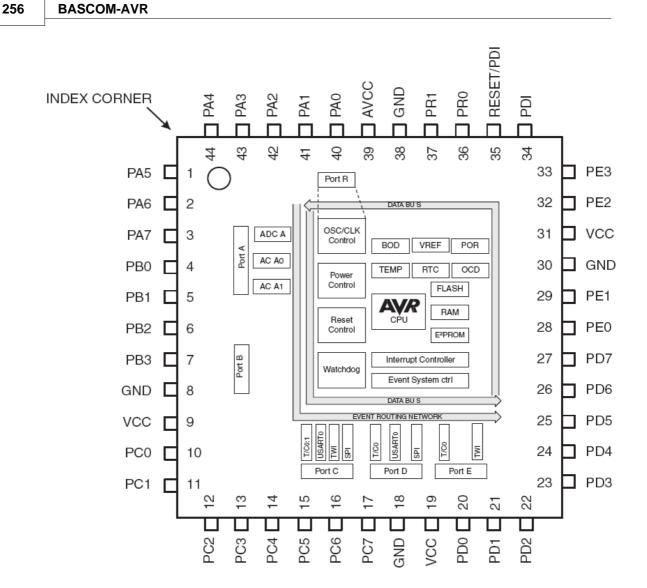


4.18.87 ATXMEGA128D3

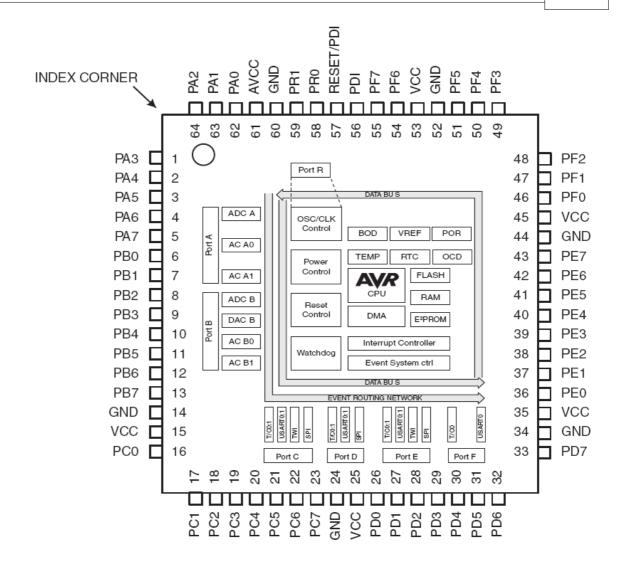


4.18.88 ATXMEGA128D4

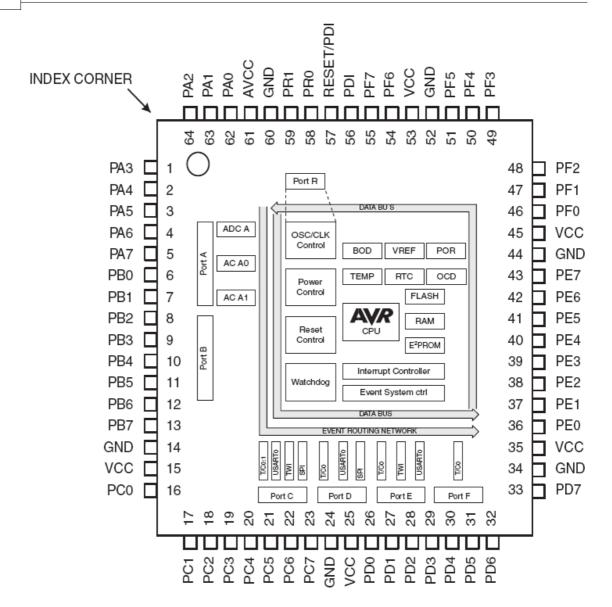
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet. Read the generic info about Xmega.



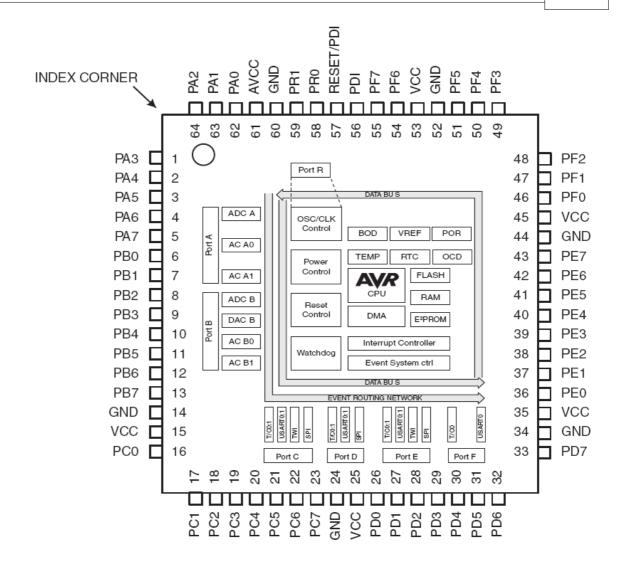
4.18.89 ATXMEGA192A3



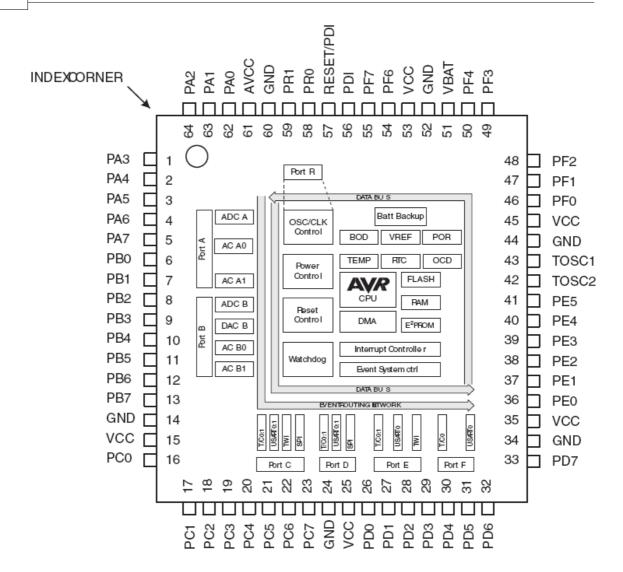
4.18.90 ATXMEGA192D3



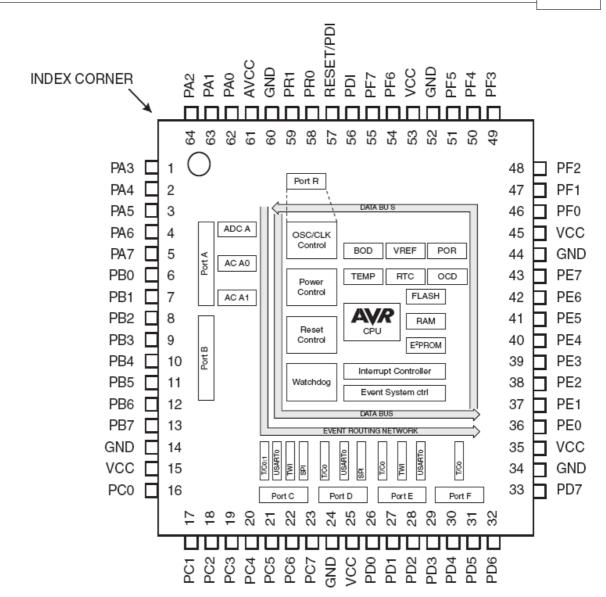
4.18.91 ATXMEGA256A3



4.18.92 ATXMEGA256A3B



4.18.93 ATXMEGA256D3



4.19 Reference Designs

4.19.1 EM4095 RFID Reader

Introduction

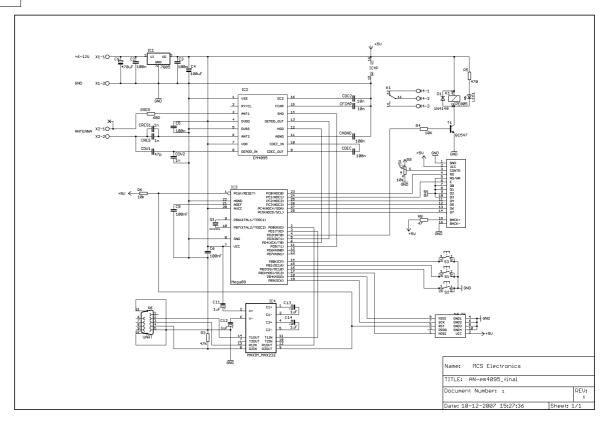
RFID technology is an exciting technology. The EM4095 chip allows us to create a reader with little code or processor resources. A complete KIT is available from the web shop at <u>www.mcselec.com</u>

This topic describes the reference design. The data sheets you can download from:

EM4095 (chip) , EM4102 (transponder)

The circuit

© 2011 MCS Electronics

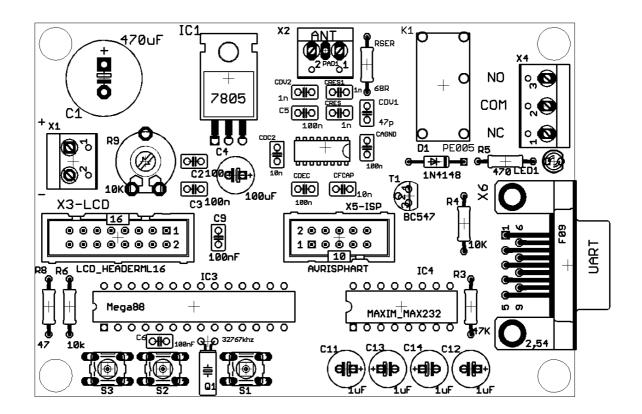


As you can see from the data sheets, the EM4095 needs little external hardware. A coil, capacitors that tune the coil for 125 KHz, are basically all that you need. IC1 is a voltage regulator that regulates the input voltage to 5V. (you can operate it from a 9V battery). The capacitors stabilize the output voltage. The DEMOD output of the EM4095 is connected to the microprocessor and the pin is used in input mode. The MOD and SHD pins are connected to micro pins that are used in output mode.

The micro(mega88) has a small 32 KHz crystal so the soft clock can be used. There are 3 switches that can be used for menu input, and there is a relay that can be used to activate a door opener. Parallel on the relay there is a LED for a visible indication. IC4 is a serial interface buffer so we can connect the PCB to our computer for logging and programming. The Mega88 is delivered with a Boot loader and thus can be serial programmed with the MCS Boot loader. That is why pin 4 of X6 (DTR) is connected via IC4(pin 8-9) to the reset pin of the micro(pin 1).

Further there is a standard 10-pins ISP programmer connector for the USB-ISP or STK200, and an LCD connector for an optional LCD display.

The PCB



Part list

Component	Value
C1	470uF/25V
C2,C3,C5,C6,C9,CDEC,CAGND	100nF (104)
C4	100uF/16V
CRES1,CRES, CDV2	1nF(102)
CDV1	47pF
CDC2,CFCAP	10nF(103)
C11,C12,C13,C14	1uF/16V
RSER	68
R4,R6	10K
R5	470
R8	47
R3	47K
R9	1K-10K pot
IC1	7805
IC2	EM4095
IC3	ATMEGA88
IC4	MAX232
20 pin IC feet, 16 pin IC feet	
X1,X2	2-pin header
X3	16 pin boxed header
Х4	3-pin header
X5	10-pin boxed header
Х6	DB-9 female connector
Τ1	BC547
D1	1N4148
LED1	3 mm LED, red
K1	Relay, 5V

S1,S2,S3	switch
Q1	32768 Hz crystal
Antenna	
M3x6 bolt and nut	
4 rubber feet	

Building the PCB

As usually we start with the components that have the lowest height. And normally we would solder all passive components first, and insert/solder the active components last. This to prevent damage to the active components(IC). But since the EM4095 is only available in SMD, we need to solder this chip first. Make sure the chip is lined out right and that pin 1 matches the small dot on the chip which is an indication for pin 1. Then solder pin 1 and 16 so the chip can not be moved anymore. Now solder the remaining pins. Use an iron with a small tip. When you use too much solder, and two feet are soldered together do not panic. Just finish soldering and when ready, use some copper braid to remove the solder between the 2 feet. This works best when you lay the braid over the 2 pins, then push the solder iron to the braid so it will heat up. Then after some seconds, add some solder which will get sucked into the braid. This will in turn suck the other solder into the braid. While it does not seem logical to add solder, it will conduct the heat better. But since the used SMD chip is relatively large there should not be any problem.

Now mount and solder the following components :

- RSER (68 ohm)
- R3 (47K)
- R4,R6 (10 K)
- R5 (470)
- R8 (47 for LCD)
- D1 (diode 1N4148). The black line must match the line on the PCB(Kathode)
- C2,C3,C5,C6,C9,CDEC,CAGND (100 nF)
- CRES1,CRES , CDV2 (1nF)
- CDV1 (47pF)
- CDC2,CFCAP (10nF)
- 28 pins IC feet for the Mega88 and 16 pins IC feet for the MAX232
- Bend the wires of IC1 and mount IC1 with the bolt and nut
- Bend the wires of the crystal and mount $\ensuremath{\text{Q1}}$
- S1,S2,S3 (switches)
- LED1. The square pad matches the longest wire of the LED(Anode)
- R9 (potmeter for LCD contrast)
- T1(transistor BC547)
- Boxed header X5 and X3. Notice the gap in the middle which must match with the PCB
- X6 (DB9-female connector)
- K1 (relay)
- C11,C12,C13,C14 (1uF/16V)
- C4 (100uF/16V)
- X1,X2 (2 pins screw connectors)
- X4 (3 pin screw connector)
- C1 (470 uF/25V)
- 4 rubber feet

Operation

Now the PCB is ready. Make sure there are no solder drops on the PCB. You can measure with an Ohm-meter if there is a short circuit.

Measure pin 1 and pin 2 of IC1 (the voltage input) and pin 3 and pin 2 of IC1 (the voltage output).

When everything is ok, insert the MAX232 and the MEGA88.

You can connect the battery cord to header X1. The red wire is the plus. Since the circuit is not for beginners, there is no reverse polarity protection. While the 7805 does not mind a short circuit, the C1 elco might not like it.

Connect the battery and measure with a Volt meter if IC1 actual outputs 5V. If not, check the input voltage, and for a possible shortcut.

Connect the antenna to connector X2. The PCB is now ready for use. When you have the LCD display, connect it to the LCD header and adjust the variable resistor R9 so you can see square blocks.

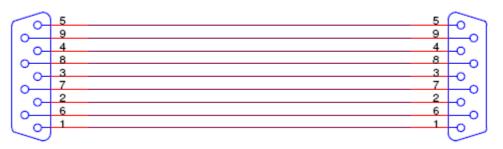
Since the chip has a boot loader, you can serial program the device. We made a simple AN that can be used as a door opener. It has simple menu, and we can add new tags. When a valid tag is held in front of the antenna, it will activate the relay for 2 seconds. The LED will be turned on as well.

Compile the program **AN_READHITAG_EM4095.BAS** and select the **MCS Boot Loader programmer**. Connect a serial cable to X6 and press F4 to program.

You need a normal straight cable.

DB9-MALE

DB9-FEMALE



When you did not used the **MCS Bootloader** before, check the COM port settings and make sure the BAUD is set to 38400 as in the following screenshot:

266 BASCOM-AVR

BASCOM-AVR Options			
Compiler Communicat	tion <u>E</u> nvironment <u>S</u> imulator	Programmer Monitor F	Printer
Programmer	MCS Bootloader	•	
Play sound			
Erase warning	🗹 Auto Flash 🛛 📝 AutoVerify npile 👘 📄 Set focus	Upload Code and I to terminal emulator after p	
Serial MCS Loader			
COM-port [1 TAU BAU	ID 38400 🔻	
STK500 EXE		F	
USB	~		
Default	V <u>D</u> k	X Cancel	

You also need to set 'RESET via DTR' on the 'MCS Loader' TAB.

Now the program will start and show some info on the LCD. Each time you hold a RFID tag before the antenna/coil, the TAG ID will be shown.

When you press S3, you can store an RFID. Press S3, and then hold the TAG before the coil. When there is room , or the tag is new, it will be stored. Otherwise it will be ignored. The TAG ID is also stored in EEPROM.

Now when you hold the tag before the coil, the relay is activated for 2 seconds. The AN is very simple and you can change and extend it easily.

One nice idea from Gerhard : use one TAG as a master tag to be able to add/remove tags.

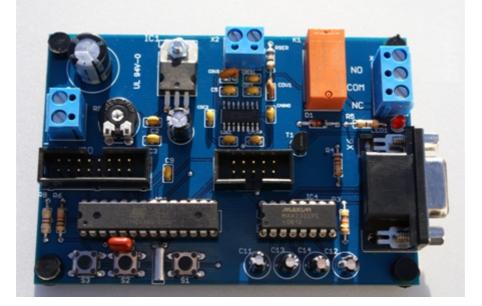
Security

To make the code more secure you could add a delay so that a valid tag must be received twice, so after the valid TAG, wait 1 second, and then start a new measurement and check if the TAG is valid again.

This will prevent where a bit generator could be used to generate all possible codes. With 64 bit times a second, it would take ages before it would work.

The other hack would be to listen with a long range 125 KHz antenna, and recording all bits. A long range scanner would be very hard to make. It would be easier to open the door with a crowbar.

When you open your door with this device, make sure you have a backup option like a key in case there is no power. Also, when the door is opened by a magnetic door opener, make sure it has the right quality for the entrance you want to protect.



AN Code

(c) 1995-2008 MCS Electronics This sample will read a HITAG chip based on the EM4095 cl Consult EM4102 and EM4095 datasheets for more info	hip	
The EM4095 was implemented after an idea of Gerhard Günzel Gerhard provided the hardware and did research at the coil and capacitors. The EM4095 is much simpler to use than the HTRC110. It need less pins. A reference design with all parts is available from MCS		
<pre>\$regfile = "M88def.dat" \$baud = 19200 \$crystal = 8000000 \$hwstack = 40 \$swstack = 40 \$framesize = 40</pre>		
Declare Function Havetag(b As Byte) As Byte		
<pre>'Make SHD and MOD low _md Alias Portd.4 Config _md = Output _md = 0</pre>		
_shd Alias Portd.5 Config _shd = Output _shd = 0		
Relay Alias Portd.2 Config Relay = Output		
S3 Alias Pinb.0 S2 Alias Pinb.2 S1 Alias Pinb.1 Portb = &B111	' these are all input p	
<pre>Config Clock = Soft Config Date = Dmy , Separator = -</pre>	'we use a clock	

```
268 BASCOM-AVR
```

```
Enable Interrupts
                                                              ' the clock and RFID co
Date = "15 - 12 - 07"
                                                              ' just a special date t
Time$ = "00:00:00"
'Config Lcd Sets The Portpins Of The Lcd
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5
Config Lcd = 16 \times 2
                                                              '16*2 type LCD screen
Cls
             Lcd " EM4095 sample"
Lowerline : Lcd "MCS Electronics"
Dim Tags(5) As Byte
                                                              'make sure the array is
Dim J As Byte , Idx As Byte
Dim Eramdum As Eram Byte
                                                              ' do not use first pos:
                                                              ' number of stored tags
Dim Etagcount As Eram Byte
                                                              'room for 20 tags
Dim Etags(100) As Eram Byte
Dim Stags(100) As Byte
                                                              'since we have enough S
Dim Btags As Byte , Tmp1 As Byte , Tmp2 As Byte
Dim K As Byte , Tel As Byte , M As Byte
Config Hitag = 64 , Type = Em4095 , Demod = Pind.3 , Int = @int1
Print "EM4095 sample"
'you could use the PCINT option too, but you must mask all pins out so it will only
 Pcmsk2 = \&B0000_0100
' On Pcint2 Checkints
' Enable Pcint2
On Intl Checkints Nosave
                                                              'we use the INT1 pin a
Config Int1 = Change
                                                              'we have to config so t
Enable Interrupts
                                                              'as last we have to ena
'read eeprom and store in sram
'when the program starts we read the EEPROM and store it in SRAM
For Idx = 1 To 100
                                                              'for all stored tags
   Stags(idx) = Etags(idx)
   Print Hex(stags(idx)) ; ",";
Next
                                                              ' get number of stored
Btags = Etagcount
                                                              ' an empty cell is FF
If Btags = 255 Then
   Print "No tags stored yet"
   Btags = 0 : Etagcount = Btags
                                                              ' reset and write to ea
                                                              ' we have some tags
Else
   For J = 1 To Btags
       Tmp2 = J * 5
                                                              'end
       Tmp1 = Tmp2 - 4
                                                              'start
       Print "RFID ; " ; J
                                                              ' just for debug
       For Idx = Tmp1 To Tmp2
         Print Hex(stags(idx)) ; ",";
       Next
       Print
  Next
End If
Do
   Print "Check..."
   Upperline : Lcd Time$ ; " Detect"
                                                              'this will enable INT1
   If Readhitag(tags(1)) = 1 Then
      Lowerline
      For J = 1 To 5
         Print Hex(tags(j)) ; ",";
         Lcd Hex(tags(j)) ; ","
```

BASCOM HARDWARE 2

```
269
```

```
Next
      M = Havetag(tags(1))
                                                              'check if we have this
      If M > 0 Then
         Print "Valid TAG ;" ; M
         Relay = 1
                                                              'turn on relay
         Waitms 2000
                                                              'wait 2 secs
                                                              'relay off
         Relay = 0
      End If
      Print
  Else
     Print "Nothing"
  End If
  If S3 = 0 Then
                                                              'user pressed button 3
     Print "Button 3"
     Cls : Lcd "Add RFID"
     Do
       If Readhitag(tags(1)) = 1 Then
                                                              'this will enable INT1
          If Havetag(tags(1)) = 0 Then
                                                              'we do not have it yet
             If Btags < 20 Then
                                                                 'will it fit?
                Incr Btags
                                                                 'add one
                Etagcount = Btags
                Idx = Btags * 5
                                                                 'offset
                Idx = Idx - 4
                Lowerline
                For J = 1 To 5
                  Lcd Hex(tags(j)) ; ","
                  Stags(idx) = Tags(j)
                  Etags(idx) = Tags(j)
                  Incr Idx
                Next
                Cls
                Lcd "TAG stored" : Waitms 1000
             End If
          End If
          Exit Do
       End If
    Loop
 End If
  If S2 = 0 Then
     Print "Button 2"
  End If
  If S1 = 0 Then
     Print "Button 1"
 End If
 Waitms 500
Loop
'check to see if a tag is stored already
'return 0 if not stored
'return value 1-20 if stored
Function Havetag(b As Byte ) As Byte
 Print "Check if we have TAG : ";
 For K = 1 To 5
     Print Hex(b(k)) ; ","
 Next
 For K = 1 To 20
    Tmp2 = K * 5
                                                              'end addres
    Tmp1 = Tmp2 - 4
                                                              'start
```

270 BASCOM-AVR

```
Tel = 0
    For Idx = Tmp1 To Tmp2
       Incr Tel
                                                              'if they do not match
       If Stags(idx) <> B(tel) Then
          Exit For
                                                              'exit and try next
       End If
    Next
    If Tel = 5 Then
                                                              'if we did found 5 mate
       Print "We have one"
       Havetag = K
                                                              'set index
       Exit Function
    End If
 Next
 Havetag = 0
                                                              'assume we have nothing
End Function
Checkints:
Call _checkhitag
                                                              'in case you have used
```

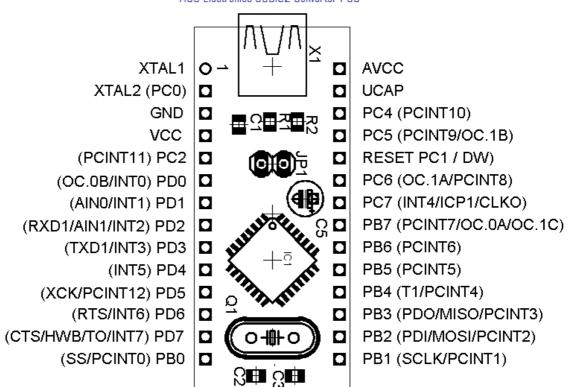
Tips and Tricks

The oscillator frequency must be 125 KHz. You can measure this with an oscilloscope. It is possible that you need to remove a few windings of the antenna coil to get an exact 125 KHz. This will result in a higher distance that you can use for the tags.

4.19.2 USB162 module

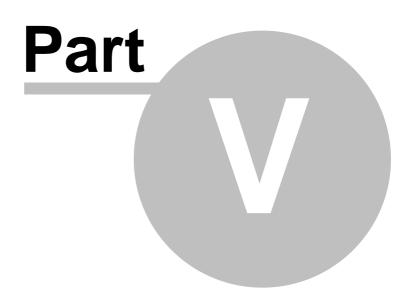
Return

The USB162 from Atmel is great new chip with USB device support. The only downside for most hobbyists will be that it is not available in DIP format. MCS Electronics created a small converter board with normal pins with the size of a 28 pin DIP chip.



MCS Electronics USB162 Converter PCB

The USB module is available from the MCS Electronics online web shop.



5 BASCOM Language Fundamentals

5.1 Changes compared to BASCOM-8051

The design goal was to make BASCOM-AVR compatible with BASCOM-8051.

For the AVR compiler some statements had to be removed. New statements were also added. And some statements were changed.

They need specific attention, but the changes to the syntax will be made available to BASCOM-8051 too in the future.

DESCRIPTION
Not needed anymore.
Code always starts at address 0 for the AVR. Added again in 1.11.6.2
Use LCD Hex(var) instead.
Not needed anymore. Added in 1.11.6.2
Not needed anymore
Can't be used anymore because there is no object code that can be used for it.
Removed.
Can't be used anymore because there is no object code that can be used for it.
AVR does no allow setting priority of interrupts
You can use Print Hex(var) now
You can use Lcd Hex(var) now

Statements that were removed

Statements that were added

STATEMENT	DESCRIPTION
FUNCTION	You can define your own user FUNCTIONS.
LOCAL	You can have LOCAL variables in SUB routines or FUNCTIONS.
^	New math statement. Var = $2 \land 3$ will return $2*2*2$
SHIFT	Because ROTATE was changed, I added the SHIFT statement. SHIFT works just like ROTATE, but when shifted left, the LS BIT is cleared and the carry doesn't go to the LS BIT.
LTRIM	LTRIM, trims the leftmost spaces of a string.
RTRIM	RTRIM, trims the rightmost spaces of a string.
TRIM	TRIM, trims both the leftmost and rightmost spaces of a string.

Statements that behave differently

STATEMEN T	DESCRIPTION
	Rotate now behaves like the ASM rotate, this means that the carry will go to the most significant bit of a variable or the least significant bit of a variable.
	String were added to the CONST statement. I also changed it to be compatible with QB.
DECLARE	BYVAL has been added since real subprograms are now supported.

274 BASCOM-AVR

DIM	You can now specify the location in memory of the variable.
	Dim v as byte AT 100, will use memory location 100.

5.2 Language Fundamentals

Characters from the BASCOM character set are put together to form labels, keywords, variables and operators.

These in turn are combined to form the statements that make up a program.

This chapter describes the character set and the format of BASCOM program lines. In particular, it discusses:

- The specific characters in the character set and the special meanings of some characters.
- The format of a line in a BASCOM program.
- Line labels.
- Program line length.

Character Set

The BASCOM BASIC character set consists of alphabetic characters, numeric characters, and special characters.

The alphabetic characters in BASCOM are the uppercase letters (A-Z) and lowercase letters (a-z) of the alphabet.

The BASCOM numeric characters are the digits 0-9.

The letters A-H can be used as parts of hexadecimal numbers. The following characters have special meanings in BASCOM statements and

expressions:

Character	Name
ENTER	Terminates input of a line
	Blank (or space)
1	Single quotation mark (apostrophe)
*	Asterisks (multiplication symbol)
+	Plus sign
,	Comma
-	Minus sign
	Period (decimal point)
/	Slash (division symbol) will be handled as \
:	Colon
"	Double quotation mark
;	Semicolon
<	Less than
=	Equal sign (assignment symbol or relational operator)
>	Greater than
\backslash	Backslash (integer/word division symbol)
^	Exponent

The BASCOM program line

BASCOM program lines have the following syntax:

[[line-identifier]] [[statement]] [[:statement]] ... [[comment]]

Using Line Identifiers

BASCOM support one type of line-identifier; alphanumeric line labels:

An alphabetic line label may be any combination of from 1 to 32 letters and digits, starting with a letter and ending with a colon. BASCOM keywords are not permitted.

The following are valid alphanumeric line labels:

Alpha: ScreenSUB: Test3A:

Case is not significant. The following line labels are equivalent:

alpha: Alpha: ALPHA:

Line labels may begin in any column, as long as they are the first characters other than blanks on the line.

Blanks are not allowed between an alphabetic label and the colon following it. A line can have only one label. When there is a label on the line, no other identifiers may be used on the same line. So the label is the sole identifier on a line.

BASCOM Statements

A BASCOM statement is either "executable" or " non-executable". An executable statement advances the flow of a programs logic by telling the program what to do next.

Non executable statement perform tasks such as allocating storage for variables, declaring and defining variable types.

The following BASCOM statements are examples of non-executable statements:

- REM or (starts a comment)
- DIM

A "comment" is a non-executable statement used to clarify a programs operation and purpose.

A comment is introduced by the REM statement or a single quote character('). The following lines are equivalent:

PRINT " Quantity remaining" : REM Print report label. PRINT " Quantity remaining" ' Print report label.

276 BASCOM-AVR

More than one BASCOM statement can be placed on a line, but colons(:) must separate statements, as illustrated below.

FOR I = 1 TO 5 : PRINT " Gday, mate." : NEXT I

BASCOM LineLength

If you enter your programs using the built-in editor, you are not limited to any line length, although it is advised to shorten your lines to 80 characters for clarity.

Data Types

Every variable in BASCOM has a data type that determines what can be stored in the variable. The next section summarizes the elementary data types.

Elementary Data Types

- Bit (1/8 byte). A bit can hold only the value 0 or 1. A group of 8 bits is called a byte.
- Byte (1 byte). Bytes are stores as unsigned 8-bit binary numbers ranging in value from 0 to 255.
- Integer (two bytes). Integers are stored as signed sixteen-bit binary numbers ranging in value from -32,768 to +32,767.
- Word (two bytes). Words are stored as unsigned sixteen-bit binary numbers ranging in value from 0 to 65535.
- Dword(fout bytes). Dwords are stored as unsigned 32-bit unsigned numbers ranging in value from 0 to 4294967295
- Long (four bytes). Longs are stored as signed 32-bit binary numbers ranging in value from -2147483648 to 2147483647.
- Single. Singles are stored as signed 32 bit binary numbers. Ranging in value from 1.5 x 10^-45 to 3.4 x 10^38
- Double. Doubles are stored as signed 64 bit binary numbers. Ranging in value from 5.0 x 10^-324 to 1.7 x 10^308
- String (up to 254 bytes). Strings are stored as bytes and are terminated with a 0byte. A string dimensioned with a length of 10 bytes will occupy 11 bytes.

Variables can be stored internal (default), external or in EEPROM.

Variables

A variable is a name that refers to an object--a particular number.

A numeric variable, can be assigned only a numeric value (either integer, byte, long, single or bit).

The following list shows some examples of variable assignments:

• A constant value:

A = 5 C = 1.1

- The value of another numeric variable: abc = def
 - k = g
- The value obtained by combining other variables, constants, and operators: Temp

= a + 5 Temp = C + 5

• The value obtained by calling a function: Temp = Asc(S)

Variable Names

A BASCOM variable name may contain up to 32 characters. The characters allowed in a variable name are letters and numbers. The first character in a variable name must be a letter.

A variable name cannot be a reserved word, but embedded reserved words are allowed.

For example, the following statement is illegal because AND is a reserved word.

AND = 8

However, the following statement is legal:

ToAND = 8

Reserved words include all BASCOM commands, statements, function names, internal registers and operator names.

(see <u>BASCOM Reserved Words</u>²⁹⁶), for a complete list of reserved words).

You can specify a hexadecimal or binary number with the prefix &H or &B. a =&HA , a =&B1010 and a = 10 are all the same.

Before assigning a variable, you must tell the compiler about it with the DIM statement.

Dim b1 As Bit, I as Integer, k as Byte , s As String * 10

The STRING type needs an additional parameter to specify the length.

You can also use <u>DEFINT</u> 648, <u>DEFBIT</u> 648, <u>DEFBYTE</u> 648, <u>DEFWORD</u> 648, <u>DEFLNG</u> 648 or DEFSNG 648.

For example, DEFINT c tells the compiler that all variables that are not dimensioned and that are beginning with the character c are of the Integer type.

Expressions and Operators

This chapter discusses how to combine, modify, compare, or get information about expressions by using the operators available in BASCOM.

Anytime you do a calculation you are using expressions and operators.

This chapter describes how expressions are formed and concludes by describing the following kind of operators:

- Arithmetic operators, used to perform calculations.
- Relational operators, used to compare numeric or string values.
- Logical operators, used to test conditions or manipulate individual bits.
- Functional operators, used to supplement simple operators.

Expressions and Operators

An expression can be a numeric constant, a variable, or a single value obtained by combining constants, variables, and other expressions with operators.

Operators perform mathematical or logical operations on values. The operators provided by BASCOM can be divided into four categories, as follows:

- 1. Arithmetic
- 2. Relational
- Logical
- 4. Functional

Arithmetic

Arithmetic operators are $+, -, *, \setminus, /$ and \wedge .

- Integer
 Integer division is denoted by the backslash (\).
 Example: Z = X \ Y
- Modulo Arithmetic

Modulo arithmetic is denoted by the modulus operator MOD. Modulo arithmetic provides the remainder, rather than the quotient, of an integer division.

Example: $X = 10 \setminus 4$: remainder = 10 MOD 4

 Overflow and division by zero Division by zero, produces an error. At the moment no message is produced, so you have to make sure yourself that this won't happen.

Relational Operators

Relational operators are used to compare two values as shown in the table below. The result can be used to make a decision regarding program flow.

Operator	Relation Tested	Expression
=	Equality	X = Y
<>	Inequality	X <> Y
<	Less than	X < Y
>	Greater than	X > Y
<=	Less than or equal to	X <= Y
>=	Greater than or equal to	X >= Y

Logical Operators

Logical operators perform tests on relations, bit manipulations, or Boolean operators. There four operators in BASCOM are :

Operator	Meaning
NOT	Logical complement
AND	Conjunction
OR	Disjunction
XOR	Exclusive or

It is possible to use logical operators to test bytes for a particular bit pattern. For example the AND operator can be used to mask all but one of the bits of a status byte, while OR can be used to merge two bytes to create a particular binary value.

Example

A = 63 And 19 PRINT A A = 10 Or 9 PRINT A

Output

19 11

Floating point SINGLE (4 BYTE)(ASM code used is supplied by Jack Tidwell) Single numbers conforming to the IEEE binary floating point standard. An eight bit exponent and 24 bit mantissa are supported. Using four bytes the format is shown below:

31 30	23 22	0
JT J0	25 22	0

s exponent mantissa

The exponent is biased by 128. Above 128 are positive exponents and below are negative. The sign bit is 0 for positive numbers and 1 for negative. The mantissa is stored in hidden bit normalized format so that 24 bits of precision can be obtained.

All mathematical operations are supported by the single. You can also convert a single to an integer or word or vise versa:

Dim I as Integer, S as Single

S = 100.1 'assign the single

I = S 'will convert the single to an integer

Here is a fragment from the Microsoft knowledge base about FP:

Floating-point mathematics is a complex topic that confuses many programmers. The tutorial below should help you recognize programming situations where floating-point errors are likely to occur and how to avoid them. It should also allow you to recognize cases that are caused by inherent floating-point math limitations as opposed to actual compiler bugs.

Decimal and Binary Number Systems

Normally, we count things in base 10. The base is completely arbitrary. The only reason that people have traditionally used base 10 is that they have 10 fingers, which have made handy counting tools.

The number 532.25 in decimal (base 10) means the following:

 $(5 * 10^2) + (3 * 10^1) + (2 * 10^0) + (2 * 10^{-1}) + (5 * 10^{-2})$

500 + 30 + 2 + 2/10 + 5/100

= 532.25

In the binary number system (base 2), each column represents a power of 2 instead of 10. For example, the number 101.01 means the following: $(1 * 2^2) + (0 * 2^1) + (1 * 2^0) + (0 * 2^{-1}) + (1 * 2^{-2})$ 4 + 0 + 1 + 0 + 1/4

= 5.25 Decimal

How Integers Are Represented in PCs

Because there is no fractional part to an integer, its machine representation is much simpler than it is for floating-point values. Normal integers on personal computers (PCs) are 2 bytes (16 bits) long with the most significant bit indicating the sign. Long integers are 4 bytes long.

Positive values are straightforward binary numbers. For example:

1 Decimal = 1 Binary 2 Decimal = 10 Binary 22 Decimal = 10110 Binary, etc.

However, negative integers are represented using the two's complement scheme. To get the two's complement representation for a negative number, take the binary representation for the number's absolute value and then flip all the bits and add 1. For example:

4 Decimal = 0000 0000 0000 0100

1111 1111 1111 1011 Flip the Bits

-4 = 1111 1111 1111 1100 Add 1

Note that adding any combination of two's complement numbers together

using ordinary binary arithmetic produces the correct result.

Floating-Point Complications

Every decimal integer can be exactly represented by a binary integer; however, this is not true for fractional numbers. In fact, every number that is irrational in base 10 will also be irrational in any system with a base smaller than 10.

For binary, in particular, only fractional numbers that can be represented in the form p/q, where q is an integer power of 2, can be expressed exactly, with a finite number of bits.

Even common decimal fractions, such as decimal 0.0001, cannot be represented exactly in binary. (0.0001 is a repeating binary fraction with a period of 104 bits!)

This explains why a simple example, such as the following

SUM = 0 FOR I% = 1 TO 10000 SUM = SUM + 0.0001 NEXT I% PRINT SUM ' Theoretically = 1.0.

will PRINT 1.000054 as output. The small error in representing 0.0001 in binary propagates to the sum.

For the same reason, you should always be very cautious when making comparisons on real numbers. The following example illustrates a common programming error:

item1# = 69.82# item2# = 69.20# + 0.62# IF item1# = item2# then print "Equality!"

This will NOT PRINT "Equality!" because 69.82 cannot be represented exactly in binary, which causes the value that results from the assignment to be SLIGHTLY different (in binary) than the value that is generated from the expression. In practice, you should always code such comparisons in such a way as to allow for some tolerance.

General Floating-Point Concepts

It is very important to realize that any binary floating-point system can represent only a finite number of floating-point values in exact form. All other values must be approximated by the closest represent able value. The IEEE standard specifies the method for rounding values to the "closest" represent able value. BASCOM supports the standard and rounds according to the IEEE rules.

Also, keep in mind that the numbers that can be represented in IEEE are spread out over a very wide range. You can imagine them on a number line. There is a high density of represent able numbers near 1.0 and -1.0 but fewer and fewer as you go towards 0 or infinity.

The goal of the IEEE standard, which is designed for engineering calculations, is to maximize accuracy (to get as close as possible to the actual number). Precision refers to the number of digits that you can represent. The IEEE standard attempts to balance the number of bits dedicated to the exponent with the number of bits used for the fractional part of the number, to keep both accuracy and precision within

acceptable limits.

IEEE Details

Floating-point numbers are represented in the following form, where [exponent] is the binary exponent:

X = Fraction * 2^(exponent - bias)

[Fraction] is the normalized fractional part of the number, normalized because the exponent is adjusted so that the leading bit is always a 1. This way, it does not have to be stored, and you get one more bit of precision. This is why there is an implied bit. You can think of this like scientific notation, where you manipulate the exponent to have one digit to the left of the decimal point, except in binary, you can always manipulate the exponent so that the first bit is a 1, since there are only 1s and 0s.

[bias] is the bias value used to avoid having to store negative exponents.

The bias for single-precision numbers is 127 and 1023 (decimal) for double-precision numbers.

The values equal to all 0's and all 1's (binary) are reserved for representing special cases. There are other special cases as well, that indicate various error conditions.

Single-Precision Examples

 $2 = 1 * 2^{1} = 0100\ 0000\ 0000\ 0000\ \dots\ 0000\ 0000 = 4000\ 0000\ hex$ Note the sign bit is zero, and the stored exponent is 128, or

100 0000 0 in binary, which is 127 plus 1. The stored mantissa is (1.) 000 0000 ... 0000 0000, which has an implied leading 1 and binary point, so the actual mantissa is 1.

 $-2 = -1 * 2^{1} = 1100\ 0000\ 0000\ 0000\ \dots\ 0000\ 0000 = C000\ 0000\ hex$ Same as +2 except that the sign bit is set. This is true for all IEEE format floating-point numbers.

 $4 = 1 * 2^2 = 0100\ 0000\ 1000\ 0000\ \dots\ 0000\ 0000 = 4080\ 0000\ hex$ Same mantissa, exponent increases by one (biased value is 129, or 100\ 0000\ 1 in binary.

 $6 = 1.5 * 2^2 = 0100\ 0000\ 1100\ 0000\ \dots\ 0000\ 0000 = 40C0\ 0000\ hex$ Same exponent, mantissa is larger by half -- it's

(1.) 100 0000 ... 0000 0000, which, since this is a binary fraction, is 1-1/2 (the values of the fractional digits are 1/2, 1/4, 1/8, etc.).

 $1 = 1 * 2^0 = 0011 1111 1000 0000 \dots 0000 0000 = 3F80 0000 hex$ Same exponent as other powers of 2, mantissa is one less than 2 at 127, or 011 1111 1 in binary.

.75 = 1.5 * 2⁻¹ = 0011 1111 0100 0000 ... 0000 0000 = 3F40 0000 hex

The biased exponent is 126, 011 1111 0 in binary, and the mantissa is (1.) 100 0000 ... 0000 0000, which is 1-1/2.

 $2.5 = 1.25 * 2^{1} = 0100\ 0000\ 0010\ 0000\ \dots\ 0000\ 0000\ = 4020\ 0000\ hex$ Exactly the same as 2 except that the bit which represents 1/4 is set in the mantissa.

0.1 = 1.6 * 2⁻⁴ = 0011 1101 1100 1100 ... 1100 1101 = 3DCC CCCD hex

1/10 is a repeating fraction in binary. The mantissa is just shy of 1.6, and the biased exponent says that 1.6 is to be divided by 16 (it is 011 1101 1 in binary, which is 123 n decimal). The true exponent is 123 - 127 = -4, which means that the factor by which to multiply is $2^{**}-4 = 1/16$. Note that the stored mantissa is rounded up in the last bit. This is an attempt to represent the un-representable number as accurately as possible. (The reason that 1/10 and 1/100 are not exactly representable in binary is similar to the way that 1/3 is not exactly representable in decimal.)

 $0 = 1.0 * 2^{-128} = all zeros -- a special case.$

Other Common Floating-Point Errors

The following are common floating-point errors:

Round-off error
 This error results when all of the bits in a binary number cannot be used in a calculation.
 Example: Adding 0.0001 to 0.9900 (Single Precision)
 Decimal 0.0001 will be represented as:
 (1.)10100011011011100010111 * 2^(-14+Bias) (13 Leading 0s in Binary!)

0.9900 will be represented as: (1.)11111010111000010100011 * 2^(-1+Bias)

Now to actually add these numbers, the decimal (binary) points must be aligned. For this they must be Unnormalized. Here is the resulting addition:

.0000000000011010001101 * 2^0 <- Only 11 of 23 Bits retained +.111111010111000010100011 * 2^0

.111111010111011100110000 * 2^0

This is called a round-off error because some computers round when shifting for addition. Others simply truncate. Round-off errors are important to consider whenever you are adding or multiplying two very different values.

2. Subtracting two almost equal values

.1235 -.1234

.0001

This will be normalized. Note that although the original numbers each had four significant digits, the result has only one significant digit.

3. Overflow and underflow

This occurs when the result is too large or too small to be represented by the data

type.

4. Quantizing error

This occurs with those numbers that cannot be represented in exact form by the floating-point standard.

Rounding

When a Long is assigned to a single, the number is rounded according to the rules of the IEEE committee.

For explanation: 1.500000 is exact the middle between 1.00000 and 2.000000. If x.500000 is always rounded up, than there is trend for higher values than the average of all numbers. So their rule says, half time to round up and half time to round down, if value behind LSB is exact ..500000000.

The rule is, round this .500000000000 to next even number, that means if LSB is 1 (half time) to round up, so the LSB is going to 0 (=even), if LSB is 0 (other half time) to round down, that means no rounding.

This rounding method is best since the absolute error is 0.

You can override the default IEEE rounding method by specifying the \$LIB LONG2FLOAT.LBX library which rounds up to the next number. This is the method used up to 1.11.7.4 of the compiler.

Double

The double is essential the same as a single. Except the double consist of 8 bytes instead of 4. The exponent is 11 bits leaving 52 bits for the mantissa.

Arrays

An array is a set of sequentially indexed elements having the same type. Each element of an array has a unique index number that identifies it. Changes made to an element of an array do not affect the other elements.

The index must be a numeric constant, a byte, an integer, word or long. The maximum number of elements is 65535. The first element of an array is always one. This means that elements are 1-based.

Arrays can be used on each place where a 'normal' variable is expected.

Example:

'create an array named a, with 10 elements (1 to 10) Dim A(10) As Byte 'create an integer Dim C As Integer 'now fill the array For C = 1 To 10 'assign array element $\begin{array}{l} A(c)=C\\ ' \mbox{ print it}\\ \mbox{Print } A(c)\\ \mbox{Next}\\ 'you \ can \ add \ an \ offset \ to \ the \ index \ too\\ C=0\\ A(c+1)=100\\ \mbox{Print } A(c+1)\\ \mbox{End} \end{array}$

Strings

A string is used to store text. A string must be dimensioned with the length specified.

DIM S as STRING * 5

Will create a string that can store a text with a maximum length of 5 bytes. The space used is 6 bytes because a string is terminated with a null byte.

To assign the string: s = "abcd"

To insert special characters into the string : $s = "AB{027}cd"$

The {ascii} will insert the ASCII value into the string.

The number of digits must be 3. $s = \{27\}$ will assign $\{27\}$ to the string instead of escape character 27!

Casting

In BASCOM-AVR when you perform operations on variables they all must be of the same data type.

long = long1 * long2 ' for example

The assigned variables data type determines what kind of math is performed. For example when you assign a long, long math will be used.

If you try to store the result of a LONG into a byte, only the LSB of the LONG will be stored into the BYTE. Byte = LONG

When LONG = 256, it will not fit into a BYTE. The result will be 256 AND 255 = 0.

Of course you are free to use different data types. The correct result is only guaranteed when you are using data types of the same kind or that result always can fit into the target data type.

When you use strings, the same rules apply. But there is one exception:

Dim b as Byte

b = 123 ' ok this is normal

b = "A" ' b = 65

When the target is a byte and the source variable is a string constant denoted by "", the ASCII value will be stored in the byte. This works also for tests :

```
IF b = "A" then ' when b = 65
```

END IF

This is different compared to QB/VB where you can not assign a string to a byte variable.

SINGLE CONVERSION

When you want to convert a SINGLE into a byte, word, integer or long the compiler will automatic convert the values when the source string is of the SINGLE data type.

integer = single

You can also convert a byte, word, integer or long into a SINGLE by assigning this variable to a SINGLE.

single = long

5.3 Mixing ASM and BASIC

BASCOM allows you to mix BASIC with assembly. This can be very useful in some situations when you need full control of the generated code.

Almost all assembly mnemonics are recognized by the compiler. The exceptions are : SUB, SWAP, CALL and OUT. These are BASIC reserved words and have priority over the ASM mnemonics. To use these mnemonics precede them with the ! - sign.

For example :

Dim a As Byte At &H60 'A is stored at location &H60 Ldi R27 , \$00 'Load R27 with MSB of address Ldi R26 , \$60 'Load R26 with LSB of address Ld R1, X 'load memory location \$60 into R1 !SWAP R1 'swap nibbles

As you can see the SWAP mnemonic is preceded by a ! sign.

Another option is to use the assembler block directives:

\$ASM

Ldi R27 , \$00 'Load R27 with MSB of address Ldi R26 , \$60 'Load R26 with LSB of address Ld R1, X 'load memory location \$60 into R1 SWAP R1 'swap nibbles **\$END ASM**

A special assembler helper function is provided to load the address into the register X or Z. Y can may not be used because it is used as the soft stack pointer.

Dim A As Byte 'reserve space LOADADR a, X 'load address of variable named A into register pair X This has the same effect as : Ldi R26 , \$60 'for example ! Ldi R27, \$00 'for example !

Some registers are used by BASCOM R4 and R5 are used to point to the stack frame or the temp data storage R6 is used to store some bit variables: R6 bit 0 = flag for integer/word conversion R6 bit 1 = temp bit space used for swapping bits R6 bit 2 = error bit (ERR variable) R6 bit 3 = show/noshow flag when using INPUT statement R8 and R9 are used as a data pointer for the READ statement.

All other registers are used depending on the used statements.

To Load the address of a variable you must enclose them in brackets. Dim B As Bit Lds R16, $\{B\}$ 'will replace $\{B\}$ with the address of variable B

To refer to the bit number you must precede the variable name by BIT.

Sbrs R16, BIT.B 'notice the point!

Since this was the first dimensioned bit the bit number is 0. Bits are stored in bytes and the first dimensioned bit goes in the LS (least significant) bit.

To load an address of a label you must use :

LDI ZL, Low(lbl * 1) LDI ZH , High(lbl * 1)

Where ZL = R30 and may be R24, R26, R28 or R30

And ZH = R31 and may be R25, R27, R29 or R31.

These are so called register pairs that form a pointer.

When you want to use the LPM instruction to retrieve data you must multiply the address with 2 since the AVR object code consist of words. LDI ZL, Low(IbI * 2) LDI ZH , High(IbI * 2) LPM ; get data into R0 LbI:

Atmel mnemonics must be used to program in assembly. You can download the pdf from www.atmel.com that shows how the different mnemonics are used.

Some points of attention : * All instructions that use a constant as a parameter only work on the upper 16 registers (r16-r31) So LDI R15,12 WILL NOT WORK

288 BASCOM-AVR

* The instruction SBR register, K will work with K from 0-255. So you can set multiple bits!

The instruction SBI port, K will work with K from 0-7 and will set only ONE bit in a IO-port register.

The same applies to the CBR and CBI instructions.

You can use constants too:

.equ myval = (10+2)/4 ldi r24,myval+2 '5 ldi r24,asc("A")+1 ; load with 66

Or in BASIC with CONST :

CONST Myval = (10+2) / 4 Ldi r24,myval

How to make your own libraries and call them from BASIC? The files for this sample can be found as libdemo.bas in the SAMPLES dir and as mylib.lib in the LIB dir.

First determine the used parameters and their type. Also consider if they are passed by reference or by value

For example the sub test has two parameters: x which is passed by value (copy of the variable) y which is passed by reference(address of the variable)

In both cases the address of the variable is put on the soft stack which is indexed by the Y pointer.

The first parameter (or a copy) is put on the soft stack first To refer to the address you must use:

ldd r26 , y + 0 ldd r27 , y + 1

This loads the address into pointer X The second parameter will also be put on the soft stack so : The reference for the x variable will be changed :

To refer to the address of x you must use: ldd r26 , y + 2 ldd r27 , y + 3

To refer to the last parameter y you must use ldd r26 , y + 0 ldd r27 , y + 1

Write the sub routine as you are used too but include the name within brackets []

```
[test]
test:
ldd r26,y+2 ; load address of x
ldd r27,y+3
ld r24,x ; get value into r24
inc r24 ; value + 1
st x,r24 ; put back
ldd r26,y+0 ; address of y
ldd r27,y+1
st x,r24 ; store
ret ; ready
[end]
```

To write a function goes the same way. A function returns a result so a function has one additional parameter. It is generated automatic and it has the name of the function. This way you can assign the result to the function name

For example:

Declare Function Test(byval x as byte , y as byte) as byte A virtual variable will be created with the name of the function in this case test. It will be pushed on the soft stack with the Y-pointer.

To reference to the result or name of the function (test) the address will be: y + 0 and y + 1 The first variable x will bring that to y + 2 and y + 3

And the third variable will cause that 3 parameters are saved on the soft stack

To reference to test you must use : ldd r26 , y + 4 ldd r27 , y + 5

To reference variable x ldd r26 , y + 2 ldd r27 , y + 3

And to reference variable y

ldd r26 , y + 0 ldd r27 , y + 1

When you use exit sub or exit function you also need to provide an additional label. It starts with sub_ and must be completed with the function / sub routine name. In our example:

sub_test:

LOCALS

When you use local variables thing become more complicated.

Each local variable address will be put on the soft stack too

When you use 1 local variable its address will become

ldd r26, y+0 ldd r27 , y + 1

All other parameters must be increased with 2 so the reference to y variable changes from

ldd r26 , y + 0 to ldd r26 , y + 2 ldd r27 , y + 1 to ldd r27 , y + 3

And of course also for the other variables.

When you have more local variables just add 2 for each.

Finally you save the file as a .lib file Use the library manager to compile it into the lbx format. The declare sub / function must be in the program where you use the sub / function.

The following is a copy of the libdemo.bas file :

'define the used library

\$lib "mylib.lib"

'also define the used routines

\$external Test

'this is needed so the parameters will be placed correct on the stack Declare Sub Test(byval X As Byte , Y As Byte)

'reserve some space Dim Z As Byte

'call our own sub routine Call Test(1, Z)

'z will be 2 in the used example End

When you use ports in your library you must use .equ to specify the address: .equ EEDR=\$1d In R24, EEDR

This way the library manager knows the address of the port during compile time.

As an alternative precede the mnemonic with a * so the code will not be compiled into the lib. The address of the register will be resolved at run time in that case.

This chapter is not intended to teach you ASM programming. But when you find a topic is missing to interface BASCOM with ASM send me an email.

Translation

In version 1.11.7.5 of the compiler some mnemonics are translated when there is a need for.

For example, SBIC will work only on normal PORT registers. This because the address may not be greater then 5 bits as 3 bits are used for the pin number(0-7).

SBIC worked well in the old AVR chips(AT90Sxxxx) but in the Mega128 where PORTG is on a high address, it will not work.

You always needs a normal register when you want to manipulate the bits of an external register.

For example : LDS r23, PORTG ; get value of PORTG register SBR r23,128 ; set bit 7 STS PORTG, R23

The mnemonics that are translated by the compiler are : IN, OUT, SBIC, SBIS, SBI and CBI.

The compiler will use register R23 for this. So make sure it is not used.

Special instructions

ADR Label ; will create a word with the address of the label name ADR2 Label ; will create a word with the address of the label name, multiplied by 2 to get the byte address since word addresses are used. This is convenient when loading the Z-pointer to use (E)LPM.

5.4 Assembler mnemonics

BASCOM supports the mnemonics as defined by Atmel.

The Assembler accepts mnemonic instructions from the instruction set.

A summary of the instruction set mnemonics and their parameters is given here. For a detailed description of the Instruction set, refer to the AVR Data Book.

Mnemonics	Operand s	Description	Operation	Flags	Cloc k
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V, H	1
ADC	Rd, Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V, H	1
SUB	Rd, Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V, H	1
SUBI	Rd, K	Subtract Immediate	Rd = Rd – K	Z,C,N,V,	1

	1	1	1	•		
				Н		
SBC	C Rd, Rr Subtract with Carry		Rd = Rd - Rr - C	Z,C,N,V, H	1	
BBCI Rd, K Subtract Immediate with Carry		Rd = Rd - K - C	Z,C,N,V, H	1		
AND	Rd, Rr	Logical AND	$Rd = Rd \cdot Rr$	Z,N,V	1	
ANDI	Rd, K	Logical AND with Immediate	$Rd = Rd \cdot K$	Z,N,V	1	
OR	Rd, Rr	Logical OR	Rd = Rd v Rr	Z,N,V	1	
ORI	Rd, K	Logical OR with Immediate	Rd = Rd v K	Z,N,V	1	
EOR	Rd, Rr	Exclusive OR	Rd = Rd Å Rr	Z,N,V	1	
СОМ	Rd	Ones Complement	Rd = \$FF - Rd	Z,C,N,V	1	
NEG	Rd	Twos Complement	Rd = \$00 - Rd	Z,C,N,V, H	1	
SBR	Rd,K	Set Bit(s) in Register	Rd = Rd v K	Z,N,V	1	
CBR	Rd,K	Clear Bit(s) in Register	Rd = Rd · (\$FFh - K)	Z,N,V	1	
INC	Rd	Increment	Rd = Rd + 1	Z,N,V	1	
DEC	Rd	Decrement	Rd = Rd - 1	Z,N,V	1	
TST	Rd	Test for Zero or Minus	$Rd = Rd \cdot Rd$	Z,N,V	1	
CLR	Rd	Clear Register	Rd = Rd Å Rd	Z,N,V	1	
SER	Rd	Set Register	Rd = \$FF	None	1	
ADIW Adiw r24, K6	Rdl, K6	Add Immediate to Word	Rdh:Rdl = Rdh: Rdl + K	Z,C,N,V, S	2	
SBIW	Rdl, K6	Subtract Immediate	Rdh:Rdl = Rdh:	ZCNV	2	
Sbiw R24,K6	KUI, KU	from Word	Rdl - K	s	Z,C,N,V, 2 S	
MUL	Rd,Rr	Multiply Unsigned	R1, R0 = Rd * Rr	С	2 *	
BRANCH INSTRUCTIONS						
RJMP	К	Relative Jump	PC = PC + k + 1	None	2	
IJMP		Indirect Jump to (Z)	PC = Z	None	2	
JMP	К	Jump	PC = k	None	3	
RCALL	К	Relative Call Subroutine	PC = PC + k + 1	None	3	
ICALL		Indirect Call to (Z)	PC = Z	None	3	
CALL	К	Call Subroutine	PC = k	None	4	
RET		Subroutine Return	PC = STACK	None	4	
RETI		Interrupt Return	PC = STACK	I	4	
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC = PC + 2 or 3	None	1/2	
СР	Rd,Rr	Compare	Rd - Rr	Z,C,N,V, H,	1	
СРС	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V, H	1	
СРІ	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V, H	1	
SBRC	Rr, b	Skip if Bit in Register Cleared	If (Rr(b)=0) PC = PC + 2 or 3	None	1/2	

292

BASCOM Language Fundamentals

293

			<u> </u>		
		Skip if Bit in Register Set	If (Rr(b)=1) PC = PC + 2 or 3		
SBIC	P, b	Skip if Bit in I/O Register Cleared	If(I/O(P,b)=0) PC = PC + 2 or 3	None	2/3
SBIS	P, b	Skip if Bit in I/O Register Set	If(I/O(P,b)=1) PC = PC + 2 or 3	None	2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC=PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC=PC+k + 1	None	1/2
BREQ	К	Branch if Equal	if $(Z = 1)$ then PC = PC + k + 1	None	1/2
BRNE	К	Branch if Not Equal	if $(Z = 0)$ then PC = PC + k + 1	None	1/2
BRCS	К	Branch if Carry Set	if (C = 1) then PC = PC + k + 1	None	1/2
BRCC	К	Branch if Carry Cleared	if (C = 0) then PC = PC + k + 1	None	1/2
BRSH	К	Branch if Same or Higher	if (C = 0) then PC = PC + k + 1	None	1/2
BRLO	К	Branch if Lower	if (C = 1) then PC = PC + k + 1	None	1/2
BRMI	К	Branch if Minus	if $(N = 1)$ then PC = PC + k + 1	None	1/2
BRPL	К	Branch if Plus	if $(N = 0)$ then PC = PC + k + 1	None	1/2
BRGE	К	Branch if Greater or Equal, Signed	if (N V= 0) then PC = PC+ $k + 1$	None	1/2
BRLT	К	Branch if Less Than, Signed	if (N V= 1) then PC = PC + $k + 1$	None	1/2
BRHS	К	Branch if Half Carry Flag Set	if (H = 1) then PC = PC + k + 1	None	1/2
BRHC	К	Branch if Half Carry Flag Cleared	if (H = 0) then PC = PC + k + 1	None	1/2
BRTS	К	Branch if T Flag Set	if $(T = 1)$ then PC = PC + k + 1	None	1/2
BRTC	К	Branch if T Flag Cleared	if $(T = 0)$ then PC = PC + k + 1	None	1/2
BRVS	К	Branch if Overflow Flag is Set	if $(V = 1)$ then PC = PC + k + 1	None	1/2
BRVC	К	Branch if Overflow Flag is Cleared	if $(V = 0)$ then PC = PC + k + 1	None	1/2
BRIE	К	Branch if Interrupt Enabled	if (I = 1) then PC = PC + k + 1	None	1/2
BRID	К	Branch if Interrupt Disabled	if (I = 0) then PC = PC + k + 1	None	1/2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Copy Register	Rd = Rr	None	1

LDI	Rd, K	Load Immediate	Rd = K	None	1
LDS	Rd, k	Load Direct	Rd = (k)	None	2
LD	Rd, X	Load Indirect	Rd = (X)	None	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd = (X), X = X + 1	None	2
LD	Rd, -X		X = X - 1, Rd = (X)	None	2
LD	Rd, Y	Load Indirect	Rd = (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd = (Y), Y = Y + 1		2
LD	Rd, -Y		- Y = Y - 1, Rd = (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd = (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd = (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd = (Z), Z = Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre- Decrement		None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd = (Z + q)	None	2
STS	k, Rr	Store Direct	(k) = Rr	None	2
ST	, X, Rr	Store Indirect	(X) = Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X) = Rr, X = X + 1		2
ST	-X, Rr	Store Indirect and Pre- Decrement	 X = X - 1, (X) = Rr	None	2
ST	Y, Rr	Store Indirect	(Y) = Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) = Rr, Y = Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre- Decrement	Y = Y - 1, (Y) = Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) = Rr	None	2
ST	Z, Rr	Store Indirect	(Z) = Rr	None	2
ST	, Z+, Rr	Store Indirect and Post-Increment	(Z) = Rr, Z = Z + 1		2
ST	-Z, Rr	Store Indirect and Pre- Decrement	Z = Z - 1, (Z) = Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) = Rr	None	2
LPM			R0 =(Z)	None	3
IN	Rd, P	In Port	Rd = P	None	1
OUT	P, Rr	Out Port	P = Rr	None	1
PUSH	Rr	Push Register on Stack		None	2
РОР	Rd	Pop Register from Stack	Rd = STACK	None	2
BIT AND BIT- TEST INSTRUCTIONS					
LSL	Rd	Logical Shift Left	Rd(n+1) =Rd(n),	Z,C,N,V,	1

BASCOM Language Fundamentals

JOE	
290	

			Rd(0)= 0,C=Rd (7)	Н	
LSR	Rd	Logical Shift Right	Rd(n) = Rd(n+1), Rd(7) =0, C=Rd (0)	Z,C,N,V	1
ROL	ROL Rd Rotate Lo Carry		e Left Through Rd(0) =C, Rd		1
ROR	Rd	Rotate Right Through Carry	Rd(7) =C,Rd(n) =Rd(n+1),C¬Rd (0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) = Rd(n+1), n=06	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(30) « Rd (74)	None	1
BSET	S	Flag Set	SREG(s) = 1	SREG(s)	1
BCLR	S	Flag Clear	SREG(s) = 0	SREG(s)	1
SBI	P, b	Set Bit in I/O Register		None	2
CBI	P, b	Clear Bit in I/O Register	I/O(P, b) = 0	None	2
BST	Rr, b	Bit Store from Register to T	T = Rr(b)	Т	1
BLD	Rd, b	Bit load from T to Register			1
SEC		Set Carry	C = 1	С	1
CLC		Clear Carry	C = 0	С	1
SEN		Set Negative Flag	N = 1	N	1
CLN		Clear Negative Flag	N = 0	N	1
SEZ		Set Zero Flag	Z = 1	Z Z	1
CLZ		Clear Zero Flag	Zero Flag Z = 0		1
SEI		Global Interrupt Enable	I = 1	I	1
CLI		Disable	I = 0	I	1
SES		Set Signed Test Flag	S = 1	S	1
CLS		Clear Signed Test Flag	S = 0	S	1
SEV		Set Twos Complement Overflow	V = 1	V	1
CLV		Clear Twos Complement Overflow	V = 0	V	1
SET		Set T in SREG	T = 1	Т	1
CLT			T = 0	Т	1
SHE		Set Half Carry Flag in SREG	H = 1	Н	1
		Clear Half Carry Flag in SREG	H = 0	Н	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1

 $\ensuremath{^*}$) Not available in base-line microcontrollers

The Assembler is not case sensitive. The operands have the following forms:

Rd: R0-R31 or R16-R31 (depending on instruction) Rr: R0-R31 b: Constant (0-7) s: Constant (0-7) P: Constant (0-31/63) K: Constant (0-255) k: Constant, value range depending on instruction. q: Constant (0-63)

Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions

5.5 Reserved Words

The following table shows the reserved BASCOM statements or characters.

```
^
!
;
$BAUD, $BAUD1, $BOOT, $CRYSTAL, $DATA, $DBG, $DEFAULT, $END, $EEPROM,
$EXTERNAL, $INCLUDE, $LCD, $LCDRS, $LCDPUTCTRL, $LCDPUTDATA, $LCDVFO,
$LIB,$MAP,$REGFILE,$SERIALINPUT,$SERIALINPUT1, $SERIALINPUT2LCD,
$SERIALOUTPUT, $SERIALOUTPUT1,
$TINY,$WAITSTATE,$XRAMSIZE, $XRAMSTART
```

1WRESET ,1WREAD ,1WWRITE

ACK ,ABS ,ALIAS ,AND ,ACOS ,AS , ASC , ASIN , AT , ATN, ATN2

 $\mathsf{BAUD}, \mathsf{BCD}$, BIN , $\mathsf{BIN2GRAY}$, BINVAL , BIT , $\mathsf{BITWAIT}$, BLINK , $\mathsf{BOOLEAN}$, BYTE , BYVAL

CALL , CAPTURE1 , CASE , CHECKSUM , CHR , CIRCLE , CLS , CLOSE , COMPARE1x , CONFIG , CONST , COS , COSH , COUNTER , COUNTERx , CPEEK , CPEEKH , CRC8 , CRC16 , CRC32 , CRYSTAL , CURSOR

DATA, DATE\$, DBG, DEBOUNCE, DECR, DECLARE, DEFBIT, DEFBYTE, DEFLNG, DEFWORD, DEG2RAD, DEGSNG, DEFLCDCHAR, DEFINT, DEFWORD, DELAY, DIM, DISABLE, DISKSIZE, DISKFREESIZE, DISPLAY, DO, DOUBLE, DOWNTO, DTMFOUT

ELSE, ELSEIF, ENABLE, END, EOF, ERAM, ERASE, ERR, EXIT, EXP, EXTERNAL, FIX, FLUSH, FOR, FOURTH, FOURTHLINE, FREEFILE, FUNCTION

GATE, GET, GETADC, GETKBD, GETATKBD , GETRC5, GLCDDATA , GLCDCMD, GOSUB, GOTO, GRAY2BIN

HEXVAL, HIGH, HOME

I2CINIT, I2CRECEIVE, I2CSEND, I2CSTART, I2CSTOP, I2CRBYTE, I2CWBYTE, IDLE, IF, INCR, INKEY, INP, INPUT, INPUTBIN, INPUTHEX, INT, INTO, INT1, INTEGER, INTERNAL, INSTR, IS, ISCHARWAITING

LCASE, LCD, LCDAT, LEFT, LEFT, LEN, LINE, LOAD, LOADLABEL, LOC , LOF , LOCAL, LOCATE, LOG , LOG10 , LONG, LOOKUP, LOOKUPSTR, LOOP, LTRIM, LOOKDOWN, LOW, LOWER, LOWERLINE

MAKEBCD, MAKEDEC, MAKEINT, MID, MIN, MAX, MOD, MODE

NACK, NEXT, NOBLINK, NOSAVE, NOT

OFF, ON, OR, OUT, OUTPUT

PEEK, POKE, PORTx, POWER, POWERDOWN, PRINT, PRINTBIN, PULSEOUT, PUT, PWM1x, RAD2DEG, RC5SEND, RC6SEND, READ, READEEPROM REM, RESET, RESTORE, RETURN, RIGHT, RIGHT, ROTATE, ROUND, RTRIM

SEEK, SELECT, SERIAL, SET, SERIN, SEROUT, SETFONT, SGN, SHIFT, SHIFTLCD, SHIFTCURSOR, SHIFTIN, SHIFTOUT, SHOWPIC, SHOWPICE, SIN, SINH, SONYSEND, SOUND, SPACE, SPC, SPIINIT, SPIIN, SPIMOVE, SPIOUT, START, STEP, STR, STRING, STOP, SUB, SWAP, SQR

TAN , TANH , THEN , TIME\$, THIRD , THIRDLINE , TIMERx , TO , TRIM , TYPE

UCASE, UNTIL, UPPER, UPPERLINE

VAL, VARPTR

WAIT, WAITKEY, WAITMS , WAITUS , WATCHDOG , WRITEEEPROM , WEND , WHILE , WORD XOR, XRAM

5.6 Error Codes

The following table lists errors that can occur.

Error	Description				
1	Unknown statement				
2	Unknown structure EXIT statement				
3	WHILE expected				
4	No more space for IRAM BIT				
5	No more space for BIT				
6	. expected in filename				
7	IF THEN expected				
8	BASIC source file not found				
9	Maximum 128 aliases allowed				
10	Unknown LCD type				
11	INPUT, OUTPUT, 0 or 1 expected				
12	Unknown CONFIG parameter				
13	CONST already specified				
14	Only IRAM bytes supported				
15	Wrong data type				
16	Unknown Definition				
17	9 parameters expected				
18	BIT only allowed with IRAM or SRAM				
19	STRING length expected (DIM S AS STRING * 12 ,for example)				
20	Unknown DATA TYPE				
21	Out of IRAM space				
22	Out of SRAM space				

1						
23	Out of XRAM space					
24	Out of EPROM space					
25	Variable already dimensioned					
26	AS expected					
27	parameter expected					
28	IF THEN expected					
29	SELECT CASE expected					
30	BIT's are GLOBAL and can not be erased					
31	Invalid data type					
32	Variable not dimensioned					
33	GLOBAL variable can not be ERASED					
34	Invalid number of parameters					
35	3 parameters expected					
36	THEN expected					
37	Invalid comparison operator					
38	Operation not possible on BITS					
39	FOR expected					
40	Variable can not be used with RESET					
41	Variable can not be used with SET					
42	Numeric parameter expected					
43	File not found					
44	2 variables expected					
45	DO expected					
46	Assignment error					
47	UNTIL expected					
50	Value doesn't fit into INTEGER					
51	Value doesn't fit into WORD					
52	Value doesn't fit into LONG					
60	Duplicate label					
61	Label not found					
62	SUB or FUNCTION expected first					
63	Integer or Long expected for ABS()					
64	, expected					
65	device was not OPEN					
66	device already OPENED					
68	channel expected					
70	BAUD rate not possible					
71	Different parameter type passed then declared					
72	Getclass error. This is an internal error.					
73	Printing this FUNCTION not yet supported					
74	3 parameters expected					
80	Code does not fit into target chip					
81	Use HEX(var) instead of PRINTHEX					
82	Use HEX(var) instead of LCDHEX					
85	Unknown interrupt source					
86	Invalid parameter for TIMER configuration					
87	ALIAS already used					
88	0 or 1 expected					

299

89	Out of range : must be 1-4
90	Address out of bounds
91	INPUT, OUTPUT, BINARY, or RANDOM expected
92	LEFT or RIGHT expected
93	Variable not dimensioned
94	Too many bits specified
95	Falling or rising expected for edge
96	Pre scale value must be 1,8,64,256 or 1024
97	SUB or FUNCTION must be DECLARED first
98	SET or RESET expected
99	TYPE expected
100	No array support for IRAM variables
101	Can't find HW-register
102	Error in internal routine
103	= expected
104	LoadReg error
105	StoreBit error
106	Unknown register
107	LoadnumValue error
108	Unknown directive in device file
109	= expected in include file for .EQU
110	Include file not found
111	SUB or FUNCTION not DECLARED
112	SUB/FUNCTION name expected
113	SUB/FUNCTION already DECLARED
114	LOCAL only allowed in SUB or FUNCTION
115	#channel expected
116	Invalid register file
117	Unknown interrupt
126	NEXT expected.
129	(or) missing.
200	.DEF not found
201	Low Pointer register expected
202	.EQU not found, probably using functions that are not supported by the selected chip
203	Error in LD or LDD statement
204	Error in ST or STD statement
205	} expected
206	Library file not found
207	Library file already registered
210	Bit definition not found
211	External routine not found
212	LOW LEVEL, RISING or FALLING expected
213	String expected for assignment
214	Size of XRAM string 0
215	Unknown ASM mnemonic
216	CONST not defined
217	No arrays allowed with BIT/BOOLEAN data type

300 BASCOM-AVR

218	Register must be in range from R16-R31					
219	INTO-INT3 are always low level triggered in the MEGA					
220	Forward jump out of range					
221	Backward jump out of range					
222	Illegal character					
223	* expected					
224	Index out of range					
225	() may not be used with constants					
226	Numeric of string constant expected					
227	SRAM start greater than SRAM end					
228	DATA line must be placed after the END statement					
229	End Sub or End Function expected					
230	You can not write to a PIN register					
231	TO expected					
232	Not supported for the selected micro					
233	READ only works for normal DATA lines, not for EPROM data					
234	') block comment expected first					
235	'(block comment expected first					
236	Value does not fit into byte					
238	Variable is not dimensioned as an array					
239	Invalid code sequence because of AVR hardware bug					
240	END FUNCTION expected					
241	END SUB expected					
242	Source variable does not match the target variable					
243	Bit index out of range for supplied data type					
244	Do not use the Y pointer					
245	No arrays supported with IRAM variable					
246	No more room for .DEF definitions					
247	. expected					
248	BYVAL should be used in declaration					
249	ISR already defined					
250	GOSUB expected					
251	Label must be named SECTIC					
252	Integer or Word expected					
253	ERAM variable can not be used					
254	Variable expected					
255	Z or Z+ expected					
256	Single expected					
257	"" expected					
258	SRAM string expected					
259	- not allowed for a byte					
260	Value larger than string length					
261	Array expected					
262	ON or OFF expected					
263	Array index out of range					
264	Use ECHO OFF and ECHO ON instead					
265	offset expected in LDD or STD like Z+1					
266	TIMER0, TIMER1 or TIMER2 expected					
	, <u> </u>					

301

r					
267	Numeric constant expected				
268	Param must be in range from 0-3				
269	END SELECT expected				
270	Address already occupied				
322	Data type not supported with statement				
323	Label too long				
324	Chip not supported by I2C slave library				
325	Pre-scale value must be 1,8,32,128,256 or 1024				
326	#ENDIF expected				
327	Maximum size is 255				
328	Not valid for SW UART				
329	FileDateTime can only be assigned to a variable				
330	Maximum value for OUT is &H3F				
332	\$END ASM expected				
334	') blockcomment end expected				
335	Use before DIM statements				
336	Could not set specified CLOCK value				
999	DEMO/BETA only supports 4096 bytes of code				
9999	I hope you do not see this one.				

Other error codes are internal ones. Please report them when you get them.

5.7 Newbie problems

When you are using the AVR without knowledge of the architecture you can experience some problems.

- -I can not set a pin high or low
- -I can not read the input on a pin

The AVR has 3 registers for each port. A port normally consists of 8 pins. A port is named with a letter from A-F. All parts have PORTB.

When you want to set a single pin high or low you can use the SET and RESET statements. But before you use them the AVR chip must know in which direction you are going to use the pins.

Therefore there is a register named DDRx for each port. In our sample it is named DDRB. When you write a 0 to the bit position of the pin you can use the pin as an input. When you write a 1 you can use it as output.

After the direction bit is set you must use either the PORTx register to set a logic level or the PINx register to READ a pin level.

Yes the third register is the PINx register. In our sample, PINB.

For example : $DDRB = \&B1111_0000$ ' upper nibble is output, lower nibble is input SET PORTB.7 'will set the MS bit to +5V RESET PORTB.7 'will set MS bit to 0 V To read a pin :

Print PINB.0 'will read LS bit and send it to the RS-232

You may also read from PORTx but it will return the value that was last written to it.

To read or write whole bytes use : PORTB = 0 'write 0 to register making all pins low PRINT PINB 'print input on pins

I want to write a special character but they are not printed correct:

Well this is not a newbie problem but I put it here so you could find it. Some ASCII characters above 127 are interpreted wrong depending on country settings. To print the right value use : PRINT "Test{123}?"

The $\{xxx\}$ will be replaced with the correct ASCII character.

You must use 3 digits otherwise the compiler will think you want to print $\{12\}$ for example. This should be $\{012\}$

My application was working but with a new micro it is slow and print funny

Most new micro's have an internal oscillator that is enabled by default. As it runs on 1 or 4 or 8 MHz, this might be slower or faster then your external crystal. This results in slow operation.

As the baud rate is derived from the clock, it will also result in wrong baud rates.

Solution : change frequency with \$crystal so the internal clock will be used. Or change the fuse bits so the external xtal will be used.

Some bits on Port C are not working

Some chips have a JTAG interface. Disable it with the proper fuse bit .

5.8 Tips and tricks

This section describes tips and tricks received from users.

Kyle Kronyak : Using all the RAM from an external RAM chip.

I have found a way to use the 607 bytes of external SRAM that are normally not available when using hardware SRAM support with BASCOM-AVR. It's actually quite simple. Basically the user just has to disconnect A15 from /CE on the SRAM module, and tie /CE to ground. This makes the chip enabled all the time. Addresses 1-32768 will then be available! The reason is because normally when going above 32768, the A15 pin would go high, disabling the chip. When A15 is not connected to /CE, the chip is always enabled, and allows the address number to "roll over". Therefore address 32162 is actually 0, 32163 is actually 1, 32164 is actually 2, etc. I have only tested this on a 32k SRAM chip. It definitely won't work on a 64k chip, and I believe it already works on any chip below 32k without modification of the circuit.

Programming problems

- When you have unreliable results, use a shielded LPT cable

- The AVR chips have a bug, if the erase is not complete. It tend's to hang at some point. Sometimes although the system reports erased but blank check report "not empty". As per Atmel Data Errata You must drop the vcc by 0.5V (a diode 1N4148 in Series) if the erase is not happening. (Such Chip's are unreliable and hence can be used only if you are sure). This can happen after you have programmed the chip many times.

5.9 **ASCII chart**

Decima	l Octa	al He	x Binary	Value	
000	000	000	00000000	NUL	(Null char.)
001	001	001	00000001	SOH	(Start of Header)
002	002	002	00000010	STX	
003	003	003	00000011	ETX	(End of Text)
004	004	004	00000100	EOT	(End of Transmission)
005	005	005	00000101	ENQ	
006	006	006	00000110	ACK	(Acknowledgment)
007	007	007	00000111	BEL	(Bell)
008	010	008	00001000	BS	(Backspace)
009	011	009	00001001	HT	(Horizontal Tab)
010	012	00A	00001010	LF	(Line Feed)
011	013	00B	00001011	VT	(Vertical Tab)
012	014	00C	00001100	FF	(Form Feed)
013	015	00D	00001101	CR	(Carriage Return)
014	016	00E	00001110	SO	(Shift Out)
015	017	00F	00001111	SI	(Shift In)
016	020	010	00010000	DLE	(Data Link Escape)
017	021	011	00010001		(XON) (Device Control 1)
018	022 023	012 013	00010010	DC2	(Device Control 2) (XOFF)(Device Control 3)
019 020	023	013	$00010011 \\ 00010100$	DC3 DC4	
020	024	014 015	00010100	NAK	
021	025	015	00010101	SYN	
022	020	017	00010110	ETB	(End of Trans. Block)
024	030	018	00011000	CAN	(Cancel)
025	031	019	00011001	EM	(End of Medium)
026	032	01A	00011010	SUB	(Substitute)
027	033	01B	00011011	ESC	(Escape)
028	034	01C	00011100	FS	(File Separator)
029	035	01D	00011101	GS	(Group Separator)
030	036	01E	00011110	RS	(Request to Send)(Record Separator)
031	037	01F	00011111	US	(Unit Separator)
032	040	020	00100000	SP	(Space)
033	041	021	00100001	!	(exclamation mark)
034	042	022	00100010	п	(double quote)
035	043	023	00100011	#	(number sign)
036	044	024	00100100	\$	(dollar sign)
037	045	025	00100101	%	(percent)
038	046	026	00100110	&	(ampersand)
039	047	027	00100111	-	(single quote)
040	050	028	00101000	((left/opening parenthesis)
041	051	029	00101001) *	(right/closing parenthesis)
042	052	02A	00101010		(asterisk)
043	053	02B	00101011	+	(plus)

044	054	02C	00101100	,	(comma)
045	055	02D	00101101	-	(minus or dash)
046	056	02E	00101110		(dot)
047	057	02F	00101111	/	(forward slash)
048	060	030	00110000	0	
049	061	031	00110001	1	
050	062	032	00110010	2	
051	063	033	00110011	3	
052	064	034	00110100	4	
053	065	035	00110101	5	
054	066	036	00110110	6	
055	067	037	00110111	7	
056	070	038	00111000	8	
057	071	039	00111001	9	
058	072	03A	00111010	:	(colon)
059	073	03B	00111011	;	(semi-colon)
060	074	03C	00111100	<	(less than)
061	075	03D	00111101	=	(equal sign)
062	076	03E	00111110	>	(greater than)
063	077	03F	00111111	?	(question mark)
064	100	040	0100000	@	(AT symbol)
065	101	041	01000001	A	
066	102	042	01000010	В	
067	103	043	01000011	C	
068	104	044	01000100	D	
069	105	045	01000101	E	
070	106	046	01000110	F	
071	107	047	01000111	G	
072	110	048	01001000	H	
073	111	049	01001001	I	
074	112	04A	$01001010 \\ 01001011$	J	
075 076	113 114	04B 04C	01001011	K L	
070	115	04C 04D	01001100	L M	
077	115	04D 04E	01001101	N	
078	117	04L 04F	01001110	0	
080	120	050		P	
080	120	051	010100001	Q	
082	122	051	01010001	R	
083	123	052	01010010	S	
084	124	054	01010100	Т	
085	125	055	01010101	Ů	
086	126	056	01010110	v	
087	127	057	01010111	Ŵ	
088	130	058	01011000	Х	
089	131	059	01011001	Ŷ	
090	132	05A	01011010	Z	
091	133	05B	01011011	[(left/opening bracket)
092	134	05C	01011100	Ň	(back slash)
093	135	05D	01011101	j	(right/closing bracket)
094	136	05E	01011110	~	(caret/circumflex)
095	137	05F	01011111	_	(underscore)
096	140	060	01100000	``	-
097	141	061	01100001	а	
098	142	062	01100010	b	
099	143	063	01100011	С	
100	144	064	01100100	d	
101	145	065	01100101	е	

102	146	066	01100110	f	
103	147	067	01100111	g	
104	150	068	01101000	h	
105	151	069	01101001	i	
106	152	06A	01101010	j	
107	153	06B	01101011	k	
108	154	06C	01101100	1	
109	155	06D	01101101	m	
110	156	06E	01101110	n	
111	157	06F	01101111	0	
112	160	070	01110000	р	
113	161	071	01110001	q	
114	162	072	01110010	r	
115	163	073	01110011	S	
116	164	074	01110100	t	
117	165	075	01110101	u	
118	166	076	01110110	v	
119	167	077	01110111	w	
120	170	078	01111000	х	
121	171	079	01111001	У	
122	172	07A	01111010	z	
123	173	07B	01111011	{	(left/opening brace)
124	174	07C	01111100	ľ	(vertical bar)
125	175	07D	01111101	;	(right/closing brace)
126	176	07E	01111110	~	(tilde)
127	177	07F	01111111	DEL	(delete)
	- · ·	•••			(



6 BASCOM Language Reference

6.1 #AUTOCODE

Action

Informs the IDE that code can be maintained by the IDE.

Syntax #AUTOCODE

CONFIG STATEMENTS

#ENDAUTOCODE

Remarks

Auto code informs the IDE that it may alter the code. A new IDE uses a property editor for the configuration. It will only update, add or delete, CONFIG statements that are enclosed in an #AUTOCODE block. #AUTOCODE must be closed with a matching #ENDAUTOCODE

You can still use CONFIG statements in other places of your code. But the property editor will only work on the ones inside the block. The compiler will ignore #AUTOCODE and #ENDAUTOCODE.

6.2 **#IF ELSE ENDIF**

Action

Conditional compilation directives intended for conditional compilation.

Syntax

#IF condition

#ELSE

#ENDIF

Remarks

Conditional compilation is supported by the compiler. What is conditional compilation? Conditional compilation will only compile parts of your code that meet the criteria of the condition.

By default all your code is compiled.

Conditional compilation needs a <u>constant</u> such to test. So before a condition can be tested you need to define a constant.

CONST test = 1 #IF TEST Print "This will be compiled" #ELSE Print "And this not" #ENDIF

Note that there is no THEN and that #ENDIF is not #END IF (no space)

You can nest the conditions and the use of #ELSE is optional.

There are a few internal constants that you can use. These are generated by the compiler:

_CHIP = 0 _RAMSIZE = 128 _ERAMSIZE = 128 _SIM = 0 _XTAL = 4000000 _BUILD = 11162

_CHIP is an integer that specifies the chip, in this case the 2313 _RAMSIZE is the size of the SRAM

ERAMSIZE is the size of the EEPROM

_SIM is set to 1 when the \$SIM directive is used

_XTAL contains the value of the specified crystal

_BUILD is the build number of the compiler.

The build number can be used to write support for statements that are not available in a certain version :

#IF _BUILD >= 11162
s = Log(1.1)
#ELSE
Print "Sorry, implemented in 1.11.6.2"
#ENDIF

Conditional compilation allows you to create different versions of your program but that you keep one source file.

For example you could make a multi lingual program like this :

CONST LANGUAGE=1

'program goes here

#IF LANGUAGE=1 DATA "Hello" #ENDIF #IF LANGUAGE=2 DATA "Guten tag" #ENDIF

By changing the just one constant you then have for example English or German data lines.

Conditional compilation does not work with the \$REGFILE directive. If you put the \$REGFILE inside a condition or not, the compiler will use the first \$REGFILE it encounters. This will be changed in a future version.

A special check was added to 1.11.8.1 to test for existence of constants or variables. #IF varexist("S")

'the variable S was dimensioned so we can use it here

#ELSE

```
'when it was not dimmed and we do need it, we can do it here
DIM S as BYTE
#ENDIF
```

See Also

CONST 594

6.3 \$AESKEY

Action

This directive accepts a 16 byte AES key and informs the compiler to encrypt the binary image.

Syntax

\$AESKEY 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

Remarks

\$AESKEY accepts 16 parameters. These are the 16 bytes which form a 128 bit key. When your code is compiled, the resulting binary code will be encrypted with the provided key.

A boot loader could then use AES and decrypt the binary file before writing to flash memory.

Only the binary image is encrypted, the HEX file is not encrypted! You can not simulate an encrypted program. Add this option when your project is ready.

See also

NONE

Example

See the Samples\boot\xmega_dos_boot_AES.zip , an Xmega boot loader with AES decryption.

6.4 \$ASM

Action

Start of inline assembly code block.

Syntax

\$ASM

Remarks

Use \$ASM together with \$END ASM to insert a block of assembler code in your BASIC code. You can also precede each line with the ! sign. Most ASM mnemonics can be used without the preceding ! too. See also the chapter Mixing BASIC and Assembly 286 and assembler mnemonics 297

Example

Dim C As Byte

Loadadr C , X 'load address of variable C into register X

```
$asm
Ldi R24,1 ; load register R24 with the constant 1
St X,R24 ; store 1 into variable c
$end Asm
Print C
End
```

6.5 \$BAUD

Action

Instruct the compiler to override the baud rate setting from the options menu.

Syntax

\$BAUD = var

Remarks

Var The baud rate that you want to use. This must be a numeric constant.

The baud rate is selectable from the <u>Compiler Settings</u> and the setting from the Compiler Settings from the Compiler Settings.

In the generated report, you can view which baud rate is actually generated. The generated baud rate does depend on the used micro and crystal.

When you simulate a program you will not notice any problems when the baud rate is not set to the value you expected. In real hardware a wrong baud rate can give weird results on the terminal emulator screen. For best results use a crystal that is a multiple of the baud rate.

In the simulator you need to select the UARTO-TAB to view the output of the UARTO, or to send data to this UART.

See also

\$CRYSTAL 316 , BAUD 407

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

```
Print "Hello"
```

```
'Now change the baud rate in a program
Baud = 9600
Print "Did you change the terminal emulator baud rate too?"
End
```

6.6 \$BAUD1

Action

Instruct the compiler to set the baud rate for the second hardware UART.

Syntax

\$BAUD1 = var

Remarks

ľ	Var	The baud rate that you want to use. This must be a
I		numeric constant.

In the generated report, you can view which baud rate is actually generated.

When you simulate a program you will not notice any problems when the baud rate is not set to the value you expected. In real hardware a wrong baud rate can give weird results on the terminal emulator screen. For best results use a crystal that is a multiple of the baud rate.

Some AVR chips have 2 UARTS. For example the Mega161, Mega162, Mega103 and Mega128. There are several other's and some new chips even have 4 UARTS.

In the simulator you need to select the UART1-TAB to view the output of the UART1, or to send data to this UART.

See also

\$CRYSTAL 3161, BAUD 4071, \$BAUD 3101

Example

```
_____
'copyright
                       : (c) 1995-2005, MCS Electronics
'micro
                       : Mega162
                       : yes
'suited for demo
'commercial addon needed : no
                       : demonstrates BAUD1 directive and BAUD1
'purpose
statement
·_____
_____
$regfile = "M162def.dat"
$baud1 = 2400
$crystal= 14000000 ' 14 MHz crystal
Open "COM2:" For BINARY As #1
```

312 BASCOM-AVR

```
Print #1 , "Hello"
'Now change the baud rate in a program
Baud1 = 9600
'
Print #1 , "Did you change the terminal emulator baud rate too?"
Close #1
End
```

6.7 \$BGF

Action

Includes a BASCOM Graphic File.

Syntax

\$BGF "file"

Remarks

file The file name of the BGF file to include.

Use SHOWPIC to display the BGF file. \$BGF only task is to store the picture into the compressed **B**ASCOM **G**raphics **F**ormat(BGF).

See also

SHOWPIC [883], PSET [813], CONFIG GRAPHLCD [506]

Example

(c) 1995-2005 MCS Electronics T6963C graphic display support demo 1_____ _____ 'The connections of the LCD used in this demo 'LCD pin connected to ' 1 GND GND ' 2 GND GND ' 3 +5V +5V '4 -9V -9V potmeter ' 5 /WR PORTC.0 '6 /RD PORTC.1 '7 /CE PORTC.2 ' 8 PORTC.3 C/D '9 NC not '10 RESET PORTC.4conneted '11-18 D0-D7 PA PORTC.5 '19 FS '20 NC not connected \$crystal = 8000000

'First we define that we use a graphic LCD

Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc , Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8 'The dataport is the portname that is connected to the data lines of the LCD 'The controlport is the portname which pins are used to control the lcd

```
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns
'Dim variables (y not used)
Dim X As Byte , Y As Byte
'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT to clear only the text display
' CLS GRAPH to clear only the graphical part
Cursor Off
Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE, COLUMN LINE can be 1-8 and column 0-30
Locate 1 , 1
'Show some text
Lcd "MCS Electronics"
'And some other text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "1234567890123456789012345678901234567890"
Wait 2
Cls Text
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
For X = 0 To 140
                                                              ' set the
   Pset X , 20 , 255
pixel
Next
Wait 2
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Showpic 0 , 0 , Plaatje
Wait 2
Cls Text
                                                             ' clear the
text
End
'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"
'You could insert other picture data here
```

on

6.8 \$BIGSTRINGS

Action

Instruct the compiler to use big strings.

Syntax \$BIGSTRINGS

Remarks

By default each string has a maximum length of 254 bytes. A null character is used to mark the end of a string.

When a longer string is needed, the compiler can not use bytes for passing the length. A word is needed to hold the length.

The \$BIGSTRINGS directive will include the bigstrings.lbx and will handle all string routines different when parameters are passed that has influence on the length.

The alternative library has modified routines for code that is not compatible with big strings.

The library is not tested with all string operations yet. Report problems to support.

See also

Example

\$BIGSTRINGS

6.9 \$BOOT

Action

Instruct the compiler to include boot loader support.

Syntax

\$BOOT = address

Remarks

address The boot loader address.

Some new AVR chips have a special boot section in the upper memory of the flash. By setting some fuse bits you can select the code size of the boot section. The code size also determines the address of the boot loader.

With the boot loader you can reprogram the chip when a certain condition occurs. The sample checks a pin to see if a new program must be loaded. When the pin is low there is a jump to the boot address.

The boot code must always be located at the end of your program. It must be written in ASM since the boot loader may not access the application flash rom. This because otherwise you could overwrite your running code!

The example is written for the M163. You can use the Upload file option of the terminal emulator to upload a new hex file. The terminal emulator must have the same baud rate as the chip. Under Options, Monitor, set the right upload speed and set a monitor delay of 20. Writing the flash take time so after every line a delay must be added while uploading a new file.

The \$BOOT directive is replaced by \$LOADER. \$LOADER works much simpler. \$BOOT is however still supported.

See also

<u>\$LOADER</u> 343

Example

See BOOT.BAS from the samples dir. But better look at the \$LOADER directive.

6.10 \$CRYPT

Action

This directive marks encrypted BASIC code.

Syntax

\$CRYPT data

Remarks

In some cases you might want to share only portions of your code. The IDE can encrypt your code, and the compiler can process this encrypted code. AES encryption is used. You do need a commercial add on to use the encryption. The \$crypt command can be processed by all bascom editions starting from version 2.0.5.0. So you only need an add on when you want to encrypt the code.

Once encrypted, you can NOT DECRYPT into source code! Thus make a BACKUP of your source code before you encrypt the code.

See also Edit Encrypt Selected Code 53

Example

\$CRYPT 6288E522B4A1429A6F16D639BFB7405B \$CRYPT 7ABCF89E7F817EB166E03AFF2EB64C4B \$CRYPT 645C88E996A87BF94D34726AA1B1BCCC \$CRYPT 9405555D91FA3B51DEEC4C2186F09ED1 \$CRYPT 6D4790DA2ADFF09DE0DA97C594C1B074

6.11 \$CRYSTAL

Action

Instruct the compiler to override the crystal frequency options setting.

Syntax

\$CRYSTAL = var

Remarks

	_
var	A numeric constant with the Frequency of the crystal.

The frequency is selectable from the <u>Compiler Settings</u> 2. It is stored in a configuration file. The \$CRYSTAL directive overrides this setting. It is best to use the \$CRYSTAL directive as the used crystal frequency is visible in your program that way.

The \$CRYSTAL directive only informs the compiler about the used frequency. It does not set any fuse bit. The frequency must be know by the compiler for a number of reasons. First when you use serial communications, and you specify <u>\$BAUD</u>[310], the compiler can calculate the proper settings for the UBR register. And second there are a number of routines like <u>WAITMS</u>[350], that use the execution time of a loop to generate a delay. When you specify \$CRYSTAL = 1000000 (1 MHz) but in reality, connect a 4 MHz XTAL, you will see that everything will work 4 times as quick.

Most new AVR chips have an internal oscillator that is enabled by default. Check the data sheet for the default value.

See also

\$BAUD 310, BAUD 407, CONFIG CLOCKDIV 465

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Coml = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Print "Hello world"
```

```
End
```

6.12 \$DATA

Action

Instruct the compiler to store the data in the DATA lines following the \$DATA directive, in code memory.



Remarks

The AVR has built-in EEPROM. With the WRITEEEPROM and READEEPROM statements, you can write to and read from the EEPROM.

To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM.

A separate file is generated with the EEP extension. This file can be used to program the EEPROM.

The compiler must know which DATA must go into the code memory and which into the EEPROM memory and therefore two compiler directives were added.

\$EEPROM and \$DATA.

\$EEPROM tells the compiler that the DATA lines following the compiler directive must be stored in the EEP file.

To switch back to the default behavior of the DATA lines, you must use the \$DATA directive.

The READ statement that is used to read the DATA info may only be used with normal DATA lines. It does not work with DATA stored in EEPROM.

Do not confuse \$DATA directive with the DATA statement.

So while normal DATA lines will store the specified data into the code memory of the micro which is called the flash memory, the \$EEPROM and \$DATA will cause the data to be stored into the EEPROM. The EEP file is a binary file.

See also

\$EEPROM 321, READEEPROM 830, WRITEEEPROM 954, DATA 612

ASM

NONE

Example

'copyright 'micro 'suited for demo 'commercial addon needed 'purpose	 (c) 1995-2005, MCS Electronics AT90S2313 yes no demonstrates \$DATA directive 	
<pre>\$regfile = "2313def.dat" \$baud = 19200 \$crystal = 4000000 crystal</pre>		' 4 MHz
Dim B As Byte Readeeprom B , 0 be 1 End		'now B will

318 BASCOM-AVR

```
Dta:
$eeprom
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
$data
End
```

6.13 \$DBG

Action

Enables debugging output to the hardware UART.

Syntax ^{\$DBG}

Remarks

Calculating the hardware, software and frame space can be a difficult task. With \$DBG the compiler will insert characters for the various spaces.

To the Frame space 'F' will be written. When you have a frame size of 4, FFFF will be written.

To the Hardware space 'H' will be written. If you have a hardware stack space of 8, HHHHHHHH will be written to this space.

To the software space 'S' will be written. If you have a software stack space of 6, SSSSSS will be written.

The idea is that when a character is overwritten, it is being used. So by watching these spaces you can determine if the space is used or not.

With the DBG statement a record is written to the HW UART. The record must be logged to a file so it can be analyzed by the stack analyzer.

Make the following steps to determine the proper values:

- Make the frame space 40, the soft stack 20 and the HW stack 50
- Add \$DBG to the top of your program
- Add a DBG statement to every Subroutine or Function
- Open the terminal emulator and open a new log file. By default it will have the name of your current program with the .log extension
- Run your program and notice that it will dump information to the terminal emulator
- When your program has executed all sub modules or options you have build in, turn off the file logging and turn off the program
- Choose the Tools Stack analyzer option
- A window will be shown with the data from the log file
- Press the Advise button that will determine the needed space. Make sure that there is at least one H, S and F in the data. Otherwise it means that all the data is overwritten and that you need to increase the size.
- Press the Use button to use the advised settings.

As an alternative you can watch the space in the simulator and determine if the characters are overwritten or not.

The DBG statement will assign an internal variable named ____SUBROUTINE Because the name of a SUB or Function may be 32 long, this variable uses 33 bytes!

____SUBROUTINE will be assigned with the name of the current SUB or FUNCTION.

When you first run a SUB named Test1234 it will be assigned with Test1234 When the next DBG statement is in a SUB named Test, it will be assigned with Test. The 234 will still be there so it will be shown in the log file.

Sub	FS	SS	HS	Frame space	Soft stack	HW stack	
TEST	1	4	4	а	SSSSÓxÒx	¥x x	
TEST	1	4	4	I	SSSSÓxÒx	¥xjx	
TEST	1	4	4	a	SSSSÓxÔx	¥x x	
TEST	1	4	4	1	SSSSÓxÒx	¥xjx	
TEST	1	4	4	а	SSSSÓxÒx	¥x x	
TEST	1	4	4	1	SSSSÓxÒx	¥xjx	
TEST	1	4	4	a	SSSSÓxÒx	¥×l×	
TEST	1	4	4	1	SSSSÓxÒx	¥xjx	
TEST	1	4	4	а	SSSSÓxÒx	¥x i x	
							_
							_

Every DBG record will be shown as a row. The columns are:

Column	Description
Sub	Name of the sub or function from where the DBG was used
	Used frame space
	Used software stack space
HS	Used hardware stack space
Frame space	Frame space
Soft stack	Soft stack space
HW stack	Hardware stack space

The Frame space is used to store temp and local variables. It also stores the variables that are passed to subs/functions by value. Because PRINT, INPUT and the FP num<>String conversion routines require a buffer, the compiler always is using 24 bytes of frame space.

When the advise is to use 2 bytes of frame space, the setting will be 24+2=26.

For example when you use : print var, var need to be converted into a string before it can be printed or shown with LCD.

An alternative for the buffer would be to setup a temp buffer and free it once finished. This gives more code overhead.

In older version of BASCOM the start of the frame was used for the buffer but that gave conflicts when variables were printed from an ISR.

See also

DBG 637

6.14 \$DEFAULT

Action

Set the default for data types dimensioning to the specified type.

Syntax

\$DEFAULT var

Remarks

Var SRAM, XRAM, ERAM

Each variable that is dimensioned will be stored into SRAM, the internal memory of the chip. You can override it by specifying the data type. Dim B As XRAM Byte , will store the data into external memory.

When you want all your variables to be stored in XRAM for example, you can use the statement : \$DEFAULT XRAM Each Dim statement will place the variable in XRAM in that case.

To switch back to the default behavior, use \$END \$DEFAULT

See also

NONE

ASM

NONE

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

\$default Xram
Dim A As Byte , B As Byte , C As Byte
'a,b and c will be stored into XRAM

\$default Sram
Dim D As Byte
'D will be stored in internal memory, SRAM

6.15 \$EEPLEAVE

Action

Instructs the compiler not to recreate or erase the EEP file.

Syntax

\$EEPLEAVE

Remarks

When you want to store data in the EEPROM, and you use an external tool to create the EEP file, you can use the \$EEPLEAVE directive.

Normally the EEP file will be created or erased, but this directive will not touch any existing EEP file.

Otherwise you would erase an existing EEP file, created with another tool.

See also

\$EEPROMHEX 322

Example

NONE

6.16 \$EEPROM

Action

Instruct the compiler to store the data in the DATA lines following the \$EEPROM directive in an EEP file.

Syntax

\$EEPROM

Remarks

The AVR has built-in EEPROM. With the WRITEEEPROM and READEEPROM statements, you can write to and read from the EEPROM.

To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM.

A separate file is generated with the EEP extension. This file can be used to program the EEPROM.

The compiler must know which DATA must go into the code memory and which into the EEPROM memory and therefore two compiler directives were added.

\$EEPROM and \$DATA.

\$EEPROM tells the compiler that the DATA lines following the compiler directive must be stored in the EEP file.

To switch back to the default behavior of the DATA lines, you must use the \$DATA directive.

The READ statement that is used to read the DATA info may only be used with normal

DATA lines. It does not work with DATA stored in EEPROM.

Do not confuse \$DATA directive with the DATA statement.

So while normal DATA lines will store the specified data into the code memory of the micro which is called the flash memory, the $\frac{EEPROM}{32h}$ and DATA will cause the data to be stored into the EEPROM. The EEP file is a binary file. The $\frac{EEPROMHEX}{322}$ directive can be used to create Intel HEX records in the EEP file

See also

\$EEPROM 321, READEEPROM 830, WRITEEEPROM 954, DATA 612, \$EEPROMHEX 322

ASM

NONE

Example

p		
'micro 'suited for demo 'commercial addon needed 'purpose	-	
<pre>\$regfile = "2313def.dat" \$baud = 19200 \$crystal = 4000000 crystal</pre>		' 4 MHz
Dim B As Byte Readeeprom B , 0 be 1 End		'now B will
Dta: \$eeprom Data 1 , 2 , 3 , 4 , 5 , \$data End	6 , 7 , 8	

6.17 **\$EEPROMHEX**

Action

Instruct the compiler to store the data in the EEP file in Intel HEX format instead of binary format.

Syntax \$EEPROMHEX

Remarks

The AVR has build in EEPROM. With the WRITEEEPROM and READEEPROM statements, you can write and read to the EEPROM.

To store information in the EEPROM, you can add DATA lines to your program that hold the data that must be stored in the EEPROM. \$EEPROM must be used to create a EEP file that holds the data.

The EEP file is by default a binary file. When you use the STK500 you need an Intel HEX file. Use \$EEPROMHEX to create an Intel Hex EEP file.

\$EEPROMHEX must be used together with \$EEPROM.

See also

<u>\$EEPROMLEAVE</u> 321

Example

```
$eeprom the following DATA lines data will go to the EEP file
Data 200 , 100,50
$data
```

This would create an EEP file of 3 bytes. With the values 200,100 and 50. Add \$eepromhex in order to create an Intel Hex file.

This is how the EEP file content looks when using \$eepromhex

:0A0000001020304050A141E283251 :00000001FF

6.18 \$EEPROMSIZE

Action

Instruct the compiler to override the EEPROM size of the micro processor.

Syntax

\$EEPROMSIZE = size

size

The size in bytes of the EEPROM.

Remarks

The AVR has build in EEPROM. With the WRITEEEPROM and READEEPROM statements, you can write and read to the EEPROM. You can also use the ERAM pseudo variables to read/write EEPROM.

When you use an external EEPROM and an alternative EEPROM library such as FM24C16 or FM25C256 you can override the internal EEPROM. All EEPROM routines will use the external EEPROM then. This way you are able to use a bigger EEPROM

324 BASCOM-AVR

than internal available. Or you can use a quicker EEPROM such as a RAMTRON FRAM EEPROM. These EEPROM's are as quick as SRAM and also can be written to almost unlimited times.

When using an external EEPROM and \$EEPROMSIZE , take care that the supported programmers can not write to this EEPROM. They assume the internal EEPROM.

See also FM24C16 (965), FM25C256 (965)

Example Seepromsize = &H8000

6.19 \$EXTERNAL

Action

Instruct the compiler to include ASM routines from a library.

Syntax

\$EXTERNAL Myroutine [, myroutine2]

Remarks

You can place ASM routines in a library file. With the \$EXTERNAL directive you tell the compiler which routines must be included in your program.

See also

\$LIB 340

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'In order to let this work you must put the mylib.lib file in the LIB
dir
'And compile it to a LBX
                          _____
1 - - - -
'define the used library
$lib"mylib.lbx"
'you can also use the original ASM :
'$LIB "mylib.LIB"
'also define the used routines
$external Test
```

'this is needed so the parameters will be placed correct on the stack **Declare Sub** Test(**byval** X Asbyte , Y Asbyte)

'reserve some space Dim Z As Byte

'call our own sub routine
Call Test(1 , Z)

'z will be 2 in the used example ${\bf End}$

6.20 \$FILE

Action

Change name of generated files.

Syntax

\$FILE = "myname.bin"

Remarks

In some cases it is desired to change the name of the output file. By default, the generated files have the same base name as the opened project file. So if your program name is "mytest.bas", all generated files will start with the base "mytest". The \$FILE directive let you change this base name.

Simulating and programming will NOT work since the IDE uses the base name of your project. If you change it with \$FILE, the files can not be located.

See also

NONE

Example

\$FILE = "mytest.bin"

6.21 \$FRAMESIZE

Action

Sets the available space for the frame.

Syntax

\$FRAMESIZE = var

Remarks

Var A numeric decimal value.

While you can configure the Frame Size in Options, Compiler, Chip, it is good practice to put the value into your code. This way you do no need the cfg(configuration) file.

The \$FRAMESIZE directive overrides the value from the IDE Options.

It is important that the \$FRAMESIZE directive occurs in your main project file. It may not be included in an \$include file as only the main file is parsed for \$FRAMESIZE.

You need a minimum frame size of 24 bytes. This space is used by a number of routines. For example string<>numeric conversion routines. If you use Print numVar, then the numeric variable "numvar" is converted into a string representation of the binary number. The framespace buffer is used for that. While the framespace server as dynamic memory, a fixed address is used. For this reason the buffer has a fixed size of 24 bytes.

See also

\$SWSTACK 365, \$HWSTACK 326

Example

1 _ _ _ _ _ _ _ _

_____ 'name : adc.bas 'copyright : (c) 1995-2005, MCS Electronics : demonstration of GETADC() function for 8535 'purpose or M163 micro : Mega163 'micro 'suited for demo : yes 'commercial addon needed : no : possible 'use in simulator ' Getadc() will also work for other AVR chips that have an ADC converter *_____ _____ \$regfile = "m163def.dat" ' we use the M163 \$crystal = 4000000\$hwstack = 32' default use 32 for the hardware stack \$swstack = 10'default use 10 for the SW stack

\$framesize = 40 40 for the frame space

6.22 \$HWSTACK

Action

Sets the available space for the Hardware stack.

Syntax

\$HWSTACK = var

Remarks

'default use

Var A numeric decimal value.

While you can configure the HW Stack in Options, Compiler, Chip, it is good practice to put the value into your code. This way you do no need the cfg(configuration) file.

The \$HWSTACK directive overrides the value from the IDE Options.

It is important that the \$HWSTACK directive occurs in your main project file. It may not be included in an \$include file as only the main file is parsed for \$HWSTACK.

The Hardware stack is room in RAM that is needed by your program. When you use GOSUB label, the microprocessor pushes the return address on the hardware stack and will use 2 bytes for that. When you use RETURN, the HW stack is popped back and the program can continue at the proper address. When you nest GOSUB, CALL or functions, you will use more stack space. Most statements use HW stack because a machine language routine is called.

See also

<u>\$SWSTACK</u> अब्जे , <u>\$FRAMESIZE</u> अब्जे

Example

: adc.bas 'name : (c) 1995-2005, MCS Electronics 'copyright 'purpose : demonstration of GETADC() function for 8535 or M163 micro Suited for demo : Veg 'micro 'commercial addon needed : no 'use in simulator : possible ' Getadc() will also work for other AVR chips that have an ADC converter ·-----\$regfile = "m163def.dat" ' we use the M163 \$crystal = 4000000\$hwstack = 32' default use 32 for the hardware stack \$swstack = 10 'default use 10 for the SW stack **\$framesize** = 40 'default use 40 for the frame space

6.23 \$HWCHECK, \$SWCHECK, \$SOFTCHECK

Action

This directive can be used to determine the required stack space.

Syntax \$HWCHECK \$FRAMECHECK \$SOFTCHECK

Remarks

All variables you DIM in your application require RAM or SRAM space. But an application needs more RAM space.

Each time you call a sub or function, or us gosub, the processor need to know at which address to return after returning from the call. For this purpose, the processor saves this address on the hardware stack. There is noting you can do about this. This hardware stack grows downwards. Some basic statements compile into code that do not need any calls. But some call a machine language function which in turn can call other functions. Which and how many other calls will be made depend on the selected processor and other options. sometimes it also depends on variable parameters.

When parameters are passed to a sub or function, the address is passed of the variables. These are word addresses thus using 2 bytes for each variable. This passing is being done via the so called soft stack. This area is located below the HW stack space. And it also grows down.

All LOCAL variables you use also need 2 bytes of the soft stack.

When you pass a parameter with BYVAL or when you create a LOCAL variable, some temporarily space is need.

Consider this example : somestring = "abc" + somestring

When the compiler assigns "abc" to somestring, the somestring variable will become "abc" and it will overwrite the content making it impossible to add it's content after the "abc".

So we first need to store the content of somestring before we can start assigning new data to this string.

This copy also requires space.

This space is created dynamically and is taken from the so called frame space. This space is located below the soft stack.

Now you can use \$DBG or some default values for most projects to determine the values.

But when you have a problem and have absolutely no idea how the settings must be made, you can use the \$HWCHECK option.

You start with including a special library named "stackcheck.lib" to your code. Then you run your application and somewhere in your code you print the value of the generated **_hw_lowest** variable.

This variable is set to &HFFFF and each time a call is made, the stack is compared to this value. If the hardware stack (SPL and SPH registers) are lower then the

_hw_lowest value, _hw_lowest is assigned with the new lowest stack value. This way you determine the lowest possible hardware stack value that occurred during the runtime of your application.

Of course it is important that your application runs all code.

You can print the value or show it on LCD. To determine the actual needed space you subtract it from the stacktop value.

For the softstack the same applies. It will store the lowest Y-pointer value to the variable named _sw_lowest.

For the framespace the the variable _fw_highest is used and this variables is increasing.

The stackcheck.bas example demonstrates how to retrieve the values when a recursive sub is used.

See also

NONE

```
Example
$regfile = "m88def.dat"
\$hwstack = 40
\$swstack = 80
$framesize = 80
$lib "stackcheck.lib"
Declare Sub Test(byval Prm As Byte)
Print "stack test"
Dim G As Byte , W As Word
Dim P As Byte
$hwcheck
                                                                'hw stack
check on
$framecheck
$softcheck
Test P
Print _hw_lowest
W = _hwstackstart - _hw_lowest
Print "HW stack needed : " ; W
Print _fw_highest
If _fw_highest > 0 Then
   W = _frame_high - _fw_highest
   Print "Frame space needed : " ; W
End If
Print _sw_lowest
W = _hwstack_low - _sw_lowest
Print "SW stack needed : " ; W
End
Sub Test(byval Prm As Byte)
   Local L As Byte
  Print "HWSTACK:" ; hw lowest
  Print "Frame:" ; _fw_highest
Print "SWSTACK:" ; _sw_lowest
   G = G + 1
                                                                ' global var
   If G >= 5 Then
     Exit Sub
   Else
     Test P
                                                                'recursive
call
  End If
End Sub
```

6.24 \$INC

Action

Includes a binary file in the program at the current position.

Syntax

\$INC label , size | nosize , "file"

Remarks

Label	The name of the label you can use to refer to the data.
	Specify either nosize or size. When you use size, the size of the data will be included. This way you know how many bytes you can retrieve.
File	Name of the file which must be included.

Use RESTORE to get a pointer to the data. And use READ, to read in the data.

The \$INC statement is an alternative for the DATA statement. While DATA works ok for little data, it is harder to use on large sets of data.

See Also

RESTORE 841, DATA 612, READ 828

Example

Restore L1

Read Size

pointer to label

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

Dim Size As Word , W As Word , B As Byte

'set

' get size

of the data
Print Size ; " bytes stored at label L1"
For W = 1 To Size
 Read B : Print Chr(b);
Next

End

'include some data here
\$inc L1 , Size , "c:\test.bas"
'when you get an error, insert a file you have on your system

6.25 \$INCLUDE

Action

Includes an ASCII file in the program at the current position.

Syntax

\$INCLUDE "file"

Remarks

File Name of the ASCII file, which must contain valid BASCOM statements. This option can be used if you make use of the same routines in many programs. You can write modules and include them into your program. If there are changes to make you only have to change the module file, not all your BASCOM programs. You can only include ASCII files!

Use \$INC when you want to include binary files.

See Also

<u>\$INC</u> 330

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

·_____

```
Print "INCLUDE.BAS"
'Note that the file 123.bas contains an error
$include "123.bas" 'include file that prints
Hello
Print "Back in INCLUDE.BAS"
End
```

6.26 \$INITMICRO

Action

Calls a user routine at startup to perform important initialization functions such as setting ports.

Syntax \$INITMICRO

Remarks

This directive will call a label named _INIT_MICRO just after the most important initialization is performed. You can put the _INIT_MICRO routine into your program, or you can put it in a library. Advantage of a library is that it is the same for all programs, and advantage of storing the code into your program is that you can change it for every program.

It is important that you end the routine with a RETURN as the label is called and expects a return.

332 BASCOM-AVR

The \$initmicro can be used to set a port direction or value as it performs before the memory is cleared which can take some mS.

The best solution for a defined logic level at startup remains the usage of pull up/pull down resistors.

See Also

NONE

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

\$initmicro

Print Portb

Do nop Loop End

Print Version()
and time of compilation

'show date

```
'do not write a complete application in this routine.
'only perform needed init functions
_init_micro:
   Config Portb = Output
   Portb = 3
Return
```

6.27 \$LCD

Action

Instruct the compiler to generate code for 8-bit LCD displays attached to the data bus.

Syntax

\$LCD = [&H]address

Remarks

Address	The address where must be written to, to enable the LCD display and the RS line of the LCD display.
	The db0-db7 lines of the LCD must be connected to the data lines D0- D7. (or is 4 bit mode, connect only D4-D7) The RS line of the LCD can be configured with the LCDRS statement.
	On systems with external RAM, it makes more sense to attach the LCD to the data bus. With an address decoder, you can select the LCD display.

Do not confuse \$LCD with the LCD statement. The compiler will create a constant named ____LCD_ADR which you could use in an alternative LCD library.

```
$LCDRS 337, CONFIG LCD 520
Example
                    _____
              (c) 1995-2005 MCS Electronics
     _____
' file: LCD.BAS
 demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
       CURSOR, DISPLAY
      _____
                      _____
'note : tested in bus mode with 4-bit on the STK200
'LCD - STK200
1 _ _ _ _ _
     _____
     D4
'D4
'D5
          D5
'D6
          D6
'D7
          D7
         WR
'WR
' E
          E
'RS
         RS
'+5V
          +5V
' GND
          GND
'V0
          V0
D0-D3 are not connected since 4 bit bus mode is used!
'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Rem with the config lcdpin statement you can override the compiler
settings
$regfile = "8515def.dat"
$lcd = &HC000
$lcdrs = &H8000
Config Lcdbus = 4
Dim A As Byte
Config Lcd = 16 * 2
                                                      'configure
lcd screen
'other options are 16 * 2 , 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines
'$LCD = address will turn LCD into 8-bit databus mode
 use this with uP with external RAM and/or ROM
τ.
      because it aint need the port pins !
Cls
                                                      'clear the
LCD display
Lcd "Hello world."
                                                      'display
this at the top line
Wait 1
Lowerline
                                                      'select the
```

See also

334 BASCOM-AVR

lower line Wait 1 Lcd "Shift this." 'display this at the lower line Wait 1 **For** A = 1 **To** 10 Shiftlcd Right 'shift the text to the right Wait 1 'wait a moment Next **For** A = 1 **To** 10 Shiftlcd Left 'shift the text to the left Wait 1 'wait a moment Next Locate 2 , 1 'set cursor position Lcd "*" 'display this Wait 1 'wait a moment 'shift the Shiftcursor Right cursor Lcd "@" 'display this Wait 1 'wait a moment Home Upper 'select line 1 and return home Lcd "Replaced." 'replace the text Wait 1 'wait a moment Cursor Off Noblink 'hide cursor 'wait a Wait 1 moment Cursor On Blink 'show cursor Wait 1 'wait a moment Display Off 'turn display off Wait 1 'wait a moment Display On 'turn display on -----NEW support for 4-line LCD-----Thirdline Lcd "Line 3" Fourthline Lcd "Line 4" Home Third 'goto home on line three Home Fourth Home F 'first letteer also works Locate 4 , 1 : Lcd "Line 4" Wait 1

```
'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line
Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240
replace ? with number (0-7)
Cls
                                                            'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0); Chr(1)
                                                            'print the
special character
         ----- Now use an internal routine ------
temp1 = 1
                                                            'value into
ACC
                                                            'put it on
!rCall _write_lcd
LCD
End
```

6.28 \$LCDPUTCTRL

Action

Specifies that LCD control output must be redirected.

Syntax

\$LCDPUTCTRL = label

Remarks

Label	The name of the assembler routine that must be called when a control
	byte is printed with the LCD statement. The character must be placed in
	register R24.

With the redirection of the LCD statement, you can use your own routines.

See also

\$LCDPUTDATA 336

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Coml = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'dimension used variables
Dim S As String* 10
Dim W As Long
'inform the compiler which routine must be called to get serial
```

```
336
       BASCOM-AVR
    'characters
    $lcdputdata= Myoutput
    $lcdputctrl= Myoutputctrl
    'make a never ending loop
    Do
      Lcd "test"
    Loop
    End
    'custom character handling routine
    'instead of saving and restoring only the used registers
    'and write full ASM code, we use Pushall and PopAll to save and 'restore
    'all registers so we can use all BASIC statements
    '$LCDPUTDATA requires that the character is passed in R24
    Myoutput:
      Pushall
                                                                   'save all
    registers
       'your code here
      Popall
                                                                   'restore
    registers
    Return
    MyoutputCtrl:
      Pushall
                                                                   'save all
    registers
       'your code here
      Popall
                                                                   'restore
    registers
    Return
```

6.29 \$LCDPUTDATA

Action

Specifies that LCD data output must be redirected.

Syntax

\$LCDPUTDATA = label

Remarks

Label The name of the assembler routine that must be called when a character is printed with the LCD statement. The character must be placed in R24.

With the redirection of the LCD statement, you can use your own routines.

See also

\$LCDPUTCTRL 335

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

```
'dimension used variables
Dim S As String* 10
Dim W As Long
'inform the compiler which routine must be called to get serial
'characters
$lcdputdata= Myoutput
$lcdputctrl= Myoutputctrl
'make a never ending loop
Do
 Lcd "test"
Loop
End
'custom character handling routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and 'restore
'all registers so we can use all BASIC statements
'$LCDPUTDATA requires that the character is passed in R24
Myoutput:
 Pushall
                                                              'save all
registers
  'your code here
 Popall
                                                              'restore
registers
Return
MyoutputCtrl:
 Pushall
                                                              'save all
registers
  'your code here
 Popall
                                                              'restore
registers
Return
```

6.30 \$LCDRS

Action

Instruct the compiler to generate code for 8-bit LCD displays attached to the data bus.

Syntax

\$LCDRS = [&H]address

Remarks

Address	The address where must be written to, to enable the LCD display.
	The db0-db7 lines of the LCD must be connected to the data lines D0-D7. (or is 4 bit mode, connect only D4-D7)
	On systems with external RAM, it makes more sense to attach the LCD to the data bus. With an address decoder, you can select the LCD display.

The compiler will create a constant named ____LCDRS_ADR which you could use in an alternative LCD library.

See also

\$LCD 3327, CONFIG LCDBUS 5207

Example

_____ (c) 1995-2005 MCS Electronics !_____ ' file: LCD.BAS demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME CURSOR, DISPLAY ·_____. _____ 'note : tested in bus mode with 4-bit on the STK200 'LCD – STK200 ·_____ 'D4 D4 'D5 D5 'D6 D6 D7 דםי WR 'WR ' E E RS 'RS '+5V +5V ' GND GND 'V0 V0 D0-D3 are not connected since 4 bit bus mode is used! 'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 , Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6 Rem with the config lcdpin statement you can override the compiler settings \$regfile = "8515def.dat" **\$lcd** = &HC000 **\$lcdrs** = &H8000 **Config** Lcdbus = 4 Dim A As Byte Config Lcd = 16 * 2'configure lcd screen 'other options are 16 * 2 , 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a 'When you dont include this option 16 * 2 is assumed '16 * 1a is intended for 16 character displays with split addresses over 2 lines '\$LCD = address will turn LCD into 8-bit databus mode use this with uP with external RAM and/or ROM 1 because it aint need the port pins ! Cls 'clear the LCD display Lcd "Hello world." 'display this at the top line Wait 1 Lowerline 'select the lower line Wait 1 Lcd "Shift this." 'display this at the lower line

Wait 1 **For** A = 1 **To** 10 Shiftlcd Right 'shift the text to the right 'wait a Wait 1 moment Next **For** A = 1 **To** 10 Shiftlcd Left 'shift the text to the left Wait 1 'wait a moment Next Locate 2 , 1 'set cursor position Lcd "*" 'display this Wait 1 'wait a moment 'shift the Shiftcursor Right cursor Lcd "@" 'display this Wait 1 'wait a moment 'select line Home Upper 1 and return home Lcd "Replaced." 'replace the text Wait 1 'wait a moment Cursor Off Noblink 'hide cursor Wait 1 'wait a moment Cursor On Blink 'show cursor Wait 1 'wait a moment Display Off 'turn display off Wait 1 'wait a moment 'turn Display On display on '-----NEW support for 4-line LCD-----Thirdline Lcd "Line 3" Fourthline Lcd "Line 4" Home Third 'goto home on line three Home Fourth Home F 'first letteer also works Locate 4 , 1 : Lcd "Line 4" Wait 1 'Now lets build a special character 'the first number is the characternumber (0-7)'The other numbers are the rowvalues

340 BASCOM-AVR

'Use the LCD tool to insert this line Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228 replace ? with number (0-7)Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240 replace ? with number (0-7)'select data Cls RAM Rem it is important that a CLS is following the deflcdchar statements because it will set the controller back in datamode Lcd Chr(0); Chr(1)'print the special character '----- Now use an internal routine ------'value into temp1 = 1 ACC !rCall _write_lcd 'put it on LCD End

6.31 \$LCDVFO

Action

Instruct the compiler to generate very short Enable pulse for VFO displays.

Syntax

\$LCDVFO

Remarks

VFO based displays need a very short Enable pulse. Normal LCD displays need a longer pulse. To support VFO displays this compiler directive has been added.

The display need to be instruction compatible with normal HD44780 based text displays.

Noritake is the biggest manufacturer of VFO displays.

The \$LCDVFO directive is intended to be used in combination with the LCD routines.

ASM

NONE

See also

Example

NONE

6.32 \$LIB

Action

Informs the compiler about the used libraries.

Syntax

\$LIB "libname1" [, "libname2"]

Remarks

Libname1 is the name of the library that holds ASM routines that are used by your program. More filenames can be specified by separating the names by a comma.

The specified libraries will be searched when you specify the routines to use with the \$EXTERNAL directive.

The search order is the same as the order you specify the library names.

The MCS.LBX will be searched last and is always included so you don't need to specify it with the \$LIB directive.

Because the MCS.LBX is searched last you can include duplicate routines in your own library. These routines will be used instead of the ones from the default MCS.LBX library. This is a good way when you want to enhance the MCS.LBX routines. Just copy the MCS.LIB to a new file and make the changes in this new file. When we make changes to the library your changes will be preserved.

Creating your own LIB file

A library file is a simple ASCII file. It can be created with the BASCOM editor, notepad or any other ASCII editor.

When you use BASCOM, make sure that the LIB extension is added to the Options, Environment, Editor, "No reformat extension".

This will prevent the editor to reformat the LIB file when you open it.

The file must include the following header information. It is not used yet but will be later.

copyright = Your name www = optional location where people can find the latest source email = your email address comment = AVR compiler library libversion = the version of the library in the format : 1.00

date = date of last modification

statement = A statement with copyright and usage information

The routine must start with the name in brackets and must end with the [END].

The following ASM routine example is from the MYLIB.LIB library.

[test] Test: Idd r26,y+2 ; load address of X Idd r27,y+3 Id r24,x ; get value into r24 Inc r24 ; value + 1 St x,r24 ; put back Idd r26,y+0 ; address of Y Idd r27,y+1 st x,r24 ; store ret ; ready [END]

After you have saved your library in the **LIB** subdirectory you must compile it with the <u>LIB Manager</u> [81⁻]. Or you can include it with the LIB extension in which case you don't have to compile it.

About the assembler.

When you reference constants that are declared in your basic program you need to put a star(*) before the line.

'basic program CONST myconst = 7

'asm lib * sbi portb, myconst

By adding the *, the line will be compiled when the basic program is compiled. It will not be changed into object code in the LBX file. When you use constants you need to use valid BASIC constants:

Ldi r24,12 Ldi r24, 1+1 Ldi r24, &B001 Ldi r24,0b001 Ldi r24,&HFF Ldi r24,\$FF Ldi r24,0xFF

Other syntax is NOT supported.

See also

\$EXTERNAL 324

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'In order to let this work you must put the mylib.lib file in the LIB
dir
'And compile it to a LBX
'-----
'define the used library
$lib"mylib.lbx"
'you can also use the original ASM :
'$LIB "mylib.LIB"
```

```
'also define the used routines
$external Test
'this is needed so the parameters will be placed correct on the stack
Declare Sub Test(byval X Asbyte , Y Asbyte)
'reserve some space
Dim Z As Byte
'call our own sub routine
Call Test(1 , Z)
'z will be 2 in the used example
End
```

6.33 \$LOADER

Action

Instruct the compiler to create a boot loader at the specified address.

Syntax

\$LOADER = address

<u>Remarks</u>

address	The address where the boot loader is located. You can find this address
	in the data sheet.

Most AVR chips have a so called boot section. Normally a chip will start at address 0 when it resets. This is also called the reset vector.

Chips that have a boot section, split the flash memory in two parts. The boot section is a small part of the normal flash and by setting a fuse bit you select that the chip runs code at the boot sector when it resets instead of the normal reset vector. Some chips also have fuse bits to select the size of the boot loader.

The MCS boot loader sample is a serial boot loader that uses the serial port. It uses the X-modem checksum protocol to receive the data. Most terminal emulators can send X-modem checksum.

The sample is written so it supports all chips with a boot section. You need to do the following :

- identify the \$regfile directive for your chip
- un-remark the line and the line with the CONST that is used for conditional compilation
- remark all other \$regfile lines and CONST lines.
- compile the file
- program the chip
- set the fuse bit so reset is pointed to the boot loader
- set the fuse bit so the boot size is 1024 words
- select the MCS Boot loader programmer.

The boot loader is written to work at a baud rate of 57600. This works for most chips that use the internal oscillator. But it is best to check it first with a simple program. When you use a crystal you might even use a higher speed.

Do not forget that the MCS boot loader must be set to the same baud rate as the boot

loader program.

Now make a new test program and compile it. Press F4 to start the MCS boot loader. You now need to reset the chip so that it will start the boot loader section. The boot loader will send a byte with value of 123 and the Bascom boot loader receives this and thus starts the loader process.

There will be a stand alone boot loader available too. And the sample will be extended to support other AVR chips with boot section too.

There is a \$BOOT directive too. It is advised to use \$LOADER as it allows you to write the boot loader in BASIC.

You can not use interrupts in your boot loader program as the interrupts will point to the reset vector which is located in the lower section of the flash. When you start to writing pages, you overwrite this part.

See also

\$BOOT 3147, \$LOADERSIZE 3487

Example

(c) 1995-2005, MCS Bootloader.bas This sample demonstrates how you can write your own bootloader in BASCOM BASIC _____ 'This sample will be extended to support other chips with bootloader 'The loader is supported from the IDE '\$reqfile = "m88def.dat" 'Const Loader = 88 '\$regfile = "m32def.dat" 'Const Loaderchip = 32 '\$reqfile = "m88def.dat" 'Const Loaderchip = 88 \$regfile = "m162def.dat" **Const** Loaderchip = 162 **#if** Loaderchip = 88 'Mega88 \$loader = \$c00'this address you can find in the datasheet 'the loader address is the same as the boot vector address **Const** Maxwordbit = 5 Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 , Clockpol = 0#endif **#if** Loaderchip = 32 ' Mega32 \$1oader = \$3c00' 1024 words **Const** Maxwordbit = 6 'Z6 is maximum bit **Config** Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,

```
Databits = 8 , Clockpol = 0
#endif
#if Loaderchip = 8
                                                             ' Mega8
                                                             ' 1024 words
    \$loader = \$c00
    Const Maxwordbit = 5
                                                             'Z5 is
maximum bit
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif
#if Loaderchip = 161
                                                              ' Megal61
    $loader = $1e00
                                                             ' 1024 words
    Const Maxwordbit = 6
                                                             'Z5 is
maximum bit
#endif
#if Loaderchip = 162
                                                              ' Megal62
    $loader = $1c00
                                                              ' 1024 words
   Const Maxwordbit = 6
                                                             'Z5 is
maximum bit
   Config Coml = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif
Const Maxword =(2 ^ Maxwordbit) * 2
                                                             '128
Const Maxwordshift = Maxwordbit + 1
$crystal = 8000000
|$crystal = 14745600
$baud = 57600
                                                              'this loader
uses serial com
'It is VERY IMPORTANT that the baud rate matches the one of the boot
loader
'do not try to use buffered com as we can not use interrupts
'Dim the used variables
Dim Bstatus As Byte , Bretries As Byte , Bblock As Byte , Bblocklocal As
Byte
Dim Bcsum1 As Byte , Bcsum2 As Byte , Buf(128) As Byte , Csum As Byte
Dim J As Byte , Spmcrval As Byte
                                                              ' self
program command byte value
Dim Z As Word
                                                              'this is the
Z pointer word
Dim Vl As Byte , Vh As Byte
                                                              ' these
bytes are used for the data values
Dim Wrd As Byte , Page As Byte
                                                             'these vars
contain the page and word address
'Mega 88 : 32 words, 128 pages
Disable Interrupts
                                                             'we do not
use ints
Waitms 1000
                                                             'wait 1 sec
'We start with receiving a file. The PC must send this binary file
'some constants used in serial com
Const Nak = &H15
```

```
346 BASCOM-AVR
```

```
Const Ack = &H06
Const Can = &H18
'we use some leds as indication in this sample , you might want to
remove it
Config Portb = Output
Portb = 255
                                                              'the stk200
has inverted logic for the leds
'$timeout = 1000000
                                                               'we use a
timeout
$timeout = 1000000
                                                              'we use a
timeout
'Do
                                                              'wait for
  Bstatus = Waitkey()
the loader to send a byte
 Print Chr(bstatus);
  If Bstatus = 123 Then
                                                              'did we
received value 123 ?
     Goto Loader
  End If
'Loop
For J = 1 To 10
                                                              'this is a
simple indication that we start the normal reset vector
  Toggle Portb : Waitms 100
Next
                                                              'goto the
Goto reset
normal reset vector at address 0
'this is the loader routine. It is a Xmodem-checksum reception routine
Loader:
For J = 1 To 3
                                                              'this is a
simple indication that we start the normal reset vector
   Toggle Portb : Waitms 500
Next
Spmcrval = 3 : Gosub Do_spm
                                                              'erase the
first page
Spmcrval = 17 : Gosub Do_spm
                                                              ' re-enable
page
Bretries = 10
                                                              'number of
retries
Do
  Csum = 0
                                                              'checksum is
0 when we start
 Print Chr(nak);
                                                              ' firt time
send a nack
 Do
                                                              'wait for
    Bstatus = Waitkey()
statuse byte
    Select Case Bstatus
       Case 1:
                                                              ' start of
heading, PC is ready to send
            Incr Bblocklocal
                                                              'increase
local block count
                                                              'checksum is
            Csum = 1
1
            Bblock = Waitkey() : Csum = Csum + Bblock
                                                              'get block
```

Bcsum1 = Waitkey() : Csum = Csum + Bcsum1 'get checksum first byte **For** J = 1 **To** 128 'get 128 bytes Buf(j) = Waitkey() : Csum = Csum + Buf(j) Next Bcsum2 = Waitkey() 'get second checksum byte If Bblocklocal = Bblock Then 'are the blocks the same? If Bcsum2 = Csum Then 'is the checksum the same? Gosub Writepage 'yes go write the page Print Chr(ack); 'acknowledge 'no match so Else send nak Print Chr(nak); End If Else 'blocks do Print Chr(nak); not match End If Case 4: ' end of transmission , file is transmitted ' send ack Print Chr(ack); and ready Portb.3 = 0' simple indication that we are finished and ok ' start new Goto _reset program ' PC aborts Case &H18: transmission Goto _reset ' ready Case Else Exit Do ' no valid data End Select Loop If Bretries > 0 Then 'attempte left? Waitms 1000 Decr Bretries 'decrease attempts Else 'reset chip Goto _reset End If Loop 'write one or more pages Writepage: For J = 1 To 128 Step 2 'we write 2 bytes into a page Vl = Buf(j) : Vh = Buf(j + 1)'get Low and High bytes lds r0, {vl} 'store them into r0 and r1 registers

'write value

lds r1, {vh}

into page at word address

Spmcrval = 1 : Gosub Do_spm

```
BASCOM-AVR
```

```
Wrd = Wrd + 2
                                                              ' word
address increases with 2 because LS bit of Z is not used
      If Wrd = Maxword Then
                                                              ' page is
f1111
          Wrd = 0
                                                              'Z pointer
needs wrd to be 0
          Spmcrval = 5 : Gosub Do_spm
                                                              'write page
                                                              'next page
          Page = Page + 1
          Spmcrval = 3 : Gosub Do_spm
                                                              ' erase
next page
                                                              ' re-enable
          Spmcrval = 17 : Gosub Do_spm
page
      End If
  Next
  Toggle Portb.2 : Waitms 10 : Toggle Portb.2
                                                              'indication
that we write
Return
Do spm:
 Bitwait Spmcsr.selfprgen , Reset
                                                              ' check for
previous SPM complete
                                                              'wait for
 Bitwait Eecr.eepe , Reset
eeprom
 Z = Page
                                                              'make equal
to page
 Shift Z , Left , Maxwordshift
                                                              'shift to
proper place
                                                              'add word
 Z = Z + Wrd
  lds r30, \{Z\}
  lds r31, {Z+1}
 Spmcsr = Spmcrval
                                                              'assign
register
                                                              'this is an
 spm
asm instruction
 nop
 nop
Return
'How you need to use this program:
'1- compile this program
'2- program into chip with sample elctronics programmer
'3- select MCS Bootloader from programmers
'4- compile a new program for example M88.bas
'5- press F4 and reset your micro
' the program will now be uploaded into the chip with Xmodem Checksum
' you can write your own loader. And we will release a command line
```

```
loader in the future
```

6.34 \$LOADERSIZE

Action

Instruct the compiler that a boot loader is used so it will not overwrite the boot space.

Syntax \$LOADERSIZE = size

348

Remarks

size

The amount of space that is used by the boot loader.

When you use a boot loader it will use space from the available flash memory. The compiler does not know if you use a boot loader or not. When your program exceeds the available space and runs into the boot sector space, it will overwrite the boot loader.

The \$loadersize directive will take the boot loader size into account so you will get an error when the target file gets too big.

When you select the MCS boot loader as programmer the IDE also will take into account the specified boot loader size.

The directive can be used when you have a different programmer selected. For example an external programmer that does not know about the boot size.

See also

\$LOADER 343

ASM

NONE

Example

NONE

6.35 \$MAP

Action

Will generate label info in the report.

Syntax

\$MAP

Remarks

The \$MAP directive will put an entry for each line number with the address into the report file. This info can be used for debugging purposes with other tools.

See also

NONE

ASM

NONE

Example

\$MAP

The report file will not contain the following section :

Code map		_
Line	Address(hex)	_
1	0	-
9	36	
26	39	
30	3B	
31	3E	
32	48	
33	4B	
36	50	
37	56	
42	5B	
43	6C	
44	7D	
45	80	
46	81	

6.36 \$NOCOMPILE

Action

Instruct the compiler not to compile the file.

Syntax \$NOCOMPILE

Remarks

This looks like an odd directive. Since you can split your program in multiple files, and you can create configuration files, you might open a file and try to compile it. Only normal project files can be compiled and you will get a number of errors and also unwanted files like error, report, etc.

To prevent that you compile a file that is intended to be included, you can insert the \$NOCOMPILE directive.

Then the file will only be compiled when it is called from your main file, or other include file.

A file that is opened as thus the main file, and which includes the \$NOCOMP directive, can not be compiled.

The IDE will see it as a successful compilation. This is important for the Batch Compiler.

See also

Batch Compiler 84

Example \$NOCOMPILE

6.37 \$NOINIT

Action

Instruct the compiler to generate code without initialization code.

Syntax \$NOINIT

Remarks

\$NOINIT is only needed in rare situations. It will instruct the compiler not to add initialization code. But that means that you need to write your own code then. \$NOINIT was added in order to support boot loaders. But the new \$LOADER directive can better be used as it does not require special ASM knowledge.

See also

<u>\$LOADER</u> 343

Example

NONE

6.38 \$NORAMCLEAR

Action

Instruct the compiler to not generate initial RAM clear code.

Syntax

\$NORAMCLEAR

Remarks

Normally the SRAM is cleared in the initialization code. When you don't want the SRAM to be cleared(set to 0) you can use this directive.

Because all variables are automatically set to 0 or ""(strings) without the \$NORAMCLEAR, using \$NORAMCLEAR will set the variables to an unknown value. That is, the variables will probably set to FF but you cannot count on it.

When you have a battery back upped circuit, you do not want to clear the RAM at start up. So that would be a situation when you could use \$NORAMCLEAR.

See also

6.39 **\$NORAMPZ**

Action

This compiler directive disables RAMPZ clearing.

Syntax **\$NORAMPZ**

Remarks

Processors with more then 64 KB of memory need to set the RAMPZ register in order to point to the proper 64 KB page.

If the RAMPZ register is used, it will be cleared when it is used for something different then accessing the flash.

BASCOM uses the Z register to access flash memory or RAM memory. Since processors with external memory capability can access more then 64KB of RAM, the RAMPZ must be set/cleared when accessing this memory.

Otherwise accessing the flash code could result in a change of RAMPZ, and after this, accessing the RAM would not point to the proper place in memory.

But setting this register requires extra code. When your application just fitted into a M128 or M256 and you do not want this RAMPZ handling because your application works fine, then you can use this \$NORAMPZ directive. To see if your processor

See also NONE

Example

NONE

\$NOTRANSFORM 6.40

Action

This option controls transformation of unsupported ASM mnemonics.

Svntax

\$NOTRANSFORM ON|OFF

Remarks

By default, assembler mnemonics that are not supported for a chip or register are transformed into different assembler mnemonics.

The IN and OUT instructions for example only work on hardware registers with an address lower then 64. Most PORT registers are located in this lower address space, but there are many chips that have more ports which are located in extended memory. For such chips, using a IN or OUT on an extended address would result in a failure.

Thus the compiler changes IN into an LDS and an OUT into an STS. When a register is required, R23 will be used.

When you develop some ASM code, you might want to get an error when you are

using an instruction the wrong way. For this purpose you can turn off the transformation.

\$NOTRANSFORM OFFwill turn off the transformation. And with \$NOTRANFORM ON you can turn it back on.

You should only use this option on your own code. When you use it on your whole program, it will not compile since the bascom libraries which use CBI, SBI, SBIS, IN, OUT, etc. will use the transformation.

See also

NONE

Example

NONE

6.41 \$PROJECTTIME

Action

This directive will keep track of time you spend on the source.

Syntax

\$PROJECTTIME

Remarks

Keeping track of project time is the only purpose of this directive. It will be ignored by the compiler.

When the IDE finds the \$PROJECTTIME directive, it will count the minutes you spend on the code.

Each time you save the code, the updated value will be shown.

The IDE will automatic insert the value after \$PROJECTTIME.

So how does this work? When you type, you start a timer. When there are no keystrokes for 2 minutes, this process stops. It is started automatic as soon as you start typing. So when you type a character each minute, each minute will be counted a a full minutes of work.

The time is counted and shown in minutes.

While you can edit the value in the source, it will be changed as soon as you save the source.

See also

Example \$PROJECTTIME

354 BASCOM-AVR

6.42 \$PROG

Action

Directive to auto program the lock and fuse bits.

Syntax

\$PROG LB, FB , FBH , FBX

Syntax Xmega

\$PROG LB, F0 , F1 , F2 , F3 ,F4 , F5

Remarks

While the lock and fuse bits make the AVR customizable, the settings for your project can give some problems.

The \$PROG directive will create a file with the project name and the PRG extension.

Every time you program the chip, it will check the lock and fuse bit settings and will change them if needed.

So in a new chip, the lock and fuse bits will be set automatically. A chip that has been programmed with the desired settings will not be changed.

The programmer has an option to create the PRG file from the current chip settings.

The LB, FH, FBH and FBX values are stored in hexadecimal format in the PRJ file. You may use any notation as long as it is a numeric constant.

Some chips might not have a setting for FBH or FBX, or you might not want to set all values. In that case, do NOT specify the value. For example:

\$PROG &H20 ,,,

This will only write the Lockbit settings.

\$PROG ,,&H30,

This will only write the FBH settings.

LB	Lockbit settings
FB	Fusebit settings
FBH	Fusebit High settings
FBX	Extended Fusebit settings

Sometimes the data sheet refers to the Fusebit as the Fusebit Low settings.

The \$PROG setting is only supported by the AVRISP, STK200/300, Sample Electronics and Universal MCS Programmer Interface. The USB-ISP programmer also supports the \$PROG directive.

When you select the wrong Fuse bit, you could lock your chip. For example when you choose the wrong oscillator option, it could mean that the micro expects an external crystal oscillator. But when you connect a simple crystal, it will not work.

355

In these cases where you can not communicate with the micro anymore, the advise is to apply a clock signal to X1 input of the micro.

You can then select the proper fuse bits again.

When you set the Lock bits, you can not read the chip content anymore. Only after erasing the chip, it could be reprogrammed again.

Conce the lock bits and fuse bits are set, it is best to remark the \$PROG directive. This because it takes more time to read and compare the bits every time.

Xmega

The Xmega has one lock byte and 6 fuse bytes. For an Xmega the Write PRG option will write the correct code.

See also

Programmers 1027, \$PROG 354

6.43 \$PROGRAMMER

Action

Will set the programmer from the source code.

Syntax

\$PROGRAMMER = number

Remarks

Number A numeric constant that identifies the programmer.

The \$PROGRAMMER directive will set the programmer just before it starts programming. When you press F4 to program a chip, the selected programmer will be made active. This is convenient when you have different project open and use different programmers.

But it can also lead to frustration as you might think that you have the 'STK200' selected, and the directive will set it to USB-ISP.

The following values can be used :

Value	Programmer
0	AVR-ISP programmer(old AN 910)
1	STK200/STK300
2	PG302
3	External programmer
4	Sample Electronics
5	Eddie Mc Mullen
6	KITSRUS K122
7	STK500
8	Universal MCS Interface
9	STK500 extended
10	Lawicel Bootloader
11	MCS USB
12	USB-ISP I

13	MCS Bootloader
14	Proggy
15	FLIP

See also

<u>\$PROG</u>354

ASM

NONE

Example

\$REGFILE

6.44 \$REGFILE

Action

Instruct the compiler to use the specified register file instead of the selected dat file.

Syntax

\$REGFILE = "name"

Remarks

	The name of the register file. The register files are stored in the BASCOM-AVR application directory and they all have the DAT extension.
	The register file holds information about the chip such as the internal registers and interrupt addresses. The register file info is derived from atmel definition files.

The \$REGFILE statement overrides the setting from the Options, Compiler, Chip menu.

The settings are stored in a <project>.CFG file.

The \$REGFILE directive must be the first statement in your program. It may not be put into an included file since only the main source file is checked for the \$REGFILE directive.

It is good practice to use the \$REGFILE directive. It has the advantage that you can see at the source which chip it was written for. The \$REGFILE directive is also needed when the PinOut state viewer or the PDF state viewer is used.

The register files contain the hardware register names from the micro. They also contain the bit names. These are constants that you may use in your program. But the names can not be used to dim a variable for example.

Example : *DIM PORTA As Byte* This will not work as PORTA is a register constant.

See also

\$SWSTACK 385 , \$HWSTACK 326 , \$FRAMESIZE 325

ASM

NONE

Example

\$REGFILE = "8515DEF.DAT"

6.45 \$RESOURCE

Action

Instruct the compiler to use a special resource file for multi language support.

Syntax

\$RESOURCE [DUMP] "lang1" [, "lang2"] **\$RESOURCE** ON | OFF

Remarks

lang1	This is the name of the first and default language. You can add a maximum of 8 languages. The names will be used in the resource editor. But they are only intended as a reference. The resource names will not end up in your application. They are used for the column names in the resource editor.
lang2	The second language. You can add multiple languages separated by a comma. The language must be specified within double quotes.
ON	This will turn on the languages resource handling. In some cases you need to turn the language handling ON or OFF which is explained later
OFF	This will turn OFF the language handling
DUMP	This mode will create a <project>.BCS file which contains all used string constants</project>

Some applications require that the interface is available in multiple languages. You write your application the same way as you always do.

When it is ready, you can add the \$RESOURCE directive to make the application suited for multiple languages.

The \$RESOURCE option will generate a BYTE variable named LANGUAGE. You can change the value in your application. The compiler will take care that the proper string is shown.

But first you need to translate the strings into the languages of your choice. For this purpose you can use the Resource Editor. The <u>Resource Editor</u> a BCS file (BASCOM String file) which contains the languages and the strings. You can then add a string for all languages.

So first make sure your application works. Then compile using the \$RESOURCE DUMP option.

When you test the languages.bas sample the content will look like this : "English", "Dutch", "German", "Italian" "Multi language test" "This" " is a test" "Name " "Hello "

As you can see, the first line contains the languages. The other lines only contain a string. Each string is only stored once in BASCOM. So even while "Mark" can have multiple meanings, it will only end up once in the BCS file. After you have translated the strings, the content of the BCR (BASCOM Resource) file will look like :

"English", "Dutch", "German", "Italian" "This", "Dit", "Dies", "Questo" "Name ", "Naam", "Name", "Nome" "Multi language test", "Meertalen test", "", "Test multilingua" "Hello ", "Hallo", "Hallo", "Ciao" " is a test", " is een test", "ist ein test", "è un test" "mark", "Mark", "Marcus", "Marco"

You may edit this file yourself, using Notepad or you can use the Resource Editor. Untranslated strings will be stored as "". Untranslated strings will be shown in the original language !

Now recompile your project and the compiler will handle every string it will find in the resource file (BCR) in a special way. Strings that are not found in the BCR file, are not processed and handled like normal. For example when you have a PRINT "check this out", and you did not put that in the BCR file, it will show the same no matter which value the LANGUAGE variable has.

But for each string found in the BCR file, the compiler will show the string depending on the LANGUAGE variable. When one of the languages is not translated, it will show as the original language.

When LANGUAGE is 0, it will show the first string (the string from the first column). When languages is 1, it will show the string from the second column, and so on. You must take care that the LANGUAGE variables has a valid value.

So by switching/changing 1 variable, you can change the language in the entire application. Strings are used for PRINT, LCD and other commands. It will work on every string that is in the BCR file. But that also brings us to the next option.

Image this code : If S = "mark" Then Print "we can not change names" End If

As you can see, we use a string. The code will fail if the string is translated (and is different in each language). You can simply remove the this string from the Resource file. But when you also need the word "mark" in the interface, you have a problem. For this purpose you can turn off the resource handling using \$RESOURCE OFF The compiler will then not process the code following the directive with the special resource handling.

And when you are done, you can turn the resource handling on again using \$RESOURCE ON.

See also

Resource Editor 88

```
Example
                           _____
                         language.bas
                 (c) 1995-2008 , MCS Electronics
'This example will only work with the resource add on
'resources are only needed for multi language applications
'By changing the LANGUAGE variable all strings used will be shown in the proper lar
_____
                                              _____
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
'a few steps are needed to create a multi language application
'STEP 1, make your program as usual
'STEP 2, generate a file with all string resources using the $RESOURCE DUMP direct:
'$resource Dump , "English" , "Dutch" , "German" , "Italian" 'we will use 4 language 'STEP 3, compile and you will find a file with the BCS extesion
'STEP 4, use Tools, Resource Editor and inport the resources
'STEP 5, add languages, translate the original strings
'STEP 6, compile your program this time with specifying the languages without the I
$resource "English" , "Dutch" , "German" , "Italian"
'this must be done before you use any other resource !
'in this sample 4 languages are used
'this because all resources found are looked up in the BCR file(BasCom Resource)
Dim S As String * 20
Dim B As Byte
Print "Multi language test"
Do
  Print "This" ;
  S = " is a test" : Print S
  Input "Name " , S
  Print "Hello " ; S
   'now something to look out for !
   'all string data not found in the BCR file is not resourced. so there is no prob
   If S = "mark" Then
     Print "we can not change names"
   End If
   'but if you want to have "mark" resourced for another sentence you have a proble
   'the solution is to turn off resourcing
   $resource Off
   Print "mark"
   If S = "mark" Then
     Print "we can not change names"
   End If
   $resource On
   Language = Language + 1
   If Language > 3 Then Language = 0
Loop
```

6.46 \$ROMSTART

Action

Instruct the compiler to generate a hex file that starts at the specified address.

Syntax

\$ROMSTART = address

Remarks

The address where the code must start. By default the first address is 0.
The bin file will still begin at address 0.

The \$ROMFILE could be used to locate code at a different address for example for a boot loader.

It is best to use the new \$LOADER directive to add boot loader support.

See also

<u>\$LOADER</u> 343

ASM

NONE

Example

\$ROMSTART = &H4000

6.47 \$SERIALINPUT

Action

Specifies that serial input must be redirected.

Syntax SERIALINPUT = label

Remarks

Label	The name of the assembler routine that must be called when a character
	is needed by the INPUT routine. The character must be returned in R24.

With the redirection of the INPUT command, you can use your own input routines.

This way you can use other devices as input devices. Note that the INPUT statement is terminated when a RETURN code (13) is received.

By default when you use INPUT or INKEY(), the compiler will expect data from the COM port. When you want to use a keyboard or remote control as the input device

361

you can write a custom routine that puts the data into register R24 once it needs this data.

See also

\$SERIALOUTPUT 363

```
_____
'name
                       : $serialinput.bas
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : demonstrates $SERIALINPUT redirection of
serial input
                       : Mega48
'micro
'suited for demo
                        : yes
'commercial addon needed : no
.....
                                _____
$regfile = "m48def.dat"
'define used crystal
$crystal = 4000000
                                                        ' default
\$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
                                                        'default use
10 for the SW stack
$framesize = 40
                                                        'default use
40 for the frame space
'dimension used variables
Dim S As String * 10
Dim W As Long
'inform the compiler which routine must be called to get serial
characters
$serialinput = Myinput
'make a never ending loop
Do
 'ask for name
 Input "name " , S
 Print S
 'error is set on time out
 Print "Error " ; Err
Loop
End
'custom character handling routine
'instead of saving and restoring only the used registers
'and write full ASM code, we use Pushall and PopAll to save and restore
'all registers so we can use all BASIC statements
'$SERIALINPUT requires that the character is passed back in R24
Myinput:
 Pushall
                                                        'save all
registers
 W = 0
                                                        'reset
counter
```

Myinput1:	
Incr W	'increase
counter	
Sbis USR, 7	' Wait for
character	
Rjmp myinput2	'no charac
waiting so check again	
Popall	'we got
something	
$\mathbf{Err} = 0$	'reset error
In _temp1, UDR	' Read
character from UART	
Return	'end of
routine	
Myinput2:	
If W > 1000000 Then	'with 4 MHz
ca 10 sec delay	
rjmp Myinput_exit	'waited too
long	
Else	
Goto Myinput1	'try again
End If	
Myinput_exit:	
Popall	'restore
registers	
$\mathbf{Err} = 1$	'set error
variable	
ldi R24, 13	'fake enter
so INPUT will end	
Return	

6.48 \$SERIALINPUT1

Action

Specifies that serial input of the second UART must be redirected.

Syntax

\$SERIALINPUT1 = label

Remarks

Label	The name of the assembler routine that must be called when a character
	is needed from the INPUT routine. The character must be returned in R24.

With the redirection of the INPUT command, you can use your own input routines.

This way you can use other devices as input devices. Note that the INPUT statement is terminated when a RETURN code (13) is received.

By default when you use INPUT or INKEY(), the compiler will expect data from the COM2 port. When you want to use a keyboard or remote control as the input device you can write a custom routine that puts the data into register R24 once it asks for this data.

See also <u>\$SERIALOUTPUT1</u> (364), <u>\$SERIALINPUT</u> (365), <u>\$SERIALOUTPUT</u> (363)

Example

See the <u>\$SERIALINPUT</u> 360 sample

6.49 \$SERIALINPUT2LCD

Action

This compiler directive will redirect all serial input to the LCD display instead of echoing to the serial port.

Syntax \$SERIALINPUT2LCD

Remarks

You can also write your own custom input or output driver with the <u>\$SERIALINPUT</u> and <u>\$SERIALOUTPUT</u> and <u>\$SERIALOUTPUT</u> and <u>\$SERIALOUTPUT</u> and <u>\$SERIALOUTPUT</u> and <u>\$SERIALINPUT2LCD</u> is handy when you use a LCD display. By adding only this directive, you can view all output form routines such as PRINT, PRINTBIN, on the LCD display.

See also

<u>\$SERIALINPUT</u> अध्ये , <u>\$SERIALOUTPUT</u> अध्ये , <u>\$SERIALOUTPUT1</u> अध्ये , <u>\$SERIALOUTPUT1</u> अध्ये ,

Example

\$regfile = "m48def.dat"
\$crystal = 4000000
\$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.5 , Db6 = Portb.6 ,
Db7 = Portb.7 , E = Portc.7 , Rs = Portc.6

```
$serialinput2lcd
Dim V As Byte
Do
    Cls
    Input "Number " , V
go to the LCD display
Loop
```

'this will

6.50 \$SERIALOUTPUT

Action

Specifies that serial output must be redirected.

Syntax SERIALOUTPUT = label

Remarks

	364	BASCOM-AVR
--	-----	------------

Label	The name of the assembler routine that must be called when a character is send to the serial buffer (UDR).
	The character is placed into R24.

With the redirection of the PRINT and other serial output related commands, you can use your own routines.

This way you can use other devices as output devices.

See also

<u>\$SERIALINPUT</u> बिको , <u>\$SERIALINPUT2LCD</u> बिको , <u>\$SERIALINPUT1</u> बिक्वी , <u>\$SERIALOUTPUT1</u> बिक्वी

Example

```
$regfile = "m48def.dat"
crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
$serialoutput = Myoutput
'your program goes here
Do
 Print "Hello"
Loop
End
myoutput:
'perform the needed actions here
'the data arrives in R24
'just set the output to PORTB
  !outportb,r24
ret
```

6.51 \$SERIALOUTPUT1

Action

Specifies that serial output of the second UART must be redirected.

Syntax

\$SERIALOUTPUT1 = label

Remarks

Label	The name of the assembler routine that must be called when a character is send to the serial buffer (UDR1).
	The character is placed into R24.

With the redirection of the PRINT and other serial output related commands, you can use your own routines.

This way you can use other devices as output devices.

See also

<u>\$SERIALINPUT1</u> अध्ये , <u>\$SERIALINPUT</u> अध्ये , <u>\$SERIALOUTPUT</u> अध्ये , <u>\$SERIALOUTPUT</u> अध्ये ,

Example

See the **<u>\$SERIALOUTPUT</u> 363** example

6.52 \$SIM

Action

Instructs the compiler to generate empty wait loops for the WAIT and WAITMS statements. This to allow faster simulation.

Syntax

\$SIM

Remarks

Simulation of a WAIT statement can take a long time especially when memory view windows are opened.

The \$SIM compiler directive instructs the compiler to not generate code for WAITMS and WAIT. This will of course allows faster simulation.

When your application is ready you must remark the \$SIM directive or otherwise the WAIT and WAITMS statements will not work as expected. When you forget to remove the \$SIM option and you try to program a chip you will receive a warning that \$SIM was used.

See also

NONE

ASM

NONE

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
$sim
Do
    Wait 1
    Print "Hello"
Loop
```

6.53 \$SWSTACK

Action

Sets the available space for the software stack.

Syntax \$SWSTACK = var

Remarks

Var A numeric decimal value.

While you can configure the SW Stack in Options, Compiler, Chip, it is good practice to put the value into your code. This way you do no need the cfg(configuration) file.

The \$SWSTACK directive overrides the value from the IDE Options.

It is important that the \$SWSTACK directive occurs in your main project file. It may not be included in an \$include file as only the main file is parsed for \$SWSTACK

See also

\$HWSTACK 3267 , \$FRAMESIZE 3257

Example

```
: adc.bas
'name
'copyright
                      : (c) 1995-2005, MCS Electronics
                      : demonstration of GETADC() function for 8535
'purpose
or M163 micro
                      : Mega163
'micro
'suited for demo
                       : yes
'commercial addon needed : no
'use in simulator
                       : possible
' Getadc() will also work for other AVR chips that have an ADC converter
·-----
                                             _____
_____
$regfile = "m163def.dat"
                                                     ' we use the
M163
$crystal = 4000000
$hwstack = 32
                                                     ' default
use 32 for the hardware stack
\$swstack = 10
                                                     'default use
10 for the SW stack
$framesize = 40
                                                     'default use
40 for the frame space
```

6.54 \$TIMEOUT

Action

Enable timeout on the hardware UART 0 and UART1.

Syntax

\$TIMEOUT = value

Remarks

Value A constant that fits into a LONG , indicating how much time must be waited before the waiting is terminated.

All RS-232 serial statements and functions(except INKEY) that use the HW UART, will halt the program until a character is received. Only with buffered serial input you can process your main program while the buffer received data on the background.

STIMEOUT is an alternative for normal serial reception. It is not intended to be used with buffered serial reception.

When you assign a constant to \$TIMEOUT, you actual assign a value to the internal created value named _____TIMEOUT.

This value will be decremented in the routine that waits for serial data. When it reaches zero, it will terminate.

So the bigger the value, the longer the wait time before the timeout occurs. The timeout is not in seconds or microseconds, it is a relative number. Only the speed of the oscillator has effect on the duration. And the value of the number of course.

When the time out is reached, a zero/null will be returned to the calling routine. Waitkey() will return 0 when used with a byte. When you use INPUT with a string, the timeout will be set for every character. So when 5 characters are expected, and they arrive just before the timeout value is reached, it may take a long time until the code is executed.

When the timeout occurs on the first character, it will return much faster.

When you already sent data, this data will be returned. For example, "123" was sent but a RETURN was never sent, INPUT will return "123". While without the \$TIMEOUT, INPUT will not return until a RETURN is received.

When you activate \$TIMEOUT, and your micro has two UARTS(Mega128 for example) it will be active for both UART0 and UART1.

\$TIMEOUT is not supported by the software UART.

See Also

INPUT 744 , WAITKEY 949

Example

```
: timeout.bas
'name
                       : (c) 1995-2005, MCS Electronics
'copyright
                       : demonstration of the $timeout option
'purpose
                       : Mega48
'micro
'suited for demo : yes
'commercial addon needed : no
 _____
$reqfile = "m48def.dat"
                                                        ' specify
the used micro
$crystal = 8000000
                                                        ' used
```

368 BASCOM-AVR

```
crystal frequency
$baud = 19200
                                                              ' use baud
rate
\$hwstack = 32
                                                              ' default
use 32 for the hardware stack
Swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
'most serial communication functions and routines wait until a character
'or end of line is received.
'This blocks execution of your program. SOmething you can change by
using buffered input
'There is also another option : using a timeout
'$timeout Does Not Work With Buffered Serial Input
Dim Sname As String * 10
Dim B As Byte
Do
   $timeout = 1000000
   Input "Name : " , Sname
   Print "Hello " ; Sname
   $timeout = 5000000
   Input "Name : " , Sname
  Print "Hello " ; Sname
Loop
```

'you can re-configure \$timeout

6.55 \$TINY

Action

Instruct the compiler to generate initialize code without setting up the stacks.

Syntax \$TINY

Remarks

The tiny11 for example is a powerful chip. It only does not have SRAM. BASCOM depends on SRAM for the hardware stack and software stack. When you like to program in ASM you can use BASCOM with the \$TINY directive.

Some BASCOM statements will also already work but the biggest part will not work. A future version will support a subset of the BASCOM statements and function to be used with the chips without SRAM.

Note that the generated code is not yet optimized for the tiny parts. Some used ASM statements for example will not work because the chip does not support it.

See also NONE

ASM NONE

Example

```
_____
                    : tiny15.bas
'name
                    : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                     : demonstrate using ATtiny15
'purpose. demons'micro: Tiny15'suited for demo: yes
'commercial addon needed : no
·-----
                                      _____
_____
$regfile = "at15def.dat"
                                                 ' specify
the used micro
$crystal = 1000000
                                                 ' used
crystal frequency
$tiny
$noramclear
Dim A As Iram Byte
Dim B As Iram Byte
A = 100 : B = 5
```

6.56 \$VERSION

nop

Action

A = A + B

This compiler directive stores version information.

Syntax \$VERSION V,S,R

Remarks

Version info is important information. If you need to maintain source code, it will make it easy to identify the code.

\$VERSION has 3 parameters. These must be numeric digits. Each time you compile your code, the release number is increased.

You can use Version(2) to print this information. version 1,2,3 will be printed as 1.2.3

See also

VERSION 947

Example

\$version 1,2,3
Print Version(2)

6.57 \$WAITSTATE

Action

Compiler directive to activate external SRAM and to insert a WAIT STATE for a slower ALE signal.

🚺 CONFIG XRAM should be used instead.

Syntax \$WAITSTATE

Remarks

The \$WAITSTATE can be used to override the Compiler Chip Options setting. Wait states are needed for slow external components that can not handle the fast ALE signal from the AVR chip.

See also

<u>\$XA</u> (370) , <u>CONFIG XRAM</u> 589)

Example

\$WAITSTATE

6.58 \$XA

Action Compiler directive to activate external memory access.

CONFIG XRAM 589should be used instead.

Syntax

\$XA

Remarks

The \$XA directive can be used to override the Compiler Chip Options setting. This way you can store the setting in your program code. It is strongly advised to do this.

See also

<u>\$WAITSTATE</u> अगे, <u>CONFIG XRAM</u> 589ो

Example

\$XA

6.59 \$XRAMSIZE

Action

Specifies the size of the external RAM memory.

Syntax

\$XRAMSIZE = [&H] size

Remarks

Size A constant with the size of the external RAM memory chip.

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _

The size of the chip can be selected from the <u>Options Compiler Chip</u> and menu. The \$XRAMSIZE overrides this setting. It is important that \$XRAMSTART precedes \$XRAMSIZE

See also

<u>\$XRAMSTART</u> अगी

Example

```
_____
                      : m128.bas
'name
                      : (c) 1995-2005, MCS Electronics
'copyright
                       : demonstrate using $XRAM directive
'purpose
                      : Mega128
'micro
'suited for demo
                       : yes
'commercial addon needed : no
! _____
                                              _____
-----
$regfile = "m128def.dat"
                                                      ' specify
the used micro
$crystal = 1000000
                                                      ' used
crystal frequency
\$hwstack = 32
                                                      ' default
use 32 for the hardware stack
\$swstack = 10
                                                      ' default
use 10 for the SW stack
                                                      ' default
$framesize = 40
use 40 for the frame space
$xramstart = &H1000
```

\$xramsize = &H1000
Dim X As X

6.60 \$XRAMSTART

Action

Specifies the location of the external RAM memory.

Syntax

\$XRAMSTART = [&H]address

<u>Remarks</u>	
Address	The (hex)-address where the data is stored.
	Or the lowest address that enables the RAM chip. You can use this option when you want to run your code in systems with external RAM memory. Address must be a constant.

By default the extended RAM will start after the internal memory so the lower addresses of the external RAM can't be used to store information.

When you want to protect an area of the chip, you can specify a higher address for the compiler to store the data. For example, you can specify &H400. The first dimensioned variable will be placed in address &H400 and not in &H260.

It is important that when you use \$XRAMSTART and \$XRAMSIZE that \$XRAMSTART comes before \$XRAMSIZE.

See also

<u>\$XRAMSIZE</u> अने

Example

'name	: m128.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: demonstrate using \$XRAM directive
'micro	: Mega128
'suited for demo	; yes
'commercial addon needed	1
· · · · · · · · · · · · · · · · · · ·	

```
$regfile = "m128def.dat" ' specify
the used micro
$crystal = 1000000 ' used
crystal frequency
$hwstack = 32 ' default
use 32 for the hardware stack
$swstack = 10 ' default
use 10 for the SW stack
$framesize = 40 ' default
use 40 for the frame space
```

\$xramstart = &H1000

\$xramsize = &H1000
Dim X As X

6.61 1WIRECOUNT

Action

This statement reads the number of 1wire devices attached to the bus.

Syntax

var2 = 1WIRECOUNT()
var2 = 1WIRECOUNT(port , pin)

Remarks

var2	A WORD variable that is assigned with the number of devices on the bus.
port	The PIN port name like PINB or PIND.
	The pin number of the port. In the range from 0-7. May be a numeric constant or variable.

The variable must be of the type word or integer.

You can use the 1wirecount() function to know how many times the 1wsearchNext() function should be called to get all the Id's on the bus.

The 1wirecount function will take 4 bytes of SRAM.

____1w_bitstorage , Byte used for bit storage : lastdeviceflag bit 0 id_bit bit 1 cmp_id_bit bit 2 search_dir bit 3 ____1wid_bit_number, Byte ___1wlast_zero, Byte ___1wlast_discrepancy , Byte

ASM

The following asm routines are called from mcs.lib.

_1wire_Count : (calls _1WIRE, _1WIRE_SEARCH_FIRST , _1WIRE_SEARCH_NEXT)

Parameters passed : R24 : pin number, R30 : port , Y+0,Y+1 : 2 bytes of soft stack, X : pointer to the frame space

Returns Y+0 and Y+1 with the value of the count. This is assigned to the target variable.

See also

1WWRITE [386], 1WRESET [375], 1WREAD [377], 1WSEARCHFIRST [379], 1WSEARCHNEXT [382], Using the 1wire protocol [180]

'name	: lwireSearch.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: demonstrates 1wsearch
'micro	: Mega48
'suited for demo	: yes
'commercial addon needed	: no

374

```
$regfile = "m48def.dat"
$crystal = 4000000
\$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              'default use
10 for the SW stack
$framesize = 40
                                                              'default use
40 for the frame space
Config 1wire = Portb.0
                                                              'use this
pin
'On the STK200 jumper B.0 must be inserted
'The following internal bytes are used by the scan routines
 ____1w_bitstorage , Byte used for bit storage :
       lastdeviceflag bit 0
       id bit
                      bit 1
                     bit 2
       cmp_id_bit
                     bit 3
       search_dir
   _1wid_bit_number, Byte
  ____1wlast_zero, Byte
 ____lwlast_discrepancy , Byte
  ___lwire_data , string * 7 (8 bytes)
'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte
'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word
'Now search for the first device on the bus
Reg_no(1) = lwsearchfirst()
For I = 1 To 8
                                                              'print the
number
   Print Hex(reg_no(i));
Next
Print
Do
  'Now search for other devices
  Reg_no(1) = lwsearchnext()
  For I = 1 To 8
    Print Hex(reg_no(i));
 Next
  Print
Loop Until Err = 1
'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = lwirecount()
'It is IMPORTANT that the lwirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W
```

```
'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = lwsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Req no(1) = 2
'now verify if the number exists
lwverify Reg_no(1)
Print Err
'err =1 when the ID passed n req no() does NOT exist
' optinal call it with pinnumber line lwverify req no(1),pinb,1
'As for the other lwire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)
                                                              'for
example look at pin PINB.1
End
```

6.62 **1WRESET**

Action

This statement brings the 1wire pin to the correct state, and sends a reset to the bus.

Syntax

1WRESET 1WRESET, PORT, PIN

Remarks

	Reset the 1WIRE bus. The error variable ERR will return 1 if an error occurred
Port	The register name of the input port. Like PINB, PIND.
	The pin number to use. In the range from 0-7. May be a numeric constant or variable.

The global variable ERR is set when an error occurs. There is also support for multi 1-wire devices on different pins.

To use this you must specify the port and pin that is used for the communication.

The 1wreset, 1wwrite and 1wread statements will work together when used with the old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE statement.

The syntax for additional 1-wire devices is :

1WRESET port , pin 1WWRITE var/constant ,bytes] , port, pin var = 1WREAD(bytes) , for the configured 1 wire pin var = 1WREAD(bytes, port, pin) ,for reading multiple bytes

See also

1WREAD 3777, 1WWRITE 386

Example

```
_____
_____
                        : lwire.bas
'name
'copyright
                        : (c) 1995-2005, MCS Electronics
'purpose
                        : demonstrates lwreset, lwwrite and lwread()
'micro
                        : Mega48
'suited for demo
                        : yes
'commercial addon needed : no
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
!-----
                                         _____
_____
$regfile = "m48def.dat"
$crystal = 4000000
$hwstack = 32
                                                        ' default
use 32 for the hardware stack
\$swstack = 10
                                                        'default use
10 for the SW stack
$framesize = 40
                                                        'default use
40 for the frame space
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"
Config 1wire = Portb.0
                                                        'use this
pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte
Do
 Wait 1
                                                        'reset the
 1wreset
device
 Print Err
                                                        'print error
1 if error
 1wwrite &H33
                                                        'read ROM
command
 For I = 1 To 8
   Ar(i) = lwread()
                                                        'place into
array
 Next
'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)
                                                         'read 8
bytes
 For I = 1 To 8
    Print Hex(ar(i));
                                                        'print
output
 Next
 Print
                                                        'linefeed
Loop
```

'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT

```
'THIS because of the DO LOOP that is never terminated!!!
'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
 Ar(i) = 0
                                                                'clear array
to see that it works
Next
1wreset Pinb , 2
                                                                'use this
port and pin for the second device
lwwrite &H33 , 1 , Pinb , 2
now the number of bytes must be specified!
                                                                'note that
'lwwrite Ar(1) , 5,pinb,2
'reading is also different
Ar(1) = 1wread(8, Pinb, 2)
                                                                'read 8
bytes from portB on pin 2
For I = 1 To 8
 Print Hex(ar(i));
Next
'you could create a loop with a variable for the bit number !
For I = 0 To 3
                                                                'for pin 0-3
  lwreset Pinb , I
 lwwrite &H33 , 1 , Pinb , I
 Ar(1) = 1wread(8, Pinb, I)
 For A = 1 To 8
    Print Hex(ar(a));
 Next
 Print
Next
End
```

6.63 1WREAD

Action

This statement reads data from the 1wire bus into a variable.

Syntax

```
var2 = 1WREAD( [ bytes] )
var2 = 1WREAD( bytes , port , pin)
```

Remarks

var2	Reads a byte from the bus and places it into variable var2.
	Optional the number of bytes to read can be specified.
Port	The PIN port name like PINB or PIND.
	The pin number of the port. In the range from 0-7. Maybe a numeric constant or variable.

Multi 1-wire devices on different pins are supported. To use this you must specify the port pin that is used for the communication.

The 1wreset, 1wwrite and 1wread statements will work together when used with the

378 BASCOM-AVR

old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE statement 444.

The syntax for additional 1-wire devices is : 1WRESET port, pin 1WWRITE var/constant , bytes, port, pin var = 1WREAD(bytes, port, pin) for reading multiple bytes

See also

1WWRITE 386 , 1WRESET 375

```
_____
_____
'name
                       : 1wire.bas
                        : (c) 1995-2005, MCS Electronics
'copyright
                       : demonstrates 1wreset, 1wwrite and 1wread()
'purpose
                       : Mega48
'micro
'suited for demo
                       : yes
'commercial addon needed : no
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
·-----
                                            _____
_____
$regfile = "m48def.dat"
$crystal = 4000000
\$hwstack = 32
                                                        ' default
use 32 for the hardware stack
\$swstack = 10
                                                        'default use
10 for the SW stack
$framesize = 40
                                                        'default use
40 for the frame space
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"
Config 1wire = Portb.0
                                                        'use this
pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte
Do
 Wait 1
 lwreset
                                                        'reset the
device
 Print Err
                                                        'print error
1 if error
 1wwrite &H33
                                                        'read ROM
command
 For I = 1 To 8
   Ar(i) = 1wread()
                                                        'place into
array
 Next
```

```
'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)
                                                              'read 8
bytes
  For I = 1 To 8
    Print Hex(ar(i));
                                                              'print
output
  Next
                                                              'linefeed
  Print
Loop
'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!
'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
 Ar(i) = 0
                                                             'clear arrav
to see that it works
Next
lwreset Pinb , 2
                                                             'use this
port and pin for the second device
1wwrite &H33 , 1 , Pinb , 2
                                                             'note that
now the number of bytes must be specified!
'lwwrite Ar(1) , 5,pinb,2
'reading is also different
Ar(1) = 1wread(8, Pinb, 2)
                                                             'read 8
bytes from portB on pin 2
For I = 1 To 8
 Print Hex(ar(i));
Next
'you could create a loop with a variable for the bit number !
For I = 0 To 3
                                                              'for pin 0-3
  lwreset Pinb , I
  lwwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8, Pinb, I)
  For A = 1 To 8
   Print Hex(ar(a));
 Next
  Print
Next
End
```

6.64 1WSEARCHFIRST

Action

This statement reads the first ID from the 1wire bus into a variable(array).

Syntax

```
var2 = 1WSEARCHFIRST()
var2 = 1WSEARCHFIRST( port , pin)
```

Remarks

380 BASCOM-AVR

var2	A variable or array that should be at least 8 bytes long that will be assigned with the 8 byte ID from the first 1 wire device on the bus.
port	The PIN port name like PINB or PIND.
pin	The pin number of the port. In the range from 0-7. Maybe a numeric constant or variable.

The 1wireSearchFirst() function must be called once to initiate the ID retrieval process. After the 1wireSearchFirst() function is used you should use successive function calls to the 1wSearchNext[382] function to retrieve other ID's on the bus.

A string can not be assigned to get the values from the bus. This because a null may be returned as a value and the null is also used as a string terminator.

I would advice to use a byte array as shown in the example.

The 1wirecount function will take 4 bytes of SRAM. _____1w__bitstorage , Byte used for bit storage : lastdeviceflag bit 0 id__bit bit 1 cmp_id__bit bit 2 search_dir bit 3 ____1wid__bit__number, Byte ____1wlast_zero, Byte ____1wlast_discrepancy , Byte

ASM

The following asm routines are called from mcs.lib. _1wire_Search_First : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS) Parameters passed : R24 : pin number, R30 : port , X : address of target array Returns nothing.

See also

1WWRITE 3388), 1WRESET 3751, 1WREAD 377), 1WSEARCHNEXT 3827, 1WIRECOUNT 3721

Example

! _ _ _ _ _ _

```
'name
                     : 1wireSearch.bas
                     : (c) 1995-2005, MCS Electronics
'copyright
                     : demonstrates 1wsearch
'purpose
                     : Mega48
'micro
'suited for demo : yes
'commercial addon needed : no
·_____
                                    _____
 _____
$regfile = "m48def.dat"
$crystal = 4000000
$hwstack = 32
                                                   ' default
use 32 for the hardware stack
\$swstack = 10
                                                   'default use
10 for the SW stack
```

```
$framesize = 40
                                                              'default use
40 for the frame space
                                                              'use this
Config 1wire = Portb.0
pin
'On the STK200 jumper B.0 must be inserted
'The following internal bytes are used by the scan routines
  ____1w_bitstorage , Byte used for bit storage :
       lastdeviceflag bit 0
       id bit
                      bit 1
       cmp id bit
                      bit 2
       search_dir
                      bit 3
 ____1wid_bit_number, Byte
 ____1wlast_zero, Byte
  __lwlast_discrepancy , Byte
   _1wire_data , string * 7 (8 bytes)
'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte
'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word
'Now search for the first device on the bus
\text{Reg no(1)} = 1 \text{wsearchfirst()}
For I = 1 To 8
                                                              'print the
number
    Print Hex(reg_no(i));
Next
Print
Do
  'Now search for other devices
  Reg_no(1) = 1wsearchnext()
  For I = 1 To 8
     Print Hex(reg_no(i));
  Next
  Print
Loop Until Err = 1
'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1 wirecount()
'It is IMPORTANT that the lwirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W
'as a bonus the next routine :
' first fill the array with an existing number
Reg no(1) = 1wsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
lwverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
```

```
382 BASCOM-AVR
```

```
' optinal call it with pinnumber line lwverify reg_no(1),pinb,1
'As for the other lwire statements/functions, you can provide the port
and pin number as anoption
'W = lwirecount(pinb , 1) 'for
example look at pin PINB.1
End
```

6.65 1WSEARCHNEXT

Action

This statement reads the next ID from the 1wire bus into a variable(array).

Syntax

```
var2 = 1WSEARCHNEXT()
var2 = 1WSEARCHNEXT( port , pin)
```

Remarks

	A variable or array that should be at least 8 bytes long that will be assigned with the 8 byte ID from the next 1 wire device on the bus.
Port	The PIN port name like PINB or PIND.
	The pin number of the port. In the range from 0-7. May be a numeric constant or variable.

The 1wireSearchFirst() function must be called once to initiate the ID retrieval process. After the 1wireSearchFirst() function is used you should use successive function calls to the 1wireSearchNext function to retrieve other ID's on the bus.

A string can not be assigned to get the values from the bus. This because a null may be returned as a value and the null is also used as a string terminator.

I would advice to use a byte array as shown in the example.

The 1wirecount function will take 4 bytes of SRAM.

____1w_bitstorage , Byte used for bit storage :
lastdeviceflag bit 0
id_bit bit 1
cmp_id_bit bit 2
search_dir bit 3
____1wid_bit_number, Byte
___1wlast_zero, Byte
___1wlast_discrepancy , Byte

ASM

The following asm routines are called from mcs.lib.

_1wire_Search_Next : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS) Parameters passed : R24 : pin number, R30 : port , X : address of target array Returns nothing.

See also

1WWRITE 388), 1WRESET 375), 1WREAD 377), 1WSEARCHFIRST 379), 1WIRECOUNT 372)

```
Example
                   _____
                       : lwireSearch.bas
'name
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : demonstrates 1wsearch
'micro
                       : Mega48
'suited for demo
                       : yes
'commercial addon needed : no
.....
                                    _____
_____
$regfile = "m48def.dat"
$crystal = 4000000
                                                       ' default
$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
                                                       'default use
10 for the SW stack
$framesize = 40
                                                       'default use
40 for the frame space
                                                       'use this
Config 1wire = Portb.0
pin
'On the STK200 jumper B.0 must be inserted
'The following internal bytes are used by the scan routines
 ____1w_bitstorage , Byte used for bit storage :
      lastdeviceflag bit 0
              bit 1
      id_bit
                 bit 2
bit 3
      cmp_id_bit
      search_dir
   _1wid_bit_number, Byte
 ____1wlast_zero, Byte
  '____1wire_data , string * 7 (8 bytes)
'[DIM variables used]
'we need some space from at least 8 bytes to store the ID
Dim Reg_no(8) As Byte
'we need a loop counter and a word/integer for counting the ID's on the
bus
Dim I As Byte , W As Word
'Now search for the first device on the bus
Reg_no(1) = lwsearchfirst()
For I = 1 To 8
                                                       'print the
number
   Print Hex(reg_no(i));
Next
Print
Do
 'Now search for other devices
 Reg_no(1) = lwsearchnext()
 For I = 1 To 8
```

```
Print Hex(reg_no(i));
  Next
  Print
Loop Until Err = 1
'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1 wirecount()
'It is IMPORTANT that the lwirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W
'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = 1wsearchfirst()
 unremark next line to chance a byte to test the ERR flag
'Reg no(1) = 2
'now verify if the number exists
lwverify Reg_no(1)
Print Err
'err =1 when the ID passed n req no() does NOT exist
' optinal call it with pinnumber line lwverify reg_no(1),pinb,1
'As for the other lwire statements/functions, you can provide the port
and pin number as anoption
'W = lwirecount(pinb , 1)
                                                              'for
example look at pin PINB.1
End
```

6.66 1WVERIFY

Action

This verifies if an ID is available on the 1wire bus.

Syntax

1WVERIFY ar(1)

Remarks

Ar(1) A byte array that holds the ID to verify.

Returns ERR set to 0 when the ID is found on the bus otherwise it will be 1.

ASM

The following asm routines are called from mcs.lib. _1wire_Search_Next : (calls _1WIRE, _ADJUST_PIN , _ADJUST_BIT_ADDRESS)

See also

1WWRITE 3887, 1WRESET 3757, 1WREAD 3777, 1WSEARCHFIRST 3797, 1WIRECOUNT 3727

Example

_____ _____ 'name : 1wireSearch.bas : (c) 1995-2005, MCS Electronics 'copyright 'purpose : demonstrates 1wsearch 'micro : Mega48 'suited for demo : yes 'commercial addon needed : no *_____ _____ \$regfile = "m48def.dat" \$crystal = 4000000\$hwstack = 32 ' default use 32 for the hardware stack \$swstack = 10'default use 10 for the SW stack **\$framesize** = 40 'default use 40 for the frame space 'use this **Config** 1wire = Portb.0 pin 'On the STK200 jumper B.0 must be inserted 'The following internal bytes are used by the scan routines '____1w_bitstorage , Byte used for bit storage : lastdeviceflag bit 0 id_bit bit 1 cmp_id_bit bit 2 search_dir bit 3 ____1wid_bit_number, Byte ____1wlast_zero, Byte ____1wlast_discrepancy , Byte ____1wire_data , string * 7 (8 bytes) '[DIM variables used] 'we need some space from at least 8 bytes to store the ID Dim Reg_no(8) As Byte 'we need a loop counter and a word/integer for counting the ID's on the bus Dim I As Byte , W As Word 'Now search for the first device on the bus Reg no(1) = 1 wsearchfirst()For I = 1 To 8 'print the number Print Hex(reg_no(i)); Next Print Do 'Now search for other devices $Reg_no(1) = 1wsearchnext()$ For I = 1 To 8 Print Hex(reg_no(i)); Next Print Loop Until Err = 1

```
'When ERR = 1 is returned it means that no device is found anymore
'You could also count the number of devices
W = 1wirecount()
'It is IMPORTANT that the lwirecount function returns a word/integer
'So the result variable must be of the type word or integer
'But you may assign it to a byte or long too of course
Print W
'as a bonus the next routine :
' first fill the array with an existing number
Reg_no(1) = lwsearchfirst()
' unremark next line to chance a byte to test the ERR flag
'Reg_no(1) = 2
'now verify if the number exists
lwverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line lwverify reg_no(1),pinb,1
'As for the other lwire statements/functions, you can provide the port
and pin number as anoption
'W = lwirecount(pinb , 1)
                                                              'for
example look at pin PINB.1
End
```

6.67 **1WWRITE**

Action

This statement writes a variable to the 1wire bus.

Syntax

1WWRITE var1
1WWRITE var1, bytes
1WWRITE var1 , bytes , port , pin

Remarks

var1	Sends the value of var1 to the bus. The number of bytes can be specified too but this is optional.
bytes	The number of bytes to write. Must be specified when port and pin are used.
port	The name of the PORT PINx register like PINB or PIND.
port pin	The pin number in the range from 0-7. May be a numeric constant or variable.

Multiple 1-wire devices on different pins are supported.

To use this you must specify the port and pin that are used for the communication.

The 1wreset, 1wwrite and 1wread statements will work together when used with the old syntax. And the pin can be configured from the compiler options or with the <u>CONFIG 1WIRE at statement</u>.

The syntax for additional 1-wire devices is : 1WRESET port , pin

1WWRITE var/constant, bytes, port , pin var = 1WREAD(bytes, port, pin) ,for reading multiple bytes

See also

1WREAD 3777 , 1WRESET 375

```
_____
'name
                        : lwire.bas
'copyright
                        : (c) 1995-2005, MCS Electronics
'purpose
                        : demonstrates 1wreset, 1wwrite and 1wread()
                        : Mega48
'micro
'suited for demo
                         : yes
'commercial addon needed : no
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
......
_____
$regfile = "m48def.dat"
$crystal = 4000000
$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 10
                                                          'default use
10 for the SW stack
$framesize = 40
                                                          'default use
40 for the frame space
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"
                                                          'use this
Config 1wire = Portb.0
pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte
Do
 Wait 1
 1wreset
                                                          'reset the
device
 Print Err
                                                          'print error
1 if error
 lwwrite &H33
                                                          'read ROM
command
 For I = 1 To 8
   Ar(i) = 1wread()
                                                          'place into
array
 Next
'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)
                                                           'read 8
bytes
 For I = 1 To 8
    Print Hex(ar(i));
                                                          'print
```

```
output
 Next
                                                               'linefeed
  Print
Loop
'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!
'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
 Ar(i) = 0
                                                               'clear array
to see that it works
Next
lwreset Pinb , 2
                                                               'use this
port and pin for the second device
lwwrite &H33 , 1 , Pinb , 2
now the number of bytes must be specified!
                                                               'note that
'lwwrite Ar(1) , 5,pinb,2
'reading is also different
Ar(1) = 1wread(8, Pinb, 2)
                                                               'read 8
bytes from portB on pin 2
For I = 1 To 8
  Print Hex(ar(i));
Next
'you could create a loop with a variable for the bit number !
For I = 0 To 3
                                                                'for pin 0-3
  lwreset Pinb , I
  lwwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8, Pinb, I)
  For A = 1 To 8
   Print Hex(ar(a));
 Next
 Print
Next
End
```

6.68 ABS

Action

Returns the absolute value of a numeric signed variable.

Syntax

var = **ABS**(var2)

Remarks

Var	Variable that is assigned with the absolute value of var2.
Var2	The source variable to retrieve the absolute value from.

var : Integer , Long, Single or Double. var2 : Integer, Long, Single or Double. The absolute value of a number is always positive.

See also

NONE

ASM

Calls: _abs16 for an Integer and _abs32 for a Long Input: R16-R17 for an Integer and R16-R19 for a Long Output:R16-R17 for an Integer and R16-R19 for a Long

Calls _Fltabsmem for a single from the fp_trig library.

Example

```
Dim a as Integer, c as Integer
a =-1000
c = Abs(a)
Print c
End
```

6.69 ACOS

Action

Returns the arccosine of a single in radians.

Syntax

var = **ACOS**(x)

Remarks

A floating point variable such as single or double, that is assigned with the ACOS of variable x.
The float to get the ACOS of. Input is valid from -1 to $+1$ and returns p to 0.
If Input is < -1 than p and input is > 1 than 0 will returned.

If Input is cause of rounding effect in float-operations a little bit over 1 or -1, the value for 1.0 (-1.0) will be returned. This is the reason to give the value of the limit-point back, if Input is beyond limit. Generally the user have to take care, that Input to this function lies within -1 to +1.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD [649], COS [596], SIN [885], TAN [926], ATN [403], ASIN [402], ATN2 [404]

390 BASCOM-AVR

```
$regfile = "m48def.dat"
                                                              ' specify
the used micro
$crystal = 8000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

```
Dim S As Single , X As Single
x= 0.5 : S = Acos(x)
Print S
End
```

6.70 ADR, ADR2

Action

Create label address.

Syntax

ADR label ADR2 label

Remarks

label The name of a label.

The AVR uses WORD addresses. ADR will create the word address. To find a byte in memory, you need to multiply by 2. For this purpose ADR2 is available. It will create the address of the label multiplied by 2.

Using ADR2 you can use tables. The sample program demonstrates this together with some more advanced ASM code.

The sample includes ADR2.LIB. This lib contains a special version of _MoveConst2String .

The normal routine in MCS.LIB will stop printing once a null byte (zero) is encountered that indicates the end of a string.

But for the sample program, we may not change the address, so the address is restored when the null byte is found.

See Also NONE

BASCOM Language Reference

391

```
'______
' This is an example of how to create an interactive menu system supporting
' sub-menus and support routines using the !ADR and !ADR2 statements
$regfile = "M644def.dat"
$crystal = 8000000
\$hwstack = 64
                                                    ' specify the hardware
\$swstack = 64
                                                     ' specify the software
                                                    ' specify the framesize
$framesize = 64
$lib "adr2.lib"
*_____
                                   _____
Dim Menupointer As Word
Dim Actionpointer As Word
Dim Entries As Byte
Dim Dummy As Byte
Dim Message As String * 32
Dim Locall As Byte
Dim Local_loop1 As Byte
Const Menu id = & HAA
                                                    ' sub-menu ID byte
                                                    ' service routine ID by
Const Routine id = &H55
                                    _____
                                                    ' point to the start of
Restore Main_menu
sts {MenuPointer}, R8
                                                      { store the pointer t
sts {MenuPointer + 1}, R9
Display_new_menu:
lds R8, {MenuPointer}
                                                      } restore the pointer
lds R9, {MenuPointer + 1}
                                                    ' get the number of ent
Read Entries
Print
For Local_loop1 = 1 To Entries
 Read Message
                                                     ' read the message
 Print Message
                                                    ' send it to the consol
Next
Read Dataptr
                                                    ' get the pointer to the
sts {ActionPointer}, R8
sts {ActionPointer + 1}, R9
                                                      } store the pointer t
Input "Entry ? " , Local1
                                                    ' ask the user which me
If Local1 = 0 Then
                                                    ' is it valid ?
 Goto Display_new_menu
                                                    ' if not, re-display th
End If
If Local1 => Entries Then
                                                    ' is it valid ?
 Goto Display_new_menu
                                                    ' if not, re-display th
End If
lds R8,{ActionPointer}
lds R9,{ActionPointer + 1}
                                                       restore the pointer
If Local1 <> 1 Then
    For Local_loop1 = 2 To Local1
```

```
τ.
  ldI R30,4
                                                i.
  clr R1
                                                \mathbf{r}
  add R8,R30
                                                τ.
  adc R9,R1
                                                   calculate the locat
   Next
End If
Read Local1
                                                ' get the menu entry's
                                                ' to handle the uP expe
Read Dummy
If Local1 = Menu_id Then
                                                ' did the user select a
 Read Dataptr
 sts {MenuPointer}, R8
                                                ' } store the start of
 sts {MenuPointer + 1}, R9
 Goto Display_new_menu
End If
                                                ' get the address of the
Read Dataptr
movw R30,R8
icall
                                                ' pass control to the e
                                                ' re-display the last r
Goto Display_new_menu
 _____
  Test support routines
......
                   _____
Hello_message:
Print
Print "You asked to print 'Hello'"
                                                ' confirmation that Mer
Return
2nd_menu_1st_entry_routine:
Print
Print "You selected Entry 1 of the 2nd menu"
                                                ' confirmation that Mer
Return
2nd_menu_2nd_entry_routine:
Print
Print "You selected Entry 2 of the 2nd menu"
                                                ' confirmation that Mer
Return
3rd_menu_1st_entry_routine:
Print
Print "You selected Entry 1 of the 3rd menu"
                                                ' confirmation that Mer
Return
3rd_menu_2nd_entry_routine:
Print
Print "You selected Entry 2 of the 3rd menu"
                                                ' confirmation the Menu
Return
End
' Data Statements
*_____
```

\$data

*_____ ' Main Menu _____ Main_menu: Data 4 ' number of entries in Data "MAIN MENU" ' } menu title Data "1. Go to Menu 2" ' } 1st menu entry ' } 2nd menu entry Data "2. Go to Menu 3" ' } 3rd menu entry Data "3. Print 'Hello' message" ' point to this menu su Adr2 Mainmenu_supporttable _____ Mainmenu_supporttable: ' identify this menu er Data Menu_id Adr2 Second_menu ' address of next menu Data Menu_id ' identify this menu er Adr2 Third_menu ' address of next menu **Data** Routine_id ' identify this menu en ' address of the support Adr Hello_message ·_____ ' Second Menu ·_____ Second_menu: Data 4 ' number of entries in ' } menu title Data "SECOND MENU" } lst menu entry
} 2nd menu entry Data "1. 2nd Menu Entry #1" Data "2. 2nd Menu Entry #2" ' } 3rd menu entry Data "3. Go to previous menu" Adr2 Secondmenu_supporttable ' point to this menu su /_____ Secondmenu_supporttable: **Data** Routine_id ' identify this menu er Adr 2nd_menu_1st_entry_routine ' support routine for 1 ' identify this menu er **Data** Routine_id Adr 2nd_menu_2nd_entry_routine ' support routine for 2 ' identify this menu er Data Menu id Adr2 Main_menu ' support routine for 3 · _____ ' Third Menu

```
Third_menu:
```

```
Data 4
```

' number of entries in

394 BASCOM-AVR

Data "THIRD MENU" Data "1. 3rd Menu Entry #1" Data "2. 3rd Menu Entry #2" Data "3. Go to previous menu"	<pre>' } menu title ' } 1st menu entry ' } 2nd menu entry ' } 3rd menu entry</pre>
Adr2 Thirdmenu_supporttable	' point to this menu su
· ·	
Thirdmenu_supporttable:	
Data Routine_id Adr 3rd_menu_1st_entry_routine	' identify this menu er ' support routine for 1
Data Routine_id Adr 3rd_menu_2nd_entry_routine	' identify this menu er ' support routine for 2
Data Menu_id Adr2 Main menu	' identify this menu er ' support routine for 3

6.71 AESDECRYPT

Action

This statement of function uses the Xmega AES encryption engine to decrypt a block of data.

Syntax

AESDECRYPT key, var, size targ = **AESDECRYPT** (key, var, size)

Remarks

key	The name of a label that contains 16 bytes of key data. Or an array holding 16 bytes of key data.
var	A variable or array containing the data to be encrypted. When you use the statement, this variable will contains the encrypted data after the conversion.
size	The number of bytes to encrypt. Encryption is done with blocks of 16 bytes. So the size should be a multiple of 16. If you supply only 14 bytes this is ok too, but the result will still be 16 bytes. It is important that your array is big enough to hold the result. Without the full 16 byte result, you can not decrypt the data.
targ	In case you use the function, this variable will hold the result.

This function only works for Xmega chips that have an AES encryption unit. 128 bit encryption is used.

You can either use a label with a fixed key, or use a variable. You should use the same key data for encryption and decryption.

See also

```
_____
                 (c) 1995-2010, MCS
                     xm128-AES.bas
  This sample demonstrates the Xmega128A1 AES encryption/decryption
   _____
$regfile = "xm128a1def.dat"
$crystal = 3200000
$hwstack = 64
\$swstack = 40
$framesize = 40
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 38400 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
'$external _aes_enc
                                                          ' room for
Dim Key(16) As Byte
key
Dim Ar(34) As Byte
Dim Arenc(34) As Byte
Dim J As Byte
Print "AES test"
Restore Keydata
For J = 1 To 16
                                                          ' load a key
to memory
  Read Key(j)
Next
'load some data
For J = 1 To 32
                                                          ' fill some
data to encrypt
 Ar(j) = J
Next
Aesencrypt Keydata , Ar(1) , 32
Print "Encrypted data"
For J = 1 To 32
                                                          ' fill some
data to encrypt
 Print Ar(j)
Next
Aesdecrypt Keydata , Ar(1) , 32
Print "Decrypted data"
```

```
For J = 1 To 32
                                                              ' fill some
data to encrypt
 Print Ar(j)
Next
Print "Encrypt function"
Arenc(1) = Aesencrypt(keydata , Ar(1) , 32)
                                                              ' fill some
For J = 1 To 32
data to encrypt
 Print Ar(j) ; "-" ; Arenc(j)
Next
Print "Decrypt function"
Ar(1) = Aesdecrypt(keydata , Arenc(1) , 32)
For J = 1 To 32
Print J ; ">" ; Ar(j) ; "-" ; Arenc(j)
Next
End
```

```
Keydata:
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 ,
16
```

6.72 AESENCRYPT

Action

This statement of function uses the Xmega AES encryption engine to encrypt a block of data.

Syntax

AESENCRYPT key, var , size targ = AESENCRYPT (key, var , size)

Remarks

key	The name of a label that contains 16 bytes of key data. Or an array holding 16 bytes of key data.
var	A variable or array containing the data to be encrypted. When you use the statement, this variable will contains the encrypted data after the conversion.
size	The number of bytes to encrypt. Encryption is done with blocks of 16 bytes. So the size should be a multiple of 16. If you supply only 14 bytes this is ok too, but the result will still be 16 bytes. It is important that your array is big enough to hold the result. Without the full 16 byte result, you can not decrypt the data.
targ	In case you use the function, this variable will hold the result.

This function only works for Xmega chips that have an AES encryption unit. 128 bit encryption is used.

You can either use a label with a fixed key, or use a variable. You should use the same key data for encryption and decryption. See also

AESDECRYPT 394

```
------
                 (c) 1995-2010, MCS
                     xm128-AES.bas
  This sample demonstrates the Xmega128A1 AES encryption/decryption
    _____
$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
\$swstack = 40
$framesize = 40
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 38400 , Mode = Asynchroneous , Parity = None , Stopbits =
1, Databits = 8
'$external _aes_enc
                                                          ' room for
Dim Key(16) As Byte
key
Dim Ar(34) As Byte
Dim Arenc(34) As Byte
Dim J As Byte
Print "AES test"
Restore Keydata
For J = 1 To 16
                                                          ' load a kev
to memory
  Read Key(j)
Next
'load some data
For J = 1 To 32
                                                          ' fill some
data to encrypt
 Ar(j) = J
Next
Aesencrypt Keydata , Ar(1) , 32
Print "Encrypted data"
For J = 1 To 32
                                                          ' fill some
data to encrypt
  Print Ar(j)
Next
```

```
Aesdecrypt Keydata , Ar(1) , 32
Print "Decrypted data"
For J = 1 To 32
                                                              ' fill some
data to encrypt
  Print Ar(j)
Next
Print "Encrypt function"
Arenc(1) = Aesencrypt(keydata , Ar(1) , 32)
For J = 1 To 32
                                                              ' fill some
data to encrypt
  Print Ar(j) ; "-" ; Arenc(j)
Next
Print "Decrypt function"
Ar(1) = Aesdecrypt(keydata , Arenc(1) , 32)
For J = 1 To 32
 Print J ; ">" ; Ar(j) ; "-" ; Arenc(j)
Next
End
```

```
Keydata:
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 ,
16
```

6.73 ALIAS

Action

Indicates that the variable can be referenced with another name.

Syntax

newvar ALIAS oldvar

Remarks

oldvar	Name of the variable such as PORTB.1
newvar	New name of the variable such as direction

Aliasing port pins can give the pin names a more meaningful name. For example, when your program uses 4 different pins to control 4 different relays, you could name them portb.1, portb.2, portb.3 and portb.4.

But it would be more convenient to refer to them as relais1, relais2, relais3 and realais4.

When you later on change your PCB and decide that relays 4 must be connected to portD.4 instead of portb.4, you only need to change the ALIAS line, and not your whole program.



Example

'		
'micro 'suited for demo 'commercial addon needed	-	
<pre>\$regfile = "m48def.dat" \$crystal = 4000000 crystal</pre>		' 4 MHz
Const On = 1 Const Off = 0		
Config Portb = Output		
Relais1 Alias Portb.1		
Relais2 Alias Portb.2		
Relais3 Alias Portd.5		
Relais4 Alias Portd.2		
Set Relais1		
Relais $2 = 0$		
Relais3 = On		
Relais4 = Off		
End		

6.74 ASC

Action

Assigns a numeric variable with the ASCII value of the first character of a string.

Syntax

var = **ASC**(string)

Remarks

Var	Target numeric variable that is assigned.
String	String variable or constant from which to retrieve the ASCII value.

Note that only the first character of the string will be used. When the string is empty, a zero will be returned.

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is they want 'plain' text with no formatting such as

tabs, bold or underscoring - the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues. Notepad.exe creates ASCII text, or in MS Word you can save a file as 'text only'

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	0	96	60	
1	01	Start of heading	33	21	1	65	41	A	97	61	a
2	02	Start of text	34	22	<i>11</i>	66	42	в	98	62	b
3	03	End of text	35	23	#	67	43	С	99	63	c
4	04	End of transmit	36	24	Ş	68	44	D	100	64	d
5	05	Enquiry	37	25	*	69	45	Е	101	65	e
6	06	Acknowledge	38	26	£	70	46	F	102	66	f
7	07	Audible bell	39	27	a -	71	47	G	103	67	g
8	08	Backspace	40	28	(72	48	Н	104	68	h
9	09	Horizontal tab	41	29)	73	49	I	105	69	i
10	OA	Line feed	42	2A	*	74	4A	J	106	6A	j
11	OB	Vertical tab	43	2 B	+	75	4B	ĸ	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	1
13	OD	Carriage return	45	2D		77	4D	М	109	6D	m
14	OE	Shift out	46	2 E	•	78	4E	N	110	6E	n
15	OF	Shift in	47	2 F	/	79	4F	0	111	6F	o
16	10	Data link escape	48	30	O	80	50	Р	112	70	р
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	Т	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	v	118	76	v
23	17	End trans. block	55	37	7	87	57	ឃ	119	77	w
24	18	Cancel	56	38	8	88	58	x	120	78	х
25	19	End of medium	57	39	9	89	59	Y	121	79	У
26	1A	Substitution	58	ЗA	:	90	5A	Z	122	7A	z
27	1B	Escape	59	ЗB	;	91	5B	C	123	7B	{
28	1C	File separator	60	3C	<	92	5C	١	124	7C	1
29	1D	Group separator	61	ЗD	=2	93	5D]	125	7D	}
30	1E	Record separator	62	ЗE	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	ЗF	2	95	5F	1000	127	7F	

Extended ASCII

As people gradually required computers to understand additional characters and nonprinting characters the ASCII set became restrictive. As with most technology, it took a while to get a single standard for these extra characters and hence there are few varying 'extended' sets. The most popular is presented below.

401

Dec	Hex	Char	Dec	Нех	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	ç	160	AO	á	192	CO	L	224	EO	α
129	81	ü	161	A1	í	193	C1	Т	225	E1	ß
130	82	é	162	A2	ó	194	C2	т	226	E 2	Г
131	83	â	163	A 3	ú	195	C3	F	227	E3	п
132	84	ä	164	A4	ñ	196	C4		228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	+	229	E5	ਰੁ
134	86	å	166	A6	2	198	C6	F	230	E6	μ
135	87	ç	167	A7	•	199	C7	⊩	231	E7	τ
136	88	ê	168	A8	č	200	C8	L	232	E8	Φ
137	89	ë	169	A9	-	201	C9	F	233	E9	۲
138	8A	è	170	AA	7	202	CA	ᄟ	234	EA	Ω
139	8B	ï	171	AB	₩2	203	CB	T	235	EB	δ
140	8C	î	172	AC	4	204	CC	ŀ	236	EC	œ
141	8D	ì	173	AD	i	205	CD		237	ED	ଛ
142	8 E	Ä	174	AE	«	206	CE	÷	238	EE	ε
143	8F	Å	175	AF	»	207	CF	⊥	239	EF	n
144	90	É	176	BO	38 8	208	DO	ш	240	FO	=
145	91	æ	177	B1		209	D1	Ŧ	241	F1	±
146	92	Æ	178	B2		210	D2	π	242	F2	≥
147	93	ô	179	B3	I.	211	D3	L	243	FЗ	≤
148	94	ö	180	В4	4	212	D4	F	244	F4	ſ
149	95	ò	181	B5	4	213	D5	F	245	F5	J
150	96	û	182	B6	-1	214	D6	г	246	F6	÷
151	97	ù	183	В7	п	215	D7	#	247	F7	*
152	98	ÿ	184	B8	F	216	D8	ŧ	248	F8	•
153	99	Ö	185	B9	╡	217	D9	L	249	F9	۰.
154	9A	Ü	186	BA	1	218	DA	г	250	FA	9
155	9B	¢	187	BB	า	219	DB		251	FB	٧
156	90	£	188	BC	П	220	DC	-	252	FC	ъ
157	9D	¥	189	BD	ш	221	DD	L	253	FD	z
158	9E	R.	190	BE	4	222	DE	1	254	FE	•
159	9F	f	191	BF	1	223	DF	-	255	FF	

See also

CHR 428

ASM

NONE

Example
\$regfile = "m48def.dat" the used micro **\$crystal** = 8000000 crystal frequency **\$baud** = 19200 rate \$hwstack = 32 use 32 for the hardware stack \$swstack = 10

' specify

' used

' use baud

' default

' default

```
use 10 for the SW stack
$framesize = 40 ' default
use 40 for the frame space
Config Coml = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim A As Byte , S As String * 10
s = "ABC"
A = Asc(s)
Print A 'will print
65
End
```

6.75 ASIN

Action

Returns the arcsine of a single in radians.

Syntax

var = ASIN(x)

Remarks

	A float variable such as single or double that is assigned with the ASIN of variable x.
х	The float to get the ASIN of. Input is valid from -1 to $+1$ and returns $-p/2$ to $+p/2$.
	If Input is < -1 than -p/2 and input is > 1 than p/2 will returned.

If Input is cause of rounding effect in single-operations a little bit over 1 or -1, the value for 1.0 (-1.0) will be returned. This is the reason to give the value of the limit-point back, if Input is beyond limit. Generally the user have to take care, that Input to this function lies within -1 to +1.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [8227], DEG2RAD [6497], COS [5967], SIN [8857], TAN [9207], ATN [4037], ACOS [3897], ATN2 [4047]

Example

\$regfile = "m48def.dat"
the used micro
\$crystal = 8000000
crystal frequency
\$baud = 19200
rate
\$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
use 10 for the SW stack

- ' specify ' used ' use baud ' default
- ' default

```
$framesize = 40 ' default
use 40 for the frame space
Config Coml = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As Single , X As Single
X = 0.5 : S = Asin(x)
Print S '0.523595867
```

End

6.76 ATN

Action

Returns the Arctangent of a single in radians.

Syntax

var = **ATN**(single)

Remarks

	A float variable that is assigned with the arctangent of variable single.
Single	The float variable to get the arctangent of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD [649], COS [596], SIN [885], TAN [920], ATN2 [404]

```
$regfile = "m48def.dat"
                                                              ' specify
the used micro
$crystal = 8000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As Single , X As Single
S = Atn(1) * 4
Print S ' prints 3.141593 PI
End
```

6.77 ATN2

Action

ATN2 is a four-quadrant arc-tangent.

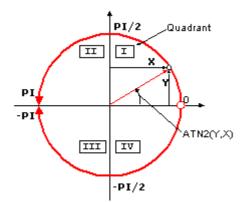
While the ATN-function returns from -p/2 (-90°) to p/2 (90°), the ATN2 function returns the whole range of a circle from -p (-180°) to +p (180°). The result depends on the ratio of Y/X and the signs of X and Y.

Syntax

var = **ATN2**(y, x)

Remarks

Var	A single variable that is assigned with the ATN2 of variable single.
Х	The single variable with the distance in x-direction.
Y	The single variable with the distance in y-direction



Quadrant	Sign Y	Sign X	ATN2
Ι	+	+	0 to p/2
II	+	-	p/2 to p
III	-	-	-p/2 to -p
IV	-	+	0 to -p/2

If you go with the ratio Y/X into ATN you will get same result for X greater zero (right side in coordinate system) as with ATN2. ATN2 uses X and Y and can give information of the angle of the point over 360° in the coordinates system.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD [649], COS [596], SIN [885], TAN [920], ATN [403]

Example

\$regfile = "m48def.dat"
the used micro
\$crystal = 8000000
crystal frequency

' specify

' used

```
$baud = 19200
                                                              ' use baud
rate
                                                              ' default
$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As Single , X As Single
X = 0.5 : S = 1.1
S = Atn2(s, X)
Print S ' prints 1.144164676
```

End

6.78 BASE64DEC

Action

Converts Base-64 data into the original data.

Syntax

Result = **BASE64DEC**(source)

Remarks

Result	A string variable that is assigned with the un-coded string.
Source	The source string that is coded with base-64.

Base-64 is not an encryption protocol. It sends data in 7-bit ASCII data format. MIME, web servers, and other Internet servers and clients use Base-64 coding.

The provided Base64Dec() function is a decoding function. It was written to add authentication to the web server sample.

When the web server asks for authentication, the client will send the user and password unencrypted, but base-64 coded to the web server.

Base-64 coded strings are always in pairs of 4 bytes. These 4 bytes represent 3 bytes.

See also

<u>CONFIG TCPIP</u> 55को, <u>GETSOCKET</u> 717े, <u>SOCKETCONNECT</u> बिको, <u>SOCKETSTAT</u> बिको , <u>TCPWRITE</u> जि2को, <u>TCPWRITESTR</u> जि2को, <u>CLOSESOCKET</u> विजे , <u>SOCKETLISTEN</u> बिको , <u>BASE64ENC</u> विको

1	specify
1	used
1	use baud
1	default
	,

```
use 32 for the hardware stack
$swstack = 10 ' default
use 10 for the SW stack
$framesize = 40 ' default
use 40 for the frame space
$lib "tcpip.lbx"
Config Coml = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As String * 15 , Z As String * 15
S = "bWFyazptYXJr"
Z = Base64dec(s)
Print Z 'mark:mark
```

End

6.79 BASE64ENC

Action

Converts a string into the Base-64 representation.

Syntax

Result = **BASE64ENC**(source)

Remarks

Result	A string variable that is assigned with the coded string.
Source	The source string that must be code with base-64.

Base-64 is not an encryption protocol. It sends data in 7-bit ASCII data format. MIME, web servers, and other Internet servers and clients use Base-64 coding.

The provided Base64Enc() function is an encoding function. You need it when you want to send attachments with POP3 for example.

The target string will use 1 additional byte for every 3 bytes.

So make sure the target string is dimensioned longer then the original string.

See also

<u>CONFIG TCPIP</u> िङको, <u>GETSOCKET</u> िगो , <u>SOCKETCONNECT</u> अको, <u>SOCKETSTAT</u> अको , <u>TCPWRITE</u> विश्वे, <u>TCPWRITESTR</u> विश्वे, <u>CLOSESOCKET</u> विश्वे , <u>SOCKETLISTEN</u> अको , <u>BASE64DEC</u> विको

<pre>\$regfile = "m48def.dat"</pre>	1	specify
the used micro		
\$crystal = 8000000	1	used
crystal frequency		
\$baud = 19200	1	use baud
rate		
<pre>\$hwstack = 32</pre>	1	default
use 32 for the hardware stack		
<pre>\$swstack = 10</pre>	1	default
use 10 for the SW stack		

```
$framesize = 40 ' default
use 40 for the frame space
$lib "tcpip.lbx"
Config Coml = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As String * 15 , Z As String * 15
S = "bWFyazptYXJr"
Z = Base64dec(s)
Print Z 'mark:mark
s = Base64Enc(z)
Print s
End
```

6.80 BAUD

Action

Changes the baud rate for the hardware UART.

Syntax

BAUD = var **BAUD** #x , const

Remarks

Var	The baud rate that you want to use.
Х	The channel number of the software UART.
Const	A numeric constant for the baud rate that you want to use.

L Do not confuse the BAUD statement with the <u>\$BAUD</u> and compiler directive.

And do not confuse \$CRYSTAL 316 and CRYSTAL 608

\$BAUD overrides the compiler setting for the baud rate and BAUD will change the current baud rate.

So \$BAUD is a global project setting in your source code while BAUD will change the baud rate during run time.

You could use BAUD to change the baud rate during run time after the user changes a setting.

BAUD = ... will work on the hardware UART.

BAUD #x, yyyy will work on the software UART.

See also

\$CRYSTAL 3161, \$BAUD 3101, BAUD1 4081



Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

Print "Hello"

'Now change the baud rate in a program
Baud = 9600
Print "Did you change the terminal emulator baud rate too?"
End

6.81 BAUD1

Action

Changes the baud rate for the second hardware UART.

Syntax

BAUD1 = var **BAUD1** #x , const

Remarks

Var	The baud rate that you want to use.
Х	The channel number of the software UART.
Const	A numeric constant for the baud rate that you want to use.

Do not confuse the BAUD1 statement with the \$BAUD1 compiler directive.

And do not confuse \$CRYSTAL 316 and CRYSTAL 608

\$BAUD1 overrides the compiler setting for the baud rate and BAUD1 will change the current baud rate.

 $BAUD1 = \dots$ will work on the hardware UART. BAUD #x, yyyy will work on the software UART.

See also

<u>\$CRYSTAL</u>316, <u>\$BAUD</u>310, <u>\$BAUD1</u>311, <u>BAUD</u>407

ASM

NONE

Example

```
-----
'copyright : (c) 1995-2005, MCS Electronics
```

```
'micro
'suited for demo
```

: Mega162

: yes

6.82 BCD

Action

Converts a variable stored in BCD format into a string.

Syntax

PRINT **BCD**(var) LCD **BCD**(var)

Remarks

Var Numeric variable to convert.

When you want to use an I2C clock device which stores its values in BCD format you can use this function to print the value correctly. BCD() displays values with a leading zero.

The BCD() function is intended for the PRINT/LCD statements. Use the MAKEBCD function to convert variables from decimal to BCD. Use the MAKEDEC function to convert variables from BCD to decimal.

See also

MAKEDEC 779 , MAKEBCD 778

ASM

Calls: _BcdStr Input: X hold address of variable Output: R0 with number of bytes, frame with data.

Example

'name : bcd.bas
'copyright : (c) 1995-2005, MCS Electronics
'purpose : demonstration of split and combine BCD Bytes

410

```
'suited for demo
                   : yes
'commercial addon needed : no
'use in simulator : possible
                                 _____
_____
$regfile = "m48def.dat"
                                              ' specify
the used micro
$crystal = 4000000
                                              ' used
crystal frequency
$baud = 19200
                                              ' use baud
rate
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
\$hwstack = 32
                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                              ' default
use 10 for the SW stack
$framesize = 40
                                              ' default
use 40 for the frame space
*-----
_____
' Set up Variables
========
Dim A As Byte
                                          'Setup A Variable
Dim B As Byte
                                          'Setup B Variable
Dim C As Byte
                                          'Setup C Variable
A = \&H89
*_____
_____
' Main
*_____
========
Main:
Print "Combined : " ; Hex(a)
                                          'Print A
*_____
_____
B = A And &B1111_0000
                                          'Mask To Get Only
High Nibble Of Byte
Shift B , Right , 4
                                          'Shift High
Nibble To Low Nibble Position , Store As B
C = A And &B0000_1111
                                          'Mask To Get Only
Low Nibble Of Byte , Store As \ensuremath{\mathsf{C}}
Print "Split : " ; B ; " " ; C
                                          'Print B (High
Nibble) , C(low Nibble)
               _____
_ _ _ _ _ _ _ _
Shift B , Left , 4
                                          'Shift Data From
Low Nibble Into High Nibble Position
A = B + C
                                          'Add B (High
Nibble) And C(low Nibble) Together
Print "Re-Combined: " ; Hex(a)
                                          'Print A (re -
combined Byte)
End
                                          'End Program
```

© 2011 MCS Electronics

6.83 BIN

Action

Convert a numeric variable into the binary string representation.

Syntax

Var = **Bin**(source)

Remarks

	The target string that will be assigned with the binary representation of the variable source.	
Source	The numeric variable that will be converted.	

The BIN() function can be used to display the state of a port. When the variable source has the value &B10100011 the string named var will be assigned with "10100011".

It can be easily printed to the serial port.

See also

HEX [722], STR [917], VAL [945], HEXVAL [722], BINVAL [412]

ASM

NONE

<pre>\$regfile = "m48def.dat" the used micro</pre>	1	specify
\$crystal = 8000000	1	used
crystal frequency \$baud = 19200	,	use baud
rate \$hwstack = 32 use 32 for the hardware stack	ı.	default
<pre>\$\$ 101 the hardware stack \$\$ swstack = 10 use 10 for the SW stack</pre>	,	default
\$framesize = 40 use 40 for the frame space	T	default
<pre>Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbi Databits = 8 , Clockpol = 0</pre>	.ts	s = 1 ,
Dim B As Byte ' assign value to B B = 45		
<pre>Dim S As String * 10 'convert to string S = Bin(b)</pre>		
'assign value to portb Portb = 33		

```
Print Bin(portb)
```

'of course it also works for other numerics $\ensuremath{\textbf{End}}$

6.84 BINVAL

Action

Converts a string representation of a binary number into a number.

Syntax

var = **Binval**(s)

Remarks

Var	A numeric variable that is assigned with the value of s.
S	Variable of the string type. Should contain only 0 and 1 digits.

See also

<u>STR</u> [917] , <u>HEXVAL</u> [722] , <u>HEX</u> [722] , <u>BIN</u> [417] , <u>VAL</u> [945]

<pre>\$regfile = "m48def.dat"</pre>	,	specify
the used micro \$crystal = 8000000 crystal frequency	ı	used
<pre>\$baud = 19200 rate</pre>	,	use baud
<pre>\$hwstack = 32</pre>	,	default
<pre>use 32 for the hardware stack \$swstack = 10</pre>	ı.	default
use 10 for the SW stack \$framesize = 40 use 40 for the frame space	ı.	default
<pre>Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopb: Databits = 8 , Clockpol = 0</pre>	its	5 = 1 ,
Dim S As String * 8 S = "11001100"		
<pre>Dim B As Byte ' assign value to B B = Binval(s)</pre>		
Print B End		

6.85 BIN2GRAY

Action

Returns the Gray-code of a variable.

Syntax

var1 = Bin2gray(var2)

Remarks

var1	Variable that will be assigned with the Gray code.	
var2	A variable that will be converted.	

Gray code is used for rotary encoders. Bin2gray() works with byte , integer, word and long variables.

The data type of the variable that will be assigned determines if a byte, word or long conversion will be done.

See also

GRAY2BIN 7207, ENCODER 678

ASM

Depending on the data type of the target variable the following routine will be called from mcs.lbx:

_grey2Bin for bytes , _grey2bin2 for integer/word and _grey2bin4 for longs.

```
_____
'name
                        : graycode.bas
'copyright
                        : (c) 1995-2005, MCS Electronics
                        : show the Bin2Gray and Gray2Bin functions
'purpose
                        : Mega48
'micro
'suited for demo
                         : yes
'commercial addon needed : no
$regfile = "m48def.dat"
                                                          ' specify
the used micro
$crystal = 4000000
                                                          ' used
crystal frequency
                                                          ' use baud
$baud = 19200
rate
$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 10
                                                          ' default
use 10 for the SW stack
$framesize = 40
                                                          ' default
use 40 for the frame space
'Bin2Gray() converts a byte, integer, word or long into grey code.
'Gray2Bin() converts a gray code into a binary value
```

' could be

BASCOM-AVR

414

```
Dim B As Byte
word,integer or long too
Print "BIN" ; Spc(8) ; "GREY"
For B = 0 To 15
    Print B ; Spc(10) ; Bin2gray(b)
Next
Print "GREY" ; Spc(8) ; "BIN"
For B = 0 To 15
    Print B ; Spc(10) ; Gray2bin(b)
Next
End
```

6.86 BITWAIT

X

Action

Wait until a bit is set or reset.

Syntax

BITWAIT x , SET/RESET

Remarks

Bit variable or internal register like PORTB.x , where x ranges from 0-7.

When using bit variables make sure that they are set/reset by software otherwise your program will stay in a loop.

When you use internal registers that can be set/reset by hardware such as PINB.0 this doesn't apply since this state can change as a result from for example a key press.

See also

NONE

ASM

Calls: NONE Input: NONE Output: NONE

Code : shown for address 0-31

label1: Sbic PINB.0,label2 Rjmp label1 Label2:

Example

\$regfile = "m48def.dat"

' specify the used mich

BASCOM Language Reference 415

' used crystal frequend scrystal = 8000000**\$baud =** 19200 ' use baud rate \$hwstack = 32' default use 32 for th ' default use 10 for th \$swstack = 10' default use 40 for th \$framesize = 40 **Config** Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 Dim A As Bit Bitwait A , Set 'wait until bit a is se 'the above will never contine because it is not set i software 'it could be set in an ISR routine Bitwait Pinb.7 , Reset 'wait until bit 7 of Po End

6.87 BITS

Action

Set all specified bits to 1.

Syntax

Var = Bits(b1 [,bn])

Remarks

Var	The BYTE/PORT variable that is assigned with the constant.
B1,bn	A list of bit numbers that must be set to 1.

While it is simple to assign a value to a byte, and there is special Boolean notation &B for assigning bits, the Bits() function makes it simple to assign a few bits.

B = &B1000001 : how many zero's are there?

This would make it more readable : B = Bits(0, 6)

You can read from the code that bit 0 and bit 6 are set to 1. It does not save code space as the effect is the same. It can only be used on bytes and port registers.

Valid bits are in range from 0 to 7.

See Also

NBITS 788

```
'name : bits-nbits.bas
'copyright : (c) 1995-2005, MCS Electronics
'purpose : demo for Bits() AND Nbits()
'micro : Mega48
'suited for demo : yes
'commercial addon needed : no
'use in simulator : possible
```

```
_ _ _ _ _ _ _ _
$regfile = "m48def.dat"
                                                                 ' specify
the used micro
$crystal = 4000000
                                                                 ' used
crystal frequency
$baud = 19200
                                                                 ' use baud
rate
$hwstack = 32
                                                                 ' default
use 32 for the hardware stack
                                                                 ' default
\$swstack = 10
use 10 for the SW stack
$framesize = 40
                                                                 ' default
use 40 for the frame space
```

Dim B As Byte

```
'while you can use &B notation for setting bits, like B = &B1000_0111
'there is also an alternative by specifying the bits to set
B = Bits(0, 1, 2, 7) 'set only
bit 0,1,2 and 7
Print B
'and while bits() will set all bits specified to 1, there is also Nbits
()
'the N is for NOT. Nbits(1,2) means, set all bits except 1 and 2
B = Nbits(7) 'do not set
bit 7
Print B
End
```

6.88 BLOAD

Action

Writes the Content of a File into SRAM

Syntax

BLoad sFileName, wSRAMPointer

Remarks

sFileName	(String) Name of the File to be read
	(Word) Variable, which holds the SRAM Address to which the content of the file should be written

This function writes the content of a file to a desired space in SRAM. A free handle is needed for this function.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [695], FILEATTR [684], SEEK [850], BSAVE [420], KILL [757], DISKFREE [660], DISKSIZE [667], GET [697], PUT [819], FILEDATE [688], FILETIME [687], FILEDATETIME [666], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_BLoad	
		Z: Pointer to Long-variable, which holds the start position of SRAM
Output	r25: Errorcode	C-Flag: Set on Error

Example

```
' THIS IS A CODE FRAGMENT, it needs AVR-DOS in order to work
'now the good old bsave and bload
Dim Ar(100)as Byte , I Asbyte
For I = 1 To 100
                                                              ' fill the
 Ar(i) = I
array
Next
Wait 2
W = Varptr(ar(1))
Bsave"josef.img", W , 100
For I = 1 To 100
 Ar(i) = 0
                                                              ' reset the
array
Next
Bload "josef.img" , W
                                                              ' Josef you
are amazing !
For I = 1 To 10
 Print Ar(i) ; " ";
Next
Print
```

6.89 BOX

Action

Create a filled box on a graphical display.

Syntax

BOX (x1,y1) - (x2,y2), color

Remarks

x1	The left corner position of the box	
y1	he top position of the box	
x2	The right corner position of the box	
y2	The bottom position of the box	
color	The color to use to fill the box	

On COLOR displays, the box will be filled with the specified color.

On B&W displays, the box will not be filled. Only the box is drawn in the specified color.

On B&W displays you can use the BOXFILL statement to create a solid box.

See also

LINE 759, CIRCLE 429, BOXFILL 419

ASM

NONE

```
_____
' The support for this display has been made possible by Peter Küsters
from (c) Display3000
' You can buy the displays from Display3000 or MCS Electronics
-----!
$lib "lcd-pcf8833.lbx"
                                                          'special
color display support
$regfile = "m88def.dat"
                                                          'ATMega 8,
change if using different processors
                                                          '8 MHz
$crystal = 8000000
'First we define that we use a graphic LCD
Config Graphled = Color , Controlport = Porte , Cs = 1 , Rs = 0 , Sel =
3 , Sda = 2
'here we define the colors
Const Blue = & B00000011
                                                          ''predefined
contants are making programming easier
Const Yellow = &B11111100
Const Red = &B11100000
Const Green = & B00011100
Const Black = & B0000000
Const White = &B11111111
Const Brightgreen = & B00111110
Const Darkgreen = &B00010100
Const Darkred = &B10100000
Const Darkblue = &B00000010
Const Brightblue = &B00011111
Const Orange = &B11111000
'clear the display
Cls
'create a cross
Line(0 , 0) -(130 , 130) , Blue
Line(130 , 0) -(0 , 130) , Red
Waitms 1000
'show an RLE encoded picture
Showpic 0 , 0 , Plaatje
Showpic 40 , 40 , Plaatje
```

Waitms 1000

```
'select a font
Setfont Color16x16
'and show some text
Lcdat 100 , 0 , "12345678" , Blue , Yellow
Waitms 1000
Circle(30, 30), 10, Blue
Waitms 1000
'make a box
Box(10, 30) -(60, 100), Red
'set some pixels
Pset 32 , 110 , Black
Pset 38 , 110 , Black
Pset 35 , 112 , Black
End
Plaatje:
$bgf "a.bgc"
$include "color.font"
$include "color16x16.font"
```

6.90 BOXFILL

Action

Create a filled box on a graphical display.

Syntax

BOXFILL (x1,y1) - (x2,y2), color

Remarks

x1	The left corner position of the box
y1	The top position of the box
x2	The right corner position of the box
y2	The bottom position of the box
color	The color to use to fill the box

The BOXFILL command will draw a number of lines which will appear as a filled box.

See also

LINE 759, CIRCLE 429, BOX 417



Example

 $\begin{array}{c} \mbox{'create} & a & \mbox{bargraph} & \mbox{effect} \\ B \, o \, x \, f \, i \, I \, I \, (0 \ , \ 0) & - \, (60 \ , \ 10) \ , \ 1 \\ B \, o \, x \, f \, i \, I \, I \, (2 \ , \ 2) & - \, (40 \ , \ 8) \ , \ 0 \end{array}$

6.91 **BSAVE**

Action

Save a range in SRAM to a File

Syntax

BSave sFileName, wSRAMPointer, wLength

Remarks

sFileName	sFileName (String) Name of the File to be written	
wSRAMPointer	wSRAMPointer (Word) Variable, which holds the SRAM Address, from where SRAM	
	should be written to a File	
wLength	(Word) Count of Bytes from SRAM, which should be written to the file	

This function writes a range from the SRAM to a file. A free file handle is needed for this function.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [698], FILEATTR [684], SEEK [850], BLOAD [416], KILL [757], DISKFREE [660], DISKSIZE [661], GET [697], PUT [819], FILEDATE [685], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_BSave	
		Z: Pointer to Long-variable, which holds the start position of SRAM
	r20/r21: Count of bytes to be written	
Output	r25: Errorcode	C-Flag: Set on Error

```
' THIS IS A CODE FRAGMENT, it needs AVR-DOS in order to work
'now the good old bsave and bload
Dim Ar(100)as Byte , I Asbyte
For I = 1 To 100
  Ar(i) = I ' fill the
array
Next
Wait 2
W = Varptr(ar(1))
Bsave"josef.img", W , 100
```

```
421
```

' reset the

' Josef you

```
For I = 1 To 100
Ar(i) = 0
array
Next
Bload "josef.img" , W
are amazing !
For I = 1 To 10
Print Ar(i) ; " ";
Next
Print
```

6.92 BUFSPACE

Action

Returns the amount of free space of a serial buffer.

Syntax

Var = BufSpace(n)

Remarks

Var	A word or integer variable that is assigned with the free buffer space.
Ν	A constant in the range from 0-3.
	A value of 0 : output buffer first UART
	A value of 1 : input buffer first UART
	A value of 2 : output buffer second UART
	A value of 3 : input buffer second UART

While serial buffers are great because you do not have to wait/block the processor, the buffer can become full when the micro has no time to empty the buffer. With the bufspace() function you can determine if there is still room in the buffer.

See Also

CONFIG SERIAL 545 , CLEAAR 432

Example

NONE

6.93 BYVAL

Action

Specifies that a variable will be passed by value.

Syntax

Sub Test(BYVAL var)

Remarks

Var Variable name

The default for passing variables to SUBS and FUNCTIONS, is by reference(BYREF). When you pass a variable by reference, the address is passed to the SUB or FUNCTION. When you pass a variable by Value, a temp variable is created on the frame and the address of the copy is passed.

When you pass by reference, changes to the variable will be made to the calling variable.

When you pass by value, changes to the variable will be made to the copy so the original value will not be changed.

By default passing by reference is used. Note that calling by reference will generate less code.

See also

CALL 422, DECLARE 644, SUB 914, FUNCTION 642

ASM

NONE

Example

Declare Sub Test(Byval X As Byte, Byref Y As Byte, Z As Byte)

6.94 CALL

Action

Call and execute a subroutine.

Syntax

CALL Test [(var1, var-n)]

Remarks

Var1	Any BASCOM variable or constant.
Var-n	Any BASCOM variable or constant.
Test	Name of the subroutine. In this case Test.

You can call sub routines with or without passing parameters.

It is important that the SUB routine is DECLARED before you make the CALL to the subroutine. Of course the number of declared parameters must match the number of passed parameters.

It is also important that when you pass constants to a SUB routine, you must DECLARE these parameters with the BYVAL argument.

With the CALL statement, you can call a procedure or subroutine.

For example: Call Test2

The call statement enables you to implement your own statements. You don't have to use the CALL statement: Test2 will also call subroutine test2

When you don't supply the CALL statement, you must leave out the parenthesis. So Call Routine(x,y,z) must be written as Routine x,y,x

Unlike normal SUB programs called with the GOSUB statement, the CALL statement enables you to pass variables to a SUB routine that may be local to the SUB.

See also

DECLARE 644, SUB 914, EXIT 682, FUNCTION 642, LOCAL 768, CONFIG SUBMODE 555

Example

\$regfile = "m48def.dat" ' specify the used micro ' used **\$crystal** = 8000000 crystal frequency **\$baud** = 19200 ' use baud rate **\$hwstack = 32** ' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space **Config** Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 , Clockpol = 0'dimension Dim A As Byte , B As Byte some variables Declare Sub Test(b1 As Byte , Byval B2 As Byte) 'declare the SUB program A = 65'assign a value to variable A **Call** Test(a , 5) 'call test with parameter A and constant 'alternative Test A , 5 call Print A 'now print the new value End Sub Test(b1 As Byte , Byval B2 As Byte) 'use the same variable names as 'the declared one Print B1 'print it Print Bcd(b2) B1 = 10 'reassign the variable $B_{2} = 15$ 'reassign the variable End Sub

One important thing to notice is that you can change b2 but that the change will

not be reflected to the calling program! Variable A is changed however.

This is the difference between the BYVAL and BYREF argument in the DECLARE ration of the SUB program.

When you use BYVAL, this means that you will pass the argument by its value. A copy of the variable is made and passed to the SUB program. So the SUB program can use the value and modify it, but the change will not be reflected to the calling parameter. It would be impossible too when you pass a numeric constant for example.

If you do not specify BYVAL, BYREF will be used by default and you will pass the address of the variable. So when you reassign B1 in the above example, you are actually changing parameter A.

6.95 CHARPOS

Action

Returns the position of a single character in a string.

Syntax

pos = CHARPOS(string , search [,start [,SAFE]])

Remarks

Pos	Numeric variable that will be assigned with the position of the sub string in the string. Returns 0 when the sub string is not found.
String	The string to search.
Search	The search string. This can be a numeric variable too. For example a byte. When a string is used, only the first character will be used for the search.
Offset	An optional start position where the searching must start.
SAFE	If you specify an offset, Charpos will check if the offset is not located after the string. For example , when the string is "abc" and you specify an offset of 10, it will be located after the string. The SAFE option is default. When you specify SPEED, the compiler will add the offset without checking. This will result in shorter and quicker code.

No constant can be used for string it must be a string variable.

L) The search is sensitive to case.

See also

SPLIT [907], INSTR [747], REPLACECHARS [846], DELCHAR [657], INSERTCHAR [746], DELCHARS [652]

Example

------' charpos.bas ' (c) 1995-2009 MCS Electronics

```
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
'-----
Dim S As String * 20
Dim Bpos As Byte
Dim Z As String * 1
Z = "*"
Do
    Input "S:" , S
    Bpos = Charpos(s , Z)
    Print Bpos
Loop Until S = ""
Do
```

```
Input "S:" , S
Bpos = Charpos(s , "A")
charpos is sensitive to case
Print Bpos
Loop
```

' notice

6.96 CHECKFLOAT

Action

This function validates the value of a floating point variable.

Syntax

targ = CHECKFLOAT(var [,option])

Remarks

targ	A numeric variable that will be assigned with the result of the validation. The following bits can be set: cBitInfinity = 0
	cmBitInfinity = 1 ;(2 ^ cBitInfinity)
	cBitZero = 1 cmBitZero = 2 ;(2 ^ cBitZero)
	cBitNAN = 2 cmBitNAN = 4 ;(2 ^ cBitNAN)
	cBitSign = 7 cmBitSign = 128 ;(2 ^ cBitSign)
	The byte values are shown in italic. The bit constants are defined in the single and double libraries.
var	A floating point variable such as a single or double to validate.
option	This is an optional numeric constant that servers as a mask. This allows to test for makes it possible to test for a single error.

A floating point value may contain an illegal value as the result of a calculation. These illegal values are NAN (not a number) and INFINITY.

The two other tests which are performed are a test for zero, and a sign test. If the result bit 0 is '1' then the number is infinity. If the result bit 1 is '1' then the number is zero. If the result bit 2 is '1' then the number if NAN. If the result bit 7 is '1' then the number is negative.

If you want to test only for NAN and INFINITY you can add the bits and pass this as the optional numeric mask. For NAN and INFINITY this would be 1+4=5 The resulting value will be AND-ed and if any of the two bits is set, the result will be non-zero, indicating an error. If both values are 0, the result will be zero.

See also

NONE

```
$regfile = "m2561def.dat"
$crystal = 8000000
$hwstack = 64
\$swstack = 64
$framesize = 64
$baud = 19200
$lib "single.lbx"
Dim S1 As Single , S2 As Single , S3 As Single
dim d1 as Double , d2 as Double , d3 as Double
dim bCheck as Byte
dim bs(4) as Byte at s3 overlay
dim bd(8) as Byte at d3 overlay
S1 = 0 : Bcheck = Checkfloat(s1) : Print Bin(bcheck)
S1 = 0 : Bcheck = Checkfloat(s1 , 2) : Print Bin(bcheck)
d1 = 1: d2 = 0: d3 = d1 / d2
   ' 1/0 should result in infinty
Bcheck = Checkfloat(d3) : Print Bin(bcheck)
                                                            ' test for
Bcheck = Checkfloat(d3 , 5) : Print Bcheck
infinity and nan
d1 = -1
d3 = sqr(d1) ' should produce NAN
Bcheck = Checkfloat(d3) : Print Bin(bcheck)
' single routines must be checked for returning IEEE-Rulues according
values
s1 = 1: s2 = 0 : s3 = s1 / s2
    ' 1/0 should result in infinty
Bcheck = Checkfloat(s3) : Print Bin(bcheck)
s1 = -1
s3 = sqr(s1)
                      ' should produce NAN
Bcheck = Checkfloat(s3) : Print Bin(bcheck)
' now check with hard-coded values for singles
```

```
bs(1) = &HFF: bs(2) = &HFF: bs(3) = &HFF: bs(4) = &H7F ' NAN
Bcheck = Checkfloat(s3) : Print Bin(bcheck)
bs(1) = &H00: bs(2) = &H00: bs(3) = &H80: bs(4) = &H7F ' infinity
Bcheck = Checkfloat(s3) : Print Bin(bcheck)
```

End

6.97 CHECKSUM CHECKSUMXOR

Action

Returns a checksum of a string.

```
Syntax
```

PRINT Checksum(var) b = Checksum(var) b = ChecksumXOR(var)

Remarks

Var	A string variable.
В	A numeric variable that is assigned with the checksum.

The checksum is computed by counting all the bytes of the string variable. The checksumXOR is computed by Xor-ing all the bytes of the string variable. Checksums are often used with serial communication. The checksum is a byte checksum. The following VB code is equivalent :

Dim Check as Byte Check = 0 For x = 1 To Len(s\$) Check = check + ASC(mid\$(s\$,x,1)) Next

The following VB code is equivalent for ChecksumXOR Dim Check as Byte Check = 0 For x = 1 To Len(s\$) Check = check XOR ASC(mid\$(s\$,x,1)) Next

See also

CRC8 601, CRC16 603, CRC32 607

Example

```
$regfile = "m48def.dat"
the used micro
$crystal = 8000000
crystal frequency
$baud = 19200
rate
$hwstack = 32
use 32 for the hardware stack
```

' specify

- ' used
- ' use baud

' default

```
428 BASCOM-AVR
```

```
' default
\$swstack = 10
use 10 for the SW stack
$framesize = 40
                                                               ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As String * 10
                                                               'dim
variable
S = "test"
                                                               'assign
variable
Print Checksum(s)
                                                               'print value
(192)
End
```

6.98 CHR

Action

Convert a numeric variable or a constant to a string with a length of 1 character. The character represents the ASCII value of the numeric value.

Syntax

PRINT **CHR**(var) s = **CHR**(var)

Remarks

Var	Numeric variable or numeric constant.
S	A string variable.

When you want to print a character to the screen or the LCD display, you must convert it with the CHR() function.

When you use PRINT numvar, the value will be printed. When you use PRINT Chr(numvar), the ASCII character itself will be printed. The Chr() function is handy in combination with the LCD custom characters where you can redefine characters 0-7 of the ASCII table.

See also

ASC 399

```
'-----
'name : chr.bas
'copyright : (c) 1995-2005, MCS Electronics
'purpose : shows how to use the CHR() and BCD()
function and
' HEX() function in combination with a PRINT
statement
'micro : Mega48
'suited for demo : yes
'commercial addon needed : no
```

```
429
```

\$regfile = "m48def.dat" ' specify
the used micro
\$crystal = 4000000 ' used
crystal frequency
\$baud = 19200 ' use baud

```
rate
$hwstack = 32 ' default
use 32 for the hardware stack
$swstack = 10 ' default
use 10 for the SW stack
$framesize = 40 ' default
use 40 for the frame space
Config Coml = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
```

```
Dim K As Byte
```

```
K = 65
Print K ; Chr(k) ; K ; Chr(66) ; Bcd(k) ; Hex(k)
End
```

6.99 CIRCLE

Action

Draws a circle on a graphic display.

Syntax

CIRCLE(x0,y0), radius, color

Remarks

X0	Starting horizontal location of the line.
Y0	Starting vertical location of the line.
Radius	Radius of the circle
Color	Color of the circle

See Also

LINE 759

```
'______'
'name : t6963_240_128.bas
'copyright : (c) 1995-2005, MCS Electronics
'purpose : T6963C graphic display support demo 240 *
128
'micro : Mega8535
'suited for demo : yes
'commercial addon needed : no
'_______
```

430

```
$regfile = "m8535.dat"
                                                            ' specify
the used micro
                                                            ' used
$crystal = 8000000
crystal frequency
$baud = 19200
                                                            ' use baud
rate
$hwstack = 32
                                                            ' default
use 32 for the hardware stack
\$swstack = 10
                                                            ' default
use 10 for the SW stack
$framesize = 40
                                                            ' default
use 40 for the frame space
·-----
                     (c) 2001-2003 MCS Electronics
÷.
                 T6963C graphic display support demo 240 * 128
·_____
'The connections of the LCD used in this demo
'LCD pin
                         connected to
' 1
          GND
                         GND
' 2
        GND
                        GND
' 3
        +5V
-9V
                        +5V
' 4
                        -9V potmeter
' 5
        /WR
                        PORTC.0
'6
        /RD
                        PORTC.1
        /CE
                       PORTC.2
'7
                       PORTC.3
' 8
         C/D
        NC
'9
                        not conneted
'10 RESET
'11-18 D0-D7
                       PORTC.4
                        PA
'19
        FS
                        PORTC.5
'20
         NC
                        not connected
'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8 'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE = 2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns
'Dim variables (y not used)
Dim X As Byte , Y As Byte
'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT to clear only the text display
' CLS GRAPH to clear only the graphical part
Cursor Off
Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE, COLUMN LINE can be 1-8 and column 0-30 \,
Locate 1 , 1
```

```
'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "1234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"
Wait 2
Cls Text
'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0 , 0) -(239 , 127) , 255
                                                             ' diagonal
line
Line(0, 127) - (239, 0), 255
                                                              ' diagonal
line
Line(0, 0) - (240, 0), 255
                                                              ' horizontal
upper line
Line(0 , 127) - (239 , 127) , 255
                                                             'horizontal
lower line
Line(0 , 0) -(0 , 127) , 255
                                                              ' vertical
left line
Line(239, 0) -(239, 127), 255
                                                              ' vertical
right line
Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
  Pset X , 20 , 255
                                                             ' set the
pixel
Next
For X = 0 To 140
                                                              ' set the
   Pset X , 127 , 255
pixel
Next
Wait 2
'circle time
'circle(X,Y), radius, color
'X,y is the middle of the circle, color must be 255 to show a pixel and 0
to clear a pixel
For X = 1 To 10
  Circle(20, 20), X, 255
                                                              ' show
circle
  Wait 1
  Circle(20 , 20) , X , 0
                                                              'remove
circle
  Wait 1
Next
Wait 2
For X = 1 To 10
  Circle(20, 20), X, 255
                                                              ' show
circle
  Waitms 200
Next
```

```
Wait 2
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Test:
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje
                                                              ' show 2
since we have a big display
Wait 2
Cls Text
                                                              ' clear the
text
End
'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"
```

'You could insert other picture data here

6.100 CLEAR

Action

Clear serial input or output buffer

Syntax

CLEAR bufname

Remarks

Bufname Serialbuffer name such as Serialin, Serialin1 , Serialout or Serialout1 For chips with more UARTS : SERIALIN2, SERIALIN3, SERIALOUT2, SERIALOUT3

When you use buffered serial input or buffered serial output, you might want to clear the buffer.

While you can make the head pointer equal to the tail pointer, an interrupt could be active which might result in an update of the buffer variables, resulting in an unexpected result.

The CLEAR statement will reset the head and tail pointers of the ring buffer, and it will set the buffer count variable to 0. The buffer count variable is new and introduced in 1.11.8.3. It counts how many bytes are in the buffer.

The internal buffercount variable is named _RS_BUFCOUNTxy , where X is ${\bf R}$ for ${\bf R}$ eccive, and ${\bf W}$ for ${\bf W}$ rite, and y is 0 for the first UART, and 1 for the second UART. The

See also

CONFIG SERIALIN 540, CONFIG SERIALOUT 545

ASM

Calls _BUF_CLEAR from MCS.LIB

Example

CLEAR SERIALIN

6.101 CLS

Action

Clear the LCD display and set the cursor to home.

Syntax CLS

Syntax for graphical LCD

CLS CLS TEXT CLS GRAPH CLS Y, X1 , X2 [, CHAR]

Remarks

Clearing the LCD display does not clear the CG-RAM in which the custom characters are stored.

For graphical LCD displays CLS will clear both the text and the graphical display. The EADOG128 and KS108 support the option to clear a portion of a line. Depending on the used graphic chip, this option might be added to other graphical LCD lib's too.

Graphical displays coordinates start with 1. To clear the entire first line you need to code : CLS 1,1,128

This will clear the first line, from the starting position X1(1) to the ending position (X2). You may specify an optional character to use. By default 0 is used. When you have inverse text, you need to use 255.

See also

<u>\$LCD</u>[332], <u>\$LCDRS</u>[337], <u>LCD</u>[752], <u>SHIFTLCD</u>[882], <u>SHIFTCURSOR</u>[877], <u>SHIFTLCD</u>[882], <u>INITLCD</u>[738]

Example

```
_____
'name
                    : lcd.bas
'name
'copyright
                    : (c) 1995-2005, MCS Electronics
                    : demo: LCD, CLS, LOWERLINE, SHIFTLCD,
'purpose
SHIFTCURSOR, HOME
                      CURSOR, DISPLAY
                    : Mega8515
'micro
'suited for demo
                     : yes
'commercial addon needed : no
!_____
                        _____
_____
$regfile = "m8515.dat"
                                                 ' specify
the used micro
$crystal = 4000000
                                                 ' used
crystal frequency
$baud = 19200
                                                 ' use baud
rate
$hwstack = 32
                                                 ' default
```

BASCOM-AVR

```
use 32 for the hardware stack
                                                              ' default
\$swstack = 10
use 10 for the SW stack
                                                              ' default
$framesize = 40
use 40 for the frame space
$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation
'note : tested in PIN mode with 4-bit
'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector
Rem with the config lcdpin statement you can override the compiler
settings
Dim A As Byte
Config Lcd = 16 \times 2
                                                              'configure
lcd screen
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * la is intended for 16 character displays with split addresses over
2 lines
'$LCD = address will turn LCD into 8-bit databus mode
       use this with uP with external RAM and/or ROM
       because it aint need the port pins !
Cls
                                                              'clear the
LCD display
Lcd "Hello world."
                                                              'display
this at the top line
Wait 1
Lowerline
                                                              'select the
lower line
Wait 1
Lcd "Shift this."
                                                              'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right
                                                              'shift the
text to the right
  Wait 1
                                                              'wait a
moment
Next
For A = 1 To 10
   Shiftlcd Left
                                                              'shift the
text to the left
```

434

BASCOM Language Reference 435 Wait 1 'wait a moment Next Locate 2 , 1 'set cursor position Lcd "*" 'display this Wait 1 'wait a moment 'shift the Shiftcursor Right cursor Lcd "@" 'display this Wait 1 'wait a moment 'select line Home Upper 1 and return home Lcd "Replaced." 'replace the text Wait 1 'wait a moment 'hide cursor Cursor Off Noblink Wait 1 'wait a moment Cursor On Blink 'show cursor Wait 1 'wait a moment Display Off 'turn display off Wait 1 'wait a moment Display On 'turn display on '-----NEW support for 4-line LCD-----Thirdline Lcd "Line 3" Fourthline Lcd "Line 4" Home Third 'goto home on line three Home Fourth 'first Home F letteer also works Locate 4 , 1 : Lcd "Line 4" Wait 1 'Now lets build a special character 'the first number is the characternumber (0-7)'The other numbers are the rowvalues 'Use the LCD tool to insert this line Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228 replace ? with number (0-7) Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240 replace ? with number (0-7)Cls 'select data RAM Rem it is important that a CLS is following the deflcdchar statements because it will set the controller back in datamode Lcd Chr(0); Chr(1)'print the

```
436 BASCOM-AVR
```

special character

```
'----- Now use an internal routine ------
_temp1 = 1 'value into
ACC
!rCall _write_lcd 'put it on
LCD
End
```

6.102 CLOCKDIVISION

Action

Will set the system clock division available in the MEGA chips.

Syntax

CLOCKDIVISON = var

Remarks

Variable or numeric constant that sets the clock division. Valid values are from 2-129.
A value of 0 will disable the division.

On the MEGA 103 and 603 the system clock frequency can be divided so you can save power for instance. A value of 0 will disable the clock divider. The divider can divide from 2 to 127. So the other valid values are from 2 - 127.

Some routines that rely on the system clock will not work proper anymore when you use the divider. WAITMS for example will take twice the time when you use a value of 2.

See also

POWERSAVE 809

Example

```
$regfile = "m103def.dat"
                                                              ' specify
the used micro
$crystal = 8000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
```

```
Databits = 8 , Clockpol = 0
```

Clockdivision = 2

6.103 CLOSE

Action

Closes an opened device.

Syntax

OPEN "device" for MODE As #channel **CLOSE** #channel

Remarks

Device	The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device. With the implementation of the software UART, the compiler must know
	to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use.
	COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stop bits.
	The format for COM1 is : COM1:
	Some chips have 2 UARTS. You can use COM2: to open the second HW UART.
	Other chips might have 4 or 8 UARTS.
	The format for the software UART is: COMpin:speed,8,N,stop bits[, INVERTED]
	Where pin is the name of the PORT-pin.
	Speed must be specified and stop bits can be 1 or 2.
	An optional parameter ,INVERTED can be specified to use inverted RS- 232.
	Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin
	PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232.
MODE	You can use BINARY or RANDOM for COM1 and COM2, but for the software UART pins, you must specify INPUT or OUTPUT.
Channel	The number of the channel to open. Must be a positive constant >0.

The statements that support the device are PRINT , INPUT and $\mathsf{INPUTHEX}$, $\mathsf{INKEY},$ <code>WAITKEY</code>.

Every opened device must be closed using the CLOSE #channel statement. Of course, you must use the same channel number.

The best place for the CLOSE statement is at the end of your program.

The INPUT statement in combination with the software UART, will not echo characters back because there is no default associated pin for this.

For the AVR-DOS file system, you may place the CLOSE at any place in your program. This because the file system supports real file handles. For the UART, SPI or other devices, you do not need to close the device. Only AVR-DOS needs a CLOSE so the file will be flushed.

See also

OPEN 7947, PRINT 8107

Example

\$regfile = "m48def.dat"
the used micro
\$crystal = 10000000
crystal frequency
\$baud = 19200
rate
\$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
use 10 for the SW stack
\$framesize = 40
use 40 for the frame space

Dim B As Byte

'Optional you can fine tune the calculated bit delay 'Why would you want to do that? 'Because chips that have an internal oscillator may not 'run at the speed specified. This depends on the voltage, temp etc. 'You can either change \$CRYSTAL or you can use 'BAUD #1,9610 'In this example file we use the DT006 from www.simmstick.com 'This allows easy testing with the existing serial port 'The MAX232 is fitted for this example. 'Because we use the hardware UART pins we MAY NOT use the hardware UART 'The hardware UART is used when you use PRINT, INPUT or other related statements 'We will use the software UART. Waitms 100 'open channel for output **Open** "comd.1:19200,8,n,1" **For Output As #1** Print #1 , "serial output" 'Now open a pin for input **Open** "comd.0:19200,8,n,1" **For Input As #**2 'since there is no relation between the input and output pin 'there is NO ECHO while keys are typed Print #1 , "Number" 'get a number **Input** #2 , B

' specify

' use baud

' default

' default

' default

' used

```
'print the number
Print #1 , B
'now loop until ESC is pressed
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
   'store in byte
  B = Inkey(#2)
   'when the value > 0 we got something
  If B > 0 Then
     Print #1 , Chr(b)
                                                             'print the
character
  End If
Loop Until B = 27
Close #2
Close #1
'OPTIONAL you may use the HARDWARE UART
'The software UART will not work on the hardware UART pins
'so you must choose other pins
'use normal hardware UART for printing
'Print B
'When you dont want to use a level inverter such as the MAX-232
'You can specify ,INVERTED :
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Now the logic is inverted and there is no need for a level converter
'But the distance of the wires must be shorter with this
End
```

6.104 CLOSESOCKET

Action

Closes a socket connection.

Syntax

CloseSocket socket [, prm]

Remarks

Socket	The socket number you want to close in the range of 0-3. When the socket is already closed, no action will be performed.
Prm	 An optional parameter to change the behavior of the CloseSocket statement. The following values are possible : 0 - The code will behave as if no parameter has been set. 1 - In normal cases, there is a test to see if all data written to the chip has been sent. When you set bit 0 (value of 1), this test is not performed. 2 - In normal cases, there is a test to see if the socket is actually closed after the command has been given to the chip. When it is not closed, you can not re-use the socket. The statement will block program execution however and you could test at a later time if the connection has been closed.

You may combine the values. So 3 will combine parameter value 1 and 2. It is advised to use option value 1 with care.

You must close a socket when you receive the SOCK_CLOSE_WAIT status. You may also close a socket if that is needed by your protocol. You will receive a SOCK_CLOSE_WAIT status when the server closes the connection.

When you use CloseSocket you actively close the connection. Note that it is not needed to wait for a SOCK_CLOSE_WAIT message in order to close a socket connection.

After you have closed the connection, you need to use GetSocket in order to use the socket number again.

In normal conditions, without using the optional parameter, the statement can block your code for a short or longer time, depending on the connection speed.

See also

```
CONFIG TCPIP 559, GETSOCKET 717, SOCKETCONNECT 888, SOCKETSTAT 890, TCPWRITE 924, TCPWRITESTR 925, TCPREAD 924, SOCKETLISTEN 889
```

Example

```
_____
                       : clienttest.bas
'name
                       : (c) 1995-2005, MCS Electronics
'copyright
                       : start the easytcp.exe program and listen to
'purpose
port 5000
'micro
                       : Megal61
'suited for demo
                        : no
'commercial addon needed : yes
·_____
                                     -----
_____
$regfile = "M161def.dat"
$crystal = 4000000
$baud = 19200
\$hwstack = 40
                                                         ' default
use 40 for the hardware stack
\$swstack = 40
                                                         ' default
use 40 for the SW stack
$framesize = 64
                                                         ' default
use64 for the frame space
Const Sock_stream = $01
                                                        ' Tcp
Const Sock_dgram = $02
                                                         ' Udp
Const Sock_ipl_raw = $03
                                                         ' Ip Layer
Raw Sock
Const Sock_macl_raw = $04
                                                         ' Mac Layer
Raw Sock
                                                         ' Confirm
Const Sel_control = 0
Socket Status
Const Sel_send = 1
                                                         ' Confirm Tx
Free Buffer Size
Const Sel_recv = 2
                                                         ' Confirm Rx
Data Size
'socket status
Const Sock_closed = $00
                                                         ' Status Of
Connection Closed
```

BASCOM Language Reference 441 **Const** Sock_arp = \$01 ' Status Of Arp **Const** Sock_listen = \$02 ' Status Of Waiting For Tcp Connection Setup **Const** Sock_synsent = \$03 ' Status Of Setting Up Tcp Connection ' Status Of **Const** Sock_synsent_ack = \$04 Setting Up Tcp Connection **Const** Sock_synrecv = \$05 ' Status Of Setting Up Tcp Connection ' Status Of **Const** Sock_established = \$06 Tcp Connection Established **Const** Sock close wait = \$07 ' Status Of Closing Tcp Connection **Const** Sock last ack = \$08 ' Status Of Closing Tcp Connection Const Sock_fin_wait1 = \$09 ' Status Of Closing Tcp Connection **Const** Sock_fin_wait2 = \$0a ' Status Of Closing Tcp Connection **Const** Sock_closing = \$0b ' Status Of Closing Tcp Connection **Const** Sock_time_wait = \$0c ' Status Of Closing Tcp Connection ' Status Of **Const** Sock_reset = \$0d Closing Tcp Connection **Const** Sock init = \$0e ' Status Of Socket Initialization **Const** Sock_udp = \$0f ' Status Of Udp **Const** Sock_raw = \$10 ' Status of IP RAW \$lib "tcpip.lbx" ' specify the tcpip library Print "Init , set IP to 192.168.0.8" ' display a message ' before we Enable Interrupts use config tcpip , we need to enable the interrupts Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx = \$55 , Rx = \$55'Use the line below if you have a gate way 'Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx = \$55 , Rx = \$55 Dim Bclient As Byte ' socket number Dim Idx As Byte Dim Result As Word ' result Dim S As String * 80 For Idx = 0 To 3 ' for all sockets Bclient = Getsocket(idx , Sock_stream , 0 , 0) ' get socket for client mode, specify port 0 so loal_port is used Print "Local port : " ; Local_port ' print local port that was used Print "Socket " ; Idx ; " " ; Bclient Result = Socketconnect(idx , 192.168.0.3 , 5000) ' connect to

```
easytcpip.exe server
 Print "Result " ; Result
Next
Do
                                                             ' is there a
  If Ischarwaiting() <> 0 Then
key waiting in the uart?
     Bclient = Waitkey()
                                                              ' get the
key
     If Bclient = 27 Then
       Input "Enter string to send " , S
                                                              ' send WHO ,
TIME or EXIT
       For Idx = 0 To 3
          Result = Tcpwritestr(idx , S , 255)
       Next
     End If
  End If
  For Idx = 0 To 3
     Result = Socketstat(idx , 0)
                                                             ' get status
     Select Case Result
       Case Sock_established
            Result = Socketstat(idx , Sel_recv)
                                                             ' get number
of bytes waiting
            If Result > 0 Then
               Do
                 Result = Tcpread(idx , S)
                 Print "Data from server: " ; Idx ; " " ; S
               Loop Until Result = 0
            End If
       Case Sock_close_wait
            Print "close_wait"
            Closesocket Idx
       Case Sock_closed
            'Print "closed"
     End Select
 Next
Loop
End
```

6.105 CONFIG

The CONFIG statement is used to configure the various hardware devices.

DIRECTIVE	RE- USABLE	XMEGA ONLY
CONFIG 1WIRE 444	NO	
CONFIG ACXX 449	YES	Х
CONFIG ACI 448	YES	
<u>CONFIG ADC 45</u> ी	NO	
CONFIG ADCx 453	YES	Х
CONFIG ATEMU 455	NO	
CONFIG BASE	NO	
CONFIG BCCARD 458	NO	
CONFIG CLOCK 46	NO	
CONFIG CLOCKDIV 465	YES	
CONFIG COM1 466	YES	

	VEC
CONFIG COM2 467 also COM3 - COM8	YES
	YES X
CONFIG DATE 473	NO
CONFIG DCF77 475	NO
CONFIG DEBOUNCE 482	NO
	YES X
CONFIG DMACHx 484	YES X
CONFIG DMXSLAVE 488	NO
CONFIG DP 490	NO
CONFIG EEPROM 49ते	NO X
	NO
CONFIG EVENT_SYSTEM 492	
CONFIG EXTENDED_PORT	NO
ାଧ୍ୟ CONFIG GRAPHLCD ରେଜି	NO
CONFIG HITAG	NO
CONFIG I2CDELAY 498	NO
CONFIG I2CSLAVE	NO
	NO
CONFIG INTx 504	YES
	NO
CONFIG KEYBOARD	NO
CONFIG LCD 516	NO
	NO
CONFIG LCDMODE 523	NO
CONFIG LCDPIN 523	NO
CONFIG OSC 527	YES X
CONFIG RC5	NO
CONFIG PORT 528	YES
CONFIG	NO X
POWER_REDUCTION 530	
CONFIG PRIORITY 533	YES X
<u>CONFIG PRINT</u> 534ो	NO
CONFIG PRINTBIN 535	NO
CONFIG PS2EMU 535	NO
CONFIG SERIALIN 540	NO
CONFIG SERIALIN1 540	NO
CONFIG SERIALIN2 540	NO
CONFIG SERIALIN3 540	NO
CONFIG SERIALOUT 545	NO
CONFIG SERIALOUT1 545	NO
CONFIG SERIALOUT2	NO
CONFIG SERIALOUT3	NO
CONFIG SERVOS 552	NO
CONFIG SHIFTIN 548	NO
CONFIG SINGLE	NO
CONFIG SDA 538	NO
CONFIG SCL	NO
CONFIG SHIFTIN बिकी CONFIG SINGLE बिकी CONFIG SDA बिउक्षे	NO NO NO

444 BASCOM-AVR

CONFIG SPI 548	NO	
CONFIG SPIx 550	YES	Х
CONFIG SUBMODE	NO	
CONFIG SYSCLOCK 556	YES	Х
CONFIG TCXX 557	YES	Х
CONFIG TCPIP 559	NO	
CONFIG TWI 569	YES	
CONFIG TWISLAVE	NO	
CONFIG TIMER0 562	YES	
CONFIG TIMER1 564	YES	
CONFIG TIMER2 and 3	YES	
CONFIG USB 575	NO	
CONFIG VPORT 582	YES	Х
CONFIG WATCHDOG 584	YES	
CONFIG WAITSUART 584	NO	
CONFIG X10 58	NO	
CONFIG XRAM 589	YES	

Some CONFIG directives are intended to be used once. Others can be used multiple times. For example you can specify that a port must be set to input after you have specified that it is used as an input.

You cannot change the LCD pins during run time. In that case the last specification will be used or an error message will be displayed.

Some Configuration commands are only available to the Xmega. An X in the 'Xmega Only' indicates that the command can only be used for an Xmega processor.

6.106 CONFIG 1WIRE

Action

Configure the pin to use for 1WIRE statements and override the compiler setting.

Syntax

CONFIG 1WIRE = pin [, extended=0|1]

Remarks

Pin	The port pin to use such as PORTB.0		
extended	An optional constant value which need to be 0 or 1.		

The CONFIG 1WIRE statement overrides the compiler setting. It is the preferred that you use it. This way the setting is stored in your source code. You can configure only one pin for the 1WIRE statements because the idea is that you can attach multiple 1WIRE devices to the 1WIRE bus.

You can however use multiple pins and thus multiple busses. All 1wire commands and functions need the port and pin in that case.

The 1wire commands and function will automatically set the DDR and PORT register bits to the proper state. You do not need to bring the pins into the right state

yourself.

It is important that you use a pull up resistor of 4K7 ohm on the 1wire pin. The pull up resistor of the AVR is not sufficient.

Also notice that some 1wire chips also need +5V. 1 wire is just marketing since you need GND anyway. The least is 2 wires and typical you need 3 wires.

Extended

The extended option is only needed when you use multiple busses/pins and if these are pins mix normal and extended addresses.

Let's clear that up. When the 1wire code was written in 1995 all the port addresses were normal I/O addresses. These are addresses that fit in the I/O space (address < &H60). To save code, register R31 was cleared in the library and the port register was passed in R30.

When Atmel introduced the extended I/O registers with address >&HFF, it was possible to set R31 to a fixed value when the user port was an extended I/O address. But when you want to mix the addresses, there is no other way then to pass the word address of the I/O register to the library code.

And that is exactly what EXTENDED=1 will do. It will use more code. This support was written for a customer that already made his PCB's. We do advise to use the same port when you use multiple pins.

See also

1WRESET 3751, 1WREAD 3771, 1WWRITE 3861, 1WIRECOUNT 3721, 1WRESET 3751, 1WSEARCHFIRST 3751, 1WSEARCHNEXT 3821

Example

```
_____
                         : 1wire.bas
'name
                         : (c) 1995-2005, MCS Electronics
'copyright
                         : demonstrates lwreset, lwwrite and lwread()
'purpose
                        : Mega48
'micro
'suited for demo
                        : yes
'commercial addon needed : no
' pull-up of 4K7 required to VCC from Portb.2
' DS2401 serial button connected to Portb.2
. _ _ _ _ _ _ _ _ _
$regfile = "m48def.dat"
$crystal = 8000000
$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 10
                                                          'default use
10 for the SW stack
$framesize = 40
                                                          'default use
40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'when only bytes are used, use the following lib for smaller code
$lib "mcsbyte.lib"
```

```
Config 1wire = Portb.0
                                                               'use this
pin
'On the STK200 jumper B.0 must be inserted
Dim Ar(8) As Byte , A As Byte , I As Byte
Do
  Wait 1
  1wreset
                                                               'reset the
device
  Print Err
                                                               'print error
1 if error
  1wwrite &H33
                                                               'read ROM
command
  For I = 1 To 8
   Ar(i) = lwread()
                                                               'place into
array
 Next
'You could also read 8 bytes a time by unremarking the next line
'and by deleting the for next above
'Ar(1) = 1wread(8)
                                                                'read 8
bytes
 For I = 1 To 8
    Print Hex(ar(i));
                                                               'print
output
 Next
  Print
                                                               'linefeed
Loop
'NOTE THAT WHEN YOU COMPILE THIS SAMPLE THE CODE WILL RUN TO THIS POINT
'THIS because of the DO LOOP that is never terminated!!!
'New is the possibility to use more than one 1 wire bus
'The following syntax must be used:
For I = 1 To 8
 Ar(i) = 0
                                                               'clear array
to see that it works
Next
1wreset Pinb , 2
                                                               'use this
port and pin for the second device
lwwrite &H33 , 1 , Pinb , 2
now the number of bytes must be specified!
                                                               'note that
'lwwrite Ar(1) , 5,pinb,2
'reading is also different
Ar(1) = 1wread(8, Pinb, 2)
                                                               'read 8
bytes from portB on pin 2
For I = 1 To 8
  Print Hex(ar(i));
Next
'you could create a loop with a variable for the bit number !
                                                               'for pin 0-3
For I = 0 To 3
  lwreset Pinb , I
  lwwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8, Pinb, I)
  For A = 1 To 8
   Print Hex(ar(a));
```

Next Print Next End

Xmega Example

```
_____
                         : XM128-1wire.bas
'name
'copyright
                         : (c) 1995-2010, MCS Electronics
'purpose
                         : demonstrates lwreset, lwwrite and lwread()
                         : Xm128A1
'micro
'suited for demo
                         : no
'commercial addon needed : no
' pull-up of 4K7 required to VCC from Portb.0
' DS2401 serial button connected to Portb.0
·-----
                                               _____
_____
$regfile = "xm128a1def.dat"
$crystal = 32000000
$lib "xmega.lib" : $external _xmegafix_clear : $external
_xmegafix_rol_r1014
\$hwstack = 32
                                                           ' default
use 32 for the hardware stack
\$swstack = 32
                                                           'default use
10 for the SW stack
$framesize = 32
                                                           'default use
40 for the frame space
'First Enable The Osc Of Your Choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
'configure UART
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
'configure 1wire pin
Config 1wire = Portb.0
                                                           'use this
pin
Dim Ar(8) As Byte , A As Byte , I As Byte
Print "start"
A = 1wirecount()
Print A ; " devices found"
'get first
Ar(1) = lwsearchfirst()
For I = 1 To 8
                                                           'print the
number
 Print Hex(ar(i));
Next
Print
Do
   'Now search for other devices
```

```
448 BASCOM-AVR
```

```
Ar(1) = 1wsearchnext()
                                                             ' get next
device
   For I = 1 To 8
   Print Hex(ar(i));
   Next
   Print
Loop Until Err = 1
Waitms 2000
Do
  1wreset
                                                             'reset the
device
 Print Err
                                                             'print error
1 if error
  1wwrite &H33
                                                             'read ROM
command
  Ar(1) = 1wread(8) you can use this instead of the code below
 For I = 1 To 8
   Ar(i) = 1wread()
                                                             'place into
array
 Next
 For I = 1 To 8
    Print Hex(ar(i));
                                                             'print
output
 Next
 Print
                                                             'linefeed
 Waitms 1000
Loop
```

End

6.107 CONFIG ACI

Action

Configures the Analog Comparator.

Syntax

CONFIG ACI = ON|OFF, COMPARE = ON|OFF, TRIGGER=TOGGLE|RISING|FALLING

Remarks

ACI	Can be switched on or off
	Can be on or off.
COMPARE	
	When switched ON, the TIMER1 in capture mode will trigger on ACI too.
	Specifies which comparator events trigger the analog comparator interrupts.

See also

NONE

Example

NONE

6.108 CONFIG ACXX

Action

Configures the Analog Comparator of the Xmega.

Syntax

CONFIG ACXX = state, TRIGGER=trigger, HISPEED=speed, HYSMODE=hys, MUXPLUS=mp, MUXMIN=mm, OUTPUT=otp, SCALE=scale, WINDOW=w, WINTMODE = wint

Remarks

ACXX	The name of the Analog comparator : ACA0,ACA1, ACB0 or ACB1 Some XMEGA chips might not have (all) comparators.		
State	ON or OFF. Select ON to turn the comparator on. By default it is off.		
HiSpeed	When ENABLED, the comparator hi speed mode is activated. Default mode is DISABLED.		
Trigger	Specifies which comparator event triggers the analog comparator interrupts. This options are : RISING, FALLING or BOTH / TOGGLE.		
Hysmode	To prevent quick toggling, a hysteresis is built in. You can chose the mode : - OFF - SMALL - LARGE		
MuxPlus	This option controls which pin is connected to the positive input of the comparator. Possible values : 0-7, DAC. When you chose 7, DAC will also be used. So 7 and DAC are equivalent.		
MuxMin	This option controls which pin is connected to the negative input of the comparator. Possible values : 0-7, DAC, BANDGAP, SCALER. 0 - connects pin 0 1 - connects pin 1 2 - connects pin 3 ! 3 - connects pin 5 4 - connects pin 7 5 - connects the DAC output (same as DAC option) 6 - connects the BANDGAP voltage (same as BANDGAP option) 7 - connects the SCALER output (same as SCALE option)		
Output	Enabled or Disabled (default). When the output is enabled, the output of the comparator is routed to pin 7 of the port.		
Scale	The input voltage of the negative mux pin can be scaled. The scale value must be in range from 0-63. The scale output voltage is calculated as : (vcc * (scale+1)) / 64 Thus a value of 63 would give VCC. And 32 would give vcc/2		
Windows	Enabled or Disabled (default). When enabled, the two comparators of the port (ACA0 + ACA1) or (ACB0 + ACB1) form a window discriminator so you can control if a voltage is in the range of the lower and upper		

	comparator.
WintMode	The status register contains the window state. (bit 6 and 7). You can also fire an interrupt at one of the states: ABOVE : interrupt on signal above window INSIDE : interrupt on signal inside window BELOW : interrupt on signal below window OUTSIDE : interrupt on signal outside window

A window is used in battery voltage meters. you could set the lower voltage to 12 V. And the upper voltage to 14 V.

If the voltage is inside this window : >=12V and <=14V then the battery is OK.

If the voltage is below the battery need to be charged.

If the voltage is above the window the battery if fully charged. The mentioned values are just an example.

See also

NONE

Example

```
_____
                     _ _ _ _ _ _ _ _ _ _ _ _ _
                  (c) 1995-2010, MCS
                     xm128-AC.bas
  This sample demonstrates the Analog Comparator
· _____
$regfile = "xml28aldef.dat"
$crystal = 32000000
\$hwstack = 64
\$swstack = 64
$framesize = 64
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'First Enable The Osc Of Your Choice , make sure to enable 32 KHz clock
or use an external 32 KHz clock
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
'setup comparator pin 0 and pin 1 are the input of portA. Pin 7 is an
output in this sample
Config Aca0 = On, Hysmode = Small, Muxplus = 0, Muxmin = 1, Output =
Enabled
Do
```

```
Print Bin(aca_status)
Print Aca_status.4
Waitms 1000
Loop
```

' output ac0

6.109 CONFIG ADC

Action

Configures the A/D converter.

Syntax CONFIG ADC = single, PRESCALER = AUTO, REFERENCE = opt

Keinai KS		
ADC	Running mode. May be SINGLE or FREE.	
	A numeric constant for the clock divider. Use AUTO to let the compiler generate the best value depending on the XTAL	
E	The options depend on the used micro. Some chips like the M163 have additional reference options. In the definition files you will find : ADC_REFMODEL = x This specifies which reference options are available. The possible values are listed in the table below.	

Chip	Modes	ADC_REFMODEL
2233,4433,4434,8535,m103,	OFF	
m603, m128103	AVCC	U
m165, m169, m325,m3250,	OFF	1
m645, m6450, m329,m3290,	AVCC	Ţ
m649, m6490,m48,m88,m168	INTERNAL or INTERNAL_1.1	h
tiny15,tiny26	OFF	2
1. 12		2
tiny13	AVCC	3
	INTERNAL	
tiny24,tiny44,tiny85	AVCC	4
	EXTERNAL or OFF	
	INTERNAL or INTERNAL_1.1	
m164,m324,m644,m640,m1280,	AREF or OFF	5
m1281,m2561,m2560	AVCC	
	INTERNAL1.1	
	INTERNAL_2.56	
tiny261,tiny461,tiny861, tiny25,	AVCC	7
tiny45,tiny85	EXTERNAL or OFF	
	INTERNAL_1.1	
	INTERNAL_2.56_NOCAP	
	INTERNAL_2.56_EXTCAP	
CAN128, PWM2_3,USB1287,	AREF or OFF	8
m128, m16, m163, m32, m323,	AVCC	
m64	INTERNAL or INTERNAL_2.56	
	You may also use VALUE=value	

When you use VALUE=value, you may specify any value. The disadvantage is that

Remarks

452 BASCOM-AVR

when you port your code from one chip to another it will not work. While the AREF, AVCC, etc. are all converter to the right settings, the value can not be converted.

The AD converter is started automatic when you use the CONFIG ADC command. You can use STOP ADC and START ADC to disable and enable the power of the AD converter.

The GETADC() function is intended to be used with the SINGLE running mode. This means that each time you call GETADC(), a conversion is started. If you use the free running mode, you need to retrieve the value from the AD converter yourself. For example by reading the internal ADC word variable.

See also

GETADC 7007, CONFIG ADCx 453

Example

_____ : adc.bas 'name : (c) 1995-2005, MCS Electronics 'copyright : demonstration of GETADC() function for 8535 'purpose or M163 micro : Megal63 'micro 'suited for demo : yes 'commercial addon needed : no 'use in simulator : possible ' Getadc() will also work for other AVR chips that have an ADC converter !_____ \$regfile = "m163def.dat" ' we use the M163 \$crystal = 4000000\$hwstack = 32' default use 32 for the hardware stack \$swstack = 10'default use 10 for the SW stack **\$framesize** = 40 'default use 40 for the frame space 'configure single mode and auto prescaler setting 'The single mode must be used with the GETADC() function 'The prescaler divides the internal clock by 2,4,8,16,32,64 or 128 'Because the ADC needs a clock from 50-200 KHz 'The AUTO feature, will select the highest clockrate possible **Config** Adc = **Single**, Prescaler = Auto 'Now give power to the chip Start Adc ' NOT required since it will start automatic 'With STOP ADC, you can remove the power from the chip 'Stop Adc Dim W As Word , Channel As Byte

```
Channel = 0
'now read A/D value from channel 0
Do
 W = Getadc(channel)
 Print "Channel " ; Channel ; " value " ; W
 Incr Channel
 If Channel > 7 Then Channel = 0
Loop
End
'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
         : AREF, internal reference turned off
'OFF
' AVCC
         : AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin
```

'Using the additional param on chip that do not have the internal reference will have no effect.

6.110 CONFIG ADCx

Action

Configures the A/D converter of the Xmega.

Syntax

CONFIG ADCA | ADCB = mode, CONVMODE=sign, RESOLUTION=res, DMA=dma, REFERENCE=ref,EVENT_MODE=evt, EVENT_CHANNEL=evtchan, PRESCALER=pre, BANDGAP=gap, TEMPREF=tref, SWEEP=sweep, CH0_GAIN=gain, CH0_INP= inp, MUX0=mux, CH1_GAIN=gain, CH1_INP= inp, MUX1=mux , CH2_GAIN=gain, CH2_INP= inp, MUX2=mux, CH3_GAIN=gain, CH3_INP= inp, MUX3=mux

Remarks

mode	Running mode. May be SINGLE or FREE.
sign	The conversion mode. This can be SIGNED or UNSIGNED . When choosing SIGNED you should assign the result to an integer. When choosing UNSIGNED you should assign the result to a word. The default is UNSIGNED.
res	The resolution of the conversion. Valid values are : - 8BIT - 12BIT . This is the default - LEFT12BIT . This will result in a left aligned 21 bit value.
dma	If you want to use the DMA channel, you can select which DMA channels must be used: - OFF (no DMA) - CH01 (channel 0 + 1) - CH012 (channel 0 + 1 + 2) - CH0123 (channel 0 + 1 + 2 + 3)
ref	Selects the reference to use. Valid options : - INT1V. For internal 1V reference - INTVCC. For internal voltage divided by 1.6 - AREFA. External reference from AREF pin on PORT A. - AREFB. External reference from AREF pin on PORT B.

5A000	BASCOM-AVR	
gap	Enables the bangap reference. Use ENABLED or DISABLED . Setting this bit enables the bandgap to prepare for ADC measurement. Note that if any other functions are using the bandgap already, this bit does not need to be set. This could be when the internal 1V reference is used in ADC or DAC, or if the Brown-out Detector is enabled.	
tref	Enables the temperature reference. Use ENABLED or DISABLED . Setting this bit enables the temperature reference to prepare for ADC measurement	
sweep	Selects which channels are included in a sweep when a channel sweep is triggered by the event system or in the free running mode. Valid options : - CH0 : channel 0 included - CH01 : channel 0 and 1 included - CH012 : channel 0-2 included - CH0123 : all channels are included	
evtchan	 Event channel selection. This selects which channel should trigger which ADC channel. Valid options: CH0123. Event channel 0, 1, 2, 3 as selected inputs CH1234. Event channel 1, 2, 3, 4 as selected inputs CH2345. Event channel 2,3, 4, 5 as selected inputs CH3456. Event channel 3, 4, 5, 6 as selected inputs CH4567. Event channel 4, 5, 6, 7 as selected inputs CH456. Event channel 4, 5, 6 as selected inputs CH456. Event channel 4, 5, 6 as selected inputs CH456. Event channel 4, 5, 6 as selected inputs CH456. Event channel 7 as selected inputs 	
evt	 Event channel mode selection. This selects how many of the selected event channel are in use. Valid options: NONE. Event system is not used CH0. Event channel with the lowest number, defined by evtchan triggers conversion on channel 0 CH01. Event channel with the two lowest numbers, defined by evtchan trigger conversion on channel 0 and 1 respectively CH012. Event channel with the three lowest numbers, defined by evtchan trigger conversion on channel 0, 1 and 2 respectively CH0123. Event channel defined by evtchan trigger conversion on channel 0, 1 and 2 respectively CH0123. Event channel defined by evtchan trigger conversion on channel 0, 1 and 2 respectively SWEEP. One sweep of all active ADC channels defined by SWEEP on incoming event channel with the lowest number, defined by evtchan SWEEP on incoming event channel with the lowest number, defined by evtchan. In addition, the conversion will be synchronized on event to ensure a very accurate timing for the conversion. 	
pre	Prescaler value. The prescaler divides the system clock and applies it to	

	Prescaler value. The prescaler divides the system clock and applies it to the A/D converter. Valid prescaler values : - 4, 8, 16, 32, 64, 128, 256 and 512
-	Each of the 4 channels can have a different gain. Valid values are : 1,2,4,8,16,32 and 64
	Each of the 4 channels can have a different mode. The 4 modes are : - INTERNAL. For example for temperature measurement

	 - SINGLE_ENDED. For measuring positive voltages - DIFF. For differential input without gain which allows to measure negative voltages. - DIFFWGAIN. Same as DIFF but with gain.
mux	Selects the MUX to use with the channel. This must be a numeric constant. The value depends on the mode. See page 284 of the XMega manual. At run time you can change the ADC \mathbf{x} _CH \mathbf{y} _MUXCTRL register. Where x is A or B, and y is the channel 0-3

See also

GETADC 7007, CONFIG ADC 4517

Example

```
·-----
```

```
'setup the ADC-A converter
Config Adca = Single , Convmode = Unsigned , Resolution = 12bit , Dma =
Off , Reference = Intlv , Event_mode = None , Prescaler = 32 , Ch0_gain
= 1 , Ch0_inp = Single_ended , Mux0 = 0 'you can setup other
channels as well
```

W = Getadc(adca, 0)

6.111 CONFIG ATEMU

Action

Configures the PS/2 keyboard data and clock pins.

Syntax

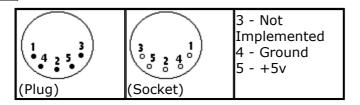
CONFIG ATEMU = int , DATA = data, CLOCK=clock [,INIT=VALUE]

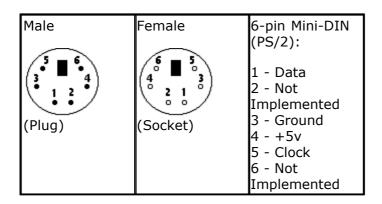
Remarks

Int	The interrupt used such as INTO or INT1.
	The pin that is connected to the DATA line. This must be the same pin as the used interrupt.
CLOCK	The pin that is connected to the CLOCK line.
	An optional value that will identify the keyboard. By default or when omitted this is &HAB83. The code that identifies a keyboard. Some mother boards/BIOS seems to require the reverse &H83AB. By making it an option you can pass any possible value. The MSB is passed first, the LSB last.

Male	Female	5-pin DIN (AT/ XT):
		1 - Clock 2 - Data

456





Old PC's are equipped with a 5-pin DIN female connector. Newer PC's have a 6-pin mini DIN female connector.

The male sockets must be used for the connection with the micro.

Besides the DATA and CLOCK you need to connect from the PC to the micro, you need to connect ground. You can use the +5V from the PC to power your microprocessor.

The config statement will setup an ISR that is triggered when the INT pin goes low. This routine you can find in the library.

The ISR will retrieve a byte from the PC and will send the proper commands back to the PC.

The SENDSCANKBD statement allows you to send keyboard commands.

Note that unlike the mouse emulator, the keyboard emulator is also recognized after your PC has booted.

The PS2 Keyboard and mouse emulator needs an additional commercial addon library.

See also

SENDSCANKBD 864

Example

'name 'copyright 'purpose

: ps2_kbdemul.bas

: (c) 1995-2005, MCS Electronics

: PS2 AT Keyboard emulator

457

```
'micro
                        : 90S2313
'suited for demo
                       : no, ADD ONE NEEDED
'commercial addon needed : yes
·_____
                                           _____
 _____
$regfile = "2313def.dat"
                                                         ' specify
the used micro
                                                         ' used
$crystal = 4000000
crystal frequency
                                                         ' use baud
$baud = 19200
rate
$hwstack = 32
                                                        ' default
use 32 for the hardware stack
                                                        ' default
\$swstack = 10
use 10 for the SW stack
$framesize = 40
                                                         ' default
use 40 for the frame space
$lib "mcsbyteint.lbx"
                                                        ' use
optional lib since we use only bytes
'configure PS2 AT pins
Enable Interrupts
                                                         ' you need
to turn on interrupts yourself since an INT is used
Config Atemu = Int1 , Data = Pind.3 , Clock = Pinb.0
                 ^----- used interrupt
                             ^----- pin connected to DATA
                                 ^-- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin
Waitms 500
                                                         ' optional
delay
'rcall _AT_KBD_INIT
Print "Press t for test, and set focus to the editor window"
Dim Key2 As Byte , Key As Byte
Do
   Key2 = Waitkey()
                                                         ' get key
from terminal
   Select Case Key2
     Case "t" :
     Waitms 1500
                                                         ' send a
     Sendscankbd Mark
scan code
    Case Else
   End Select
Loop
Print Hex(key)
                                                        ' send mark
Mark:
Data 12 , &H3A , &HF0 , &H3A , &H1C , &HF0 , &H1C , &H2D , &HF0 , &H2D ,
&H42 , &HF0 , &H42
 ^ send 12 bytes
1
                                                  r
          m
                              а
k
```

6.112 CONFIG BASE

Action

This option specifies the lower boundary of all arrays.

Syntax

CONFIG BASE= value

Remarks

By default the first element of an array starts at 1. With CONFIG BASE=0 you can override this default so that all arrays start at 0.

In some cases it is simpler that elements start at 0.

A constant named $_\mathsf{BASE}$ reflects the setting. You can not change the <code>BASE</code> at run time.

When you change this setting in existing code, you need to alter your code. For example when you used this code:

Dim a(10) as byte : a(10) = 10

And you set CONFIG BASE=0, it will mean that element 10 is invalid.

While in QB an additional element is created, this is not a good idea in bascom because it will require more space.

See also

NONE

Example

CONFIG BASE=0 Dim ar(10) as byte , j as byte For j=0 to 9 ar(j)=j Next

6.113 CONFIG BCCARD

Action

Initializes the pins that are connected to the BasicCard.

Syntax

CONFIG BCCARD = port , IO=pin, RESET=pin

Remarks

Port	The PORT of the micro that is connected to the BasicCard. This can be PORTB or PORTD and will depend on the used micro.
	The pin number that is connected to the IO of the BasicCard. Must be in the range from 0-7
	The pin number that is connected to the RESET of the BasicCard. Must be in the range from 0-7

The variables SW1, SW2 and $_BC_PCB$ are automatically dimensioned by the CONFIG BCCARD statement.

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

See Also

BCRESET 1002, BCDEF 996, BCCALL 996

Example

BCCARD.BAS ' This AN shows how to use the BasicCard from Zeitcontrol www.basiccard.com _____ _____ 'connections: ' C1 = +5V' C2 = PORTD.4 - RESET ' C3 = PIN 4 - CLOCK ' C5 = GND ' C7 = PORTD.5 - I/O /-----\ C1 C5 C2 C6 ÷ C3 C7 C4 C8 _____ '----- configure the pins we use ------**Config** Bccard = PORTD , Io = 5 , **Reset** = 4 ^ PORTD.4 ^----- PORTD.5 ^----- PORT D 'Load the sample calc.bas into the basiccard ' Now define the procedure in BASCOM ' We pass a string and also receive a string Bcdef Calc(string) 'We need to dim the following variables 'SW1 and SW2 are returned by the BasicCard 'BC_PCB must be set to 0 before you start a session 'Our program uses a string to pass the data so DIM it Dim S As String * 15 'Baudrate might be changed **\$baud** = 9600 ' Crystal used must be 3579545 since it is connected to the Card too **\$crystal** = 3579545

```
460 BASCOM-AVR
```

```
'Perform an ATR
Bcreset
'Now we call the procedure in the BasicCard
'bccall funcname(nad,cla,ins,p1,p2,PRM as TYPE,PRM as TYPE)
                                                          ' we want to
S = "1+1+3"
calculate the result of this expression
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
                                  ^--- variable to pass that holds the
expression
                             ^---- P2
                         ^----- P1
                      ^____ INS
                 ^----- CLA
           ^____ NAD
'For info about NAD, CLA, INS, P1 and P2 see your BasicCard manual
'if an error occurs ERR is set
' The BCCALL returns also the variables SW1 and SW2 \,
Print "Result of calc : " ; S
Print "SW1 = " ; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb) ' for test you can see that it toggles between 0
and 40
Print "Error : " ; Err
'You can call this or another function again in this session
S = "2+2"
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Result of calc : " ; S
Print "SW1 = " ; Hex(sw1)
Print "SW2 = " ; Hex(sw2)
'Print Hex(_bc_pcb) ' for test you can see that it toggles between 0
and 40
Print "Error : " ; Err
'perform another ATR
Bcreset
Input "expression " , S
Bccall Calc(0 , &H20 , 1 , 0 , 0 , S)
Print "Answer : " ; S
'----and now perform an ATR as a function
Dim Buf(25) As Byte , I As Byte
Buf(1) = Bcreset()
For I = 1 To 25
 Print I ; " " ; Hex(buf(i))
Next
'typical returns :
'TS = 3B
'TO = EF
'TB1 = 00
'TC1 = FF
'TD1 = 81 T=1 indication
'TD2 = 31 TA3, TB3 follow T=1 indicator
'TA3 = 50 or 20 IFSC ,50 =Compact Card, 20 = Enhanced Card
'TB3 = 45 BWT blocl waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00
        BasicCard ZC 12
                                                  3
```

```
'and another test
'define the procedure in the BasicCard program
Bcdef Paramtest(byte , Word , Long )
'dim some variables
Dim B As Byte , W As Word , L As Long
'assign the variables
B = 1 : W = &H1234 : L = &H12345678
Bccall Paramtest(0 , &HF6 , 1 , 0 , 0 , B , W , L)
Print Hex(sw1) ; Spc(3) ; Hex(sw2)
'and see that the variables are changed by the BasicCard !
Print B ; Spc(3) ; Hex(w) ; " " ; Hex(1)
'try the echotest command
Bcdef Echotest(byte)
Bccall Echotest(0 , &HC0 , &H14 , 1 , 0 , B)
Print B
```

'end program

6.114 CONFIG CLOCK

Action

End

Configures the timer to be used for the TIME\$ and DATE\$ variables.

Syntax

CONFIG CLOCK = soft | USER [, GOSUB = SECTIC] [,RTC=rtc]

Remarks

Soft	Use SOFT for using the software based clock routines. Use USER to write/ use your own code in combination with an I2C clock chip for example.
Sectic	This option allows to jump to a user routine with the label sectic.
	Since the interrupt occurs every second you may handle various tasks in the sectic label. It is important that you use the name SECTIC and that you return with a RETURN statement from this label.
	The usage of the optional SECTIC routine will use 30 bytes of the hardware stack. This option only works with the SOFT clock mode. It does not work in USER mode.
RTC	This option is only available for processors with an RTC (XMEGA). This option sets the RTC clock source.
	Valid parameters are : 1KHZ_INT32KHZ_ULP 1 kHz from internal 32 kHz ULP 1KHZ_32KHZ_CRYSTOSC 1 kHz from 32 kHz Crystal Oscillator on TOSC 1KHZ_INT32KHZ_RCOSC 1 kHz from internal 32 kHz RC Oscillator 32KHZ_32KHZ_CRYSTOSC 32 kHz from 32 kHz Crystal Oscillator on TOSC
	The 1KHz clocks will load the PER register with 1000-1 and the 32 KHz clock will load PER with a value of 32768-1.

462	BASCOM-AVR
-----	------------

The overflow mode is used and you can use the compare overflow if required. Do not forget to enable the 32 KHz oscillator and the interrupts as shown in the Xmega example.

When you use the CONFIG CLOCK directive the compiler will DIM the following variables automatic : _sec , _min , _hour, _day , _month , _year The variables TIME\$ and DATE\$ will also be dimensioned. These are special variables since they are treated different. See $\underline{TIME\$}$ and $\underline{DATE\$}$ and $\underline{DATE\$}$

The _sec, _min and other internal variables can be changed by the user too. But of course changing their values will change the DATE\$/TIME\$ variables.

The compiler also creates an ISR that gets updates once a second. This works only for the 8535, M163 and M103 and M603, or other AVR chips that have a timer that can work in asynchrony mode.

For the 90S8535, timer2 is used. It can not be used my the user anymore! This is also true for the other chips async timer.

Notice that you need to connect a 32768 Hz crystal in order to use the timer in async mode, the mode that is used for the clock timer.

When you choose the USER option, only the internal variables are created. With the USER option you need to write the clock code yourself.

See the datetime.bas example that shows how you can use a DS1307 clock chip for the date and time generation.

Numeric Values to calculate with Date and Time:

- SecOfDay: (Type LONG) Seconds elapsed since Midnight. 00:00:00 start with 0 to 85399 at 23:59:59.
- SysSec: (Type LONG) Seconds elapsed since begin of century (at 2000-01-01!). 00:00:00 at 2000-01-01 start with 0 to 2147483647 (overflow of LONG-Type) at 2068-01-19 03:14:07
- DayOfYear: (Type WORD) Days elapsed since first January of the current year.
- First January start with 0 to 364 (365 in a leap year)
- SysDay: (Type WORD) Days elapsed since begin of century (at 2000-01-01!). 2000-01-01 starts with 0 to 36524 at 2099-12-31
- DayOfWeek: (Type Byte) Days elapsed since Monday of current week. Monday start with 0 to Sunday = 6

With the numeric type calculations with Time and date are possible. Type 1 (discrete Bytes) and 2 (Strings) can be converted to an according numeric value. Than Seconds (at SecOfDay and SysSec) or Days (at DayOfYear, SysDay), can be added or subtracted. The Result can be converted back.

See also

TIME\$ 9300 , DATE\$ 6251 , CONFIG DATE 4731

ASM

The following ASM routines are called from datetime.lib _soft_clock. This is the ISR that gets called once per second.

Example

```
_____
                        : megaclock.bas
'name
'copyright
                        : (c) 1995-2005, MCS Electronics
                        : shows the new TIME$ and DATE$ reserved
'purpose
variables
'micro
                        : Mega103
'suited for demo
                         : yes
'commercial addon needed : no
·-----
                                    _____
_____
$regfile = "m103def.dat"
                                                          ' specify
the used micro
$crystal = 4000000
                                                          ' used
crystal frequency
$baud = 19200
                                                          ' use baud
rate
$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 10
                                                          ' default
use 10 for the SW stack
$framesize = 40
                                                          ' default
use 40 for the frame space
'With the 8535 and timer2 or the Mega103 and TIMER0 you can
'easily implement a clock by attaching a 32768 Hz xtal to the timer
'And of course some BASCOM code
'This example is written for the STK300 with M103
Enable Interrupts
'[configure LCD]
$lcd = &HC000
                                                          'address for
E and RS
$lcdrs = &H8000
                                                          'address for
only E
Config Lcd = 20 \times 4
                                                          'nice
display from bg micro
Config Lcdbus = 4
                                                          'we run it
in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus
                                                          'tell about
the bus mode
'[now init the clock]
Config Date = Mdy , Separator = /
                                                          ' ANSI-
Format
Config Clock = Soft
                                                          'this is how
simple it is
'The above statement will bind in an ISR so you can not use the TIMER
anymore!
'For the M103 in this case it means that TIMER0 can not be used by the
user anymore
'assign the date to the reserved date$
```

464 BASCOM-AVR

```
'The format is MM/DD/YY
Date$ = "11/11/00"
'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used
Time$ = "02:20:00"
*_____
'clear the LCD display
Cls
Do
 Home
                                                            'cursor home
 Lcd Date$ ; " " ; Time$
                                                            'show the
date and time
Loop
'The clock routine does use the following internal variables:
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
day = 1
'For the _year variable only the year is stored, not the century
End
```

Xmega Sample

-----(c) 1995-2010, MCS xm128-RTC.bas ' This sample demonstrates the Xmega128A1 RTC \$regfile = "xml28aldef.dat"
\$crystal = 32000000 hwstack = 64\$swstack = 64\$framesize = 64 'include the following lib and code, the routines will be replaced since they are a workaround \$lib "xmega.lib" \$external _xmegafix_clear \$external _xmegafix_rol_r1014 Config Portb = Output 'First Enable The Osc Of Your Choice , make sure to enable 32 KHz clock or use an external 32 KHz clock Config Osc = Enabled , 32mhzosc = Enabled , 32khzosc = Enabled ' For the CLOCK we use the RTC so make sure the 32 KHZ osc is enabled!!! 'configure the systemclock **Config** Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1 **Config** Com1 = 19200 , **Mode** = Asynchroneous , Parity = None , Stopbits = 1, Databits = 8Open "COM1:" For Binary As #1 Config Clock = Soft , Rtc = 1khz_int32khz_ulp ' we select the internal 1 KHz clock from the 32KHz internal oscillator 'the following clocks can be used to clock the RTC 1KHZ_INT32KHZ_ULP
 1 kHz from internal 32 kHz ULP
 1 kHz_32KHZ_CRYSTOSC
 1 kHz from 32 kHz Crystal Oscillator on TOSC

BASCOM Language Reference 465

```
' 1KHZ INT32KHZ RCOSC
                         1 kHz from internal 32 kHz RC Oscillator
' 32KHZ 32KHZ CRYSTOSC 32 kHz from 32 kHz Crystal Oscillator on TOSC
Config Priority = Static , Vector = Application , Lo = Enabled
the RTC uses LO priority interrupts so these must be enabled !!!
Enable Interrupts
                                                             ' as usual
interrupts must be enabled
Do
 Print TimeS
                                                             ' print the
time
 Waitms 1000
Loop
Sectic:
 Toggle Portb
                                                             'optional
toggle some leds when using the gosub=sectic option
Return
```

6.115 CONFIG CLOCKDIV

Action

Sets the clock divisor.

Syntax

CONFIG CLOCKDIV = constant

Remarks

constant The clock division factor to use. Possible values are 1, 2, 4, 8, 16, 32, 64, 128 and 256.

The options to set the clock divisor is available in most new chips. Under normal conditions the clock divisor is one. Thus an oscillator value of 8 MHz will result in a system clock of 8 MHz. With a clock divisor of 8, you would get a system clock of 1 MHz.

Low speeds can be used to generate an accurate system frequency and for low power consumption.

Some chips have a 8 or 16 division enabled by default by a fuse bit.

You can then reprogram the fuse bit or you can set the divisor from code.

When you set the clock divisor take care that you adjust the \$CRYSTAL directive also. \$CRYSTAL specifies the clock frequency of the system. So with 8 MHz clock and divisor of 8 you would specify \$CRYSTAL = 1000000.

See also

SCRYSTAL 316

Example

CONFIG $\overline{CLOCKDIV} = 8$ 'we divide 8 MHz crystal clock by 8 resulting in 1 MHz speed

6.116 CONFIG COM1

466

Action

Configures the UART of AVR chips that have an extended UART like the M8.

Syntax

CONFIG COM1 = baud , synchrone=0|1,parity=none|disabled|even|odd,stopbits=1| 2,databits=4|6|7|8|9,clockpol=0|1

Remarks

baud	Baud rate to use. Use 'dummy' to leave the baud rate at the \$baud value.
synchrone	0 for asynchrone operation (default) and 1 for synchrone operation.
Parity	None, disabled, even or odd
Stopbits	The number of stop bits : 1 or 2
Databits	The number of data bits : 4,5,7,8 or 9.
Clockpol	Clock polarity. 0 or 1.

Note that not all AVR chips have the extended UART. Most AVR chips have a UART with fixed communication parameters. These are : No parity, 1 stop bit, 8 data bits.

Normally you set the BAUD rate with \$BAUD or at run time with BAUD. You may also set the baud rate when you open the COM channel. It is intended for the Mega2560 that has 4 UARTS and it is simpler to specify the baud rate when you open the channel. It may also be used with the first and second UART but it will generate additional code since using the first UART will always result in generating BAUD rate init code.

See Also

CONFIG COM2 4667, CONFIG COMX 4697

Example

· ·		-
'name 'copyright 'purpose 'micro 'suited for demo 'commercial addon needed	: : (c) 1995-2005, MCS Electronics : test for M128 support in M128 mode : Megal28 : yes : no	_
<pre>\$regfile = "m128def.dat"</pre>	' specify	
the used micro \$crystal = 4000000	' used	
crystal frequency \$baud = 19200	' use baud	
rate \$baud1 = 19200 \$hwstack = 32	' default	

```
use 32 for the hardware stack
\$swstack = 10
                                                             ' default
use 10 for the SW stack
$framesize = 40
                                                             ' default
use 40 for the frame space
'By default the M128 has the M103 compatibility fuse set. Set the fuse
to M128
'It also runs on a 1 MHz internal oscillator by default
'Set the internal osc to 4 MHz for this example DCBA=1100
'use the ml28def.dat file when you wanto to use the Ml28 in Ml28 mode
'The M128 mode will use memory from $60-$9F for the extended registers
'Since some ports are located in extended registers it means that some
statements
'will not work on these ports. Especially statements that will set or
reset a bit
'in a register. You can set any bit yourself with the PORTF.1=1
statement for example
'But the I2C routines use ASM instructions to set the bit of a port.
These ASM instructions may
'only be used on port registers. PORTF and PORTG will not work with I2C.
'The M128 has an extended UART.
'when CONFIG COMx is not used, the default N,8,1 will be used
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'try the second hardware UART
Open "com2:" For Binary As #1
'try to access an extended register
Config Portf = Output
'Config Portf = Input
Print "Hello"
Dim B As Byte
Do
   Input "test serial port 0" , B
   Print B
  Print #1 , "test serial port 2"
Loop
Close #1
End
```

6.117 CONFIG COM2

Action

Configures the UART of AVR chips that have a second extended UART like the M128.

Syntax

```
CONFIG COM2 = baud , synchrone=0|1,parity=none|disabled|even|odd,stopbits=1|
2,databits=4|6|7|8|9,clockpol=0|1
```

Remarks

baud	Baud rate to use. Use 'dummy' to leave the baud rate at the \$baud1 value.	
synchrone	0 for asynchrone operation (default) and 1 for synchrone operation.	
Parity	None, disabled, even or odd	
Stopbits	The number of stopbits : 1 or 2	
Databits	The number of databits : 4,5,7,8 or 9.	
Clockpol	l Clock polarity. 0 or 1.	

Normally you set the BAUD rate with \$BAUD or at run time with BAUD. You may also set the baud rate when you open the COM channel. It is intended for the Mega2560 that has 4 UARTS and it is simpler to specify the baud rate when you open the channel. It may also be used with the first and second UART but it will generate additional code since using the first or second UART will always result in generating BAUD rate init code.

Note that not all AVR chips have the extended UART. Most AVR chips have a UART with fixed communication parameters. They are : No parity, 1 stopbit, 8 data bits.

See Also

CONFIG COM1 4667, CONFIG COMX 4697

Example

·	
'name	:
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: test for M128 support in M128 mode
'micro	: Mega128
'suited for demo	: yes
'commercial addon needed	: no

```
$regfile = "m128def.dat"
                                                               ' specify
the used micro
$crystal = 4000000
                                                               ' used
crystal frequency
$baud = 19200
                                                               ' use baud
rate
$baud1 = 19200
$hwstack = 32
                                                               ' default
use 32 for the hardware stack
\$swstack = 10
                                                               ' default
use 10 for the SW stack
                                                               ' default
$framesize = 40
use 40 for the frame space
```

'By default the M128 has the M103 compatibility fuse set. Set the fuse to M128 'It also runs on a 1 MHz internal oscillator by default 'Set the internal osc to 4 MHz for this example DCBA=1100

'use the m128def.dat file when you wanto to use the M128 in M128 mode

BASCOM Language Reference

```
nce 469
```

```
'The M128 mode will use memory from $60-$9F for the extended registers
'Since some ports are located in extended registers it means that some
statements
'will not work on these ports. Especially statements that will set or
reset a bit
'in a register. You can set any bit yourself with the PORTF.1=1
statement for example
'But the I2C routines use ASM instructions to set the bit of a port.
These ASM instructions may
'only be used on port registers. PORTF and PORTG will not work with I2C.
'The M128 has an extended UART.
'when CONFIG COMx is not used, the default N,8,1 will be used
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'try the second hardware UART
Open "com2:" For Binary As #1
'try to access an extended register
Config Portf = Output
'Config Portf = Input
Print "Hello"
Dim B As Byte
Do
   Input "test serial port 0" , B
  Print B
   Print #1 , "test serial port 2"
Loop
Close #1
End
```

6.118 CONFIG COMx

Action

Configures the UART of AVR chips that have an extended UART like the M2560.

Syntax

CONFIG COMx = baud , synchrone=0|1,parity=none|disabled|even|odd,stopbits=1| 2,databits=4|6|7|8|9,clockpol=0|1

Syntax Xmega

CONFIG COMx = baud , Mode=mode, Parity=parity, Stopbits=stopbits, Databits=databits,clockpol=Clockpol

Remarks

COMx	The COM port to configure. Value in range from 1-4
baud	Baud rate to use.
synchrone	0 for asynchrone operation (default) and 1 for synchrone operation.
Parity	None, disabled, even or odd

470 BASCOM-AVR

Stopbits	The number of stop bits : 1 or 2
Databits	The number of data bits : 4,5,7,8 or 9.
Clockpol	Clock polarity. 0 or 1.

Note that not all AVR chips have the extended UART. Most AVR chips have a UART with fixed communication parameters. These are : No parity, 1 stopbit, 8 data bits.

The Mega2560 does support 4 UART's.

Remarks Xmega

COMx	The COM port to configure. Value in range from 1-8	
baud	Baud rate to use. If the baud rate can be generated accurately depends on the system clock.	
mode	The USART mode, this can be : - ASYNCHRONEOUS or 0 (default) for asynchronous operation. - SYNCHRONEOUS or 1 , for synchronous operation. - IRDA or IRCOM for IRDA operation - SPI or MSPI for operation as SPI controller	
Parity	None, disabled, even or odd	
Stopbits	The number of stop bits : 1 or 2	
Databits	The number of data bits : 5,6,7,8 or 9.	

In the Xmega the registers have a fixed offset. This allows to use dynamic UARTS : you can change settings at run time by using a variable. This will use some more code when using just one UART but will save code when using multiple UARTS because you need only one copy of the code.

In the Xmega you MUST use CONFIG COM before you can use the UART. The CONFIG commands makes a call to _INIT_XMEGA_UART where the various parameters are passed to setup the UART. You also need to specify the baud rate. Do not use \$BAUD.

The CLOCKPOL for the SPI mode has been removed, it will be added to a configuration command for the SPI.

UART	TX pin	RX pin
COM1	PORTC.3	PORTC.2
COM2	PORTC.7	PORTC.6
СОМЗ	PORTD.3	PORTD.2
COM4	PORTD.7	PORTD.6
COM5	PORTE.3	PORTE.2
COM6	PORTE.7	PORTE.6
COM7	PORTF.3	PORTF.2
COM8	PORTF.7	PORTF.6

The CONFIG COM will set the TX pin to output mode. This are the following pins :

See Also

CONFIG COM1 4667, CONFIG COM2 4667

Example

```
'name
                           :
                           : (c) 1995-2008, MCS Electronics
'copyright
'purpose
                            : test for M2560 support
'micro
                            : Mega2560
'suited for demo
                            : yes
'commercial addon needed : no
·_____
$regfile = "m2560def.dat"
                                                                 ' specify the used mich
                                                                 ' used crystal frequence
$crystal = 8000000
                                                                 ' default use 32 for th
\$hwstack = 40
                                                                ' default use 10 for th
\$swstack = 40
$framesize = 40
                                                                ' default use 40 for th
'The M128 has an extended UART.
'when CO'NFIG COMx is not used, the default N,8,1 will be used
Config Com1 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Config Com3 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8
Config Com4 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8
'Open all UARTS
Open "com2:" For Binary As #1
Open "Com3:" For Binary As #2
Open "Com4:" For Binary As #3
Print "Hello"
                                                                 'first uart
Dim B As Byte
Dim Tel As Word
Do
  Incr Tel
  Print Tel ; " test serial port 1"
  Print #1 , Tel ; " test serial port 2"
  Print #2 , Tel ; " test serial port 3"
  Print #3 , Tel ; " test serial port 4"
  B = Inkey(#3)
  If B <> 0 Then
     Print #3 , B ; " from port 4"
  End If
  Waitms 500
Loop
Close #1
Close #2
Close #3
End
```

6.119 CONFIG DACx

Action

This statement configures the DACA or DACB in the Xmega.

Syntax

CONFIG DACx=dac, IO0=IO0, IO1=IO1, INTERNAL_OUTPUT =INTOTP, CHANNEL=channel, TRIGGER_CH0=trig0, TRIGGER_CH1=trig1, REFERENCE= ref, LEFT_ADJUSTED=adjusted, EVENT_CHANNEL=event, INTERVAL=interval, REFRESH=refresh

<u>Remarks</u>	
DACX	Chose either DACA or DACB. DACA is connected to PORTA. DACB is connected to PORTB.
dac	ENABLED or DISABLED. Chose ENABLED to enable the DAC.
IOO	ENABLED or DISABLED. Chose ENABLED to enable output 0. Each DAC has 2 outputs. When multiple outputs are used, the DAC is using S&H.
IO1	ENABLED or DISABLED. Chose ENABLED to enable output 1.
Intotp	ENABLED or DISABLED. Chose ENABLED to enable the internal output.
Channel	SINGLE or DUAL. If both outputs are used, you need to enable the second output with IO1.
Trig0	ENABLED or DISABLED. Chose ENABLED to enable the trigger of channel 0.
Trig1	ENABLED or DISABLED. Chose ENABLED to enable the trigger of channel 1.
Ref	 The DAC needs a stable voltage reference. You can chose one of the following: INT1V. This will select the internal 1V reference AVCC. This will use AVCC as reference. AREFA. This will use AREFA as reference. AREFB. This will use AREFB as reference. The output of the DAC can never be higher then the voltage reference. When you chose INT1V, the output is from 0-1V in 4096 steps.
Adjusted	ENABLED or DISABLED. By default the DAC output is right adjusted. You can left alight the result.
Event	The event channel to use for the event system.
Interval	The minimum interval between 2 conversions. This is a value of : 1,2,4,8,16,32,64 or 128. The default in the register is 64. A value of 64 will give an interval of 64 clock cycles.
Refresh	The DAC channel refresh timing. This is the interval refresh time in DUAL channel mode. Possible values: OFF 16, 32, 128, 256, 512, 1014, 2048, 4096, 8192, 16384, 32768, 65536. A value of 16 means an interval of 16 clock cycles. The default loaded is 64.

Remarks

The DAC data register is available in the DACA0, DACA1 and DACB0 and DACB1 variables.

See also START 9047, STOP 9107

Example

```
Config Daca = Enabled , Io0 = Enabled , Channel = Single , Reference =
Intlv , Interval = 64 , Refresh = 64
Daca0 = 4095 '1 V output
on porta.2
```

See also the example **xm128a1.bas** from the samples\chips folder

6.120 CONFIG DATE

Action

Configure the Format of the Date String for Input to and Output from BASCOM – Date functions

Syntax

CONFIG DATE = DMY , Separator = char

Remarks

DMY	The Day, month and year order. Use DMY, MDY or YMD.
Char	A character used to separate the day, month and year.
	Use / , - or . (dot)

The following table shows the common formats of date and the associated statements.

Country	Format	Statement
American	mm/dd/yy	Config Date = MDY, Separator = /
ANSI	yy.mm.dd	Config Date = YMD, Separator = .
Britisch/ French	dd/mm/yy	Config Date = DMY, Separator = /
German	dd.mm.yy	Config Date = DMY, Separator = .
Italian	dd-mm-yy	Config Date = DMY, Separator = -
Japan/Taiwan	yy/mm/dd	Config Date = YMD, Separator = /
USA	mm-dd-yy	Config Date = MDY, Separator = -

When you live in Holland you would use : CONFIG DATE = DMY, separator = -This would print 24-04-02 for 24 November 2002.

When you line in the US, you would use : CONFIG DATE = MDY , separator = / This would print 04/24/02 for 24 November 2002.

See also

<u>CONFIG CLOCK</u> 46ने, <u>DATE TIME functions</u> (993), <u>DayOfWeek</u> 615), <u>DayOfYear</u> 624),

474 BASCOM-AVR

SecOfDay [849], SecElapsed [849], SysDay [917], SysSec[918], SysSecElapsed [916], Time [937], Date [627]

Example

```
_____
'name
                        : megaclock.bas
'copyright
                        : (c) 1995-2005, MCS Electronics
                        : shows the new TIME$ and DATE$ reserved
'purpose
variables
'micro
                        : Mega103
'suited for demo
                         : yes
'commercial addon needed : no
·_____
_____
$regfile = "m103def.dat"
                                                          ' specify
the used micro
$crystal = 4000000
                                                          ' used
crystal frequency
$baud = 19200
                                                          ' use baud
rate
$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 10
                                                          ' default
use 10 for the SW stack
$framesize = 40
                                                          ' default
use 40 for the frame space
'With the 8535 and timer2 or the Mega103 and TIMERO you can
'easily implement a clock by attaching a 32768 Hz xtal to the timer
'And of course some BASCOM code
'This example is written for the STK300 with M103
Enable Interrupts
'[configure LCD]
$1cd = &HC000
                                                          'address for
E and RS
$lcdrs = &H8000
                                                          'address for
only E
Config Lcd = 20 \times 4
                                                          'nice
display from bg micro
Config Lcdbus = 4
                                                          'we run it
in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus
                                                          'tell about
the bus mode
'[now init the clock]
Config Date = Mdy , Separator = /
                                                          ' ANSI-
Format
Config Clock = Soft
                                                          'this is how
simple it is
'The above statement will bind in an ISR so you can not use the TIMER
anymore!
'For the M103 in this case it means that TIMER0 can not be used by the
user anymore
'assign the date to the reserved date$
'The format is MM/DD/YY
Date$ = "11/11/00"
```

```
'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used
Times = "02:20:00"
*_____
'clear the LCD display
Cls
DO
 Home
                                                        'cursor home
 Lcd Date$ ; " " ; Time$
                                                        'show the
date and time
Loop
'The clock routine does use the following internal variables:
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
_day = 1
'For the _year variable only the year is stored, not the century
End
```

6.121 CONFIG DCF77

Action

Instruct the compiler to use DCF-77 radio signal to get atom clock precision time

Syntax

CONFIG DCF77 = pin , timer = timer [INVERTED=inv, CHECK=check, UPDATE=upd, UPDATETIME=updtime , TIMER1SEC=tmr1sec, SWITCHPOWER=swpwr, POWERPIN=pin, POWERLEVEL = pwrlvl , SECONDTICKS=sectick ,DEBUG=dbg , GOSUB = Sectic]

Remarks

PIN	The input pin that is connected to the DCF-77 signal. This can be any micro processor pin that can be used as an input.
TIMER	The timer that is used to generate the compare interrupts, needed to determine the level of the DCF signal. Supported timers are : TIMER1.
INVERTED	This value is 0 by default. When you specify 1, the compiler will assume you use an inverted DCF signal. Most DCF-77 receivers have a normal output and an inverted output.
CHECK	 Check is 1 by default. The possible values are : O - The DCF-77 parity bits are checked. No other checks are performed. Use it when you have exceptional signal strength 1 - The received minutes are compared with the previous received minutes. And the difference must be 1. 2 - All received values(minutes, hours, etc.) are compared with their previous received values. Only the minutes must differ with 1, the other values must be exactly the same. This value uses more internal ram but it gives the best check. Use this when you have bad signal reception.

476 BASCOM-AVR

	Upd determines how often the internal date/time variables are updated with the DCF received values. The default value is 0 . There are 3 possible values : 0 - Continuous update. The date and time variables are updated every time the correct values have been received 1 - Hourly update. The date and time variables are updated once an hour. 2 - Daily update. The date and time variables are updated once a day. The UPDATE value also determines the maximum value of the UPDATETIME option. This value depends on the used UPDATE parameter. When UPDATE is 1, the value must be in the range from 0-59. Start every hour at this minute with the new update.
	When UPDATE is 2, the value must be in the range from 0-23. Start every day at this hour with the new update. The default is 0 .
TIMER1SEC	16 bit timers with the right crystal value can generate a precise interrupt that fires every second. This can be used to synchronize only once a day or hour with the DCF values. The remaining time, the 1-sec interrupt will update the soft clock. By default this value is 0.
SWITCHPOWER	This option can be used to turn on/off the DCF-77 module with the control of a port pin. The default is 0 . When you specify a value of 1 , the DCF receiver will be switched off to save power, as soon as the clock is synchronized.
POWERPIN	The name of a pin like pinB.2 that will be used to turn on/off the DCF module.
	This option controls the level of the output pin that will result in a power ON for the module. 0 - When a logic 0 is applied to the power pin, the module is ON. 1 - When a logic 1 is applied to the power pin, the module is ON. Use a transistor to power the module. Do not power it from a port PIN directly. When you do power from a pin, make sure you sink the current. Ie : connect VCC to module, and GND of the module to ground. A logic 0 will then turn on the module.
	The number of times that the DCF signal state is read. This is the number of times per second that the interrupt is executed. This value is calculated by the compiler. The highest possible timer pre scale value is used and the lowest possible number of times that the interrupt is executed. This gives least impact on your main application. You can override the value by defining your own value. For example when you want to run some own code in the interrupt and need it to execute more often.
	Optional value to fill 2 variables with debug info. DEBUG is on when a value of 1 is specified. By default, DEBUG is off. This has nothing to do with other DEBUG options of the compiler, it is only for the DCF77 code! When 1 is specified the compiler will create 2 internal variable named : bDCF_Pause and bDCF_Impuls. These values contain the DCF pulse length of the pause and the impulse. In the sample these values are printed.
GOSUB	The Sectic option will call a label in the main program every second. You have to insert this label yourself. You must also end it with a RETURN. The option is the same as used with <u>CONFIG CLOCK</u> [461]

The DCF decoding routines use a status byte. This byte can be examined as in the example.

ine t	bits have the following meaning.
Bit	Explanation
0	The last reading of the DCF pin.
1	This bit is reserved.
	This Bit is set, if after a complete time-stamp at second 58 the time-stamp is checked and it is OK. If after a minute mark (2 sec pause) this bit is set, the time from the DCF-Part is copied to the Clock-Part and this bit reset too. Every second mark also resets this bit. So time is only set, if after second 58 a minute mark follows. Normally this bit is only at value 1 from Second 58 to second 60/00.
	This Bit indicates, that the DCF-Part should be stopped, if time is set. (at the option of updating once per hour or day).
4	This Bit indicated that the DCF-Part is stopped.
5	This bit indicates, that the CLOCK is configured the way, that during DCF-Clock is stopped, there is only one ISR-Call in one second.
6	This Bit determines the level of the DCF input-pin at the pulse (100/200 mSec part).
7	This bit indicates, that the DCF-Part has set the time of the Clock-part.

See Also

DCF77TIMEZONE 637

You can read the Status-Bit 7 (DCF_Status.7), to check whether the internal clock was synchronized by the DCF-Part. You can also reset this Bit with <u>RESET</u> and DCF_Status.7. The DCF-Part will set this bit again, if a valid time-stamp is received. You can read all other bits, but don't change them.

The DCF-77 signal is broadcasted by the German Time and Frequency department. The following information is copied from : <u>http://www.ptb.de/en/org/4/44/ index.</u> <u>htm</u>

The main task of the department time and frequency is the realization and dissemination of the base unit time (second) and the dissemination of the legal time in the Federal Republic of Germany.

The second is defined as the duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyper fine levels of the ground state of the cesium-133 atom.

For the realization and dissemination of the unit of time, the department develops and operates cesium atomic clocks as primary standards of time and frequency. In the past decades, these, as the worldwide most accurate atomic clocks, have contributed to the international atomic time scale (TAI) and represent the basis for the legal time in Germany. Dissemination of the legal time to the various users in industry, society, and research is performed via satellite, via a low frequency transmitter DCF77 and via an internet- and telephone service.

The department participates in the tests for the future European satellite navigation system "Galileo".

478 BASCOM-AVR

Presently the primary clocks realizing the time unit are augmented by Cs clocks with laser cooled atoms ("Cs-fountain clocks") whose accuracy presently exceeds the clocks with thermal beams by a factor of 10 (frequency uncertainty of 1 . 10-15).

Future atomic clocks will most likely be based on atomic transitions in the optical range of single stored ions. Such standards are presently being developed along with the means to relate their optical frequencies without errors to radio-frequencies or 1 second pulsed.

As one may expect transitions in nuclei of atoms to be better shielded from environmental perturbations than electron-shell transitions which have been used so far as atomic clock references, the department attempts to use an optical transition in the nucleus of 229Th for a future generation of atomic clocks.

The work of the department is complemented by research in nonlinear optics (Solitons) and precision time transfer techniques, funded in the frame of several European projects and by national funding by Deutsche Forschungsgemeinschaft particularly in the frame of Sonderforschungsbereich 407 jointly with Hannover University.

The following information is copied from wikipedia : <u>http://en.wikipedia.org/wiki/</u> DCF77

The signal can be received in this area:



DCF77 is a long wave time signal and standard-frequency radio station. Its primary and backup transmitter are located in Mainflingen, about 25 km south-east of Frankfurt, Germany. It is operated by T-Systems Media Broadcast, a subsidiary of Deutsche Telekom AG, on behalf of the Physikalisch-Technische Bundesanstalt, Germany's national physics laboratory. DCF77 has been in service as a standardfrequency station since 1959; date and time information was added in 1973.

The 77.5 kHz carrier signal is generated from local atomic clocks that are linked with the German master clocks in Braunschweig. With a relatively-high power of 50 kW, the station can be received in large parts of Europe, as far as 2000 km from Frankfurt. Its signal carries an amplitude-modulated, pulse-width coded 1 bit/s data signal. The same data signal is also phase modulated onto the carrier using a 511-bit long pseudo random sequence (direct-sequence spread spectrum modulation). The transmitted data repeats each minute

Map showing the range of the DCF77 signal. Map showing the range of the DCF77 signal.

- * the current date and time;
- * a leap second warning bit;
- * a summer time bit;
- * a primary/backup transmitter identification bit;
- * several parity bits.

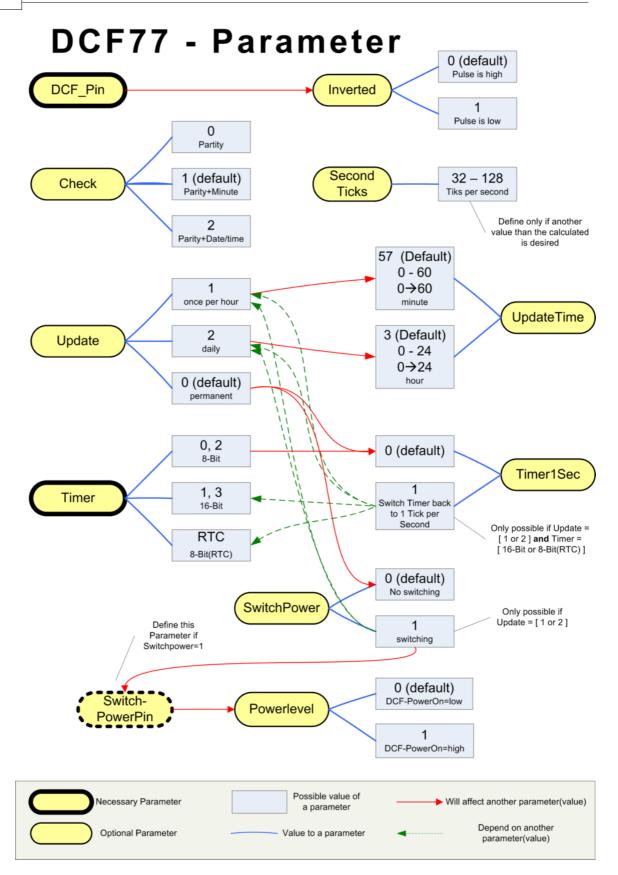
Since 2003, 14 previously unused bits of the time code have been used for civil defence emergency signals. This is still an experimental service, aimed to replace one day the German network of civil defense sirens.

The call sign stands for D=Deutschland (Germany), C=long wave signal, F=Frankfurt, 77=frequency: 77.5 kHz. It is transmitted three times per hour in morse code.

Radio clocks have been very popular in Europe since the late 1980s and most of them use the DCF77 signal to set their time automatically.

For further reference see wikipedia, a great on line information resource.

The DCF library parameters state diagram looks as following:



If the SECTIC option is used, the Sectic Interrupt routine should not need more time, than to the next timer interrupt. If you use a timer for dcf (and softclock) usually with 40 tics per second, the Sectic routine should take only less than 25msec. If the Sectic routines needs more than this limit, you will lose accuracy of the

softclock time (especially during the time, where the clock is not synchronized by DCF) and also measurement of the length of the DCF-pulses.

If the SECTIC routine needs more time than the short DCF-pulse (100ms, with some instability in DCF-receiver may be 80ms) you will lose synchronization with the DCF-signal.

It is the principle of the DCF-routine, that the timer-interrupt measures the DCF-Pulse length and if you need more time in the interrupt routine as the duration from one timer interrupt to the next, you will get a problem.

Thus keep the SECTIC routine as short as possible and set a flag in the SECTIC routine, which is checked in a loop of the main-program.

See also

CONFIG DATE 473

ASM

_DCF77 from DCF77.LBX is included by the compiler when you use the CONFIG statement.

Example

```
$reqfile = "M88def.dat"
$crystal = 8000000
$hwstack = 128
\$swstack = 128
$framesize = 128
$baud = 19200
'Config Dcf77 = Pind.2 , Debug = 1 , Inverted = 0 , Check = 2 , Update =
0 , Updatetime = 30 , Switchpower = 0 , Secondticks = 50 , Timer1sec = 1
, Powerlevel = 1 , Timer = 1
Config Dcf77 = Pind.2 , Timer = 1 , Timer1sec = 1 , Debug = 1
Enable Interrupts
Config Date = Dmy , Separator = .
Dim I As Integer
Dim Sec_old As Byte , Dcfsec_old As Byte
                                                              1:
Sec_old = 99 : Dcfsec_old = 99
DCF\_Debug\_Timer = 0
' Testroutine für die DCF77 Clock
Print "Test DCF77 Version 1.00"
Do
   For I = 1 To 78
      Waitms 10
      If Sec_old <> _sec Then
         Exit For
      End If
      If Dcfsec_old <> Dcf_sec Then
         Exit For
      End If
  Next
   Waitms 220
```

```
Sec_old = _sec
Dcfsec_old = Dcf_sec
Print Time$ ; " " ; Date$ ; " " ; Time(dcf_sec) ; " " ; Date(dcf_day)
; " " ; Bin(dcf_status) ; " " ; Bin(dcf_bits) ; " " ; Bdcf_impuls ; " "
; Bdcf_pause
Loop
End
```

6.122 CONFIG DEBOUNCE

Action

Configures the delay time for the DEBOUNCE statement.

Syntax

CONFIG DEBOUNCE = time

Remarks

Time

A numeric constant which specifies the delay time in mS.

When debounce time is not configured, 25 mS will be used as a default.

See also

DEBOUNCE 639

Example

```
_____
-----
_____
'name
                     : deboun.bas
'copyright
                     : (c) 1995-2005, MCS Electronics
'purpose
                     : demonstrates DEBOUNCE
                     : Mega48
'micro
'suited for demo
                      : yes
'commercial addon needed : no
1_____
_____
$regfile = "m48def.dat"
                                                   ' specify
the used micro
$crystal = 4000000
                                                   ' used
crystal frequency
$baud = 19200
                                                   ' use baud
rate
$hwstack = 32
                                                   ' default
use 32 for the hardware stack
\$swstack = 10
                                                   ' default
use 10 for the SW stack
$framesize = 40
                                                   ' default
use 40 for the frame space
                                                   'when the
Config Debounce = 30
config statement is not used a default of 25mS will be used
 'Debounce Pind.0 , 1 , Pr 'try this for branching when high(1)
 Debounce Pind.0 , 0 , Pr , Sub
```

```
Debounce Pind.0 , 0 , Pr , Sub
' ^----- label to branch to
  ÷.
                  ^----- Branch when P1.0 goes low(0)
            ^----- Examine P1.0
  r.
  'When Pind.O goes low jump to subroutine Pr
  'Pind.0 must go high again before it jumps again
  'to the label Pr when Pind.0 is low
 Debounce Pind.0 , 1 , Pr
                                                            'no branch
 Debounce Pind.0 , 1 , Pr
                                                            'will result
in a return without gosub
End
Pr:
 Print "PIND.0 was/is low"
Return
```

6.123 CONFIG DMA

Action

Configures the direct memory access (DMA) module of the XMEGA.

Syntax

CONFIG DMA=enabled|disabled, DOUBLEBUF=db, CPM=cpm

Remarks

DMA	By default the DMA is disabled. Use ENABLED to enable the module.
db	DOUBLE BUFFER This options will set the double buffer mode. By default is is DISABLED. To allow for continuous transfer, two channels can be interlinked so that the second takes over the transfer when the first is finished and vice versa. This is called double buffering. When a transmission is completed for the first channel, the second channel is enabled. When a request is detected on the second channel, the transfer starts and when this is completed the first channel is enabled again
	Modes : - DISABLED : No double buffer enabled - CH01 : Double buffer enabled on channel0/1 - CH23 : Double buffer enabled on channel2/3 - CH01CH23 : Double buffer enabled on channel0/1 and channel2/3
cpm	Channel Priority Mode If several channels request data transfer at the same time a priority scheme is available to determine which channel is allowed to transfer data. Application software can decide whether one or more channels should have a fixed priority or if a round robin scheme should be used. A round robin scheme means that the channel that last transferred data will have the lowest priority
	Modes : RR : Round Robin CH0RR123 : Channel0 > Round Robin (Channel 1, 2 and 3) CH01RR23 : Channel0 > Channel1 > Round Robin (Channel 2 and 3)

CH0123 : Channel0 > Channel1 > Channel2 > Channel3

You also need to set the individual DMA channels using CONFIG DMACHx.

See also

CONFIG DMACHx 4847, START DMACHx 9047

Example

See CONFIG DMACHx 484

6.124 CONFIG DMACHx

Action

Configures the direct memory access (DMA) channel of the XMEGA.

Syntax

CONFIG DMACHx=enabled|disabled,BURSTLEN=bl, CHANRPT=chrpt, CTR=ctr, SINGLESHOT=ss, TCI=tci, EIL=eil,SAR=sar, SAM=sam,DAR=dar,DAM=dam, TRIGGER,trig, BTC=btc, REPEAT=rpt,SADR=sadr, DADR=dadr

Remarks

In order to understand the various options better, we first have a better look at DMA. Normally, when you want to transfer data, the processor need to execute a number of operations.

The BASCOM MEMCOPY for example will use processor instructions like LD (load data) and ST(store data) in a loop.

If you want to clear 32KB of memory you need at least 32 K instructions. This will consume time, and all this time the processor can not handle other tasks.

In a PC, you do not want to use the processor to be busy when you load a file from disk. The DMA controller will handle this. It can move blocks of memory between devices.

DMA Transaction

A complete DMA read and write operation between memories and/or peripherals is called a DMA transaction.

A transaction is done in data blocks and the size of the transaction (number of bytes to transfer) is selectable from software and controlled by the block size and repeat counter

settings. Each block transfer is divided into smaller bursts

Block Transfer and Repeat

The size of the block transfer is set by the Block Transfer Count Register, and can be anything from 1 byte to 64 KBytes.

A repeat counter can be enabled to set a number of repeated block transfers before a transaction is complete. The repeat is from 1 to 255 and unlimited repeat count can be achieved by

setting the repeat count to zero.

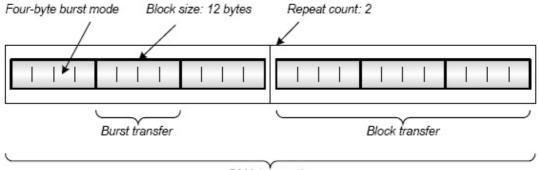
Burst Transfer

As the AVR CPU and DMA controller use the same data buses a block transfer is divided into smaller burst transfers. The burst transfer is selectable to 1, 2, 4, or 8 bytes.

This means that, if the DMA acquires a data bus and a transfer request is pending it will occupy the bus until all bytes in the burst transfer is transferred.

A bus arbiter controls when the DMA controller and the AVR CPU can use the bus. The CPU always has priority, so as long as the CPU request access to the bus, any pending burst transfer

must wait. The CPU requests bus access when it executes an instruction that write or read data to SRAM, I/O memory, EEPROM and the External Bus Interface



DMA transaction

DMACHx	There are 4 DMA channels numbered 0-3. By default these DMA channels are disabled. Use ENABLED to enable the channel.
bl	BURSTLEN Each DMA channel has an internal transfer buffer that is used for 2, 4 and 8 byte burst transfers. When a transfer is triggered, a DMA channel will wait until the transfer buffer contains two bytes before the transfer starts. For 4 or 8 byte transfer, any remaining bytes is transferred as soon as they are ready for a DMA channel. The buffer is used to reduce the time the DMA controller occupy the bus. Options : - 1 : 1 byte burst mode - 2 : 2 byte burst mode - 4 : 4 byte burst mode - 8 : 8 byte burst mode
chanrpt	Channel Repeat Setting this bit enables the repeat mode. In repeat mode, this bit is cleared by hardware in the beginning of the last block transfer. The REPCNT register should be configured before setting the REPEAT bit. When using the CONFIG command, the compiler will handle this. Options : Enabled : enabled repeat mode Disabled : disabled repeat mode
ctr	DMA Channel Transfer Request Setting this bit requests a data transfer on the DMA Channel. This bit is automatically cleared at the beginning of the data transfer Options : Enabled : request transfer
SS	DMA Channel Single Shot Data transfer Setting this bit enables the single shot mode. The channel will then do a

	burst transfer of BL bytes on the transfer trigger. This bit can not be
	changed if the channel is busy.
	Options :
	Enabled : enable SS mode.
tci	DMA Channel Transaction Complete Interrupt Level
	The interrupt can be turned OFF, or be given a pritoriy LO, MED or HI
eil	DMA Channel Error Interrupt Level
	The interrupt can be turned OFF, or be given a pritoriy LO, MED or HI
sar	Source Address Reload
	The channel source address can be reloaded the following way:
	NONE : No reload performed.
	BLOCK : DMA source address register is reloaded with initial value at
	end of
	each block transfer.
	BURST : DMA source address register is reloaded with initial value at
	end of
	each burst transfer.
	TRANSACTION : DMA source address register is reloaded with initial
	value at
	end of each transaction.
sam	Source Address Mode
	The address can be altered the following way :
	FIXED : The address remains the same.
	INC : The address is incremented by one
	DEC : The address is decremented by one
	If you want to write to a PORT, for example to generate a wave, you
	would chose FIXED. But if you want to move a block of memory, you
	want to use INC so the the source address is increased after each byte.
dar	Channel Destination Address Reload
uar	The channel destiny address can be reloaded the following way:
	NONE : No reload performed.
	BLOCK : DMA destiny address register is reloaded with initial value at
	and of
	end of
	each block transfer.
	each block transfer. BURST : DMA destiny address register is reloaded with initial value at
	each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of
	each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer.
	each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial
	each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at
	each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction.
dam	each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode
dam	each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way :
dam	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same.
dam	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one
dam	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one
dam	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you
dam	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you
dam	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you
	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you
dam trigger	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. Trigger Source Select
	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. Trigger Source Select The trigger selected which device triggers the DMA transfer. A zero (0)
	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. Trigger Source Select The trigger selected which device triggers the DMA transfer. A zero (0) will disable a trigger. You can find the values in the datasheet.
	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. Trigger Source Select The trigger selected which device triggers the DMA transfer. A zero (0) will disable a trigger. You can find the values in the datasheet. For example, EVENTSYS channel 0 would be 1. And EVENSTYS channel 1
trigger	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. Trigger Source Select The trigger selected which device triggers the DMA transfer. A zero (0) will disable a trigger. You can find the values in the datasheet. For example, EVENTSYS channel 0 would be 1. And EVENSTYS channel 1 would be 1.
	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. Trigger Source Select The trigger selected which device triggers the DMA transfer. A zero (0) will disable a trigger. You can find the values in the datasheet. For example, EVENTSYS channel 0 would be 1. And EVENSTYS channel 1 would be 1.
trigger	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. Trigger Source Select The trigger selected which device triggers the DMA transfer. A zero (0) will disable a trigger. You can find the values in the datasheet. For example, EVENTSYS channel 0 would be 1. And EVENSTYS channel 1 would be 1. Block Transfer Count The BTC represents the 16-bit value TRFCNT. TRFCNT
trigger	 each block transfer. BURST : DMA destiny address register is reloaded with initial value at end of each burst transfer. TRANSACTION : DMA destiny address register is reloaded with initial value at end of each transaction. Destiny Address Mode The address can be altered the following way : FIXED : The address remains the same. INC : The address is incremented by one DEC : The address is decremented by one If you want to write to a PORT, for example to generate a wave, you would chose FIXED. But if you want to move a block of memory, you want to use INC so the the source address is increased after each byte. Trigger Source Select The trigger selected which device triggers the DMA transfer. A zero (0) will disable a trigger. You can find the values in the datasheet. For example, EVENTSYS channel 0 would be 1. And EVENSTYS channel 1 would be 1.

487

	decremented after each byte read by the DMA channel. When TRFCNT reaches zero, the register is reloaded with the last value written to it. When repeat is 1, this is the total amount of bytes to send in the DMA transaction.
repeat	Repeat Counter Register REPCNTcounts how many times a block transfer is performed. For each block transfer this register will be decremented.
sadr	Source Address This is the address of the DMA source. For example, the address of a variable. Or the address of a register. Use <u>VARPTR</u> [946]() to find the address of a variable.
dadr	Destination Address The destiny address. Memory will be transferred to this address.

After you have configured the DMA channel, you can start the transfer with the START DMACHx statement.

This will write the TRFREQ bit in the CTRLA register.

See also

CONFIG DMA 483 , START DMACHx 904

Example

```
_____
                (c) 1995-2011, MCS
                  xm128A1-DMA.bas
 This sample demonstrates DMA with an Xmega128A1
$regfile = "xm128a1def.dat"
$crystal = 32000000
hwstack = 64
\$swstack = 40
$framesize = 40
'include the following lib and code, the routines will be
replaced since they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 38400 , Mode = Asynchroneous , Parity = None ,
Stopbits = 1 , Databits = 8
dim ar(100) as byte, dest(100) as byte, j as byte ,w as word
for j=1 to 100
 ar(j)=j ' create an array and assign a value
next
```

```
print "DMA DEMO"
config dma= enabled, doublebuf=disabled,cpm = RR ' enable DMA
'you can configure 4 DMA channels
config dmach0=enabled ,burstlen=8,chanrpt=enabled, tci=off,eil=
off, sar=none,sam=inc,dar=none,dam=inc ,trigger=0,btc=100 ,repeat
=1,sadr=varptr(ar(1)),dadr=varptr(dest(1))
start dmach0 ' this will do a manual/software DMA transfer, when
trigger<>0 you can use a hardware event as a trigger source
for j=1 to 50
  print j;"-";ar(j);"-";dest(j) ' print the values
next
end
```

6.125 CONFIG DMXSLAVE

Action

Configures the DMX-512 slave.

Syntax

CONFIG DMXSLAVE = com, Channels=nchannels, Start = nstart, Store=nstore

Remarks

com	The UART you want to use for the communication with the DMX-512 bus. This depends on the micro processor. In most cases this is COM1.	
Channels	A numeric constant that defines the maximum number of channels you can receive. When you like to process all DMX data, you need to use 512 since 512 is the maximum. When you make a simple device a number of 8 would be sufficient.	
Start	The slave starting address. This is 1 by default. You will receive data starting at address 'Start'.	
Store	The number of bytes you will receive and store.	

You must chose the crystal/oscillator speed in a way that 250000 baud will give no errors. Typical 4, 8 and 16 MHz will work fine.

When you want to be sure, check the compiler report. It should have 0% error.

Since the DMX slave is running in interrupt mode on the background, you must ENABLE interrupts.

The serial interrupts used, is enabled by the CONFIG DMXSLAVE command.

So how does this work? When you configure the DMXSLAVE, it will receive data in interrupt mode. It will store the data into a byte arrays named _DMX_RECEIVED The first byte stored into this array is the value for address 'STAR' : the address you defined with START.

The number of bytes stored in the array depends on the 'STORE' setting.

Example : Config Dmxslave = Com1 , Channels = 16 , Start = 3 , Store = 1 This will setup an array _DMX_RECEIVED that can hold 16 bytes. So the maximum value for STORE would be 16 too. In the example our address is 3, and we store only

address 3.

We can dynamic change the START address and the number of bytes to get ! For this purpose you can change the automatic generated internal variables _DMX_ADDRESS and _DMX_CHANNELS_TOGET

_DMX_ADDRESS defines the starting address. And _DMX_CHANNELS_TOGET defines the number of bytes to store after the address matches.

See also

NONE

Example

dmx-receive.bas (c) 1995-2009 MCS Electronics ' this sample demonstates receiving a DMX datastream in the background 'we use a chip with 2 UARTS so we can print some data \$regfile = "m162def.dat" you need to use a crystal that can generate a good 250 KHz baud 'For example 8 Mhz, 16 or 20 Mhz **\$crystal** = 8000000 'define the stack \$hwstack = 40\$swstack = 32\$framesize = 32 'these are the pins we use. COM1/UART1 is used for the DMX data RX ΤX ' COM1 PD.1 PD.0 DMX PB.3 PB.2 ' COM2 RS-232 **Config** Dmxslave = Com1 , Channels = 16 , **Start** = 3 , Store = 1 'this will set up the code. an array named _dmx_channels will contain the data 'the channels will define the size. So when you want to receive data for 8 channels, you set it to 8. 'the maximum size is 512 for retrieving all data 'START defines the starting address. By default it is 1. Thus the array will be filled starting at address 3 in the example 'STORE defines how many bytes you want to store 'By default, 1 channel is read. But you can alter the variable _dmx_channelels_toget to specify how many bytes you want to receive 'So essential you need to chose how many bytes you like to receive. Most slaves only need 1 - 3 bytes. It would be a waste of space to define more channels then, 'Then you set the slave address with the variable : _dmx_address , which is also set by the optional [START] 'And finally you chose how many bytes you want to receive that start at the specified address. You do this by setting the dmx channels toget variable. 'Example : Config Dmxslave = Com1 , Channels = 16 , Start = 300 , Store = 4 this would store the bytes from address 300 - 303. the maximum would be 315 since channels is set to 16 Config Dmxslave = Com1 , Channels = 8 , Start = 1 , Store = 8 this would store the bytes from address 1 - 8. the maximum would be 8 since channels is set to 8 Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 ,

```
490 BASCOM-AVR
```

```
Databits = 8 , Clockpol = 0
Open "COM2:" For Binary As #1
Print #1 , "MCS DMX-512 test"
'since DMX data is received in an ISR routine, you must enable the
global interrupts
Enable Interrupts
Dim J As Byte
Do
    If Inkey(#1) = 32 Then
                                                                 ' when you
press the space bar
      For J = 1 To _dmx_channels
                                                                 ' show the
data we received
         Print #1 , _dmx_received(j) ; " " ;
      Next
      Print #1,
   Elseif Inkey(\#1) = 27 Then
                                                                 'you ca
dynamic change the start address and the channels
     Input #1 , "start " , _dmx_address
Input #1 , "channels " , _dmx_channels_toget
   End If
Loop
'typical you would read a DIP switch and use the value as the address
```

End

6.126 CONFIG DP

Action

This option sets the character used for the decimal point for singles and fusing.

Syntax

CONFIG DP= "dp"

Remarks

The decimal point is a dot (.) by default. The STR() and FUSING functions convert a single into a string. The fraction is separated by a dot. In a number of counties the comma is used as a separator. Valid options are : CONFIG DP = "." and CONFIG DP = ","

This options only sets the character for str() and fusing for singles. In your code you still need to code with a dot : var = 1234.333

See also

Example CONFIG DP="," Dim s as single S=1234.56 print s

6.127 CONFIG EEPROM

Action

Setup memory mode for EEPROM in XMEGA.

Syntax CONFIG EEPROM=mode

Remarks

In Xmega, the EEPROM can be mapped so it can be used with pointer operations such as LD,ST, LDS and STS.

When EEPROM is mapped, it will start at &H1000. The advantage of mapping the EEPROM is that reading the EEPROM becomes much more simpler.

In release 1.11.9.6, only MAPPED is supported. When you use the BASCOM EEPROM routines, you must include this statement before you use the EEPROM.

To maintain compatibility with code and other AVR chips you can still use address 0 for the EEPROM. The library will add an offset of &H1000 to the address.

See also

NONE

Example

Config Eeprom = Mapped

6.128 CONFIG ERROR

Action

Instructs the compiler to ignore one or more errors.

Syntax

CONFIG ERROR=ignore, err=ignore [err1=ignore]

Remarks

In some situations you might want to ignore an error. For example if a new version adds a certain check that was not available in a previous version you will get errors. If you ignore the error, the code will compile without errors. This will not work in any situation. Some errors can not be ignored. You should never use this option for a finished product.

See also NONE

Example

Config Error = Ignore , 369 = Ignore

Lbl:

```
Dim Lbl As Word ' this would generate an error 369 without the ignore !!!
```

6.129 CONFIG EVENT_SYSTEM

Action

This statement configures the Xmega event routing.

Syntax

CONFIG EVENT_SYSTEM = dummy, MUX**x**=MUX, QD**x**=QD, QDI**x**=QDI, QDIRM**x** =QDIRM,DIGFLT**x**=DIGFLT

The letter X is used to indicate that a value between 0 and 7 can be used. So there is MUX0, MUX1, MUX2,MUX3 etc.

Remarks

The Event System is a set of features for inter peripheral communication. It enables the possibility

for a change of state in one peripheral to automatically trigger actions in other peripherals.

The change of state in a peripheral that will trigger actions in other peripherals is configurable in

software. It is a simple, but powerful system as it allows for autonomous control of peripherals

without any use of interrupt, CPU or DMA resources.

There are 8 multiplexers and 8 control registers. Register 0, 2 and 4 can be used for quadrature decoding.

MUX	There are 8 multiplexers, named MUX0-MUX7. The MUX is used to select
	an event source. There are many sources for events :
	NONE : disabled, default
	RTC_OVF : Real Timer overflow
	RTC_CMP : Real Timer compare match
	ACA_CH0 : analog comparator ACA, channel 0
	ACA_CH1 : analog comparator ACA, channel 1
	ACA_WIN : analog comparator ACA, window
	ACB_CH0 : analog comparator ACB, channel 0
	ACB_CH1 : analog comparator ACB, channel 1
	ACB_WIN : analog comparator ACB, window
	ADCA_CH0- ADCA_CH3 : ADCA channel 0-3
	ADCB_CH0- ADCB_CH3 : ADCB channel 0-3
	PORTA.0 - PORTA.7 : PORT A pin 0-7
	PORTB.0 - PORTB.7 : PORT B pin 0-7
	PORTC.0 - PORTC.7 : PORT C pin 0-7
	PORTD.0 - PORTD.7 : PORT D pin 0-7

	PORTE.0 - PORTE.7 : PORT E pin 0-7 PORTF.0 - PORTF.7 : PORT F pin 0-7 PRESCALER1, PRESCALER2, PRESCALER4, PRESCALER8, PRESCALER16, PRESCALER32, PRESCALER64,PRESCALER128,PRESCALER256, PRESCALER512,PRESCALER1024,PRESCALER2048,PRESCALER4096, PRESCALER8192,PRESCALER16384 : The clock divided by 1,2,4,8,16,32,64,128,256 etc. TCC0_OVF : Timer TC0 overflow TCC0_ERR : Timer TC0 error TCC0_CCA : Timer TC0 capture or compare match A TCC0_CCB : Timer TC0 capture or compare match B TCC0_CCC : Timer TC0 capture or compare match C TCC0_CCC : Timer TC0 capture or compare match D TCC1_OVF : Timer TC1 overflow TCC1_ERR : Timer TC1 error TCC1_CCA : Timer TC1 capture or compare match A TCC1_CCB : Timer TC1 capture or compare match A
	TCC1_CCD : Timer TC1 capture or compare match D Dito for TCD0, TCD1, TCE0, TCE1, TCF0 and TCF1
QD	Enables or disables the quadrature decoder. Will only work on QD0,QD2 and QD4.
QDI	Enables or disables the quadrature decode index. Will only work on QDI0, QDI2 and QDI4.
QDIRM	Quadrature decode index recognition mode. This is a numeric constant between 0 and 3. Each value represents the 2 possible bit values for the two input signals. Will only work on QDIRM0, QDIRM2 and QDIRM4.
DIGFLT	Defines the length of digital filtering used. Events will be passed through to the event channel only when the event source has been active and sampled with the same level for a number of peripheral clock for the number of cycles as defined by DIGFLT. The number of samples is in the range from 1-8. The default is 1 sample.

See also

NONE

Example

6.130 CONFIG EXTENDED_PORT

Action

Configures compiler to generate warning or error when transforming extended port register.

Syntax

CONFIG EXTENDED_PORT = WARNING|ERROR

Remarks

A lot of AVR chips have so called extended registers. When the AVR was designed the designers did not set aside enough space for the hardware registers. A number of instructions work only with the lower 32 addresses, and a number only work on registers with an address till &H3F.

SRAM memory was moved up and the space after &H5F was used for registers. These are extended registers.

For these chips, the SRAM starts at &H100 or higher.

Because INP, OUT, SBI, SBI, SBIC, SBIS, etc. will not work on these extended registers, the compiler changes this automatic when needed. When INP or OUT is used, this is not a problem. LDS or STS can be used with the same register. But an instruction like SBIC that will test a pin , needs a temporarily register. Register R23 is used for this.

When you write your own ASM you might want to get a warning or an error. For this purpose you can use CONFIG EXTENDED_PORT.

When you use WARNING there will be a warning in the report file. When you use ERROR, you will get an error and your code will not compile.

See also

6.131 CONFIG HITAG

Action

Configures the timer and HITAG variables.

Syntax

CONFIG HITAG = prescale, TYPE=tp, DOUT = dout, DIN=din , CLOCK=clock, INT=int **CONFIG HITAG** = prescale, TYPE=tp, DEMOD= demod, INT=@int

Remarks

	syntax for HTRC110	
prescale	The pre scaler value that is used by TIMER0. A value of 8 and 256 will work at 8 MHz.	
tp	The kind of RFID chip you use. Use HTRC110.	
DOUT	The pin that is connected to the DOUT pin of the HTRC110. This pin is used in input mode since DOUT is an output. A pin that support the pin- change interrupt or the PCINT should be selected.	
DIN	The pin that is connected to the DIN pin of the HTRC110. This pin is used in output mode. You can chose any pin that can be used in output mode.	
CLOCK	The pin that is connected tot the CLOCK pin of the HTRC110. This pin is used in output mode. You can chose any pin that can be used in output mode.	
INT	The interrupt used. Note that you need to precede the interrupt with an @ sign. For example for INT1 you provide : @INT1	

syntax for EM4095

prescale	The pre scaler value that is used by TIMER0. A value of 8 and 256 will work at 8 MHz.
tp	The kind of RFID chip you use. Use EM4095.
demod	The pin that is connected to the DEMOD pin of the EM4095. This pin is used in input mode. A pin that support the pin-change interrupt or the PCINT should be selected.
INT	The interrupt used. Note that you need to precede the interrupt with an @ sign. For example for INT1 you provide : @INT1

The CONFIG HITAGE command will generate a number of internal used variables and constants.

Constants : _TAG_MIN_SHORT, _TAG_MAX_SHORT , _TAG_MIN_LONG and _TAG_MAX_LONG.

See the description of READHITAG to see how they are calculated. The actual value will depend on the prescale value you use.

Variables for HTRC110 :

_htr_statemachine , a byte that is used to maintain a state machine.

_htcbit , a byte that will hold the received bit.

_htcbitcount , a byte to store the number of received bits.

_htcmpulse , a byte that stores the pulse

_htr_pulse_state , a byte that is used to maintain the pulse state machine.

_htc_retries, a byte that is used for the number of retries.

_tagdelta , a byte that will held the delta time between 2 edges.

_tagtime , a byte with the actual timer0 value when an edge is detected.

_taglasttime , a byte with the previous edge time, needed to calculate the delta time. _tagparbit , a byte that will held the parity.

_tagdata , a byte where the bits are stored before they are loaded into the serial number array.

_tagid , a word that points to the serial number array

The HTRC110.LBX contains a number of other constants that are used to control the HTRC chip.

The _init_Tag routine is called automatically.

The clock output of the Mega88 is used to drive the HTRC110. Since the clock output of the internal oscillator is 8 MHz, the HTRC110 is also configured to work at 8 MHz. The .equ for Tag_set_config_page3 = &H40 + 48 + Fsel0 in the LBX. You can set it to 12 and 16 MHz too but you can not drive it from the clock output then.

Variables for EM4095 :

_tagflag , a byte that stores the return flag that will be loaded with 1 when a valid tag is detected

_tag_insync ,a byte that is used to store the state of the bit stream.

_tag_bitcount , a byte that stores the total bits when not in sync yet

_tag_tbit , a byte that stores the total received bits

_tag_par , a byte that stores the parity

_tag_timeout ,a byte that is loaded with the time that will be tried to detect an RFID chip

_taglasttime , a byte that stores the last time a valid edge was detected

_tagid , a word that points to the serial number array

See also

READHITAG 832

Example HTRC110

```
_____
                     (c) 1995-2008 , MCS Electronics
' sample : readhitag.bas
' demonstrates usage of the READHITAG() function
$regfile = "m88def.dat"
                                                           ' specify chip
$crystal = 8000000
                                                            ' used speed
$baud = 19200
                                                           'baud rate
'Notice that the CLOCK OUTPUT of the micro is connected to the clock input of the H
'PORTB.0 of the Mega88 can optional output the clock. You need to set the fusebit i
'This way all parts use the Mega88 internal oscillator
'The code is based on Philips(NXP) datasheets and code. We have signed an NDA to ge
'You can find more info on Philips website if you want their code
Print "HTC110 demo"
Config Hitag = 64 , Type = Htrc110 , Dout = Pind.2 , Din = Pind.3 , Clock = Pind.4
                ^ use timer0 and select prescale value 64
                      * we used htrc110 chip
                                      ^-- dout of HTRC110 is connected to PIND.2 w
                                                     ^ DIN of HTRC100 is connected
                                                                    ^clock of HTR
'the config statement will generate a number of constante and internal variables us
'the htrc110.1bx library is called
Dim Tags(5) As Byte
                                                           'each tag has 5 byte se
Dim J As Byte
                                                            ' a loop counter
'you need to use a pin that can detect a pin level change
'most INT pins have this option
'OR , you can use the PCINT interrupt that is available on some chips
'In case you want PCINT option
' Pcmsk2 = &B0000_0100 'set the mask to ONLY use the pin connected to DOUT
 On Pcint2 Checkints
                             'label to be called
' Enable Pcint2
                             'enable this interrupt
'In case you want to use INT option
On Int0 Checkints
                                                            ' PIND.2 is INTO
Config Int0 = Change
                                                           'you must configure the
Enable Interrupts
                                                           ' enable global interru
Do
                                                           'check if there is a ne
If Readhitag(tags(1)) = 1 Then
    For J = 1 To 5
                                                           'print the 5 bytes
        Print Hex(tags(j)) ; ",";
    Next
 Else
                                                           'there was nothing
   Print "Nothing"
 End If
 Waitms 500
                                                           'some delay
Loop
```

'this routine is called by the interrupt routine

```
Checkints:

Call _checkhitag 'you must call this lak

'you can do other things here but keep time to a minimum

Return
```

Example EM4095

```
_____
                  (c) 1995-2008 MCS Electronics
  This sample will read a HITAG chip based on the EM4095 chip
  Consult EM4102 and EM4095 datasheets for more info
!_____
                                                            _____
  The EM4095 was implemented after an idea of Gerhard Günzel
  Gerhard provided the hardware and did research at the coil and capacitors.
  The EM4095 is much simpler to use than the HTRC110. It need less pins.
  A reference design with all parts is available from MCS
$regfile = "M88def.dat"
$baud = 19200
$crystal = 8000000
\$hwstack = 40
\$swstack = 40
$framesize = 40
'Make SHD and MOD low
                                                         'make sure the array is
Dim Tags(5) As Byte
Dim J As Byte
Config Hitag = 64 , Type = Em4095 , Demod = Pind.3 , Int = @int1
Print "Test EM4095"
'you could use the PCINT option too, but you must mask all pins out so it will only
' Pcmsk2 = &B0000 0100
' On Pcint2 Checkints
' Enable Pcint2
On Intl Checkints Nosave
                                                         'we use the INT1 pin a
                                                         'we have to config so t
Config Int1 = Change
Enable Interrupts
                                                         'as last we have to ena
Do
  Print "Check..."
                                                         'this will enable INT1
  If Readhitag(tags(1)) = 1 Then
     For J = 1 To 5
        Print Hex(tags(j)) ; ",";
     Next
     Print
 Else
    Print "Nothing"
 End If
 Waitms 500
Loop
Checkints:
Call _checkhitag
                                                         'in case you have used
Return
```

6.132 CONFIG I2CDELAY

Action

498

Compiler directive that overrides the internal I2C delay routine.

Syntax

CONFIG I2CDELAY = value

Remarks

value	A numeric value in the range from 1 to 255.	
	A higher value means a slower I2C clock.	

For the I2C routines the clock rate is calculated depending on the used crystal. In order to make it work for all I2C devices the slow mode is used. When you have faster I2C devices you can specify a low value.

By default a value of 5 is used. This will give a 200 kHZ clock. When you specify 10, 10 uS will be used resulting in a 100 KHz clock.

When you use a very low crystal frequency, it is not possible to work with high clock frequencies.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files. For chips that have hardware TWI, you can use the MasterTWI lib.

See also

CONFIG SCL 539 , CONFIG SDA 538

Example

```
_____
_____
'name
                     : i2c.bas
'name
'copyright
                     : (c) 1995-2005, MCS Electronics
'purpose
                      : demo: I2CSEND and I2CRECEIVE
'micro
                      : Mega48
'micro : Meg.
'suited for demo : yes
'commercial addon needed : no
·_____.
_____
$regfile = "m48def.dat"
                                                   ' specify
the used micro
$crystal = 4000000
                                                   ' used
crystal frequency
$baud = 19200
                                                   ' use baud
rate
$hwstack = 32
                                                   ' default
use 32 for the hardware stack
$swstack = 10
                                                   ' default
use 10 for the SW stack
$framesize = 40
                                                   ' default
use 40 for the frame space
```

Config Scl = Portb.4 **Config** Sda = Portb.5 Declare Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte) Declare Sub Read_eeprom(byval Adres As Byte , Value As Byte) **Const** Addressw = 174 'slave write address **Const** Addressr = 175 'slave read address Dim B1 As Byte , Adres As Byte , Value As Byte 'dim byte **Call** Write_eeprom(1, 3) 'write value of three to address 1 of EEPROM 'read it Call Read_eeprom(1 , Value) : Print Value back **Call** Read eeprom(5 , Value) : **Print** Value 'again for address 5 '----- now write to a PCF8474 I/O expander ------**12csend** &H40 , 255 'all outputs hiah I2creceive &H40 , B1 'retrieve input 'print it Print "Received data " ; B1 End Rem Note That The Slaveaddress Is Adjusted Automaticly With I2csend & I2creceive Rem This Means You Can Specify The Baseaddress Of The Chip. 'sample of writing a byte to EEPROM AT2404 Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte) I2cstart 'start condition I2cwbyte Addressw 'slave address 'asdress of 12cwbyte Adres EEPROM 12cwbyte Value 'value to write I2cstop 'stop condition Waitms 10 'wait for 10 milliseconds End Sub 'sample of reading a byte from EEPROM AT2404 Sub Read_eeprom(byval Adres As Byte , Value As Byte) I2cstart 'generate start 'slave 12cwbyte Addressw adsress 'address of 12cwbyte Adres EEPROM

I2cstart start	'repeated
I2cwbyte Addressr	'slave
address (read) I2crbyte Value , Nack I2cstop stop End Sub	'read byte 'generate
<pre>' when you want to control a chip with a larger memory like requires an additional byte ' to be sent (consult the datasheet): ' Wires from the I2C address that are not connected will ded most cases!</pre>	
' I2cstart	'start
condition ' I2cwbyte &B1010_0000 address	'slave
' I2cwbyte H	'high
address ' I2cwbyte L ' I2cwbyte Value	'low address 'value to
write ' I2cstop condition ' Waitms 10	'stop

6.133 CONFIG I2CSLAVE

Action

Configures the I2C slave mode.

Syntax

CONFIG I2CSLAVE = address , INT = interrupt , TIMER = tmr

Remarks

ir	
Address	The slave address you want to assign to the I2C slave chip. This is an address that must be even like 60. So 61 cannot be used.
	I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C.
	This means that an I2C address of 1 becomes $\&B10 = 2$. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases.
Interrupt	The interrupt that must be used. This is INT0 by default.
Tmr	The timer that must be used. This is TIMER0 by default.

While the interrupt can be specified, you need to change the library code when you use a non-default interrupt. For example when you like to use INT1 instead of the default INT0.

The same applies to the TIMER. You need to change the library when you like to use another timer.

See Also

CONFIG TWI 569

Example

```
_____
                         : i2c_pcf8574.bas
'name
'copyright
                         : (c) 1995-2005, MCS Electronics
'purpose
                         : shows how you could use the I2C slave
library to create a PCF8574
                         : AT90S2313
'micro
'micro : AT90S2313
'suited for demo : NO, ADDON NEEDED
'commercial addon needed : yes
!_____
_____
$regfile = "2313def.dat"
                                                            ' specify
the used micro
$crystal = 3684000
                                                            ' used
crystal frequency
$baud = 19200
                                                            ' use baud
rate
$hwstack = 32
                                                            ' default
use 32 for the hardware stack
\$swstack = 10
                                                            ' default
use 10 for the SW stack
Sframesize = 40
                                                            ' default
use 40 for the frame space
'This program shows how you could use the I2C slave library to create a
PCF8574
'The PCF8574 is an IO extender chip that has 8 pins.
'The pins can be set to a logic level by writing the address followed by
a value
'In order to read from the pins you need to make them '1' first
'This program uses a AT90S2313, PORTB is used as the PCF8574 PORT
'The slave library needs INTO and TIMERO in order to work.
'SCL is PORTD.4 (T0)
'SDA is PORTD.2 (INT0)
'Use 10K pull up resistors for both SCL and SDA
'The Slave library will only work for chips that have T0 and INT0
connected to the same PORT.
'These chips are : 2313,2323, 2333,2343,4433,tiny22, tiny12,tiny15, M8
'The other chips have build in hardware I2C(slave) support.
'specify the slave address. This is &H40 for the PCF8574
'You always need to specify the address used for write. In this case
&H40 ,
'The config i2cslave command will enable the global interrupt enable
flag !
Config I2cslave = &B0100000
                                                           ' same as
&H40
'Config I2cslave = &H40 , Int = Int0 , Timer = Timer0
'A byte named _i2c_slave_address_received is generated by the compiler.
'This byte will hold the received address.
'A byte named _i2c_slave_address is generated by the compiler.
'This byte must be assigned with the slave address of your choice
```

502 BASCOM-AVR

```
'the following constants will be created that are used by the slave
library:
 _i2c_pinmask = &H14
 _i2c_slave_port = Portd
  i2c slave pin = Pind
  i2c slave ddr = Ddrd
  _i2c_slave_scl = 4
 i2c slave sda = 2
'These values are adjusted automatic depending on the selected chip.
'You do not need to worry about it, only provided as additional info
'by default the PCF8574 port is set to input
Config Portb = Input
Portb = 255
                                                        'all pins
high by default
'DIM a byte that is not needed but shows how you can store/write the I2C
DATA
Dim Bfake As Byte
'empty loop
   ' you could put your other program code here
  'In any case, do not use END since it will disable interrupts
Loop
'here you can write your other program code
'But do not forget, do not use END. Use STOP when needed
1111111
            The following labels are called from the slave library
1111111
'When the master wants to read a byte, the following label is allways
called
'You must put the data you want to send to the master in variable _a1
which is register R16
I2c_master_needs_data:
 'when your code is short, you need to put in a waitms statement
 'Take in mind that during this routine, a wait state is active and the
master will wait
  'After the return, the waitstate is ended
 Config Portb = Input
                                                        ' make it an
input
 _al = Pinb
                                                        ' Get input
from portB and assign it
Return
'When the master writes a byte, the following label is always called
'It is your task to retrieve variable _A1 and do something with it
'_Al is register R16 that could be destroyed/altered by BASIC statements
'For that reason it is important that you first save this variable
```

I2c_master_has_data:
 'when your code is short, you need to put in a waitms statement
 'Take in mind that during this routine, a wait state is active and the
master will wait

```
'After the return, the waitstate is ended
                                                       ' this is
 Bfake = _a1
not needed but it shows how you can store _A1 in a byte
 'after you have stored the received data into bFake, you can alter R16
 Config Portb = Output
                                                       ' make it an
output since it could be an input
                                                       'assign Al
 Portb = _a1
(R16)
Return
1111111
'You could simply extend this sample so it will use 3 pins of PORT D for
the address selection
'For example portD.1 , portd.2 and portD.3 could be used for the address
selection
'Then after the CONFIG I2CSLAVE = &H40 statement, you can put code like:
'Dim switches as Byte ' dim byte
                      ' get dip switch value
'switches = PIND
'switches = switches and &H1110 ' we only need the lower nibble without
the LS bit
'_i2c_slave_address = &H40 + switches ' set the proper address
```

6.134 CONFIG INPUT

Action

Instruct the compiler to modify serial input line terminator behaviour

Syntax

CONFIG INPUT = term , ECHO=echo

Syntax Xmega

CONFIG INPUT1|INPUT2|INPUT3|INPUT4|INPUT5|INPUT6|INPUT7|INPUT8 = term , ECHO=echo

Remarks

Term	A parameter with one of the following values : CR - Carriage Return (default) LF - Line Feed CRLF - Carriage Return followed by a Line Feed LFCR - Line Feed followed by a Carriage Return
Echo	A parameter with one of the following values : CR - Carriage Return LF - Line Feed CRLF - Carriage Return followed by a Line Feed (default) LFCR - Line Feed followed by a Carriage Return

The 'term' parameter specifies which character(s) are expected to terminate the INPUT statement with serial communication. It has no impact on the DOS file system INPUT.

In most cases, when you press <ENTER> , a carriage return(ASCII 13) will be sent. In some cases, a line feed (LF) will also be sent after the CR. It depends on the terminal emulator or serial communication OCX control you use. The 'echo' parameter specifies which character(s) are send back to the terminal emulator after the INPUT terminator is received. By default CR and LF is sent. But you can specify which characters are sent. This can be different characters then the 'term' characters. So when you send in your VB application a string, and end it with a CR, you can send back a LF only when you want.

When NOECHO is used, **NO** characters are sent back even while configured with CONFIG INPUT

For the XMega you can specify for each UART how it should handle input and echo.

See also

INPUT 744

ASM

NONE

Example

Config Input1 = CR , Echo = CRLF Dim S as String * 20 Input "Hello ",s

6.135 CONFIG INTx

Action

Configures the way the interrupts 0,1 and 4-7 will be triggered.

Syntax

CONFIG INTx = state Where X can be 0,1 and 4 to 7 in the MEGA chips.

Remarks

LOW LEVEL to generate an interrupt while the pin is held low. Holding
the pin low will generate an interrupt over and over again.
FALLING to generate an interrupt on the falling edge.
RISING to generate an interrupt on the rising edge.
CHANGE to generate an interrupt on the change of the edge. Not all
microprocessors support CHANGE.

The MEGA103 has also INTO-INT3. These are always low level triggered so there is no need /possibility for configuration.

The number of interrupt pins depend on the used chip. Most chips only have int0 and int1.

Example

'name 'copyright

- : spi-softslave.bas
- : (c) 1995-2005, MCS Electronics

BASCOM Language Reference

505

: shows how to implement a SPI SLAVE with 'purpose software 'micro : AT90S2313 'suited for demo : yes 'commercial addon needed : no !_____ _____ _____ \$regfile = "2313def.dat" ' specify the used micro Scrystal = 4000000' used crystal frequency ' use baud **\$baud** = 19200 rate ' default \$hwstack = 32use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space 'Some atmel chips like the 2313 do not have a SPI port. 'The BASCOM SPI routines are all master mode routines 'This example show how to create a slave using the 2313 'ISP slave code 'define the constants used by the SPI slave Const _softslavespi_port = Portd ' we used portD **Const** _softslavespi_pin = Pind 'we use the PIND register for reading **Const** _softslavespi_ddr = Ddrd ' data direction of port D **Const** _softslavespi_clock = 5 'pD.5 is used for the CLOCK **Const** _softslavespi_miso = 3 'pD.3 is MISO **Const** _softslavespi_mosi = 4 'pd.4 is MOST ' pd.2 is SS Const _softslavespi_ss = 2 'while you may choose all pins you must use the INTO pin for the SS 'for the 2313 this is pin 2 'PD.3(7), MISO must be output 'PD.4(8), MOSI 'Pd.5(9) , Clock 'PD.2(6), SS /INT0 'define the spi slave lib \$lib "spislave.lbx" 'sepcify wich routine to use **\$external** _spisoftslave 'we use the int0 interrupt to detect that our slave is addressed On IntO Isr_sspi Nosave 'we enable the int0 interrupt Enable Int0 'we configure the INTO interrupt to trigger when a falling edge is detected **Config Int0 = Falling** 'finally we enabled interrupts Enable Interrupts

```
÷.
Dim _ssspdr As Byte
                                                                ' this is
out SPI SLAVE SPDR register
                                                                ' SPI
Dim _ssspif As Bit
interrupt revceive bit
Dim Bsend As Byte , I As Byte , B As Byte
                                                                ' some other
demo variables
_ssspdr = 0
0 the first time the master sends data
                                                                ' we send a
Do
   If ssspif = 1 Then
   Print "received: " ; _ssspdr
  Reset _ssspif
   _ssspdr = _ssspdr + 1
                                                                ' we send
this the next time
  End If
Loop
```

6.136 CONFIG GRAPHLCD

Action

Configures the Graphical LCD display.

Syntax

Config GRAPHLCD = type , DATAPORT = port, CONTROLPORT=port , CE = pin , CD = pin , WR = pin, RD=pin, RESET= pin, FS=pin, MODE = mode

Remarks

Туре	This must be 240X64, 128X128, 128X64 , 160X48 , 240X128, 192X64 , SED180X32 or 192X64SED.
	For SED displays use 128X64sed or 120X64SED or SED180X32 For 132x132 color displays, use COLOR For EADOG128x64 use 128X64EADOGM For SSD1325 96x64 use 96X64SSD1325. See <u>SSD1325lib</u> 974).
	The name of the port that is used to put the data on the LCD data pins db0-db7. PORTA for example.
	This is the name of the port that is used to control the LCD control pins. PORTC for example
Ce	The pin number that is used to enable the chip on the LCD.
Cd	The pin number that is used to control the CD pin of the display.
WR	The pin number that is used to control the /WR pin of the display.
RD	The pin number that is used to control the /RD pin of the display.
FS	The pin number that is used to control the FS pin of the display. Not needed for SED based displays.
RESET	The pin number that is used to control the RESET pin of the display.
MODE	The number of columns for use as text display. Use 8 for X-pixels / 8 = 30 columns for a 240 pixel screen. When you specify 6, 240 / 6 = 40 columns can be used.

507

	EADOG128M pins for SPI mode. This display only can write data. As a result, a number of grapical commands are not supported.
CS1	Chip select for EADOG128x64
A0	A0 line for EADOG128x64. This is the line that controls data/command
SI	This is the serial input pin for the EADOG128x64.
SCLK	This is the clock pin for the EADOG128x64.
	ST7565R parallel data mode A 128x64 graphical display which supports all graphic commands
dataport	The data port connected to the display. For example portJ
CS1	the chip enabled line
A0	the chip data/command mode pin
RST	the reset pin of the chip
WR	The /WR line of the chip
RD	The /RD line of the chip
C86	This pin selects the transfer mode.
PM	Some displays have this PM pin which sets the parallel mode
example	Config Graphlcd = 128 * 64eadogm ,dataport=portj, Cs1 = Porth.0 , A0 = Porth.2 , rst= Porth.1 , wr = Porth.3 , Rd = Porth.4,c86=porth.6

The first chip supported was T6963C. There are also driver for other LCD's such as SED and KS0108. The most popular LCD's will be supported with a custom driver.

The following connections were used for the T6963C:

PORTA.0 to PORTA.7 to DB0-DB7 of the LCD PORTC.5 to FS, font select of LCD PORTC.2 to CE, chip enable of LCD PORTC.3 to CD, code/data select of LCD PORTC.0 to WR of LCD, write PORTC.1 to RD of LCD, read PORTC.4 to RESET of LCD, reset LCD

The LCD used from www.conrad.de needs a negative voltage for the contrast.

Two 9V batteries were used with a pot meter. Some displays have a Vout that can be used for the contrast(Vo)

The T6963C displays have both a graphical area and a text area. They can be used together. The routines use the XOR mode to display both text and graphics layered over each other.

The statements that can be used with the graphical LCD are :

CLS [433], will clear the graphic display and the text display CLS GRAPH will clear only the graphic part of the display CLS TEXT will only clear the text part of the display

 $\underline{\text{LOCATE}}_{771}$ row, column : Will place the cursor at the specified row and column The row may vary from 1 to 16 and the column from 1 to 40. This depends on the size and mode of the display.

CURSOR [ab] ON/OFF BLINK/NOBLINK can be used the same way as for text displays.

LCD [752] : can be handled the same way as for text displays.

<u>SHOWPIC</u> (883) X, Y, Label : Show image where X and Y are the column and row and Label is the label where the picture info is placed.

PSET [813] X, Y, color : Will set or reset a pixel. X can range from 0-239 and Y from 9-63. When color is 0 the pixel will turned off. When it is 1 the pixel will be set on.

<u>\$BGF</u>[312] "file.bgf" : inserts a BGF file at the current location

LINE [759] (x0,y0) – (x1,y1), color : Will draw a line from the coordinate x0,y0 to x1,y1. Color must be 0 to clear the line and 255 for a black line.

BOX 417 (x0,y0)-(x1,y1), color : Will draw a box from x0,y0 to x1,y1. Color must be 0 to clear the box and 255 for a black line.

<u>BOXFILL</u> 419 (x0,y0)-(x1,y1), color : Will draw a filled box from x0,y0 to x1,y1. Color must be 0 or 255.

The Graphic routines are located in the glib.lib or glib.lbx files.

You can hard wire the FS and RESET and change the code from the glib.lib file so these pins can be used for other tasks.

COLOR LCD

Color displays were always relatively expensive. The mobile phone market changed that. And Display3000.com , sorted out how to connect these small nice colorful displays.

You can buy brand new Color displays from Display3000. MCS Electronics offers the same displays.

There are two different chip sets used. One chipset is from EPSON and the other from Philips. For this reason there are two different libraries. When you select the wrong one it will not work, but you will not damage anything.

LCD-EPSON.LBX need to be used with the EPSON chipset.

LCD-PCF8833.LBX need to be used with the Philihps chipset.

eening erapi	
Controlport	The port that is used to control the pins. PORTA, PORTB, etc.
	The chip select pin of the display screen. Specify the pin number. 1 will mean PORTC.1
RS	The RESET pin of the display
SCL	The clock pin of the display
SDA	The data pin of the display

Config Graphled = Color , Controlport = Porte , Cs = 1 , Rs = 0 , Sel = 3 , Sda = 2

As the color display does not have a built in font, you need to generate the fonts yourself.

You can use the <u>Fonteditor</u> [148] for this task.

A number of statements accept a color parameter. See the samples below in **bold**.

LINE	Line(0 , 0) -(130 , 130) , Blue
LCDAT	Lcdat 100 , 0 , "12345678" , Blue , Yellow
CIRCLE	Circle(30, 30), 10, Blue
PSET	32 , 110 , Black
BOX	Box(10 , 30) -(60 , 100) , Red

See also

SHOWPIC [883], PSET [813], \$BGF [312], LINE [759], LCD [332], BOX [417], BOXFILL [419]

```
_____
                       : t6963 240 128.bas
'name
'copyright
                        : (c) 1995-2005, MCS Electronics
'purpose
                        : T6963C graphic display support demo 240 *
128
'micro
                        : Mega8535
'suited for demo
                        : yes
'commercial addon needed : no
                                      ......
_____
$regfile = "m8535.dat"
                                                        ' specify
the used micro
$crystal = 8000000
                                                        ' used
crystal frequency
$baud = 19200
                                                        ' use baud
rate
$hwstack = 32
                                                        ' default
use 32 for the hardware stack
\$swstack = 10
                                                        ' default
use 10 for the SW stack
Sframesize = 40
                                                        ' default
use 40 for the frame space
1_____
                    (c) 2001-2008 MCS Electronics
                T6963C graphic display support demo 240 * 128
'The connections of the LCD used in this demo
'LCD pin
                       connected to
' 1
         GND
                       GND
' 2
        GND
                      GND
' 3
        +5V
                      +5V
' 4
         -9V
                       -9V potmeter
       /WR
' 5
                      PORTC.0
       /RD
'6
                       PORTC.1
'7
         /CE
                       PORTC.2
        .
C/D
' 8
                      PORTC.3
'9
        NC
                      not conneted
'10 RESET
                      PORTC.4
'11-18 D0-D7
                       PA
                      PORTC.5
'19
    FS
'20
        NC
                      not connected
'First we define that we use a graphic LCD
' Only 240*64 supported yet
Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8 'The dataport is the portname that is connected to the data lines of the
LCD
'The controlport is the portname which pins are used to control the lcd
'CE, CD etc. are the pin number of the CONTROLPORT.
' For example CE =2 because it is connected to PORTC.2
'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns
'Dim variables (y not used)
```

510 BASCOM-AVR

```
Dim X As Byte , Y As Byte
'Clear the screen will both clear text and graph display
Cls
'Other options are :
' CLS TEXT to clear only the text display
' CLS GRAPH to clear only the graphical part
Cursor Off
Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE, COLUMN LINE can be 1-8 and column 0-30
Locate 1 , 1
'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "1234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"
Wait 2
Cls Text
'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0, 0) -(239, 127), 255
                                                               ' diagonal
line
Line(0, 127) - (239, 0), 255
                                                               ' diagonal
line
Line(0, 0) - (240, 0), 255
                                                               ' horizontal
upper line
Line(0 , 127) - (239 , 127) , 255
                                                               'horizontal
lower line
Line(0, 0) - (0, 127), 255
                                                               ' vertical
left line
Line(239, 0) -(239, 127), 255
                                                               ' vertical
right line
Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
  Pset X , 20 , 255
                                                               ' set the
pixel
Next
For X = 0 To 140
  Pset X , 127 , 255
                                                               ' set the
pixel
Next
Wait 2
'circle time
'circle(X,Y), radius, color
```

511

```
'X,y is the middle of the circle,color must be 255 to show a pixel and 0
to clear a pixel
For X = 1 To 10
  Circle(20 , 20) , X , 255
                                                              ' show
circle
  Wait 1
  Circle(20 , 20) , X , 0
                                                              'remove
circle
  Wait 1
Next
Wait 2
For X = 1 To 10
 Circle(20 , 20) , X , 255
                                                               ' show
circle
  Waitms 200
Next
Wait 2
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Test:
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje
                                                              ' show 2
since we have a big display
Wait 2
Cls Text
                                                              ' clear the
text
End
```

```
'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"
'You could insert other picture data here
```

6.137 CONFIG KBD

Action

Configure the GETKBD() function and tell which port to use.

Syntax

CONFIG KBD = PORTx , DEBOUNCE = value [, DELAY = value] [,COLS=cols]

PORTx	The name of the PORT to use such as PORTB or PORTD.
DEBOUNCE	By default the debounce value is 20. A higher value might be needed. The maximum is 255.
	An optional parameter that will cause Getkbd() to wait the specified amount of time after the key is detected. This parameter might be added when you call GetKbd() repeatedly in a loop. Because of noise and static electricity, wrong values can be returned. A delay of say 100 mS, can eliminate this problem.

Remarks

COLS	This value is 4 by default. Some chips do not have port pin 7 and for	
these cases you can use COLS=3, or COLS=2.		
	This does assume that columns are connected to the high port nibble.	

The GETKBD() function can be used to read the pressed key from a matrix keypad attached to a port of the uP.

You can define the port with the CONFIG KBD statement.

In addition to the default behavior you can configure the keyboard to have 6 rows instead of 4 rows.

CONFIG KBD = PORTx , DEBOUNCE = value , rows=6, row5=pinD.6, row6=pind.7

This would specify that row5 is connected to pind.6 and row7 to pind.7 Note that you can only use rows=6. Other values will not work.

See also

GETKBD 709

```
_____
_____
'name
                     : getkbd.bas
'copyright
                     : (c) 1995-2005, MCS Electronics
'purpose
                     : demo : GETKBD
                     : Mega48
'micro
'suited for demo
                      : yes
'commercial addon needed : no
*_____
_____
$regfile = "m48def.dat"
                                                    ' specify
the used micro
$crystal = 4000000
                                                    ' used
crystal frequency
$baud = 19200
                                                    ' use baud
rate
                                                    ' default
$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
                                                    ' default
use 10 for the SW stack
$framesize = 40
                                                    ' default
use 40 for the frame space
'specify which port must be used
'all 8 pins of the port are used
Config Kbd = Portb
'dimension a variable that receives the value of the pressed key
Dim B As Byte
'loop for ever
Do
 B = Getkbd()
 'look in the help file on how to connect the matrix keyboard
 'when you simulate the getkbd() it is important that you press/click
the keyboard button
 ' before running the getkbd() line !!!
 Print B
 'when no key is pressed 16 will be returned
```

```
'use the Lookup() function to translate the value to another one
' this because the returned value does not match the number on the
keyboad
Loop
End
```

6.138 CONFIG KEYBOARD

Action

Configure the GETATKBD() function and tell which port pins to use.

Syntax

CONFIG KEYBOARD = PINX.y , DATA = PINX.y , KEYDATA = table

Remarks

KEYBOARD	The PIN that serves as the CLOCK input.
DATA	The PIN that serves as the DATA input.
	The label where the key translation can be found. The AT keyboard returns scan codes instead of normal ASCII codes. So a translation table s needed to convert the keys. BASCOM allows the use of shifted keys too. Special keys like function keys are not supported.

The AT keyboard can be connected with only 4 wires: clock,data, gnd and vcc. Some info is displayed below. This is copied from an Atmel data sheet.

The INTO or INT1 shown can be in fact any pin that can serve as an INPUT pin.

The application note from Atmel works in interrupt mode. For BASCOM we rewrote the code so that no interrupt is needed/used.

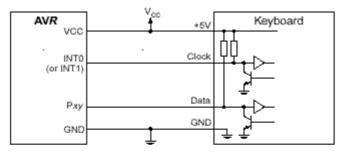


Table 1. AT Keyboard Connector Pin Assignments

AT Computer	4 1 3 3	
Signals	DIN41524, Female at Computer, 5-pin DIN 180°	6-pin Mini DIN PS2 Style Female at Computer
Clock	1	5
Data	2	1
nc	3	2,6
GND	4	3
+5V	5	4
Shield	Shell	Shell

See also

GETATKBD 703

Example

```
_____
                               _____
    _____
        _____
'name
                       : getatkbd.bas
'copyright
                       : (c) 1995-2005, MCS Electronics
'purpose
                       : PC AT-KEYBOARD Sample
'micro
                       : Mega48
'suited for demo
                        : yes
'commercial addon needed : no
! _____
  _____
$regfile = "8535def.dat"
                                                        ' specify
the used micro
$crystal = 4000000
                                                       ' used
crystal frequency
$baud = 19200
                                                        ' use baud
rate
\$hwstack = 32
                                                        ' default
use 32 for the hardware stack
$swstack = 10
                                                       ' default
use 10 for the SW stack
$framesize = 40
                                                        ' default
use 40 for the frame space
'For this example :
'connect PC AT keyboard clock to PIND.2 on the 8535
'connect PC AT keyboard data to PIND.4 on the 8535
```

© 2011 MCS Electronics

```
'The GetATKBD() function does not use an interrupt.
'But it waits until a key was pressed!
'configure the pins to use for the clock and data
'can be any pin that can serve as an input
'Keydata is the label of the key translation table
Config Keyboard = Pind.2 , Data = Pind.4 , Keydata = Keydata
'Dim some used variables
Dim S As String * 12
Dim B As Byte
'In this example we use SERIAL(COM) INPUT redirection
$serialinput = Kbdinput
'Show the program is running
Print "hello"
Do
  'The following code is remarked but show how to use the GetATKBD()
function
                       'get a byte and store it into byte variable
  ' B = Getatkbd()
  'When no real key is pressed the result is 0
  'So test if the result was > 0
  ' If B > 0 Then
  .
      Print B ; Chr(b)
  ' End If
  'The purpose of this sample was how to use a PC AT keyboard
  'The input that normally comes from the serial port is redirected to
the
  'external keyboard so you use it to type
  Input "Name " , S
  'and show the result
 Print S
  'now wait for the F1 key , we defined the number 200 for F1 in the
table
 Do
   B = Getatkbd()
 Loop Until B <> 0
 Print B
Loop
End
'Since we do a redirection we call the routine from the redirection
routine
Kbdinput:
'we come here when input is required from the COM port
'So we pass the key into R24 with the GetATkbd function
' We need some ASM code to save the registers used by the function
$asm
push r16
                  ; save used register
push r25
push r26
push r27
Kbdinput1:
rCall _getatkbd
                  ; call the function
tst r24
                  ; check for zero
                 ; yes so try again
breg Kbdinputl
pop r27
                   ; we got a valid key so restore registers
pop r26
```

```
pop r25
pop r16
$end Asm
'just return
Return
'The tricky part is that you MUST include a normal call to the routine
'otherwise you get an error
'This is no clean solution and will be changed
B = Getatkbd()
'This is the key translation table
Kevdata:
'normal keys lower case
Data 0 , 0 , 0 , 0 , 0 , 113 , 49 , 0 , 0 , 0 , 122 , 115 , 97 , 119 ,
50,0
Data 0 , 99 , 120 , 100 , 101 , 52 , 51 , 0 , 0 , 32 , 118 , 102 , 116 ,
114 , 53 , 0
Data 0 , 110 , 98 , 104 , 103 , 121 , 54 , 7 , 8 , 44 , 109 , 106 , 117
, 55 , 56 , 0
Data 0 , 44 , 107 , 105 , 111 , 48 , 57 , 0 , 0 , 46 , 45 , 108 , 48 ,
112 , 43 , 0
Data 0 , 0 , 0 , 0 , 0 , 92 , 0 , 0 , 0 , 0 , 13 , 0 , 0 , 92 , 0 , 0
Data 0 , 60 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 ,
0,0
'shifted keys UPPER case
Data 0 , 0 , 0 , 0 , 0 , 81 , 33 , 0 , 0 , 0 , 90 , 83 , 65 , 87 , 34 ,
0
Data 0 , 67 , 88 , 68 , 69 , 0 , 35 , 0 , 0 , 32 , 86 , 70 , 84 , 82 ,
37,0
Data 0 , 78 , 66 , 72 , 71 , 89 , 38 , 0 , 0 , 76 , 77 , 74 , 85 , 47 ,
40 , 0
Data 0 , 59 , 75 , 73 , 79 , 61 , 41 , 0 , 0 , 58 , 95 , 76 , 48 , 80 ,
63 , 0
Data 0 , 0 , 0 , 0 , 0 , 96 , 0 , 0 , 0 , 0 , 13 , 94 , 0 , 42 , 0 , 0
Data 0 , 62 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 ,
0,0
```

6.139 CONFIG LCD

Action

Configure the LCD display and override the compiler setting.

Syntax

CONFIG LCD = LCDtype , CHIPSET=KS077 | Dogm163v5 | DOG163V3 | DOG162V5 | DOG162V3 [,CONTRAST=value]

Remarks

LCDtype	The type of LCD display used. This can be :		
	40 * 4,16 * 1, 16 * 2, 16 * 4, 16 * 4, 20 * 2 or 20 * 4 or 16 * 1a or 20*4A.		

	Default 16 * 2 is assumed.
KS077	Most text based LCD displays use the same chip from Hitachi. But some use the KS077 which is highly compatible but needs an additional function register to be set. This parameter will cause that this register is set when you initialize the display.
DOGM	The DOGM chip set uses a special function register that need to be set. The 16 x 2 LCD displays need DOG162V3 for 3V operation or DOG162V5 for 5V operation. The 16 x 3 LCD displays need DOG163V3 for 3V operation or Dogm163v5 for 5V operation
	The optional contrast parameter is only supported for the EADOG displays. By default a value from the manufacture is used. But you might want to override this value with a custom setting.

When you have a 16 * 2 display, you don't have to use this statement. The 16 * 1a is special. It is used for 2 * 8 displays that have the address of line 2, starting at location &H8.

The 20*4A is also special. It uses the addresses &H00, &H20, &H40 and &H60 for the 4 lines. It will also set a special function register.

The CONFIG LCD can only be used once. You can not dynamic(at run time) change the pins.

When you want to initialize the LCD during run time, you can use the $\underline{INITLCD}$ ras statement.

See Also

CONFIG LCDPIN 523, CONFIG LCDBUS 520

'name 'copyright 'purpose SHIFTCURSOR, HOME ' 'micro 'suited for demo 'commercial addon needed	1	TLCD,
<pre>\$regfile = "m8515.dat" the used micro \$crystal = 4000000 crystal frequency \$baud = 19200 rate \$hwstack = 32 use 32 for the hardware st \$swstack = 10 use 10 for the SW stack \$framesize = 40 use 40 for the frame space</pre>		' specify ' used ' use baud ' default ' default ' default

518 BASCOM-AVR

```
'REMOVE the above command for the real program !!
'$sim is used for faster simulation
'note : tested in PIN mode with 4-bit
'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector
Rem with the config lcdpin statement you can override the compiler
settings
Dim A As Byte
Config Lcd = 16 * 2
                                                              'configure
lcd screen
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 \,\star\, 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines
'$LCD = address will turn LCD into 8-bit databus mode
        use this with uP with external RAM and/or ROM
τ.
        because it aint need the port pins !
Cls
                                                              'clear the
LCD display
Lcd "Hello world."
                                                              'display
this at the top line
Wait 1
Lowerline
                                                              'select the
lower line
Wait 1
Lcd "Shift this."
                                                              'display
this at the lower line
Wait 1
For A = 1 To 10
   Shiftlcd Right
                                                              'shift the
text to the right
  Wait 1
                                                              'wait a
moment
Next
For A = 1 To 10
   Shiftlcd Left
                                                              'shift the
text to the left
  Wait 1
                                                              'wait a
moment
Next
Locate 2 , 1
                                                              'set cursor
position
Lcd "*"
                                                              'display
this
```

	BASCOM Language Reference	519
Wait 1 moment	'wait	a
Shiftcursor Right	'shift	t the
cursor Lcd "@"	'disp	lay
this Wait 1 moment	'wait	a
Home Upper	'selec	ct line
1 and return home Lcd "Replaced."	'repla	ace the
text Wait 1 moment	'wait	a
Cursor Off Noblink Wait 1	'hide 'wait	cursor a
moment Cursor On Blink		cursor
Wait 1 moment Display Off	'wait 'turn	a
display off Wait 1	'wait	a
moment Display On display on	'turn	
'NEW support for 4-line Thirdline Lcd "Line 3" Fourthline Lcd "Line 4" Home Third on line three	'goto	home
Home Fourth Home F letteer also works Locate 4 , 1 : Lcd "Line 4" Wait 1	'first	t
'Now lets build a special character 'the first number is the characternumber 'The other numbers are the rowvalues 'Use the LCD tool to insert this line	r (0-7)	
Deflcdchar 1 , 225 , 227 , 226 , 226 , 2 replace ? with number (0-7)		1
Deflcdchar 0 , 240 , 224 , 224 , 255 , 2 replace ? with number (0-7) Cls		ct data
RAM Rem it is important that a CLS is follow		ents
<pre>because it will set the controller back Lcd Chr(0) ; Chr(1) special character</pre>	in datamode 'print	t the
' Now use an internal r _temp1 = 1		e into
ACC !rCall _write_lcd LCD End	'put :	it on

```
Example2
                              . . . . . . . . .
                                                                                . . . . . . . . . . . . . . . .
                                         EADOG-M163.bas
                  Demonstration for EADOG 163 display
(c) 1995-2006, MCS Electronics
                     -----
$regfile = "M8515.dat"
$crystal = 4000000
'I used the following settings
'Config Lcdpin = Pin , Db4 = Portb.2 , Db5 = Portb.3 , Db6 = Portb.4 , Db7 = Portb.5 , E =
Portb.1 , Rs = Portb.0
'CONNECT vin TO 5 VOLT

Config Lcd = 16 * 3 , Chipset = Dogm163v5

'other options for chipset are DOG163V3 for 3Volt operation
                                                                                                       '16*3 type LCD display
'Config Lcd = 16 ^{*} 3 , Chipset = Dogm163v3 , Contrast = &H702 '16*3 'The CONTRAST can be specified when the default value is not what you need
                                                                                                                         '16*3 type LCD display
'The EADOG-M162 is also supported :
'Chipset params for the DOGM162 : DOG162V5, DOG162V3
CIs
                                                                                                        'Dit maakt het scherm leeg
Locate 1, 1: Lcd "Hello
Locate 2, 1: Lcd "line
Locate 3, 1: Lcd "line
                                              World"
                                            2"
3"
End
```

6.140 CONFIG LCDBUS

Action

Configures the LCD data bus and overrides the compiler setting.

Syntax

CONFIG LCDBUS = constant

Remarks

Constant 4 for 4-bit operation, 8 for 8-bit mode (default)

Use this statement together with the \$LCD = address statement.

When you use the LCD display in the bus mode the default is to connect all the data lines. With the 4-bit mode, you only have to connect data lines d7-d4.

See also

CONFIG LCD 516

```
' (c) 1995-2005 MCS Electronics
' file: LCD.BAS
' demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME
' CURSOR, DISPLAY
```

521

_____ _____ 'note : tested in bus mode with 4-bit on the STK200 'LCD - STK200 ____ 'D4 D4 'D5 D5 'D6 D6 'D7 D7 'WR WR ' E E 'RS RS '+5V +5V ' GND GND 'V0 V0 D0-D3 are not connected since 4 bit bus mode is used! 'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 , Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6 Rem with the config lcdpin statement you can override the compiler settings \$regfile = "8515def.dat" **\$lcd** = &HC000 **\$lcdrs** = &H8000 **Config** Lcdbus = 4 Dim A As Byte Config Lcd = 16×2 'configure lcd screen 'other options are 16 * 2 , 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a 'When you dont include this option 16 * 2 is assumed '16 * la is intended for 16 character displays with split addresses over 2 lines '\$LCD = address will turn LCD into 8-bit databus mode use this with uP with external RAM and/or ROM 1 because it aint need the port pins ! Cls 'clear the LCD display Lcd "Hello world." 'display this at the top line Wait 1 Lowerline 'select the lower line Wait 1 Lcd "Shift this." 'display this at the lower line Wait 1 **For** A = 1 **To** 10 Shiftlcd Right 'shift the text to the right Wait 1 'wait a moment Next **For** A = 1 **To** 10 Shiftlcd Left 'shift the text to the left Wait 1 'wait a moment Next

Locate 2 , 1 'set cursor position Lcd "*" 'display this Wait 1 'wait a moment Shiftcursor Right 'shift the cursor Lcd "@" 'display this Wait 1 'wait a moment Home Upper 'select line 1 and return home Lcd "Replaced." 'replace the text Wait 1 'wait a moment Cursor Off Noblink 'hide cursor Wait 1 'wait a moment Cursor On Blink 'show cursor 'wait a Wait 1 moment Display Off 'turn display off Wait 1 'wait a moment Display On 'turn display on -----NEW support for 4-line LCD-----!_____ Thirdline Lcd "Line 3" Fourthline Lcd "Line 4" Home Third 'goto home on line three Home Fourth Home F 'first letteer also works Locate 4 , 1 : Lcd "Line 4" Wait 1 'Now lets build a special character 'the first number is the characternumber (0-7)'The other numbers are the rowvalues 'Use the LCD tool to insert this line Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228 replace ? with number (0-7)Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240 ī. replace ? with number (0-7)Cls 'select data RAM Rem it is important that a CLS is following the deflcdchar statements because it will set the controller back in datamode Lcd Chr(0); Chr(1)'print the special character '----- Now use an internal routine ------

```
_temp1 = 1
ACC
!rCall _write_lcd
LCD
```

'value into

523

'put it on

6.141 CONFIG LCDMODE

Action

Configures the LCD operation mode and overrides the compiler setting.

Syntax

CONFIG LCDMODE = type

Remarks

	pins.
have The F decor and t LCD o	will drive the LCD in bus mode and in this mode is meant when you external RAM and so have an address and data bus on your system. RS and E line of the LCD display can be connected to an address der. Simply writing to an external memory location select the LCD the data is sent to the LCD display. This means the data-lines of the display are fixed to the data-bus lines.

See also

CONFIG LCD [516], \$LCD [332], \$LCDRS [337]

Example

```
Config LCDMODE = PORT 'the report will show the settings
Config LCDBUS = 4 '4 bit mode
LCD "hello"
```

6.142 CONFIG LCDPIN

Action

Override the LCD-PIN select options.

Syntax

CONFIG LCDPIN = PIN , DB4= PN,DB5=PN, DB6=PN, DB7=PN, E=PN, RS=PN [WR=PIN] [BUSY=PIN] [MODE=mode] **CONFIG LCDPIN** = PIN , PORT=PORTx, E=PN, RS=PN

Remarks

PN	The name of the PORT pin such as PORTB.2 for example.
PORTX	When you want to use the LCD in 8 bit data, pin mode, you must specify the PORT to use.
PIN	A port pin that is connected to the busy pin. The busy pin is only supported by the 20x4VFD display.
MODE	A mode for the 20x4VFD display. Options : 0 : 4 bit parallel upper nibble first 1 : 4 bit parallel lower nibble first

You can override the PIN selection from the Compiler Settings with this statement, so a second configuration lets you not choose more pins for a second LCD display.

The config command is preferred over the option settings since the code makes clear which pins are used. The CONFIG statement overrides the Options setting.

The PIN and MODE are only for the 20x4VFD display. See also LCDAUTODIM 757

The WR pin is optional. When you select the WR pin, an alternative library will be used. This library uses the WR pin and reads the BUSY signal from the LCD. The library lcd4busy_anypin will be used, which is based on Luciano's LUC_lcd4busy library.

Notice that since 2040 version, the compiler will generate LCD port pin info which you can use for your own libs.

By default the WR pin is optional and the WR signal of the LCD should be connected to ground. This saves the pin for other purposes. When you have enough pins, you better use the WR-pin.

See also

CONFIG LCD [516], CONFIG LCDMODE [523], CONFIG LCDBUS [520]

'name	: lcd.bas	
'copyright	: (c) 1995-2005, MCS Electronics	
'purpose	: demo: LCD, CLS, LOWERLINE, SHIFTL	CD,
SHIFTCURSOR, HOME		
	CURSOR, DISPLAY	
'micro	: Mega8515	
'suited for demo	: yes	
'commercial addon needed	: no	
\$regfile = "m8515.dat"	1	specify
the used micro		
\$crystal = 4000000	1	used
crystal frequency		
\$baud = 19200	1	use baud
rate		
<pre>\$hwstack = 32</pre>	1	default
use 32 for the hardware s	tack	

525

```
\$swstack = 10
                                                             ' default
use 10 for the SW stack
                                                             ' default
$framesize = 40
use 40 for the frame space
$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation
'note : tested in PIN mode with 4-bit
'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector
Rem with the config lcdpin statement you can override the compiler
settings
Dim A As Byte
Config Lcd = 16 * 2
                                                             'configure
lcd screen
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines
'$LCD = address will turn LCD into 8-bit databus mode
    use this with uP with external RAM and/or ROM
       because it aint need the port pins !
Cls
                                                             'clear the
LCD display
Lcd "Hello world."
                                                             'display
this at the top line
Wait 1
Lowerline
                                                             'select the
lower line
Wait 1
Lcd "Shift this."
                                                             'display
this at the lower line
Wait 1
For A = 1 To 10
  Shiftlcd Right
                                                             'shift the
text to the right
  Wait 1
                                                             'wait a
moment
Next
For A = 1 To 10
   Shiftlcd Left
                                                             'shift the
text to the left
   Wait 1
                                                             'wait a
```

.

moment Next Locate 2 , 1 'set cursor position Lcd "*" 'display this Wait 1 'wait a moment 'shift the Shiftcursor Right cursor Lcd "@" 'display this Wait 1 'wait a moment Home Upper 'select line 1 and return home Lcd "Replaced." 'replace the text Wait 1 'wait a moment 'hide cursor Cursor Off Noblink Wait 1 'wait a moment Cursor On Blink 'show cursor Wait 1 'wait a moment Display Off 'turn display off Wait 1 'wait a moment Display On 'turn display on -----NEW support for 4-line LCD-----Thirdline Lcd "Line 3" Fourthline Lcd "Line 4" Home Third 'goto home on line three Home Fourth Home F 'first letteer also works Locate 4 , 1 : Lcd "Line 4" Wait 1 'Now lets build a special character 'the first number is the characternumber (0-7)'The other numbers are the rowvalues 'Use the LCD tool to insert this line **Deflcdchar** 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228 replace ? with number (0-7) Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240 i replace ? with number (0-7)Cls 'select data RAM Rem it is important that a CLS is following the deflcdchar statements because it will set the controller back in datamode Lcd Chr(0); Chr(1)'print the special character

527

6.143 CONFIG OSC

Action

Select and enable the oscillators available to the Xmega

Syntax

CONFIG OSC=ENABLED|DISABLED, **PLLOSC**=ENABLED|DISABLED, **EXTOSC**= ENABLED|DISABLED, **32KHZOSC**=ENABLED|DISABLED, **32MHZOSC**=ENABLED| DISABLED, **RANGE**=range, **32KHZPOWERMODE**=powermode, **STARTUP**=startup

Remarks

OSC	Use ENABLED to enable the internal 2 MHZ oscillator. This oscillator is enabled by default. Use DISABLED to disable the internal oscillator.
PLLOSC	Use ENABLED to enable the PLL oscillator. The oscillator is disabled by default.
EXTOSC	Use ENABLED to enable the external oscillator. The external oscillator is disabled by default.
32KHZOSC	Use ENABLED to enable the internal 32 KHz oscillator. This oscillator is disabled by default.
32MHZOSC	Use ENABLED to enable the internal 32 MHz oscillator. This oscillator is disabled by default.
RANGE	Specify the range of the external oscillator. - 400KHZ_2MHZ - 2MHZ_9MHZ - 9MHZ_12MHZ - 12MHZ_16MHZ This option is only needed when using the external oscillator.
32KHZPOWERMODE	Select the power mode of the 32 KHz interal oscillator. This can be NORMAL or LOW_POWER. The default is NORMAL
STARTUP	The startup time can be specified. Use a value of : - EXTCLK (6 CLK) - 32KHZ (for 16 CLK) - XTAL_256CLK (for 256 CLK) - XTAL_1KCLK (for 1K CLK) - XTAL_16CLK (for 16K CLK)

See also

CONFIG SYSCLOCK 556

BASCOM-AVR

528

```
Config Osc = Enabled , 32mhzosc = Enabled ' enable 2 MHz and 32 MHz interal oscillators
```

6.144 CONFIG PORT

Action

Sets the port or a port pin to the right data direction.

Syntax

CONFIG PORTx = state **CONFIG PINx.y** = state

Remarks

state	A numeric constant that can be INPUT or OUTPUT.
	INPUT will set the data direction register to input for port X. OUTPUT will set the data direction to output for port X. You can also use a number for state. &B 00001111, will set the upper nibble to input and the lower nibble to output.
	You can also set a single port pin with the CONFIG PIN = state, statement. Again, you can use INPUT, OUTPUT or a number. In this case the number can be only zero or one.
x	A valid port letter such as A,B,C etc. Example : CONFIG PORT \mathbf{B} = Output
У	A valid pin number in the range of 0-7. Example : CONFIG PINB. 0 =INPUT

The best way to set the data direction for more than 1 pin, is to use the CONFIG PORT, statement and not multiple lines with CONFIG PIN statements.

You may not use variables for the port letters and pin numbers. If you need to dynamically set a pin direction, you can use this form : SET PORTB.somepin , where somepin may be a constant or a variable.

If the the port itself is also dynamic, then you could use OUT with the proper address.

See Also

AVR Internal hardware ports

BASCOM Language Reference 529

\$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud** = 19200 ' use baud rate **\$hwstack = 32** ' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Dim A As Byte , Count As Byte 'configure PORT D for input mode Config Portd = Input 'reading the PORT, will read the latch, that is the value 'you have written to the PORT. 'This is not the same as reading the logical values on the pins! 'When you want to know the logical state of the attached hardware, 'you MUST use the PIN register. A^{-} = Pind 'a port or SFR can be treated as a byte A = A And Portd Print A 'print it Bitwait Pind.7 , Reset 'wait until bit is low 'We will use port B for output Config Portb = Output 'assign value Portb = 10'set port B to 10 Portb = Portb And 2 Set Portb.0 'set bit 0 of port B to 1 Incr Portb 'Now a light show on the STK200 Count = 0Do Incr Count Portb = 1**For** A = 1 **To** 8 'rotate bits Rotate Portb , Left left Wait 1 Next 'the following 2 lines do the same as the previous loop 'but there is no delay Portb = 1 Rotate Portb , Left , 8 Loop Until Count = 10 Print "Ready"

'Again, note that the AVR port pins have a data direction register 'when you want to use a pin as an input it must be set low first 'you can do this by writing zeros to the DDRx: 'DDRB =&B1110000 'this will set portbl.0,portb.1,portb.2 and portb.3 to use as inputs. 'So : when you want to use a pin as an input set it low first in the DDRx! ' and read with PINx ' and when you want to use the pin as output, write a 1 first and write the value to PORTx End

6.145 CONFIG POWER_REDUCTION

Action

This option configures the power reduction registers to reduce power consumption.

Syntax

CONFIG POWER_REDUCTION = dummy, device=ON|OFF

Remarks

The Power Reduction (PR) registers provides a method to stop the clock to individual peripherals.

When this is done the current state of the peripheral is frozen and the associated I/O registers cannot be read or written. Resources used by the peripheral will remain occupied;

hence the peripheral should in most cases be disabled before stopping the clock. Enabling the

clock to a peripheral again, puts the peripheral in the same state as before it was stopped. This

can be used in Idle mode and Active mode to reduce the overall power consumption significantly.

In all other sleep modes, the peripheral clock is already stopped.

Not all devices have all the peripherals associated with a bit in the power reduction registers.

Setting a power reduction bit for a peripheral that is not available will have no effect.

A hardware resource of the Xmega. The following hardware resources can be deactivated to reduce power: AES EBI RTC EVSYS DMA DACA, DACB ACA,ACB ADCA,ADCB TWIC,TWID,TWIE,TWIF USARTC0,USARTC1, USARTD0,USARTD1,USARTE0, USARTE1,USARTF0,USARTF1 SPIC,SPID,SPIE,SPIF TCC0,TCC1,TCD0,TDC1,TCE0,TCE1,TCF0,TCF1 HIRESC,HIRESD,HIRESE,HIRESF

531

A value of ON will leave the resource enabled and a value of OFF will activate the power reduction.		A value of ON will leave the resource enabled and a value of OFF will activate the power reduction.	
---	--	---	--

You should use the CONFIG POWER_REDUCTION at start up to disable all unused resources. All the power reduction registers will be set for the provided resources. But the existing configuration will not be preserved. When you need to enable/disable an individual resource at run time, you can manual access the register with a SET or RESET command.

For example, the DMA, EVSYS, RTC, EBI and AES bits are located in the PRGEN register. If you disable DMA and AES the compiler will write a value of 17 (dma +aes) to the PRGEN register.

It will not first read the existing value, and preserve the other bits. That is why this statement should be used once.

When you specify one value, for example DMA, it will write 1 to the PRGEN register and thus overwriting the previous AES bit that was 1, with a 0.

The additional code to mask and set the bits did not seem useful at implementation time. At user request this behaviour can be changed in a future version.

See also

NONE

Example

```
XM128A1-POWER-REDUCTION.BAS
(c) 1995-2011 MCS Electronics
sample provided by MAK3
CONFIG POWER_REDUCTION and USING EVENT SYSTEM
```

 $^{\prime}$ This Example show how to use the config power_reduction and give first insights to the XMEGA EVENT SYSTEM

' Regarding the Eventsytem this example easy show after event configuration that one Port Pin is routed to another Port Pin. ' You can see it works even during the WAIT 4 command and there are no PORT READ OR WRITE commands in the Do Loop ! ' It also shows how to manual fire an Event

```
$regfile = "xm128aldef.dat"
$crystal = 2000000 '2MHz
$hwstack = 64
$swstack = 40
$framesize = 40
$lib "xmega.lib" :$external _xmegafix_clear : $external
_xmegafix_rol_r1014
Config Osc = Enabled
Config Sysclock = 2mhz '2MHz
'YOU CAN MINIMIZE POWER CONSUMPTION FOR EXAMPLE WITH :
'1. Use Low supply voltage
'2. Use Sleep Modes
'3. Keep Clock Frequencys low (also with Precsalers)
```

BASCOM-AVR

532

```
' 4. Use Powe Reduction Registers to shut down unused peripherals
'With Power_reduction you can shut down specific peripherals that are
not used in your application
'Paramters:
aes, dma, ebi, rtc, evsys, daca, dacb, adca, adcb, aca, acb, twic, usartc0, usartc1, s
pic, hiresc, tcc0, tcc1
Config Power_reduction = Dummy , Aes = Off , Twic = Off , Twid = Off ,
Twie = Off , Aca = Off , Adcb = Off , Tcc0 = Off , Tcc1 = Off , Dma =
Off
'For the following we need the EVENT System therefore we do not shut
down EVENT SYSTEM
Config Com1 = 9600 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Open "COM1:" For Binary As #1
Waitms 2
Print #1 ,
Print #1, "-----S T A R T-----"
'Configure PCO for input, triggered on falling edge
Config Pinc.0 = Input
Portc_pin0ctrl = &B00_011_010
                      1.
                      1.4
                          React on falling edge (010)
                      1.4
                      'enable Pullup
'Select PCO as input to event channel O
'select the event source for Event Channel 0
Evsys_ch0mux = &B0110_0_000
                                                  'Event Source for
Event Channel 0 = Portc.0
                 1.4
                         •
                 1.
                         Pin0
                 'portC
                                                  '8 SAMPLES for Digital
Evsys_ch0ctrl = \&B0_00_0_0_111
Filter
                            1.4
                            'Digital Filter config
Config Pinc.7 = Output
'Event Channel 0 Ouput Configuration
Portcfg_clkevout = &B0_0_01_0_00
                                                  'Output on PINC.7
/Clock Out must be disabled
Print #1 , "Portcfg_clkevout = " ; Bin(portcfg_clkevout)
Print #1 , "Mainloop -->"
Do
  'IMPORTANT: YOU WILL SEE THE PIN CHANGES ALSO DURING WAIT 4 BECAUSE IT
USE EVENT SYSTEM
  Wait 4
  'This shows how to manual fire an Event
```

Set Evsys_strobe.0

Loop

End

'end program

6.146 CONFIG PRIORITY

Action

Configures the interrupt system and priority for Xmega

Syntax

Remarks

CONFIG PRIORITY = prio, VECTOR = vector, HI = hi, LO = lo, MED = med

<u>Nemarks</u>	
prio	STATIC or ROUNDROBIN. In the AVR the lowest interrupt address has the highest priority. When you chose STATIC the interrupts behave as in non-Xmega chips. To prevent that a low priority interrupt never get executed you can select ROUNDROBIN
vector	APPLICATION or BOOT. Application is the default. This will place the interrupt vectors at address 0, the starting address. When you chose BOOT, the interrupt vectors are placed at the beginning of the boot section. This makes it possible to use interrupts in a boot application.
hi	ENABLED or DISABLED. Chose ENABLED to enable the HI priority interrupts.
lo	ENABLED or DISABLED. Chose ENABLED to enable the LO priority interrupts.
med	ENABLED or DISABLED. Chose ENABLED to enable the MED priority interrupts.

In the XMEGA, you must enable HI, LO or MED interrupts before you can use them. When you enable an interrupt you also must specify the priority. For example : Enable Usartc0_rxc , Lo This would enable the USARTC0_RX interrupt and would assign it a low priority.

In this case, at least the LO priority should be enabled : Config Priority = Static , Vector = Application , Lo = Enabled

When you use LO and MED interrupts, you need to enable the both.

See also

ENABLE 6761, DISABLE 6571, ON 7891

Example

Config Priority = Static , Vector = Application , Lo = Enabled

```
On Usartc0_rxc Rxc_isr
Enable Usartc0_rxc , Lo
Enable Interrupts
```

6.147 CONFIG PRINT

Action

Configure the UART to be used for RS-485

Syntax

CONFIG PRINTO = pin CONFIG PRINT1 = pin CONFIG PRINT2 = pin CONFIG PRINT3 = pin

Remarks

	The name of the PORT pin that is used to control the direction of an RS-485 driver.
mode	SET or RESET

Use PRINT or PRINT0 for the first serial port. Use PRINT1 for the second serial port. PRINT2 for the third UART and PRINT3 for the fourth UART.

When you use RS-485 half duplex communication you need a pin for the direction of the data. The CONFIG PRINT automates the manual setting/resetting. It will either SET or RESET the logic level of the specified pin before data is printed with the BASCOM print routines. After the data is sent, it will inverse the pin so it goes into receive mode.

You need to set the direction of the used pin to output mode yourself.

See also

CONFIG PRINTBIN 535

'name 'copyright 'purpose 'micro 'suited for demo 'commercial addon needed	: rs485.bas : (c) 1995-2006, MCS Electronics : demonstrates : Mega48 : yes : no		
<pre>\$regfile = "m48def.dat" M48 \$crystal = 8000000 \$baud = 19200</pre>		' we	use the
<pre>\$hwstack = 32 \$swstack = 32 \$framesize = 32</pre>			
Config Print0 = Portb.0, Config Pinb.0 = Output direction yourself	Mode = Set	'set	the

Dim Resp As String * 10 Do Print "test message" Input Resp response Loop

get

6.148 CONFIG PRINTBIN

Action

Configure PRINTBIN behavior

Syntax

CONFIG PRINTBIN = extended

Remarks

This mode is the only mode. It allows to send huge arrays(more then 255 elements) to the serial port. Without the CONFIG PRINTBIN option, the maximum number of elements is 255. Because support for big
arrays cost more code, it is made optional.

See also

CONFIG PRINT 534

Example

```
$regfile = "m103def.dat"
                                                              ' specify
the used micro
                                                              ' used
$crystal = 8000000
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Config Printbin = Extended
Dim A(1000)
Printbin A(1) ; 1000
```

6.149 CONFIG PS2EMU

Action

Configures the PS2 mouse data and clock pins.

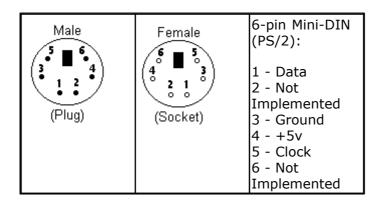
Syntax

CONFIG PS2EMU= int , DATA = data, CLOCK=clock

Remarks

Int	The interrupt used such as INT0 or INT1.
	The pin that is connected to the DATA line. This must be the same pin as the used interrupt.
CLOCK	The pin that is connected to the CLOCK line.

Male	Female	5-pin DIN (AT/ XT):
(Plug)	(Socket)	1 - Clock 2 - Data 3 - Not Implemented 4 - Ground 5 - +5v



Old PC's are equipped with a 5-pin DIN female connector. Newer PC's have a 6-pin mini DIN female connector.

The male sockets must be used for the connection with the micro.

Besides the DATA and CLOCK you need to connect from the PC to the micro, you need to connect ground. You can use the +5V from the PC to power your microprocessor.

The config statement will setup an ISR that is triggered when the INT pin goes low. This routine you can find in the library.

The ISR will retrieve a byte from the PC and will send the proper commands back to the PC.

The SENDSCAN and PS2MOUSEXY statements allow you to send mouse commands.

Note that the mouse emulator is only recognized after you have booted your PC. Mouse devices can not be plugged into your PC once it has booted. Inserting a mouse or mouse device when the PC is already booted, may damage your PC.

See also

SENDSCAN 862, PS2MOUSEXY 816

```
-----
_____
                        : ps2_emul.bas
'name
                        : (c) 1995-2005, MCS Electronics
'copyright
                        : PS2 Mouse emulator
'purpose
'micro
                        : 90S2313
'micro : 9052313
'suited for demo : NO, commercial addon needed
'commercial addon needed : yes
                                 _____
·-----
_____
$regfile = "2313def.dat"
                                                        ' specify
the used micro
$crystal = 4000000
                                                        ' used
crystal frequency
$baud = 19200
                                                        ' use baud
rate
$hwstack = 32
                                                        ' default
use 32 for the hardware stack
\$swstack = 10
                                                        ' default
use 10 for the SW stack
$framesize = 40
                                                        ' default
use 40 for the frame space
$lib "mcsbyteint.lbx"
                                                        ' use
optional lib since we use only bytes
'configure PS2 pins
Config Ps2emu = Int1 , Data = Pind.3 , Clock = Pinb.0
               ^----- used interrupt
                            ^---- pin connected to DATA
^-- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin
Waitms 500
                                                        ' optional
delay
Enable Interrupts
                                                        ' you need
to turn on interrupts yourself since an INT is used
Print "Press u,d,l,r,b, or t"
Dim Key As Byte
Do
   Key = Waitkey()
                                                        ' get key
from terminal
   Select Case Key
     Case "u" : Ps2mousexy 0 , 10 , 0
                                                        ' up
     Case "d" : Ps2mousexy 0 , -10 , 0
                                                        ' down
     Case "1" : Ps2mousexy -10 , 0 , 0
                                                        ' left
     Case "r" : Ps2mousexy 10 , 0 , 0
                                                        ' right
     Case "b" : Ps2mousexy 0 , 0 , 1
                                                        ' left
button pressed
                                                        ' left
               Ps2mousexy 0 , 0 , 0
button released
     Case "t" : Sendscan Mouseup
                                                        ' send a
scan code
     Case Else
```

```
End Select
Loop
```

```
Mouseup:
Data 3 , &H08 , &H00 , &H01
by 1 unit
```

' mouse up

6.150 CONFIG RC5

Action

Overrides the RC5 pin assignment from the Option Compiler Settings 3.

Syntax

CONFIG RC5 = pin [,TIMER=2] [,WAIT=value]

Remarks

Pin	The port pin to which the RC5 receiver is connected.
TIMER	Must be 2. The micro must have a timer2 when you want to use this option. This additional parameter will cause that TIMER2 will be used instead of the default TIMER0.
WAIT	The default value is 100. Each unit is ca. 64 us. This gives a time out of 6.4 ms. Since a start bit is 3.5 ms, you can reduce the value to 56. When you make it lower, it will not work. When you want the old behavior you need to specify a value of 2000 which is ca. 131 ms.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the RC5 pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. In BASCOM-AVR the settings are also stored in the project. CFG file.

See also

GETRC5 712

Example

CONFIG RC5 = PIND.5 'PORTD.5 is the RC5 input line

6.151 CONFIG SDA

Action

Overrides the SDA pin assignment from the Option Compiler Settings 3.

Syntax

CONFIG SDA = pin

Remarks

Pin

The port pin to which the I2C-SDA line is connected.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the SDA pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. In BASCOM-AVR the settings are also stored in the project. CFG file.

When using the Hardware TWI, you only need CONFIG SDA when you use the I2CINIT statement

See also

CONFIG SCL 539, CONFIG I2CDELAY 498, I2CINIT 725

Example

CONFIG SDA = PORTB.7 'PORTB.7 is the SDA line

6.152 CONFIG SCL

Action

Overrides the SCL pin assignment from the Option Compiler Settings 3.

Syntax

CONFIG SCL = pin

Remarks

Pin

The port pin to which the I2C-SCL line is connected.

When you use different pins in different projects, you can use this statement to override the Options Compiler setting for the SCL pin. This way you will remember which pin you used because it is in your code and you do not have to change the settings from the options. Of course BASCOM-AVR also stores the settings in a project.CFG file.

When using the Hardware TWI, you only need CONFIG SCL when you use the I2CINIT statement

See also

CONFIG SDA 538, CONFIG I2CDELAY 498, I2CINIT 725

Example

CONFIG SCL = PORTB.5 'PORTB.5 is the SCL line

6.153 CONFIG SERIALIN

Action

540

Configures the hardware UART to use a buffer for input

Syntax

CONFIG SERIALIN | SERIALIN1 | SERIALIN2 | SERIALIN3 = BUFFERED, SIZE = size [, BYTEMATCH=ALL|BYTE|NONE] [,CTS=pin, RTS=pin, Threshold_full=num, Threshold_empty=num]

Remarks

SerialIn	Some chips have multiple HW UARTS. Use the following parameter values: • SERIALIN : first UART/UART0 • SERIALIN1 : second UART/UART1 • SERIALIN2 : third UART/UART2 • SERIALIN3 : fourth UART/UART3
Size	A numeric constant that specifies how large the input buffer should be. The space is taken from the SRAM. The maximum is 255.
Bytematch	The ASCII value of the byte that will result in calling a user label. When you specify ALL , the user label will be called for every byte that is received. You must include the label yourself in your code and end it with a return. The following label names must be used when you check for a specific byte value:
	 SerialOCharMatch (for SERIALIN or the first UART/UARTO) Serial1CharMatch (for SERIALIN1 or the second UART/UART1) Serial2CharMatch (for SERIALIN2 or the third UART/UART2) Serial3CharMatch (for SERIALIN3 or the fourth UART/UART3)
	The following label names must be used when you check for any value:
	 SerialOByteReceived (for SERIALIN or the first UART/UARTO) Serial1ByteReceived (for SERIALIN1 or the second UART/UART1) Serial2ByteReceived (for SERIALIN2 or the third UART/UART2) Serial3ByteReceived (for SERIALIN3 or the fourth UART/UART3)
	When you specify NONE , it is the same as not specifying this optional parameter.
CTS	The pin used for the CTS.(Clear to send). For example PIND.6
RTS	The pin used for RTS. (Ready to send). For example PIND.7
Threshold_full	The number of bytes that will cause RTS to be set to '1'. This is an indication to the sender, that the buffer is full.
Threshold_empt y	The number of free bytes that must be in the buffer before CTS may be made '0' again.

The following internal variables will be generated for UARTO:

_RS_HEAD_PTR**0** , a byte counter that stores the head of the buffer

_RS_TAIL_PTR**0**, a byte counter that stores the tail of the buffer.

_RS232INBUF**0** , an array of bytes that serves as a ring buffer for the received

characters.

_RS_BUFCOUNTR**0**, a byte that holds the number of bytes that are in the buffer.

For the other UARTS, the variables are named similar. But they do have a different number.

A **1** for the second UART, a **3** for the third UART and a **4** for the fourth UART. Yes, the '**2**' is skipped.

While you can read and write the internal variables, we advise not to write to them. The variables are updated inside interrupts routines, and just when you write a value to them, an ISR can overwrite the value.

The optional **BYTEMATCH** can be used to monitor the incoming bytes and call a label when the specified label is found. This label is a fixed label as mentioned in the table above.

This way you can determine the start of a serial stream.

While bytematch allows you to trap the incoming bytes, take care that you do not delay the program execution too much. After all the serial input interrupt is used in order not to miss incoming data. When you add delays or code that will delay execution too much you might loose incoming data.

When using the BYTEMATCH option, you must preserve the registers you alter. If you do not know which one, use <u>PUSHALL</u> BIB and <u>POPALL</u> BIB.

To clear the buffer, use <u>CLEAR</u> (432) SERIALIN. Do not read and write the internal buffer variables yourself.

CTS-RTS is hardware flow control. Both the sender and receiver need to use CTS-RTS when CTS-RTS is used. When one of the parties does not use CTS-RTS, no communication will be possible.

CTS-RTS use two extra lines. The receiver must check the CTS pin to see if it may send. The CTS pin is a input pin as the receiver looks at the level that the sender can change.

The receiver can set the RTS pin to indicate to the sender that it can accept data. In the start condition, RTS is made '0' by the receiver. The sender will then check this logic level with it's CTS pin, and will start to send data. The receiver will store the data into the buffer and when the buffer is almost full, or better said, when the Threshold_full is the same as the number of bytes in the receive buffer, the receiver will stop sending data. And will continue when the RTS is made '0' again.

The receiver can send data to the sender and it will check the CTS pin to see if it may send data.

In order to work with CTS-RTS, you need both a serial input buffer, and a serial output buffer. So use both CONFIG SERIALIN and CONFIG SERIALOUT to specify the buffers.

The CTS-RTS can only be configured with the CONFIG SERIALIN statement.

The thresholds are needed for high baud rates where it will take some time to react on a CTS-RTS.

You need to experiment with the thresholds but good start values are 80% full, and 20% empty.

You need to use a pin that is bit addressable. For most chips this is a pin from port A, B,C or D.

ASM

Routines called from MCS.LIB :

_GotChar. This is an ISR that gets called when ever a character is received. When there is no room for the data it will not be stored. So the buffer must be emptied periodic by reading from the serial port using the normal statements like INKEY() and INPUT.

Since URXC interrupt is used by _GotChar, you can not use this interrupt anymore. Unless you modify the _gotchar routine of course.

See also

CONFIG SERIALOUT 545, ISCHARWAITING 750, CLEAR 432

Example

_____ _____ : rs232buffer.bas 'name : (c) 1995-2005, MCS Electronics 'copyright 'purpose : example shows the difference between normal and buffered serial INPUT : Megal61 'micro 'suited for demo : yes 'commercial addon needed : no -------\$regfile = "m161def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 9600 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10 ' default use 10 for the SW stack ' default **\$framesize** = 40 use 40 for the frame space 'first compile and run this program with the line below remarked **Config** Serialin = Buffered , Size = 20 Dim Na As String * 10 'the enabling of interrupts is not needed for the normal serial mode 'So the line below must be remarked to for the first test Enable Interrupts Print "Start"

Joo 'get a char from the UART

```
543
```

```
If Ischarwaiting() = 1 Then
                                                             'was there a
char?
      Input Na
      Print Na
                                                             'print it
   End If
  Wait 1
                                                             'wait 1
second
Loop
'You will see that when you slowly enter characters in the terminal
emulator
'they will be received/displayed.
'When you enter them fast you will see that you loose some chars
'NOW remove the remarks from line 11 and 18
'and compile and program and run again
'This time the chars are received by an interrupt routine and are
'stored in a buffer. This way you will not loose characters providing
that
'you empty the buffer
'So when you fast type abcdefg, they will be printed after each other
with the
'1 second delay
'Using the CONFIG SERIAL=BUFFERED, SIZE = 10 for example will
'use some SRAM memory
'The following internal variables will be generated :
'_Rs_head_ptr0
               BYTE , a pointer to the location of the start of the
buffer
'_Rs_tail_ptr0 BYTE , a pointer to the location of tail of the buffer
'_RS232INBUF0 BYTE ARRAY , the actual buffer with the size of SIZE
```

Example2

'name : (c) 1995-2008, MCS Electronics test for M2560 support copyright 'purpose : Mega2560 micro 'suited for demo : yes 'commercial addon needed : no \$regfile = "m2560def.dat"
\$crystal = 8000000
\$hwstack = 40
hardware stack . specify the used micro crystal . used crystal frequency default use 32 for the \$swstack = 40 default use 10 for the SW stack \$framesize = 40 default use 40 for the frame space '\$timeout = 1000000 'The M128 has an extended UART. when CO'NFIG COMx is not used, the default N,8,1 will be used **Config** Com1 = 19200, Synchrone = 0, Parity = None, Stopbits = 1, Databits = 8, **Config** Com2 = 19200, Synchrone = 0, Clockpol = 0Parity = None , Stopbits = 1, Databits = 8, **Config** Com3 = 19200, Synchrone = 0, Stopbits = 1, Parity = None, Databits = 8. Clockpol = 0 $\begin{array}{l} \textbf{Config} \text{ Com4} = 19200 \\ \text{Clockpol} = 0 \end{array}$ Synchrone = 0, Parity = None, Stopbits = 1, Databits = 8, **Enable Interrupts** Size = 20 Size = 20 , Size = 20 , Serialin = Serialin1 = Config Config Buffered Buffered , Bytematch = 65Buffered , Config Serialin2 = Bvtematch = 66Serialin3 = Size = 20, Bytematch = All Config Buffered

'Open all UARTS Open "COM2:" For Binary As #2 Open "COM3:" For Binary As #3 Open "COM4:" For Binary As #4 Print "Hello" Dim B1 As Byte, B2 As Byte, B3 As Byte, B4 As Byte Dim Tel As Word, Nm As String * 16 'first uart 'unremark to test second UART 'Input #2 , "Name ?" , Nm 'Print #2 , "Hello " ; Nm Do Incr Tel Print Tel ; " test serial port Print #2 , Tel ; " test serial Print #3 , Tel ; " test serial Print #4 , Tel ; " test serial serial port 1" test serial port 2" port port 3" Δ" B1 = Inkey() 'first uart B2 = Inkey(#2) B3 = Inkey(#3) B4 = Inkey(#4) I f B1 <> 0 Then Print B1 ; " from port 1" End I f I f B2 <> 0 Then Print #2 , B2 ; " from port 2" End I f I f B3 <> 0 Then Print #3 , B3 ; " from port 3" End I f I f B4 <> 0 Then Print #4 , B4 ; " from port 4" End I f Waitms 500 Loop 'Label called when UART2 received an A Serial1charmatch: Print #2, "we got an A" Return 'Label called when UART2 received a B Serial2charmatch: Print #3, "we got a B" Return 'Label called when UART3 receives a char Serial3bytereceived: Print #4 , "we got a char" Return End Close #2 Close #3 Close #4 \$eeprom Data 1, 2

6.154 CONFIG SERIALOUT

Action

Configures the hardware UART to use a buffer for output

Syntax

CONFIG SERIALOUT | SERIALOUT1 | SERIALOUT2 | SERIALOUT3 =

BUFFERED , SIZE = size

Remarks

SerialOut	Some chips have multiple HW UARTS. Use the following parameter values: • SERIALOUT : first UART/UART0 • SERIALOUT1 : second UART/UART1 • SERIALOUT2 : third UART/UART2 • SERIALOUT3 : fourth UART/UART3
size	A numeric constant that specifies how large the output buffer should be. The space is taken from the SRAM. The maximum value is 255.

The following internal variables will be used when you use CONFIG SERIALOUT

_RS_HEAD_PTRW0 , byte that stores the head of the buffer

_RS_TAIL_PTRW**0** , byte that stores the tail of the buffer

_RS2320UTBUF**0**, array of bytes for the ring buffer that stores the printed data.

_RS_BUFCOUNTW**0**, a byte that holds the number of bytes in the buffer.

For the other UARTS, the variables are named similar. But they do have a different number.

A **1** for the second UART, a **3** for the third UART and a **4** for the fourth UART. Yes, the '**2**' is skipped.

Serial buffered output can be used when you use a low baud rate. It would take relatively much time to print all data without a buffer. When you use a buffer, the data is printed on the background when the micro UART byte buffer is empty. It will get a byte from the buffer then and transmit it.

As with any buffer you have, you must make sure that it is emptied at one moment in time.

You can not keep filling it as it will become full. When you do not empty it, you will have the same situation as without a buffer !!! When the roof is leaking and you put a bucket on the floor and in the morning you empty it, it will work. But when you will go away for a day, the bucket will overflow and the result is that the floor is still wet.

Another important consideration is data loss. When you print a long string of 100 bytes, and there is only room in the buffer for 80 bytes, there is still a wait evolved since after 80 bytes, the code will wait for the buffer to become empty. When the buffer is empty it will continue to print the data. The advantage is that you do not loose any data, the disadvantage is that it blocks program execution just like a normal un-buffered PRINT would do.

ASM

Routines called from MCS.LIB :

_CHECKSENDCHAR. This is an ISR that gets called when ever the transmission buffer is empty.

546 BASCOM-AVR

Since UDRE interrupt is used , you can not use this interrupt anymore. Unless you modify the _CheckSendChar routine of course.

When you use the PRINT statement to send data to the serial port, the UDRE interrupt will be enabled. And so the _CheckSendChar routine will send the data from the buffer.

See also

CONFIG SERIALIN 540

Example

_____ 'name : rs232bufferout.bas : (c) 1995-2005, MCS Electronics 'copyright : demonstrates how to use a serial output 'purpose buffer : Megal28 'micro 'suited for demo : yes 'commercial addon needed : no ·_____ ------\$regfile = "m128def.dat" ' specify the used micro ' used \$crystal = 4000000crystal frequency **\$baud =** 9600 ' use baud rate \$hwstack = 40' default use 32 for the hardware stack \$swstack = 40' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 , Clockpol = 0Config Com2 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 , Clockpol = 0'setup to use a serial output buffer 'and reserve 20 bytes for the buffer Config Serialout = Buffered , Size = 255 'It is important since UDRE interrupt is used that you enable the interrupts Enable Interrupts Print "Hello world" Print "test1" Do Wait 1 'notice that using the UDRE interrupt will slown down execution of waiting loops like waitms Print "test" Loop End

6.155 CONFIG SINGLE

Action

Instruct the compiler to use an alternative conversion routine for representation of a single.

Syntax

CONFIG SINGLE = SCIENTIFIC , DIGITS = value

Remarks

Digits	A numeric constant with a value between 0 and 7. A value of 0 will result in no trailing zero's. A value between 1-7 can be used to specify the number of digits behind the comma.
--------	---

When a conversion is performed from numeric single variable, to a string, for example when you PRINT a single, or when you use the STR() function to convert a single into a string, a special conversion routine is used that will convert into human readable output. You will get an output of digits and a decimal point.

This is well suited for showing the value on an LCD display. But there is a downside also. The routine is limited in the way that it can not shown very big or very small numbers correct.

The CONFIG SINGLE will instruct the compiler to use a special version of the conversion routine. This version will use scientific notation such as : 12e3. You can specify how many digits you want to be included after the decimal point.

See also

NONE

ASM

Uses single.lbx library

Example

```
(c) 1995-2005, MCS
                            single_scientific.bas
   demonstation of scientific, single output
   . . . . . . . . . . . . . . . . . . .
$regfile = "m88def.dat"
$crystal = 8000000
$baud = 19200
'you can view the difference by compiling and simulating
                                                                      this sample
with
      the
     below remarked
'line
                        and active
Config Single =
                       Scientific , Digits = 7
Dim S As Single
S = 1
Do
  S = S / 10
  Print S
Loop
```

6.156 CONFIG SHIFTIN

Action

548

Instruct the compiler to use new behaviour of the SHIFTIN statement.

Syntax

CONFIG SHIFTIN = value

Remarks

value	This must be COMPATIBLE or NEW. By default the old behaviour is used. So in order to use the new behaviour you must use : CONFIG SHIFTIN=NEW

The SHIFTOUT has been enhanced with a number of options which make it incompatible to the old SHIFTOUT.

In order to maintain compatibility with your old code, this option has been added so you have control over which SHIFTIN version is used.

See also

SHIFTIN 877

6.157 CONFIG SPI

Action

Configures the SPI related statements.

Syntax for software SPI

CONFIG SPI = SOFT, DIN = PIN, DOUT = PIN , SS = PIN|NONE, CLOCK = PIN , SPIIN=value , MODE=mode

Syntax for hardware SPI

CONFIG SPI = HARD, INTERRUPT=ON|OFF, DATA ORDER = LSB|MSB, MASTER = YES|NO, POLARITY = HIGH|LOW, PHASE = 0|1, CLOCKRATE = 4|16|64|128, NOSS=1|0, SPIIN=value

Remarks

SPI	SOFT for software emulation of SPI, this allows you to choose the PINS to use. Only works in master mode. HARD for the internal SPI hardware, that will use fixed pins of the microprocessor.
DIN	Data input or MISO. Pin is the pin number to use such as PINB.0
DOUT	Data output or MOSI. Pin is the pin number to use such as PORTB.1
SS	Slave Select. Pin is the pin number to use such as PORTB.2 Use NONE when you do not want the SS signal to be generated. See

549

	remarks		
CLOCK	Clock. Pin is the pin number to use such as PORTB.3		
DATA ORDER	Selects if MSB or LSB is transferred first.		
MASTER	Selects if the SI	PI is run in master or slave	mode.
POLARITY	Select HIGH to make the CLOCK line high while the SPI is idle. LOW will make clock LOW while idle.		
PHASE	Refer to a data sheet to learn about the different settings in combination with polarity.		
CLOCKRATE	The clock rate selects the division of the of the oscillator frequency that serves as the SPI clock. So with 4 you will have a clock rate of $4.000000 / 4 = 1 \text{ MHz}$, when a 4 MHZ XTAL is used.		
NOSS	1 or 0. Use 1 when you do not want the SS signal to be generated in master mode.		
INTERRUPT	Specify ON or OFF. ON will enable the SPI interrupts to occur. While OFF disables SPI interrupts. ENABLE SPI and DISABLE SPI will accomplish the same.		
SPIIN	When reading from the SPI slave, it should not matter what kind of data you send. But some chips require a value of 255 while others require a value of 0. By default, when the SPIIN option is not provided, a value of 0 will be sent to the SPI slave. With this SPIIN option you can override this value.		
MODE	A constant in the range from 0-3 which defines the SPI MODE. Without MODE, the default mode 1 will be used. Also, when using MODE, new SPI code will be used. When using MODE, you can also specify SPEED and SETUP.		
	Mode 0 1 2 3	Leading Edge Rising, Sample Rising, Setup Falling, Sample Falling, Setup	Trailing Edge Falling, Setup Falling, Sample Rising, Setup Rising, Sample
SPEED	Is a numeric constant for an optional delay. This delay is in us. When you specify 1, it will result in 2 us delay : 1 use before and 1 us after the clock. By default there is no delay. Slower chip would require a delay.		
SETUP	Setup is the del constant must b	ay in uS before sampling th be used.	ne MISO pin. A numeric

The default setting for hardware SPI when set from the Compiler, Options, SPI menu is MSB first, POLARITY = HIGH, MASTER = YES, PHASE = 0, CLOCKRATE = 4

When you use CONFIG SPI = HARD alone without the other parameters, the SPI will only be enabled. It will work in slave mode then with CPOL =0 and CPH=0.

In hardware mode the SPIINIT statement will set the SPI pins to : sbi DDRB,7 ; SCK output cbi DDRB,6 ; MISO input sbi DDRB,5 ; MOSI output

In softmode the SPIINIT statement will set the SPI pins for example to : sbi PORTB,5 ;set latch bit hi (inactive)SS sbi DDRB,5 ;make it an output SS cbi PORTB,4 ;set clk line lo sbi DDRB,4 ;make it an output

550 BASCOM-AVR

cbi PORTB,6 ;set data-out lo MOSI sbi DDRB,6 ;make it an output MOSI cbi DDRB,7 ;MISO input Ret

When you want to address multiple slaves with the software SPI you need multiple pins to select/activate the slave chip. Specify NONE for SS in that case. This also means that before every SPI command you need to set the logic level to 0 to address the chip and after the SPI command you need to set it back to a logic high level.

The hardware SPI also has this option. The NOSS parameter with a value of 1, will not set the SS line to logic 0 when the SPI operation begins. You need to set SS or any other pin of your choice to a logic 0 yourself. After the SPI command(s) are used you need to set it back to a logic 1 to deselect the slave chip.

In order to use the SPI in master mode, you need to set the SS pin to output. In input mode, this pin can be used to set the SPI bus into slave mode. You only need to set the pin to output when you use the NOSS=1 option.

All SPI routines are SPI-master routines. Example 2 below demonstrates how to create a soft SPI slave. In the samples directory you will also find a SPI hardware master and SPI hardware slave sample.

See also

SPIIN [899], SPIOUT [907], SPIINIT [907], SPI [183], SPIMOVE [907]

Example

Config SPI = SOFT, DIN = PINB.0 , DOUT = PORTB.1, SS = PORTB.2, CLOCK =
PORTB.3
Dim var As Byte
SPIINIT 'Init SPI state and pins.
SPIOUT var , 1 'send 1 byte

6.158 CONFIG SPIx

Action

Configures the SPI mode of the Xmega.

Syntax

CONFIG SPIx = HARD, MASTER = YES|NO, MODE=0-3, CLOCKDIV=div, DATA_ORDER = LSB|MSB

Remarks

SPIx	There are 4 SPI interfaces on the Xmega. You need to specify SPIC, SPID, SPIE or SPIF for SPIx. The value must be HARD.
	Selects if the SPI is running in master or slave mode. Possible values : YES, NO, 1 or 0.
	The mode of the SPI interface. There are 4 modes in the range from 0-3.

551

	The mode decides weather the first edge in a clock cycles is rising or falling, and if data setup and sample is on leading or trailing edge.		
	Mode 0 1 2 3	Leading Edge Rising, Sample Rising, Setup Falling, Sample Falling, Setup	Falling, Setup Falling, Sample Rising, Setup
CLOCKDIV	The SPI is clocked by the system clock which is divided by a the SPI divider. If you select a division factor of 4, and the system clock is 4 MHz, then the SPI clock will be 1 MHz. The possible values are : CLK2, CLK4, CLK8, CLK16, CLK32, CLK64 and CLK128. Some modes use the internal CLK2X bit.		
DATA ORDER	Selects if MSB or LSB is transferred first. The SPI can send the Least Significant bit (LSB) or the Most Significant Bit(MSB) first.		
SS	Slave select option. The possible values are : - NONE, the SS will not be set or used - AUTO, the dedicated pin is used, this is portC.4 for SPIC, portD.4 for SPID, portE.4 for SPIE and portF.4 for SPIF.		

The SPI settings for the Xmega differ from the SPI settings for normal AVR chips. In order to be able to use the four different SPI interfaces the Xmega uses a channel which you need to OPEN.

After you have opened the device, you can send/receive data using PRINT and INPUT.

The SS pin, MOSI and CLOCK pins are set to output mode automatic in master mode. The SS pin is also made high. The SS pin is only configured when you have selected SS=AUTO.

If you need to use a different pin for SS or when you need to switch the logic level yourself for SS, and thus you use the SS=NONE option, you must setup the SS pin, even if you do not use it, yourself. You must prevent that the SS pin will be made low in input mode since that will set the SPI into SLAVE mode, even while it was in MASTER mode.

When SS is in auto mode, the SS pin will be made low before each SPI transfer and be made high when the SPI transfer is finished. SS can be used when multiple slaves are used, or to synchronize data packets.

The pins are configured before the SPI control register is set. If you do not use the AUTO mode, you must set the pin direction and state yourself before using the CONFIG SPI. The following table shows which pins you have to set when NOT using the AUTO mode.

Pin	Master Mode	Slave Mode
MOSI	User set	Input
MISO	Input	User set
SCK	User set	Input
SS	User set	Input

It is very important that you set the pin direction and level BEFORE you use the CONFIG SPI statement. This because the CONFIG SPI will enable the SPI interface

and once enabled you can not change data direction/level.

If you want to change pin levels , you must disable the SPI interface first by clearing bit 6 :

Spid_ctrl.6 = 0 Config Portd.4 = Output Set Portd.0.4 Spid_ctrl.6 = 1 ' disable ' set direction ' set level ' enable

See also

INPUT 744ो, PRINT डाले, OPEN 794ो SPIIN डिम्मे , SPIOUT जिले , SPIINIT जिले , SPI ाठी , SPIMOVE जिले

Example

Dim Bspivar As Byte , Ar(4) As Byte , W As Word
Bspivar = 1
Config Spic = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk2 ,
Data_order = Msb
Config Spid = Hard , Master = Yes , Mode = 1 , Clockdiv = Clk8 ,
Data_order = Lsb
Config Spie = Hard , Master = Yes , Mode = 2 , Clockdiv = Clk4 ,
Data_order = Msb
Config Spif = Hard , Master = Yes , Mode = 3 , Clockdiv = Clk32 ,
Data_order = Msb
Open "SPIC" For Binary As #10

```
Open "SPID" For Binary As #11
Open "SPIE" For Binary As #12
Open "SPIF" For Binary As #13
Open "SPI" For Binary As #bspivar
dynamic channel
'SPI channel only suppor PRINT and INPUT
Print #10 , "to spi" ; W
Input #10 , Ar(1) , W
Print #bspivar , W
Input #bspivar , W
```

6.159 CONFIG SERVOS

Action

Configures how much servo's will be controlled.

Syntax

CONFIG SERVOS = X , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = rl , INTERVAL=t

Remarks

Х

Servo's need a variable pulse in order to operate. The CONFIG SERVOS directive will set up a byte array with the servo pulse width values and will initialize an ISR that uses TIMER0.

The number of servo's you want to control. Each used servo will use one
byte of SRAM.

' use a

553

servox	The port pin the servo is attached too.
reload	The reload value for the ISR in uS.
	The update interval. Using the interval option will result in using alternative servo code optimized for servos.

When you use for example :

Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 10 The internal ISR will execute every 10 uS.

An arrays named SERVO() will be created and it can hold 2 bytes : servo(1) and servo(2).

By setting the value of the servo() array you control how long the positive pulse will last. After it has reached this value it will be reset to 0.

The reload value should be set to 10. After 20 mS, a new pulse will be generated. You can use other reload values but it will also mean that the repeat value will change.

The PORT pins specified must be set to work as an output pin by the user. CONFIG PINB.0 = OUTPUT Will set a pin to output mode.

The CONFIG SERVOS only works with servo's that rotate 180 degrees. These are the servo's found in RC models.

There are also continuous rotation servos which work different. The servo code will NOT work on these servos.

Alternative Servocode

When using the INTERVAL option, you can use alternative code which is optimized for servo's.

You should use a RELOAD value of 100 in that case and an interval of 100 should be used for best results.

Using a reload of 100 uS will give more time to the main application. This does give lower resolution but this is not a problem for most model servos. With an interval of 100, the refresh will be done in 100x100 us which results in 10 mS. The following test code was used:

```
Config Servos = 2 , Servo1 = Portd.7 , Servo2 = Portb.1 , Reload = 100 , Interval =
100
Servo(1) = 10
Servo(2) = 5
Enable Interrupts
Do
For J = 8 To 16
Servo(1) = J
Waitms 5000 ' some time to check if the servo is stable
Next
Waitms 5000
Loop
```

Resources used

TIMER0 is used to create the ISR.

ASM

NONE

Example

```
_____
_____
'name
                         : servos.bas
'copyright
                         : (c) 1995-2005, MCS Electronics
'purpose
                         : demonstrates the SERVO option
'micro
                         : 90S2313
'suited for demo
                         : yes
'commercial addon needed : no
·_____
_____
$regfile = "2313def.dat"
                                                          ' specify
the used micro
$crystal = 4000000
                                                          ' used
crystal frequency
$baud = 19200
                                                          ' use baud
rate
$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 10
                                                          ' default
use 10 for the SW stack
$framesize = 40
                                                          ' default
use 40 for the frame space
'Servo's need a pulse in order to operate
'with the config statement CONFIG SERVOS we can specify how many servo's
we
'will use and which port pins are used
'A maximum of 14 servos might be used
'The SERVO statements use one byte for an interrupt counter and the
TTMER0
'This means that you can not use TIMER0 anymore
'The reload value specifies the interval of the timer in uS
'Config Servos = 2 , Servol = Portb.0 , Servo2 = Portb.1 , Reload = 10
Config Servos = 1 , Servol = Portb.0 , Reload = 10
'as an option you can use TIMER1
'Config Servos = 2 , Servol = Portb.0 , Servo2 = Portb.1 , Reload = 10 ,
Timer = Timer1
'we use 2 servos with 10 uS resolution(steps)
'we must configure the port pins used to act as output
Config Portb = Output
'finally we must turn on the global interrupt
Enable Interrupts
'the servo() array is created automatic. You can used it to set the
'time the servo must be on
Servo(1) = 10
                                                          '10 times 10
= 100 uS on
'Servo(2) = 20
                                                           '20 times
10 = 200 \text{ uS on}
Do
Loop
```

```
Dim I As Byte
Do
For I = 0 To 100
   Servo(1) = I
   Waitms 1000
Next
For I = 100 To 0 Step -1
' Servo(1) = I
   Waitms 1000
Next
Loop
End
```

6.160 CONFIG SUBMODE

Action

This option sets how the compiler deals with Subs, Functions and Declarations.

```
Syntax
```

CONFIG SUBMODE = NEW|OLD

Remarks

When the SUBMODE option is not configured, the default 'OLD' will be used. This is the old mode used in versions up to 2070.

This old mode demands that you DECLARE a function or sub, before you call it. It also binds in the sub/function at the same location as in your code.

When working with \$include files, this requires that you put the \$include file at the end of your code, and that you put an \$include file at the start of your code. Or that you use a GOTO to jump over the Sub/Function code.

When you use CONFIG SUBMODE=NEW, most behaviour is changed :

- there is no need to DECLARE a sub/function before you call it. But, the actual sub/ function code must be placed before the actual call!

- only the used sub/functions are included

- the sub/function code is placed after the main program. this is something you do not need to worry about.

- you can \$include the modules without a GOTO to jump over the code.

- sub/functions behave like macro's : only when used they are included

See also

DECLARE SUB 644, SUB 914, DECLARE FUNCTION 642, CALL 422

Example

```
$regfile = "m88def.dat"
$crystal = 8000000
config submode=new
declare sub test1()
required
```

' not

```
sub test2()
this sub is not used and will not be compiled
    print "test2"
end sub
function myfunc() as byte
called from test1
   myfunc = 1
end function
sub test1()
    print "test1"
    print myfunc()
uses myfunc
end sub
print "test"
test1
call test1
```

'12

1

6.161 CONFIG SYSCLOCK

Action

end

Selects the oscillator source for the system clock.

Syntax

CONFIG SYSCLOCK=sysclock , PRESCALEA=prescaleA, PRESCALEBC=prescaleBC

Remarks

SYSCLOCK	The oscillator used for generation of the system clock. This oscillator must be running. You MUST use CONFIG OSC before you use CONFIG SYSCLOCK. The CONFIG SYSCLOCK will wait till the oscillator is running stable. Possible values: - 2MHZ - 32MHZ - EXTERNAL - PLL
PRESCALEA	The Xmega has 3 prescalers. With PRESCALEA you configure the clock division of the first prescaler. Possible values: 1, 2, 4, 8, 16, 32, 64, 128,256,512
PRESCALEBC	The Xmega has 3 prescalers. With PRESCALEBC you configure the clock division of the second and the third prescaler. Possible values: - $1_1 (1 + 1 \text{ division})$

- 1_2 (1+2 division) - 4_1 (4 + 1 division) - 2_2 (2 + 2 division)
This 1_2 will make the second prescaler divide by 1 and the third prescaler divide by 2.

See also

CONFIG OSC 527

Example

Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1 ' use 32 MHz

6.162 CONFIG TCXX

Action

Configures the Xmega TIMER.

Syntax

CONFIG TCxx = wg , PRESCALE=pre, COMPAREA=ca, COMPAREB=cb, COMPAREC=cc, COMPARED=cd, EVENT_SOURCE= event, EVENT_ACTION=act, EVENT_DELAY=ed, RESOLUTION=res

Remarks

Depending on the Xmega processor of your choice, there are one or more timers. The Xmega uses the name of the port as part of the name. The first port that has a timer is portC. The first timer is named TCC0. Most timer ports have 2 timers. The next timer is named TCC1. Xmega timers are 16 bit but can be cascaded to 32 bit timers or be set to 8 but mode.

The possible timer names are : TCC0, TCC1, TCD0, TCD1, TCE0, TCE1, TCF0 and TCF1.

WG	This options sets the Timer and/or Wave Generation mode. Possible values : - NORMAL, no wave generation - FREQ , frequency generation - PWM , pulse width modulation single slope - PWM_TOP, pwm dual slope - PWM_BOT, pwm dual slope - PWM_TOPBOT, pwm dual slope - A value between 0-7 will load the mode. See table 2.
PRESCALE	The presclaler can divide the system clock that is applied to the timer. The prescaler will only divide the system clock. Possible values : - 1 , 2, 4, 8, 64, 256, 1024 - OFF, timer is disabled - E0, E1, E2, E3, E4, E5, E6, E7 . Event channel 0-7 - value between 0-15. This will write the value to the CTRLA register.

BASCOM-AVR	
COMPAREx	Where x is A, B, C, or D. This is the COMPARE or CAPTURE register
	setup. You may use either COMPARE or CAPTURE since the same registers are used. Each COMPARE/CAPTURE pin must be enabled if the input/ output pin is used. By default they are disabled. Each TCx0 timer has 4 compare registers/pins. The TCx1 timer has two capture registers/pins. Possible values :
	ENABLED : this will enable the capture/compare register DISABLED : this will disable the capture/compare register 0 : this will set the logic level of the compare output pin to 0. 1 : this will set the logic level of the compare output pin to 1.
	In FREQ and PWM modes the compare pins will be set to output mode. In CAPTURE mode, the capture pin will be set to input mode.
	The event channel source. Possible values :
E	- OFF (default)
-	- E0-E7
	- A value between 0-15
EVENT ACTIO	The event action the timer will perform. Possible values :
N	- OFF
	- CAPTURE, input capture
	- UPDOWN, external controlled up/down count
	- QDEC, quadrature decode
	- RESTART , restart waveform period
	- FREQ, frequency capture
	- PWC, pulse width capture
	Enabled, or disabled(default). When this bit is set, the selected event source is delayed by one peripheral clock cycle. This feature
	is intended for 32-bit input capture operation. Adding the event delay is necessary for compensating for the carry propagation delay that is inserted when
	cascading two counters via the Event System.
RESOLUTION	Timer resolution is 16 by default. A value of 8 will set the timer to 8 bit resolution. 32 is reserved for future use.(cascading timers)

Table 2.

Value	Mode	ТОР	UPDATE	EVENT
0	NORMAL	PER	ТОР	ТОР
1	FREQ	CCA	ТОР	ТОР
2	reserved			
3	PWM, single slope	PER	воттом	воттом
4	reserved			
5	PWM, dual slope	PER	воттом	ТОР
6	PWM, dual slope	PER	воттом	TOP and BOTTOM
7	PWM, dual slope	PER	воттом	воттом

A CONFIG TCxx statement will update the timer control registers immediately. A pre scale value other than OFF will also START the timer at once.

Example

```
(c) 1995-2010, MCS
                  xm128-TIMER-S1.bas
  This sample demonstrates the TIMER sample 1 from AVR1501
   This sample uses TIMER TCD0 since TCC0 isused for the UART
   _____
$regfile = "xm128a1def.dat"
crystal = 3200000
\$hwstack = 64
\$swstack = 64
$framesize = 64
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'First Enable The Osc Of Your Choice , make sure to enable 32 KHz clock
or use an external 32 KHz clock
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1, Databits = 8
'connect portE bit 0 and 1 to some LED
Config Porte = Output
'config timer to normal mode
Config Tcd0 = Normal , Prescale = 64
Tcd0_per = \&H30
                                                             ' period
register
Do
  If Inkey() <> 0 Then
     Tcd0_per = Tcd0_per + 100
                                                            ' increase
period
    Print "period:" ; Tcd0_per
                                                            ' vou will
see that a larger PERIOD value will cause the TIMER to overflow later
and this generating a bigger delay
 End If
 Bitwait Tcd0_intflags.0 , Set
                                                             ' wait for
overflow
 Tcd0 intflags.0 = 1
                                                             ' clear flag
by writing 1
 Toggle Porte
                                                             ' toggle led
Loop
```

6.163 CONFIG TCPIP

Action

Configures the TCP/IP W3100A chip.

Syntax

CONFIG TCPIP = int , MAC = mac , IP = ip, SUBMASK = mask, GATEWAY = gateway, LOCALPORT= port, TX= tx, RX= rx , NOINIT= 0|1 , TWI=address , Clock =

speed [, baseaddress = address] [,TimeOut=tmOut]

<u>Reillai KS</u>	emarks		
Int	The interrupt to use such as INT0 or INT1.		
	For the Easy TCP/IP PCB, use INT0.		
MAC	The MAC address you want to assign to the W3100A.		
	The MAC address is a unique number that identifies your chip. You must use a different address for every W3100A chip in your network. Example : 123.00.12.34.56.78		
	You need to specify 6 bytes that must be separated by dots. The bytes must be specified in decimal notation.		
IP	The IP address you want to assign to the W3100A.		
	The IP address must be unique for every W3100A in your network. When you have a LAN, 192.168.0.10 can be used. 192.168.0.x is used for LAN's since the address is not an assigned internet address.		
SUBMASK	The submask you want to assign to the W3100A.		
	The submask is in most cases 255.255.255.0		
GATEWAY	This is the gateway address of the W3100A.		
	The gateway address you can determine with the IPCONFIG command at the command prompt :		
	C:\>ipconfig		
	Windows 2000 IP Configuration		
	Ethernet adapter Local Area Connection 2:		
	Connection-specific DNS Suffix . : IP Address : 192.168.0.3 Subnet Mask : 255.255.255.0 Default Gateway : 192.168.0.1		
	Use 192.168.0.1 in this case.		
LOCALPORT	A word value that is assigned to the LOCAL_PORT internal variable. See also Getsocket.		
	As a default you can assign a value of 5000.		
ТХ	A byte which specifies the transmit buffer size of the W3100A. The W3100A has 4 sockets.		
	A value of 00 will assign 1024 bytes, a value of 01 will assign 2048 bytes. A value of 10 will assign 4096 bytes and a value of 11 will assign 8192 bytes.		
	This is binary notation. And the Msbits specify the size of socket 3.		
	For example, you want to assign 2048 bytes to each socket for transmission : TX = &B01010101		
	Since the transmission buffer size may be 8KB in total, you can split		

Remarks

	them up in 4 parts of 2048 bytes : 01.
	When you want to use 1 socket with 8KB size, you would use : TX = &B11. You can use only 1 socket in that case : socket 0.
RX	A byte which specifies the receive buffer size of the W3100A. The W3100A has 4 sockets.
	A value of 00 will assign 1024 bytes, a value of 01 will assign 2048 bytes. A value of 10 will assign 4096 bytes and a value of 11 will assign 8192 bytes.
	This is binary notation. And the Msbits specify the size of socket 3.
	For example, you want to assign 2048 bytes to each socket for reception : RX = &B01010101
	Since the receive buffer size may be 8KB in total, you can split them up in 4 parts of 2048 bytes : 01.
	When you want to use 1 socket with 8KB size, you would use : $RX = \&B11$. You can use only 1 socket in that case : socket 0.
	Consult the W3100A pdf for more info.
Noinit	Make this 1 when you want to configure the TCP, MAC, Subnetmask and GateWay dynamic. Noinit will only make some important settings and you need to use SETTCP in order to finish the setup.
TWI	The slave address of the W3100A/NM7010. When you specify TWI, your micro must have a TWI interface such as Mega128, Mega88, Mega32.
Clock	The clock frequency to use with the TWI interface
Baseaddress	An optional value for the chip select of the W3100A. This is default &H8000 when not specified. When you create your own board, you can override it.
TimeOut	You can specify an optional timeout when sending UDP data. The Wiznet API does wait for the CSEND status. But it means that it will block your application. In such cases, you can use the timeout value. The timeout constant is a counter which decreases every time the status is checked. When it reaches 0, it will get out of the loop. Thus a higher value will result in a longer delay. Notice that it has nothing to do with the chip timeout registers/values. Without the software timeout, the chip will also time out.

The CONFIG TCPIP statement may be used only once.

Interrupts must be enabled before you use CONFIG TCPIP.

Configuring the W3100A will init the chip.

After the CONFIG TCPIP, you can already PING the chip!

The TWI mode works only when your micro support the TWI mode. You need to have 4k7 pull up resistors.

MCS Electronics has a small adapter PCB and KIT available that can be connected easily to your microprocessor.

The new TWI mode makes your PCB design much simpler. TWI is not as fast as bus mode. While you can use every supported TCP/IP function, it will run at a lower speed.

See also

GETSOCKET [717], SOCKETCONNECT [886], SOCKETSTAT [896], TCPWRITE [924], TCPWRITESTR [925], TCPREAD [924], CLOSESOCKET [439], SOCKETLISTEN [889]

Syntax Example

Config Tcpip = Int0 , Mac = 00.00.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx = \$55 , Rx = \$55

'Now use PING at the command line to send a ping:

PING 192.168.0.8

Or use the easytcp application to ping the chip.

6.164 CONFIG TIMER0

Action

Configure TIMER0.

Syntax

```
CONFIG TIMER0 = COUNTER, EDGE=RISING/FALLING, CLEAR TIMER = 1|0 [,
CONFIGURATION=NAME]
CONFIG TIMER0 = TIMER, PRESCALE= 1|8|64|256|1024 [,CONFIGURATION=NAME]
```

Remarks

TIMER0 is a 8 bit counter. See the hardware description of TIMER0.

When configured as a COUNTER:

EDGE You can select whether the TIMER will count on the falling or rising edge.

When configured as a TIMER:

The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter.
Valid values are 1 , 8, 64, 256 or 1024

Note that some new AVR chips have different pre scale values. You can use these.

CONFIGURATION is optional. When you add configuration=mysetting, you can use this setting when you start the timer : START TIMER0 , mysetting If you have multiple settings, you can start the timer with these different settings.

Notice that the Help was written with the AT90S2313 and AT90S8515 timers in mind.

When you use the CONFIG TIMER0 statement, the mode is stored by the compiler and the TCCRO register is set.

When you use the STOP TIMER0 statement, the TIMER is stopped.

When you use the START TIMER0 statement, the TIMER TCCR0 register is loaded with the last value that was configured with the CONFIG TIMER0 statement.

So before using the <u>START</u> and <u>STOP</u> and <u></u>

Example

```
!_____
                  _____
_____
'name
                        : timer0.bas
'copyright
                        : (c) 1995-2005, MCS Electronics
                        : shows how to use TIMER0 related statements
'purpose
'micro
                        : 90S2313
'suited for demo
                        : yes
'commercial addon needed : no
_____
$regfile = "2313def.dat"
                                                          ' specify
the used micro
$crystal = 8000000
                                                          ' used
crystal frequency
$baud = 19200
                                                          ' use baud
rate
$hwstack = 32
                                                         ' default
use 32 for the hardware stack
\$swstack = 10
                                                          ' default
use 10 for the SW stack
$framesize = 40
                                                         ' default
use 40 for the frame space
'First you must configure the timer to operate as a counter or as a
timer
'Lets configure it as a COUNTER now
'You must also specify if it will count on a rising or falling edge
Config Timer0 = Counter , Edge = Rising
'Config Timer0 = Counter , Edge = falling
'unremark the line aboven to use timer0 to count on falling edge
'To get/set the value from the timer access the timer/counter register
'lets reset it to 0
Tcnt0 = 0
Do
 Print Tcnt0
Loop Until Tcnt0 >= 10
'when 10 pulses are count the loop is exited
'or use the special variable TIMER0
Timer0 = 0
'Now configire it as a TIMER
'The TIMER can have the systemclock as an input or the systemclock
divided
'by 8,64,256 or 1024
'The prescale parameter excepts 1,8,64,256 or 1024
Config Timer0 = Timer, Prescale = 1
'The TIMER is started now automaticly
```

BASCOM-AVR

564

```
'You can STOP the timer with the following statement :
Stop Timer0
'Now the timer is stopped
'To START it again in the last configured mode, use :
Start Timer0
'Again you can access the value with the tcnt0 register
Print Tcnt0
'or
Print Timer0
'when the timer overflows, a flag named TOV0 in register TIFR is set
'You can use this to execute an ISR
'To reset the flag manual in non ISR mode you must write a 1 to the bit
position
'in TIFR:
Set Tifr.1
'The following code shows how to use the TIMERO in interrupt mode
'The code is block remarked with '( en ')
' (
'Configute the timer to use the clock divided by 1024
Config Timer0 = Timer , Prescale = 1024
'Define the ISR handler
On Ovf0 Tim0_isr
'you may also use TIMERO for OVFO, it is the same
Enable Timer0
                                                              ' enable the
timer interrupt
Enable Interrupts
                                                              'allow
interrupts to occur
Do
   'your program goes here
Loop
'the following code is executed when the timer rolls over
Tim0_isr:
 Print "*";
Return
')
End
```

6.165 CONFIG TIMER1

Action

Configure TIMER1.

Syntax

```
CONFIG TIMER1 = COUNTER | TIMER | PWM ,
EDGE=RISING | FALLING , PRESCALE= 1|8|64|256|1024 ,
NOISE CANCEL=0 |1, CAPTURE EDGE = RISING | FALLING ,
CLEAR TIMER = 1|0,
COMPARE A = CLEAR | SET | TOGGLE I DISCONNECT ,
COMPARE B = CLEAR | SET | TOGGLE I DISCONNECT ,
PWM = 8 | 9 10 ,
```

COMPARE A PWM = CLEAR UP| CLEAR DOWN | DISCONNECT COMPARE B PWM = CLEAR UP| CLEAR DOWN | DISCONNECT [,CONFIGURATION=NAME]

Remarks

The TIMER1 is a 16 bit counter. See the hardware description of TIMER1. It depends on the chip if COMPARE B is available or not. Some chips even have a COMARE C.

The syntax shown above must be on one line. Not all the options need to be selected.

EDGE	You can select whether the TIMER will count on the falling or rising edge. Only for COUNTER mode.	
CAPTURE EDGE	You can choose to capture the TIMER registers to the INPUT CAPTURE registers	
	With the CAPTURE EDGE = FALLING/RISING, you can specify to capture on the falling or rising edge of pin ICP	
NOISE CANCELING	To allow noise canceling you can provide a value of 1.	
PRESCALE	The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter.	
	Valid values are 1 , 8, 64, 256 or 1024 PRESCALE can't be used in COUNTER mode.	

Here is the effect of the various options.

The TIMER1 also has two compare registers A and B

When the timer value matches a compare register, an action can be performed

COMPARE	The action can be:
A	
	SET will set the OC1X pin
	CLEAR will clear the OC1X pin
	TOGGLE will toggle the OC1X pin
	DISCONNECT will disconnect the TIMER from output pin OC1X

And the TIMER can be used in PWM mode. You have the choice between 8, 9 or 10 bit PWM mode

Also you can specify if the counter must count UP or down after a match to the compare registers

Note that there are two compare registers A and B

PWM	Can be 8, 9 or 10.
COMPARE A PWM	PWM compare mode. Can be CLEAR UP or CLEAR DOWN

Using COMPARE A, COMPARE B, COMPARE A PWM or COMPARE B PWM will set the corresponding pin for output. When this is not desired you can use the alternative NO_OUTPUT version that will not alter the output pin.

For example : COMPARE A NO_OUTPUT , COMPARE A PWM NO_OUTPUT

CONFIGURATION is optional. When you add configuration=mysetting, you can use this setting when you start the timer : START TIMER0 , mysetting If you have multiple settings, you can start the timer with these different settings.

Example

\$regfile = "8515def.dat"
the used micro
\$crystal = 4000000
crystal frequency
\$baud = 19200
rate
\$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
use 10 for the SW stack
\$framesize = 40
use 40 for the frame space

Dim W As Word

'The TIMER1 is a versatile 16 bit TIMER. 'This example shows how to configure the TIMER

'First like TIMER0 , it can be set to act as a TIMER or COUNTER 'Lets configure it as a TIMER that means that it will count and that 'the input is provided by the internal clock. 'The internal clock can be divided by 1,8,64,256 or 1024 **Config Timer1 = Timer**, Prescale = 1024

```
'You can read or write to the timer with the COUNTER1 or TIMER1 variable W = Timer1
Timer1 = W
```

```
'To use it as a COUNTER, you can choose on which edge it is trigereed

Config Timer1 = Counter , Edge = Falling

'Config Timer1 = Counter , Edge = Rising
```

'Also you can choose to capture the TIMER registers to the INPUT CAPTURE
registers
'With the CAPTURE EDGE = , you can specify to capture on the falling or
rising edge of
'pin ICP
Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling
'Config Timer1 = Counter , Edge = Falling , Capture Edge = Rising

'To allow noise canceling you can also provide :

' specify

' use baud

' default

' default

' default

' used

567

```
Config Timer1 = Counter, Edge = Falling, Capture Edge = Falling,
Noise Cancel = 1
'to read the input capture register :
W = Capture1
'to write to the capture register :
Capture1 = W
'The TIMER also has two compare registers A and B
'When the timer value matches a compare register, an action can be
performed
Config Timer1 = Counter, Edge = Falling, Compare A = Set, Compare B =
Toggle , Clear Timer = 1
'SET , will set the OC1X pin
'CLEAR, will clear the OC1X pin
'TOGGLE, will toggle the OC1X pin
'DISCONNECT, will disconnect the TIMER from output pin OC1X
'CLEAR TIMER will clear the timer on a compare A match
'To read write the compare registers, you can use the COMPARE1A and
COMPARE1B variables
Comparela = W
W = Comparela
'And the TIMER can be used in PWM mode
'You have the choice between 8,9 or 10 bit PWM mode
'Also you can specify if the counter must count UP or down after a match
'to the compare registers
'Note that there are two compare registers A and B
Config Timer1 = Pwm , Pwm = 8 , Compare A Pwm = Clear Up , Compare B
Pwm = Clear Down , Prescale = 1
'to set the PWM registers, just assign a value to the compare A and B
registers
Comparela = 100
Compare1b = 200
'Or for better reading :
\mathbf{Pwm1a} = 100
Pwmlb = 200
End
```

6.166 CONFIG TIMER2

Action

Configure TIMER2.

Syntax for the 8535

CONFIG TIMER2 = TIMER | PWM , ASYNC=ON |OFF, PRESCALE = 1 | 8 | 32 | 64 | 128 | 256 | 1024 , COMPARE = CLEAR | SET | TOGGLE I DISCONNECT , PWM = ON | OFF , COMPARE PWM = CLEAR UP| CLEAR DOWN | DISCONNECT , CLEAR TIMER = 1|0 [,CONFIGURATION=NAME]

Syntax for the M103

CONFIG TIMER2 = COUNTER| TIMER | PWM , EDGE= FALLING |RISING, PRESCALE = 1 | 8 | 64 | 256 | 1024 , COMPARE = CLEAR | SET | TOGGLE I DISCONNECT , PWM = ON | OFF , COMPARE PWM = CLEAR UP| CLEAR DOWN | DISCONNECT , CLEAR TIMER = 1|0 [,CONFIGURATION=NAME]

Remarks

The TIMER2 is an 8 bit counter. It depends on the chip if it can work as a counter or not. The syntax shown above must be on one line. Not all the options need to be selected.

Here is the effect of the various options.

You can select whether the TIMER will count on the falling or rising
edge. Only for COUNTER mode.

PRESCALE	The TIMER is connected to the system clock in this case. You can select the division of the system clock with this parameter.
	Valid values are 1 , 8, 64, 256 or 1024 or
	1 , 8, 32 , 64 , 256 or 1024 for the M103
	Prescale can not be used in COUNTER mode.

The TIMER2 also has a compare registers

When the timer value matches a compare register, an action can be performed

COMPARE	The action can be:
	SET will set the OC2 pin CLEAR will clear the OC2 pin TOGGLE will toggle the OC2 pin DISCONNECT will disconnect the TIMER from output pin OC2

And the TIMER can be used in 8 bit PWM mode

You can specify if the counter must count UP or down after a match to the compare registers

COMPARE PWM PWM compare mode. Can be CLEAR UP or CLEAR DOWN
--

CONFIGURATION is optional. When you add configuration=mysetting, you can use this setting when you start the timer : START TIMER0 , mysetting If you have multiple settings, you can start the timer with these different settings.

Example

Dim W As Byte Config Timer2 = Timer , ASYNC = 1 , Prescale = 128 On TIMER2 Myisr ENABLE INTERRUPTS ENABLE TIMER2 DO

LOOP

MYISR: 'get here every second with a 32768 Hz xtal RETURN

'You can read or write to the timer with the COUNTER2 or TIMER2 variable W = Timer2Timer2 = W

6.167 CONFIG TWI

Action

Configure the TWI (two wire serial interface).

Syntax

CONFIG TWI = clockspeed

CONFIG TWIC | TWID | TWIE | TWIF = clockspeed

Remarks

clockspeed The desired clock frequency for SCL

CONFIG TWI will set TWSR pre scaler bits 0 and 1, and TWBR depending on the used <u>\$CRYSTAL</u> [316] frequency and the desired SCL clock speed. Typical you need a speed of 400 KHz. Some devices will work on 100 KHz as well.

When TWI is used in SLAVE mode, you need to have a faster clock speed as the master.

XMEGA

The XMEGA can contain up to 4 TWI units. When not specifying TWIC, TWID, TWIE or TWIF, the **TWIC** will be used as the default.

You MUST dimension a variable named TWI_START as a byte. It is used by the xmega TWI library code. Without it, you will get an error.

It is important that you specify the proper crystal frequency. Otherwise it will result in a wrong TWI clock frequency.

See also

\$CRYSTAL 316 , OPEN 794

Example

' (c) 2004 MCS Electronics ' This demo shows an example of the TWI ' Not all AVR chips have TWI (hardware I2C)

'The chip will work in TWI/I2C master mode 'Connected is a PCF8574A 8-bits port extender

\$regfile="M8def.dat"' the used chip \$crystal= 4000000 ' frequency used \$baud= 19200 ' baud rate

\$lib"i2c_twi.lbx"' we do not use software emulated I2C but the TWI

Config Scl =Portc.5 ' we need to provide the SCL pin name Config Sda =Portc.4 ' we need to provide the SDA pin name

'On the Mega8, On the PCF8574A 'scl=PC5 , pin 28 pin 14 'sda=PC4 , pin 27 pin 15

I2cinit' we need to set the pins in the proper state

Config Twi = 100000 ' wanted clock frequency 'will set TWBR and TWSR 'Twbr = 12 'bit rate register 'Twsr = 0 'pre scaler bits

Dim B AsByte, X AsByte Print"TWI master" Do Incr B ' increase value I2csend&B01110000 , B ' send the value Print"Error : ";Err' show error status I2creceive&B01110000 , X ' get a byte Print X ;" ";Err' show error Waitms 500 'wait a bit Loop End

6.168 CONFIG TWISLAVE

Action

Configure the TWI Slave address and bit rate

Syntax

CONFIG TWISLAVE = address , BTR = value , BITRATE = value , SAVE=option [, GENCALL=value] [,USERACK=ack]

Remarks

Address	The slave address that is assigned to the slave chip. This must be an Even number. Bit 0 of the address is used to activate the general call address. The GENCAL option will set this bit automatic. I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C 7-bit address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the
	data sheets which are in the same format in most cases. So if you work with 7 bit address, you need to multiply the address by 2.
BTR	Bytes to receive. With this constant you specify how many bytes will be expected when the master reads data from the slave. And thus how many bytes will be sent to the master.
Bit rate	This is the I2C/TWI clock frequency. Most chips support 400 KHz (400000) but all I2C chips support 100000.
SAVE	SAVE = NOSAVE : this can be used when you do not change a lot of registers in the interrupt. SAVE = SAVE : this is best to be used when you do not use ASM in the TWI interrupt. See the explanation below. When you do not specify SAVE, the default will be SAVE=SAVE.
GENCALL	General call address activated or not. When you specify 1 or YES, the General call address will be activated which mean that the slave will respond not only to it's own address, but also to the general call address 0. When you omit the option or specify 0 or NO, the general call address will not be honored.
USERACK	Default is OFF. When you use ON, an alternative library will be used. This library will create a variable named TWI_ACK. Each time your code is called this variable is filled with the value 255. If you do not alter the value, the slave will send an ACK as it is supposed to. If you reset the value to 0, the slave will send a NACK. You can use this to send data with variable length to the slave. In this case, BTR only serves as an index. You must make sure to reset TWI_ACK when you have send the last byte to the master.

The variables $\, {\rm Twi}$, ${\rm Twi}_{\rm b}{\rm tr}$ and ${\rm Twi}_{\rm b}{\rm tw}$ are created by the compiler. These are all bytes

The TWI interrupt is enabled but you need to enabled the global interrupt

The TWI Slave code is running as an interrupt process. Each time there is a TWI interrupt some slave code is executed. Your BASIC code is called from the low level

572 BASCOM-AVR

slave code under a number of events. You must include all these labels in your Slave application. You do not need to write code in all these sub routines.

Label	Event
Twi_stop_rstart_received	The Master sent a stop(i2CSTOP) or repeated start. Typical you do not need to do anything here.
Twi_addressed_goread	The master has addressed the slave and will now continue to send data to the slave. You do not need to take action here.
Twi_addressed_gowrite	The master has addressed the slave and will now continue to receive data from the slave. You do not need to take action here.
Twi_gotdata	The master has sent data. The variable TWI holds the received value. The byte TWI_BTW is an index that holds the value of the number of received bytes. The first received byte will have an index value of 1.
Twi_master_needs_byte	The master reads from the slave and needs a value. The variable TWI_BTR can be inspected to see which index byte was needed. With the CONFIG BTR , you specify how many bytes the master will read.

In most cases your main application is just an empty DO LOOP. But when you write a slave that performs other tasks on the background these other tasks are interrupted by the TWI traffic.

Take in mind that the interrupt with the lowest address has the highest priority. So do NOT write blocking code inside an interrupt. While servicing another interrupt, the TWI interrupt can not be serviced.

The TWI Slave code will save all used registers.

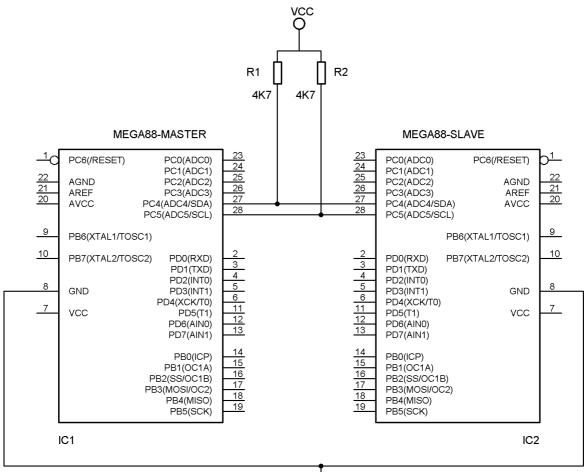
But since it will call your BASIC application when the TWI interrupt occurs, your BASIC code could be in the middle of say a PRINT statement.

When you then execute another PRINT statement , you will destroy registers.

So keep the code in the sub routines to a minimum, and use SAVE option to save all registers. This is the default.

While two printing commands will give odd results (print 12345 and 456 in the middle of the first print will give 1234545) at least no register is destroyed.

A typical configuration is shown below.



GND

To test the above hardware, use the samples : twi-master.bas and twi-slave.bas Optional you can use i2cscan.bas to test the general call address.

When you want to change the address of the slave at run time you need to write to the **TWAR** register.

The TWAR register contains the slave address. Bit 0 which is used to indicate a read or write transaction should be cleared. When you set it, the slave will also recognize the general call address. The GENCALL option just sets bit 0 of the slave.

See also

CONFIG TWI 569 , CONFIG SCL 539 , CONFIG SDA 538

ASM NONE

Example1(master)

1	(c) 2004 MCS Electronics
1	This demo shows an example of the TWI
1	Not all AVR chips have TWI (hardware I2C)

574 BASCOM-AVR

```
'The chip will work in TWI/I2C master mode
'Connected is a PCF8574A 8-bits port extender
$regfile = "M88def.dat"
                                                              ' the used chip
scrystal = 8000000
                                                              frequency used
$baud = 19200
                                                              ' baud rate
$lib "i2c_twi.lbx"
                                                              ' we do not use softwar
Config Scl = Portc.5
                                                              ' we need to provide the
Config Sda = Portc.4
                                                              ' we need to provide the
                        On the PCF8574A
'On the Mega88,
'scl=PC5 , pin 28
                        pin 14
'sda=PC4 , pin 27
                            pin 15
                                                              ' we need to set the part
I2cinit
Config Twi = 100000
                                                              ' wanted clock frequence
'will set TWBR and TWSR
'Twbr = 12
                                                               'bit rate register
'Twsr = 0
                                                              'pre scaler bits
Dim B As Byte , X As Byte
Print "TWI master"
Do
  Incr B
                                                              ' increase value
  I2csend &H0 , B
                                                              ' send the value to gen
 I2csend &H70 , B
                                                              ' send the value
 Print "Error : " ; Err
                                                              ' show error status
 I2creceive &H70 , X
                                                              ' get a byte
                                                              ' show error
 Print X ; " " ; Err
 Waitms 500
                                                             'wait a bit
Loop
```

```
Example2(slave)
```

End

' (c) 2004 MCS Electronics ' This demo shows an example of the TWI in SLAVE mode Not all AVR chips have TWI (hardware I2C) ' IMPORTANT : this example ONLY works when you have the TWI slave library which is a commercial add on library, not part of BASCOM 'Use this sample in combination with i2cscan.bas and/or twi-master.bas ' **\$regfile = "M88def.dat"** ' the chip we use **\$crystal = 800000** ' crystal oscillator va **\$baud = 19200** ' baud rate

Print "MCS Electronics TWI-slave demo"

Config Twislave = &H70 , Btr = 1 , Bitrate = 100000 , Gencall = 1
'In i2c the address has 7 bits. The LS bit is used to indicate read or write
'When the bit is 0, it means a write and a 1 means a read
'When you address a slave with the master in bascom, the LS bit will be set/reset a
'The TWAR register in the AVR is 8 bit with the slave address also in the most left
'This means that when you setup the slave address as &H70, TWAR will be set to &H00
'And in the master you address the slave with address &H70 too.
'The AVR TWI can also recognize the general call address 0. You need to either set

```
'by using &H71 as a slave address, or by using GENCALL=1
'as you might need other interrupts as well, you need to enable them all manual
Enable Interrupts
'this is just an empty loop but you could perform other tasks there
Do
  nop
Loop
End
'A master can send or receive bytes.
'A master protocol can also send some bytes, then receive some bytes
'The master and slave must match.
'the following labels are called from the library
Twi_stop_rstart_received:
 Print "Master sent stop or repeated start"
Return
Twi_addressed_goread:
  Print "We were addressed and master will send data"
Return
Twi_addressed_gowrite:
  Print "We were addressed and master will read data"
Return
'this label is called when the master sends data and the slave has received the byt
'the variable TWI holds the received value
Twi_gotdata:
   Print "received : " ; Twi
Return
'this label is called when the master receives data and needs a byte
'the variable twi_btr is a byte variable that holds the index of the needed byte
'so when sending multiple bytes from an array, twi_btr can be used for the index
Twi_master_needs_byte:
  Print "Master needs byte : " ; Twi_btr
  Twi = 65
                                                             ' twi must be filled wa
Return
'when the mast has all bytes received this label will be called
Twi_master_need_nomore_byte:
  Print "Master does not need anymore bytes"
Return
```

6.169 CONFIG USB

Action

Create settings related to USB.

Syntax

```
CONFIG USB = dev, Language= lang, Manufact= "man", Product="prod", Serial="serial"
```

Remarks

	The possible options are Device and Host. Host is not supported yet.
Lang	A language identifier. &H0409 for US/English
Man	A string constant with the manufacture name.
Prod	A string constant with the product name.
Serial	A string constant with the serial number.

The above settings determine how your device is displayed by the operating system. Since these settings end up in flash code space, it is best to chose short names. There is no limit to the length other then the USB specifications impose, but keep it short as possible. Strings in USB are UNI coded. Which mean that a word is used for each character. with normal ASCII coding, only a byte is used for each character.

For a commercial USB device you need to give it a unique VID & PID combination. When you plan to use it at home, this is not needed.

You can buy a Vendor ID (VID) from the USB organization. This cost 2000 \$. As a service MCS offers a PID in the on line shop. This cost little and it gives you a unique Product ID(PID) but with the MCS Electronics VID.

Notice that using CONFIG USB will include a file named **USBINC.BAS**. This file is not part of the BASCOM setup/distribution. It is available as a commercial add on. The add on package includes 3 samples , the include file, and a special activeX for the HID demo.

None of the samples require a driver. A small UB162 module with normal pins is available from the on line shop too.

The first supported USB devices are USB1287, USB162.

See also

NONE

Example

```
$regfile = "usb162.dat"
scrystal = 8000000
$baud = 19200
Const Mdbg = 1
Config Clockdiv = 1
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Const Vendor_id = &H16D0
                                                             ' MCS Vendor
TD
                                                             ' MCS
Const Product_id = &H201D
product ID, you can buy a VID&PID in the MCS shop
Const Ep_control_length = 32
Const User_conf_size = 41
Const Size_of_report = 53
Const Device_class = 0
```

```
Const Device sub class = 0
Const Device protocol = 0
Const Release number = &H1000
Const Length_of_report_in = 8
Const Length_of_report_out = 8
Const Interface_nb = 0
Const Alternate = 0
Const Nb_endpoint = 2
                                                              ' HTD
Const Interface_class = 3
Const Interface_sub_class = 0
Const Interface_protocol = 0
Const Interface_index = 0
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Print "USB GENERIC test'
Declare Sub Usb_user_endpoint_init
Declare Sub Hid_test_hit()
Declare Sub Hid_task()
Declare Sub Hid_task_init()
Const Usb_config_attributes_reserved = &H80
Const Usb_config_buspowered = Usb_config_attributes_reserved
Const Usb_config_selfpowered = Usb_config_attributes_reserved Or &H40
Const Usb_config_remotewakeup = Usb_config_attributes_reserved Or &H20
Const Nb_interface = 1
Const Conf_nb = 1
Const Conf_index = 0
Const Conf_attributes = Usb_config_buspowered
Const Max_power = 50
                                                              ' 100 mA
Const Interface_nb_mouse = 0
Const Alternate_mouse = 0
Const Nb_endpoint_mouse = 1
                                                              ' HID Class
Const Interface_class_mouse = 3
                                                              ' Sub Class
Const Interface_sub_class_mouse = 1
is Mouse
                                                              ' Mouse
Const Interface protocol mouse = 2
Const Interface_index_mouse = 0
Const Nb_endpoints = 2
                                                              ' number of
endpoints in the application including control endpoint
Const Ep_kbd_in = 1
                                                              ' Number of
the mouse interrupt IN endpoint
Const Ep_hid_in = 1
Const Ep_hid_out = 2
Const Endpoint_nb_1 = Ep_hid_in Or &H80
Const Ep_attributes_1 = 3
                                                              ' BIILK =
0 \times 02, INTERUPT = 0 \times 03
Const Ep_in_length_1 = 8
Const Ep_size_1 = Ep_in_length_1
Const Ep interval 1 = 20
                                                              ' Interrupt
polling interval from host
Const Endpoint_nb_2 = Ep_hid_out
Const Ep_attributes_2 = 3
                                                              ' BULK =
0 \times 02, INTERUPT = 0 \times 03
Const Ep_out_length = 8
Const Ep_size_2 = Ep_out_length
```

578 BASCOM-AVR

```
Const Ep_interval_2 = 20
                                                             ' interrupt
polling from host
Config Usb = Device , Language = \&H0409 , Manufact = "MCS" , Product =
"MCSHID162" , Serial = "MC0001"
'Dim some user vars
Dim Usb_kbd_state As Byte , Usb_key As Byte , Usb_data_to_send As Byte
Dim Dummy As Byte , Dummy1 As Byte , Dummy2 As Byte
Print "task init"
Usb task init
Hid_task_init
Do
  Usb_task
  Hid task
  'you can call your sub program here
Loop
'nothing needed to init
Sub Hid_task_init()
  'nothing
end sub
'HID task must be checked regular
Sub Hid_task()
    If Usb_connected = 1 Then
                                                             ' Check USB
HID is enumerated
       Usb_select_endpoint Ep_hid_out
                                                             ' Get Data
Repport From Host
       If Ueintx.rxouti = 1 Then
Is_usb_receive_out())
          Dummy1 = Uedatx : Print "Got : " ; Dummy1
          Dummy2 = Uedatx : Print "Got : " ; Dummy2
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Dummy = Uedatx : Print "Got : " ; Dummy
          Usb_ack_receive_out
      End If
      If Dummy1 = &H55 And Dummy2 = &HAA Then
                                                             ' Check if
we received DFU mode command from host
                                                             ' Detach
         Usb_detach
Actual Generic Hid Application
         Waitms 500
         Goto &H1800
                                                             'goto
bootloader
         'here you could call the bootloader then
      End If
      Usb_select_endpoint Ep_hid_in
                                                             ' Ready to
send these information to the host application
      If Ueintx.txini = 1 Then
Is_usb_in_ready())
         Uedatx = 1
         Uedatx = 2
         Uedatx = 3
         Uedatx = 4
```

```
Uedatx = 5
         Uedatx = 6
         Uedatx = 7
         Uedatx = 8
                                                              ' Send data
         Usb_ack_fifocon
over the USB
      End If
    End If
End Sub
Function Usb_user_read_request(type As Byte , Request As Byte) As Byte
  #if Mdbq
   Print "USB USER READ REQ"
  #endif
  Usb_string_type = Uedatx
'Usb_read_byte();
 Usb_descriptor_type = Uedatx
'Usb_read_byte();
  Usb_user_read_request = 0
  Select Case Request
     Case Get descriptor:
        Select Case Usb_descriptor_type
           Case Report : Call Hid_get_report()
                         Usb_user_read_request = 1
           Case Hid : Call Hid get hid descriptor()
                      Usb_user_read_request = 1
           Case Else
              Usb_user_read_request = 0
        End Select
    Case Set_configuration:
        Select Case Usb_descriptor_type
         Case Set_report : Call Hid_set_report()
                Usb_user_read_request = 1
         Case Else
               Usb_user_read_request = 0
        End Select
   Case Get_interface:
        1//
                 usb_hid_set_idle();
      Call Usb_hid_get_interface()
      Usb_user_read_request = 1
   Case Else
     Usb_user_read_request = 0
 End Select
End Function
'usb_init_device.
'This function initializes the USB device controller and
'configures the Default Control Endpoint.
Sub Usb_init_device()
  #if Usbfunc
     Usb_select_device
   #endif
   #if Usbfunc
  If Usbsta.id = 1 Then
                                                              'is it an
USB device?
   #endif
      Uenum = Ep_control
                                                              ' select USB
endpoint
      If Ueconx.epen = 0 Then
                                                              ' usb
endpoint not enabled yet
         Call Usb_configure_endpoint(ep_control, Type_control,
Direction_out , Size_32 , One_bank , Nyet_disabled)
      End If
```

```
#if Usbfunc
End If
#endif
End Sub
```

Sub Usb_user_endpoint_init(byval Nm As Byte)
Call Usb_configure_endpoint(ep_hid_in , Type_interrupt , Direction_in
, Size_8 , One_bank , Nyet_enabled)
Call Usb_configure_endpoint(ep_hid_out , Type_interrupt ,
Direction_out , Size_8 , One_bank , Nyet_enabled)
End Sub

Usb_dev_desc: Data 18 , Device_descriptor 'size and device descriptor **Data** 0 , 2 'Usb_write_word_enum_struc(USB_SPECIFICATION) Data Device_class , Device_sub_class DEVICE_CLASS and DEVICE_SUB_CLASS Data Device_protocol , Ep_control_length ' device protol and ep_control_length Data Vendor id% Usb_write_word_enum_struc(VENDOR_ID) Data Product_id% Usb_write_word_enum_struc(PRODUCT_ID) τ. Data Release_number% Usb write word enum struc(RELEASE NUMBER) ' MAN INDEX Data Man_index , Prod_index and PROD INDEX **Data** Sn_index , Nb_configuration ' SN INDEX and NB_CONFIGURATION

Usb_conf_desc: Data 9 , Configuration_descriptor ' length , CONFIGURATION descriptor Data User_conf_size% ' total length of data returned Data Nb_interface , Conf_nb ' number of interfaces for this conf. , value for SetConfiguration resquest Data Conf_index , Conf_attributes ' index of string descriptor , Configuration characteristics ' maximum Data Max_power power consumption

Data 9 , Interface_descriptor 'length , INTERFACE descriptor type Data Interface_nb , Alternate 'Number of interface , value to select alternate setting Data Nb_endpoint , Interface_class 'Number of EP except EP 0 ,Class code assigned by the USB Data Interface_sub_class , Interface_protocol 'Sub-class code assigned by the USB , Protocol code assigned by the USB Data Interface_index 'Index Of String Descriptor

Data 9, Hid_descriptor HID descriptor type 'length ,

BASCOM Language Reference 581

Data Hid bdc% , 8 ' Binay Coded Decimal Spec. release , Hid_country_code Data Hid_class_desc_nb , Hid_descriptor_type 'Number of HID class descriptors to follow , Report descriptor type Data Size_of_report% 'HTD KEYBOARD LENGTH Data 7 , Endpoint descriptor ' Size Of This Descriptor In Bytes , ENDPOINT descriptor type Data Endpoint_nb_1 , Ep_attributes_1 ' Address of the endpoint , Endpoint's attributes Data Ep_size_18 ' Maximum packet size for this EP , Interval for polling EP in ms Data Ep interval 1 ' Size Of Data 7 , Endpoint_descriptor This Descriptor In Bytes , ENDPOINT descriptor type Data Endpoint_nb_2 , Ep_attributes_2 ' Address of the endpoint , Endpoint's attributes **Data** Ep_size_2% ' Maximum packet size for this EP **Data** Ep_interval_2 ' Interval for polling EP in ms Usb_hid_report: Data &HO6 , &HFF , &HFF 04 2 Usage Page (vendordefined?) ' 08|1 **Data** &H09 , &H01 Usage (vendordefined Data &HA1 , &H01 ' A0 1 , Collection (Application) // IN report Data $\& \rm H09$, $\& \rm H02$ ' 08 1 , Usage (vendordefined) **Data &**H09 , **&**H03 ' 08|1 Usage (vendordefined) **Data** &H15 , &H00 ' 14|1 Logical Minimum(0 for signed byte?) Data &H26 , &HFF , &H00 ' 24 1 Logical Maximum(255 for signed byte?) **Data** &H75 , &H08 ' 74 1 Report Size(8) = field size in bits = 1 byte Data &H95 , Length_of_report_in 94 1:ReportCount(size) = repeat count of previous item Data &H81 , &H02 ' 80 1: IN report (Data, Variable, Absolute) // OUT report Data $\& \rm H09$, $\& \rm H04$ ' 08 1 Usage (vendordefined) Data $\& \rm H09$, $\& \rm H05$ ' 08 1 Usage (vendordefined) **Data** &H15 , &H00 ' 14 1 Logical Minimum(0 for signed byte?) Data &H26 , &HFF , &H00 24 1 Logical Maximum(255 for signed byte?) ' 74|1 **Data** & H75 , & H08 , Report Size(8) = field size in bits = 1 byte Data &H95 , Length_of_report_out 94 1: ReportCount(size) = repeat count of previous item ' 90|1: OUT Data &H91 , &H02 report (Data,Variable, Absolute) // Feature report

582 BASCOM-AVR

Data &H09 , &H06	1	08 1	,
Usage (vendordefined)			
Data &H09 , &H07	1	08 1	,
Usage (vendordefined)			
Data &H15 , &H00	1	14 1	,
LogicalMinimum(0 for signed byte)			
Data &H26 , &HFF , &H00	1	24 1	,
Logical Maximum(255 for signed byte)			
Data &H75 , &H08	1	74 1	,
Report Size(8) =field size in bits = 1 byte			
Data &H95 , &H04	1		
94 1: ReportCount			
Data &HB1 , &H02	1	B0 1:	
Feature report			
Data &HCO	1	C0 0	,
End Collection			

6.170 CONFIG VPORT

Action

Maps an XMEGA port to a virtual port.

Syntax

CONFIG VPORT0 = port [, VPORT1=port, VPORT2=port, VPORT3=port]

Remarks

There are 4 virtual port registers. When setting up these registers, you need to use VPORTx, where X is 0,1,2 or 3, indicating the virtual port. The virtual port itself is accesed via it's registers PORTy, PINy and DDRy where Y is a 0,1,2 or 3. The normal ports have named like PORTA, PORTB, etc. A virtual port will access the same port but using a different register.
The last letter of the real port name. For example A for PORTA, B for PORTB, C for PORTC etc.

All ports in the Xmega are located in the extended address area. This space can only be accessed with instructions like LDS,STS, LD and ST. Special bit instructions only work on the lower IO-registers.

Xmega example :

again: Lds r24, PINA ; read port input value sbrs r24,7 ; skip next instruction if bit 7 is set (1) rjmp again ; try again

Now the same code for a normal AVR again: sbis PINA,7 ; skip if pina.7 is set rjmp again

Not only less code is required, but the LDS takes 3 cycles

With the virtual mapping, you can access any PORT register (PORT,PIN and DDR) via it's virtual name PORT0, PIN0 or DDR0. Since there are 4 virtual mapping registers, you can define PORT0, PORT1, PORT2

and PORT3.

When you write to PORT**n**, the compiler can use the smaller/quicker code.

Devices like graphical LCD can benefit from this.

See Also

CONFIG PORT 528

Example

```
(c) 1995-2010, MCS
           Mapping Real Ports to Virtual Ports.bas
  This sample demonstrates mapping ports to virtual ports
  based on MAK3's sample
·_____
$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
\$swstack = 40
$framesize = 40
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Print "Map VPorts"
'map portD to virtual port0, map portE to virtual port1, map portC to
virtual port2
'map portR to virtual port 3
Config VPort0 = D , VPort1 = E , VPort2 = C , VPort3 = R
'Each virtual port is available as PORT0, PORT1, PORT2 and PORT3
       data direct is available as DDR0 , DDR1, DDR2 and DDR3
        PIN input is available as PIN0 , PIN1, PIN2 and PIN3
'The advantage of virtual port registers is that shorter asm instruction
can be used which also use only 1 cycle
Dim Var As Byte
'Real Port Direction
Ddr1 = \&B0000_{000}
                                                           ' Port E =
TNDUT
Ddr0 = &B1111_111
                                                            ' Port D =
OUTPUT
```

'Continously copy the value from PORTE to PORTD using the virtual ports.

```
Do
Var = Pin1
Virtual Port 0
Port0 = Var
Virtual Port 1
Loop
End
```

'Read

'Write

'end program

6.171 CONFIG WAITSUART

Action

Compiler directive that specifies that software UART waits after sending the last byte.

Syntax

CONFIG WAITSUART = value

Remarks

value	A numeric value in the range of 1-255.
	A higher value means a longer delay in mS.

When the software UART routine are used in combination with serial LCD displays it can be convenient to specify a delay so the display can process the data.

See also

OPEN 794

Example

See <u>OPEN</u> 794 example for more details.

6.172 CONFIG WATCHDOG

Action

Configures the watchdog timer.

Syntax

CONFIG WATCHDOG = time

Remarks

Time	The interval constant in mS the watchdog timer will count to before it will reset your program.
	Possible settings : 16 , 32, 64 , 128 , 256 , 512 , 1024 and 2048. Some newer chips : 4096, 8192.
	The XMEGA has a 1 KHz clocked watchdog. For Xmega the following

value in millisecond need to be used :
8 ,16,32,64,125,250,500,1000,2000,4000,8000 So 2000 will sets a timeout of 2 seconds.
So 2000 will sets a timeout of 2 seconds.

Note that some new AVR's might have additional reset values such as 4096 and 8192.

When the WD is started, a reset will occur after the specified number of mS. With 2048, a reset will occur after 2 seconds, so you need to reset the WD in your programs periodically with the RESET WATCHDOG statement.

Some AVR's might have the WD timer enabled by default. You can change this with the Fuse Bits.

After the CONFIG WATCHDOG statement, the watchdog timer is disabled. You can also use CONFIG WATCHDOG to change the time out value. This will stop the watchdog timer and load the new value.

After a CONFIG WATCHDOG, you always need to start the Watchdog with the START WATCHDOG statement.

Most new AVR chips have an MCUSR register that contains some flags. One of the flags is the WDRF bit. This bit is set when the chip was reset by a Watchdog overflow. The CONFIG WATCHDOG will clear this bit, providing that the register and bit is available in the micro.

When it is important to examine at startup if the micro was reset by a Watchdog overflow, you need to examine this MCUSR.WDRF flag before you use CONFIG WATCHDOG, since that will clear the flag.

For chips that have an enhanced WD timer, the WD timer is cleared as part of the chip initialize procedure. This because otherwise the WD timer will only work once. If it is important that you know the cause of the reset, you can read the register R0 before you run other code.

The sample below demonstrates how to store the WDRF bit if you need it, and print it later.

See also

<u>START WATCHDOG</u> बिभी, <u>STOP WATCHDOG</u> बिभी, <u>RESET WATCHDOG</u> बिभी

Example

\$swstack = 32

\$framesize = 40

· '	
'name 'copyright 'purpose 'micro 'suited for demo 'commercial addon needed '	<pre>watchd.bas (c) 1995-2008, MCS Electronics demonstrates the watchdog timer Mega88 yes no</pre>
<pre>\$regfile = "m88def.dat" \$crystal = 8000000 \$baud = 19200 \$hwstack = 32</pre>	' specify the used mic ' used crystal frequen ' use baud rate ' default use 32 for t

default use 32 for th

' default use 40 for th

```
Dim B As Byte
Dim Wdbit As Bit
Dim bWD As Byte
bWD=peek(0)
                                                              ' read the wd flag
Print "Watchdog test"
If bwd.wdrf = 1 Then
                                                              ' there was a WD overf
   Wdbit = 1
                                                              'store the flag
End If
                                                              'reset after 2048 mSec
Config Watchdog = 2048
                                                              'just print it now sind
If Wdbit = 1 Then
    Print "Micro was reset by Watchdog overflow"
End If
Start Watchdog
                                                              'start the watchdog tir
Dim I As Word
For I = 1 To 1000
  Waitms 100
  Print I
                                                              'print value
  B = Inkey()
                                                              ' get a key from the se
                                                              'letter A pressed
  If B = 65 Then
                                                              ' test if the WD will s
     Stop Watchdog
                                                              'letter B pressed
  Elseif B = 66 Then
     Config Watchdog = 4096
                                                              'reconfig to 4 sec
                                                              'CONFIG WATCHDOG will o
     Start Watchdog
  Elseif B = 67 Then
                                                              'C pressed
     Config Watchdog = 8192
                                                              ' some have 8 sec times
     'observe that the WD timer is OFF
  Elseif B = 68 Then
                                                              'D pressed
     Start Watchdog
                                                              ' start it
  End If
  'Reset Watchdog
  'you will notice that the for next doesnt finish because of the reset
  'when you unmark the RESET WATCHDOG statement it will finish because the
  'wd-timer is reset before it reaches 2048 msec
  'When you press 'A' you will see that the WD will stop
  'When you press 'B' you will see that the WD will time out after 4 Sec
  'When you press 'C' you will see the WD will stop
  'When you press 'D' you will see the WD will start again timing out after 8 secs
Next
End
```

```
And this shows how to read the register r0:

Dim Breset As Byte

Breset = Peek(0)

When you show this value on an LCD display you will see a value of 7 the first time, and later a va
```

Xmega Sample

```
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'First Enable The Osc Of Your Choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Config Input1 = Cr , Echo = Crlf
                                                             ' CR is used
for input, we echo back CR and LF
Open "COM1:" For Binary As #1
       ^^^^ change from COM1-COM8
Print #1 , "Xmega revision:" ; Mcu_revid
                                                             ' make sure
it is 7 or higher !!! lower revs have many flaws
Config Watchdog = 4000
                                                             'after 4
seconds a reset will occur if the watchdog is enabled
'possible value : 8 ,16,32,64,125,250,500,1000,2000,4000,8000
'these values are clock cycles, based on a 1 KHz clock !!!
Dim W As Word , B As Byte
Do
  W = W + 1
 Print W
 Waitms 500
 B = Inkey()
  If B = "a" Then
    Start Watchdog
    Print "start"
  Elseif B = "b" Then
    Stop Watchdog
    Print "stop"
  Elseif B = "c" Then
    Config Watchdog = 8000
    Print "8 sec"
  Elseif B = "d" Then
   Reset Watchdog
   Print "reset'
 End If
Loop
```

6.173 CONFIG X10

Action

Configures the pins used for X10.

Syntax

CONFIG X10 = pinZC , TX = portpin

Remarks

PinZC The pin that is connected to the zero cross output of the TW-523. This is

588 BASCOM-AVR

	a pin that will be used as INPUT.
Portpin	The pin that is connected to the TX pin of the TW-523.
	TX is used to send X10 data to the TW-523. This pin will be used in output mode.

The TW-523 RJ-11 connector has the following pinout:

Pin	Description	Connect to micro
1	Zero Cross	Input pin. Add 5.1K pull up.
2	GND	GND
3	RX	Not used.
4	ТХ	Output pin. Add 1K pull up.

See also

X10DETECT 956 , X10SEND 958

Example

_____ _____ 'name : x10.bas : (c) 1995-2005, MCS Electronics 'copyright 'purpose : example needs a TW-523 X10 interface : Mega48 'micro : yes 'suited for demo 'commercial addon needed : no 1 = 1_____ _____ \$regfile = "m48def.dat" ' specify the used micro **\$crystal** = 8000000 ' used crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32 ' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space 'define the house code **Const** House = "M" ' use code A-P Waitms 500 ' optional delay not really needed 'dim the used variables Dim X As Byte 'configure the zero cross pin and TX pin **Config** X10 = Pind.4 , Tx = Portb.0 1 ^--zero cross i. ^--- transmission pin

```
'detect the TW-523
X = X10detect()
                                                              ' 0 means
Print X
error, 1 means 50 Hz, 2 means 60 Hz
Do
   Input "Send (1-32) " , X
   'enter a key code from 1-31
   '1-16 to address a unit
   '17 all units off
   '18 all lights on
   '19 ON
   '20 OFF
   '21 DIM
   '22 BRIGHT
   '23 All lights off
   '24 extended code
   '25 hail request
   '26 hail acknowledge
   '27 preset dim
   '28 preset dim
   '29 extended data analog
   '30 status on
   '31 status off
   '32 status request
   X10send House , X
                                                               ' send the
code
Loop
Dim Ar(4) As Byte
X10send House , X , Ar(1) , 4
                                                               ' send 4
additional bytes
End
```

6.174 CONFIG XRAM

Action

Instruct the compiler to set options for external memory access.

Syntax

CONFIG XRAM = mode [, WaitstateLS=wls, WaitStateHS=whs]

Syntax Xmega

CONFIG XRAM = mode, sdbus=sdbus,lpc=lpc,sdcol=sdcol,sdcas=sdcas, sdrow=sdrow,refresh=refresh,initdelay=initdelay,modedelay=modedelay, rowcycledelay=rowcycledelay,rowprechargedelay=rowprechargedelay, wrdelay=wrdelay,ersdelay=esrdelay, rowcoldelay=rowcoldelay,modesel0=sel, adrsize0=adr,baseadr0=base,modesel1=sel,adrsize1=adr,baseadr1=base, modesel2=sel,adrsize2=adr,baseadr2=base,modesel3=sel,adrsize3=adr, baseadr3=base

Remarks AVR

Mode The memory mode. This is either enabled or disabled. By default, external memory access is disabled.

Wls	 When external memory access is enabled, some chips allow you to set a wait state. The number of modes depend on the chip. A modern chip such as the Mega8515 has 4 modes : 0 - no wait states 1 - 1 cycle wait state during read/write 2 - 2 cycle wait state during read/write 3 - 2 cycle wait state during read/write and 1 before new address output WLS works on the lower sector. Provided that the chip supports this.
Whs	 When external memory access is enabled, some chips allow you to set a wait state. The number of modes depend on the chip. A modern chip such as the Mega8515 has 4 modes : 0 - no wait states 1 - 1 cycle wait state during read/write 2 - 2 cycle wait state during read/write 3 - 2 cycle wait state during read/write and 1 before new address output WHS works on the high sector. Provided that the chip supports this.

Wait states are needed in case you connect equipment to the bus, that is relatively slow. Especial older electronics/chips.

Some AVR chips also allow you to divide the memory map into sections. By default the total XRAM memory address is selected when you set a wait state.

The \$XA directive should not be used anymore. It is the same as CONFIG XRAM=Enabled.

When using IDLE or another power down mode, it might be needed to use CONFIG XRAM again, after the chip wakes from the power down mode.

XMEGA

Mode	The memory mode. There are 4 options: - DISABLED, this will turn off the EBI and is the default - 3PORT. For using EBI in 3 PORT mode. - 4PORT. For using EBI in 4 PORT mode. - 2PORT. For using EBI in 2 PORT mode. The EBI uses specific ports for each of the modes.
sdbus	When using SDRAM, you need to configure 4 bit or 8 bit data width. For the 3 PORT mode you need to use 4 bit SDRAM. Options are : 4 and 8.
sdcol	When using SDRAM, you need to configure the number of columns of the chip. This depends on the chip. You can find this info in the datasheet of the SDRAM chip. For example a chip with column address A0-A9 would use 10 bits. Options : 8 ,9, 10 or 11.
sdrow	When using SDRAM, you need to configure the number of rows of the chip. This depends on the chip. You can find this info in the datasheet of the SDRAM chip. Options : 11 or 12.
sdcas	When using SDRAM you can configure the CAS latency as a number of Peripheral 2x Clock cycles.

	By default this is two Peripheral 2x Clock cycles. Options are : -2 CAS latency is two Peripheral 2x Clock cycles
	-3 CAS latency is three Peripheral 2x Clock cycles
refresh	When using SDRAM this value sets the refresh period as a number of peripheral clock cycles. Use a value between 0-1023. The value depends on the chip.
initdelay	When using SDRAM this value sets the delay of the initialization sequence that is sent after the voltages have been stabilized and the SDRAM clock is stable. The value is in the range of 0-16384
modedelay	When using SDRAM this value select the delay between Mode Register command and an Activate command in number of Peripheral 2x clock (CLKPER2) cycles. The range is between 0-3
rowcycledelay	When using SDRAM this value select the delay between a refresh an and Activate command in number of Peripheral 2x clock (CLKPER2) cycles. The range is between 0-7
rowprecharge delay	When using SDRAM this value select the delay between a pre-charge command and another command in number of Peripheral 2x clock (CLKPER2) cycles. The range is between 0-7
wrdelay	When using SDRAM this value selects the write recovery time in number of Peripheral 2x clock (CLK_{PER2}) cycles. The range is between 0-3
esrdelay	When using SDRAM this value selects the delay between CKE set high and activate command in number of Peripheral 2x clock (CLKPER2) cycles. The range is between 0-7
rowcoldelay	When using SDRAM this value selects the delay between an activate command and a read/write command as a number of Peripheral 2x clock (CLKPER2) cycles. The range is between 0-7
	The options ending with x , are available multiple times.(0-3) So there is an option named selfrefresh0, selfrefresh1, selfrefresh2 and selfrefresh3.
selfrefresh X	When using SDRAM this options can turn on/off self refresh of the SDRAM. Not all SDRAM have this capability. Valid options are : - ENABLED - DISABLED. This is the default.
sdmode X	When using SDRAM this option sets the SDRAM mode. This is either NORMAL (default) or LOAD.
modesel X	This option selects the MODE of the CS line. There are 4 CS lines and modes. When using SDRAM you can only select modesel3 to configure the SDRAM. The following options are possible: - DISABLE - SRAM - LPC (this is SRAM in low pin count mode) - SDRAM
adrsize X	This options sets the address size for the chip select. This is the size of the block above the base address and determines which address lines are compared to generate the CS. Options are: 256b , 256 bytes, address 8:23 512b, 512 bytes, address 9:23 1K , 1 KB , address 10:23 2K , 2 KB , address 11:23 4K , 4 KB , address 12:23 8K, 8 KB , address 13:23

0	
baseadr X	16K , 16 KB , address 14:23 32K , 32 KB , address 15:23 64K , 64 KB , address 16:23 128K , 128 KB, address 17:23 256K , 256 KB , address 18:23 512K , 512 KB , address 19:23 1M , 1 MB, address 20:23 2M , 2 MB , address 21:23 4M , 4 MB , address 22:23 8M , 8 MB, address 23 16M , 16 MB This option sets the chip base address which is the lowest address in the address space enabled by the chip select.
	The value is a word and sets address bits 12:23. Bits 9:11 are unused and need to be 0. For an 8 MB SDRAM the valid values are 0 and &H800000. Since the lower byte is not used you need to divide the address by 256(shift right 8 times). This gives a value of &H8000. When using 0, the memory overlaps the SRAM which is not a big problem with 8MB of ram!
	In SRAM mode there are some other options you must set
lpc	This sets the ALE mode in LPC SRAM mode. Options are : ALE1 : data multiplexed with address byte 0 ALE12 : data multiplexed with address byte 0 and 1
ale	This sets the ALE mode in normal SRAM mode. Options are : ALE1 : address byte 0 and 1 multiplexed ALE2 : address byte 0 and 2 multiplexed ALE12 : address byte 0, 1 and 2 multiplexed NOALE : No address multiplexing
waitstate X	The wait state selects the wait states for SRAM and SRAM LPC access as a number of peripheral 2x clock cycles. This is a value in the range from 0-7

While the EBI can configured to use a big 8 MB SDRAM, the compiler need to be changed in order to support more then 64KB of RAM. This will be handled in a future update. Using more then 64KB needs LDS/STS changed into using RAMPD and LD/ST in using the RAMP registers.

For 3PORT , 4-bit SDRAM mode the ports are set to the right direction and level. For all other modes you need to do this.

An example on how to determine the columns and rows is shown below:

Table	1:	Address	Table
lable	1:	Address	lable

	16 Meg x 4	8 Meg x 8	4 Meg x 16
Configuration	4 Meg x 4 x 4 banks	2 Meg x 8 x 4 banks	1 Meg x 16 x 4 banks
Refresh count	4K	4K	4K
Row addressing	4K (A0–A11)	4K (A0–A11)	4K (A0–A11)
Bank addressing	4 (BA0, BA1)	4 (BA0, BA1)	4 (BA0, BA1)
Column addressing	1K (A0–A9)	512 (A0-A8)	256 (A0-A7)

In 4 bit data mode, you use 16 Meg x 4, the row addressing is A0-A11 thus 12 bit and the column addressing is A0-A9 thus 10 bit.

See also <u>\$XA</u> এফা , <u>\$WAITSTATE</u> অসী

ASM

NONE

Example

CONFIG XRAM = Enabled, WaitstateLS=1 , WaitstateHS=2

Xmega Example

(c) 1995-2010, MCS xm128-XRAM-SDRAM-XPLAIN.bas . This sample demonstrates the Xmega128A1 XRAM SDRAM \$regfile = "xm128a1def.dat" **\$crystal** = 3200000 \$hwstack = 64\$swstack = 64**\$framesize** = 64 **\$xramsize** = &H800000 'include the following lib and code, the routines will be replaced since they are a workaround \$lib "xmega.lib" \$external _xmegafix_clear \$external _xmegafix_rol_r1014 'First Enable The Osc Of Your Choice Config Osc = Enabled , 32mhzosc = Enabled 'configure the systemclock **Config** Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1 'for xplain we need 9600 baud Config Com1 = 9600 , Mode = Asynchroneous , Parity = None , Stopbits = 1, Databits = 8Dim B As Byte , B1 As Byte , B2 As Byte **Config** Porte = **Output For** B = 1 **To** 5 Toggle Porte Waitms 1000 Next Print "Xplain SDRAM test" 'the XPLAIN has a 64 MBit SDRAM which is 8 MByte, it is connected in 3 port, 4 bit databus mode 'in the PDF of the SDRAM you can see it is connected as 16 Meg x 4. Refreshcount is 4K and the row address is AO-All, column addressing is A0-A9 Config Xram = 3port , Sdbus = 4 , Sdcol = 10 , Sdcas = 3 , Sdrow = 12 ,

BASCOM-AVR

594

```
Refresh = 500 , Initdelay = 3200 , Modedelay = 2 , Rowcycledelay = 7 ,
Rowprechargedelay = 7 , Wrdelay = 1 , Esrdelay = 7 , Rowcoldelay = 7 ,
Modesel3 = Sdram , Adrsize3 = 8m , Baseadr3 = &H0000
'the config above will set the port registers correct. it will also wait
for Ebi_cs3_ctrlb.7
'for all other modes you need to do this yourself !
Dim X(65000) As Xram Byte
Rampd = 0
                                                                    ' first xram
page
                                                                    ' load a
ldi r24,65
value
sts {x(60000)},r24
                                                                    ' write
lds r16, \{x(60000)\}
                                                                    ' read
                                                                    'set rampd
Rampd = 1
to second xram page
ldi r24,66
                                                                    ' load value
                                                                    ' write
sts {x(60000)},r24
Rampd = 0
                                                                    'back to
first page
lds r17, \{x(60000)\}
                                                                    'read
Rampd = 1
                                                                    'back to
second page
lds r18, \{x(60000)\}
                                                                    ' read
Rampd = 0
                                                                    'make sure
to switch back
sts {b1},r16
                                                                    ' load into
var
sts {b2},r17
sts {b},r18
Print "result : " ; B1 ; " " ; B2 ; " " ; B
Print "SRAM"
X(10000) = 100
                                                                    ' this will
use normal SRAM
B = X(10000)
Print "result : " ; B
```

End

6.175 CONST

Action

Declares a symbolic constant.

Syntax

CONST symbol = numconst **CONST** symbol = stringconst **CONST** symbol = expression

Remarks

Symbol	The name of the symbol.
Numconst	The numeric value to assign to the symbol.

Stringconst	The string to assign to the symbol
Expression	An expression that returns a value to assign the constant

Assigned constants consume no program memory because they only serve as a reference to the compiler.

The compiler will replace all occurrences of the symbol with the assigned value.

You can use a constant to give a value a more meaningful name. For example : variable = 1

const optHeaterOn = 1 variable = optHeaterOn

The source code is better to read when you assign a constant. Even better when the values change later, for example when HeaterOn becomes 2, you only need to replace 1 line of code.

See also

ALIAS 398

Example

```
_____
_____
                     : const.bas
'name
'copyright
                     : (c) 1995-2005, MCS Electronics
                     : demo for constants
'purpose
                     : Mega48
'micro
'suited for demo
                     : yes
'commercial addon needed : no
*_____
_____
$regfile = "m48def.dat"
                                                   ' specify
the used micro
$crystal = 4000000
                                                   ' used
crystal frequency
$baud = 19200
                                                   ' use baud
rate
$hwstack = 32
                                                   ' default
use 32 for the hardware stack
$swstack = 10
                                                   ' default
use 10 for the SW stack
$framesize = 40
                                                  ' default
use 40 for the frame space
'dimension some variables
Dim Z As String * 10
Dim B As Byte
'assign some constants
'constants dont use program memory
Const S = "test"
                                                   'declare a
Const A = 5
as a constant
Const B1 = &B1001
'or use an expression to assign a constant
Const X = (b1 * 3) + 2
Const Ssingle = Sin(1)
```

```
Print X
Print Ssingle
B = A
'the same as b = 5
Z = S
'the same as Z = "test"
Print A
Print B1
Print S
'you can use constants with conditional compilation
\#if A = 5
                                                              ' note there
is no then
 Print "constant a is 5"
  #if S = "test"
   Print "nested example"
                                                               ' else is
  #else
optional
  #endif
#else
#endif
End
```

6.176 COS

Action

Returns the cosine of a single

Syntax

var = COS(single)

Remarks

Var	A numeric variable that is assigned with cosine of variable single.
Single	The single variable to get the cosine of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD [649], ATN [403], SIN [885], TAN [920]

Example

```
$regfile = "m48def.dat"
the used micro
$crystal = 8000000
crystal frequency
$baud = 19200
```

' specify

' used

' use baud

```
rate
                                                              ' default
$hwstack = 32
use 32 for the hardware stack
                                                              ' default
\$swstack = 10
use 10 for the SW stack
                                                              ' default
$framesize = 40
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As Single , X As Single
S = 0.5 : X = Tan(s) : Print X
                                                              ' prints
0.546302195
S = 0.5 : X = Sin(s) : Print X
                                                              ' prints
0.479419108
S = 0.5 : X = Cos(s) : Print X
                                                              ' prints
0.877588389
End
```

6.177 COSH

Action

Returns the cosine hyperbole of a single

Syntax

var = COSH(single)

Remarks

	A numeric variable that is assigned with cosine hyperbole of variable single.
Single	The single or double variable to get the cosine hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD [649], ATN [403], COS [596], SIN [885], TANH [929], SINH [886]

Example

Show sample 983

6.178 COUNTER0 and COUNTER1

Action

Set or retrieve the internal 16 bit hardware register.

Syntax

COUNTER0 = var	TIMER0 can also be used
var = COUNTER0	

598 BASCOM-AVR

COUNTER1 = var var = COUNTER1	TIMER1 can also be used
CAPTURE1 = var var = CAPTURE1	TIMER1 capture register
COMPARE1A = var var = COMPARE1A	TIMER1 COMPARE A register
COMARE1B = var var = COMPARE1B	TIMER1 COMPARE B register
PWM1A = var var = PWM1A	TIMER1 COMPAREA register. (Is used for PWM)
PWM1B = var var = PRM1B	TIMER1 COMPARE B register. (Is used for PWM)

Remarks

Var	A byte, Integer/Word variable or constant that is assigned to the
	register or is read from the register.

Because the above 16 bit register pairs must be accessed somewhat differently than you may expect, they are implemented as variables.

The exception is TIMER0/COUNTER0, this is a normal 8 bit register and is supplied for compatibility with the syntax.

When the CPU reads the low byte of the register, the data of the low byte is sent to the CPU and the data of the high byte is placed in a temp register. When the CPU reads the data in the high byte, the CPU receives the data in the temp register.

When the CPU writes to the high byte of the register pair, the written data is placed in a temp register. Next when the CPU writes the low byte, this byte of data is combined with the byte data in the temp register and all 16 bits are written to the register pairs. So the MSB must be accessed first.

All of the above is handled automatically by BASCOM when accessing the above registers.

Note that the available registers may vary from chip to chip.

The BASCOM documentation used the 90S8515 to describe the different hardware registers.

6.179 CPEEK

Action

Returns a byte stored in code memory.

Syntax

var = CPEEK(address)

Remarks

Var	Numeric variable that is assigned with the content of the program
	memory at

	address
Address	Numeric variable or constant with the address location

There is no CPOKE statement because you can not write into program memory. Cpeek(0) will return the first byte of the file. Cpeek(1) will return the second byte of the binary file.

See also

PEEK 803 , POKE 804 , INP 740 , OUT 802

Example

_____ _____ : peek.bas 'name : (c) 1995-2005, MCS Electronics 'copyright 'purpose : demonstrates PEEk, POKE, CPEEK, INP and OUT : Mega48 'micro 'suited for demo : yes 'commercial addon needed : no _____ \$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Dim I As Integer , B1 As Byte 'dump internal memory **For** I = 0 **To** 31'only 32 registers in AVR B1 = **Peek(**i) 'get byte from internal memory
 Print Hex(b1) ; " "; 'Poke I , 1 'write a value into memory Next Print 'new line 'be careful when writing into internal memory !! 'now dump a part of the code-memory(program) **For** I = 0 **To** 255 B1 = Cpeek(i) 'get byte from internal memory Print Hex(b1) ; " "; Next 'note that you can not write into codememory !! **Out** &H8000 , 1 'write 1 into XRAM at address 8000 'return B1 = Inp(&H8000)value from XRAM

Print B1 End

6.180 CPEEKH

Action

Returns a byte stored in upper page of code memory of micro with more then 64KB such as M103, M128.

Syntax

var = CPEEKH(address [,page])

Remarks

	Numeric variable that is assigned with the content of the program memory at address
address	Numeric variable or constant with the address location
page	A numeric variable or constant with the page address. Each page is 64 KB.

CpeekH(0) will return the first byte of the upper 64KB.

Since the M103 has 64K words of code space the LPM instruction can not access the 64 upper Kbytes.

The CpeekH() function peeks in the upper 64 KB.

This function should be used with the M103 or M128 only.

CpeekH(address,0) will work on the first page (first 64 KB)

CpeekH(addres,1) will work on the second page (second 64 KB)

See also

PEEK 803 , POKE 804 , INP 740 , OUT 802

Example

'name 'copyright 'purpose 'micro 'suited for demo 'commercial addon needed	<pre>: peek.bas : (c) 1995-2005, MCS Electronics : demonstrates PEEk, POKE, CPEEK, : Mega48 : yes : no</pre>	INP and OUT
<pre>\$regfile = "m48def.dat"</pre>		' specify

```
the used micro
$crystal = 4000000
crystal frequency
$baud = 19200
rate
$hwstack = 32
use 32 for the hardware stack
$swstack = 10
use 10 for the SW stack
$framesize = 40
```

600

' used

' use baud

' default

' default

' default

```
use 40 for the frame space
Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31
                                                              'only 32
registers in AVR
  B1 = Peek(i)
                                                              'get byte
from internal memory
  Print Hex(b1) ; " ";
                                          'write a value into memory
  'Poke I , 1
Next
Print
                                                              'new line
'be careful when writing into internal memory !!
'now dump a part of the code-memory(program)
For I = 0 To 255
 B1 = Cpeek(i)
                                                              'get byte
from internal memory
 Print Hex(b1) ; " ";
Next
'note that you can not write into codememory !!
Out &H8000 , 1
                                                              'write 1
into XRAM at address 8000
B1 = Inp(\&H8000)
                                                              'return
value from XRAM
Print B1
End
```

6.181 CRC8

Action

Returns the CRC8 value of a variable or array.

Syntax

Var = CRC8(source , L)

Remarks

Var	The variable that is assigned with the CRC8 of variable source.
Source	The source variable or first element of the array to get the CRC8 of.
L	The number of bytes to check.

CRC8 is used in communication protocols to check if there are no transmission errors. The 1wire for example returns a CRC byte as the last byte from it's ID.

The code below shows a VB function of CRC8

Function Docrc8(s As String) As Byte Dim j As Byte Dim k As Byte Dim crc8 As Byte crc8 = 0 For m = 1 To Len(s) x = Asc(Mid(s, m, 1)) For k = 0 To 7 j = 1 And (x Xor crc8)

```
crc8 = Fix(crc8 / 2) And &HFF
x = Fix(x / 2) And &HFF
If j <> 0 Then
crc8 = crc8 Xor &H8C
End If
Next k
Next
Docrc8 = crc8
End Function
```

See also

CHECKSUM 427 , CRC16 603, CRC16UNI 605 , CRC32 607 , TCPCHECKSUM 920

ASM

The following routine is called from mcs.lib : _CRC8 The routine must be called with Z pointing to the data and R24 must contain the number of bytes to check. On return, R16 contains the CRC8 value. The used registers are : R16-R19, R25.

;##### X = Crc8(ar(1), 7) Ldi R24,\$07 ; number of bytes Ldi R30,\$64 ; address of ar(1) Ldi R31,\$00 ; load constant in register Rcall _Crc8 ; call routine Ldi R26,\$60 ; address of X St X,R16 ; store crc8

Example

\$regfile = "m48def.dat" ' specify the used micro **\$crystal** = 8000000 ' used crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10 ' default use 10 for the SW stack ' default **\$framesize** = 40 use 40 for the frame space **Config** Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 , Clockpol = 0Dim Ar(10) As Byte

Dim Ar(10) As Byte Dim J As Byte

Ar(1) = 1Ar(2) = 2Ar(3) = 3

```
J = Crc8(ar(1) , 3)
value which is 216
Print J
End
```

'calculate

6.182 CRC16

Action

Returns the CRC16 value of a variable or array.

Syntax

Var = CRC16(source , L)

Remarks

	The variable that is assigned with the CRC16 of variable source. Should be a word or integer variable.
Source	The source variable or first element of the array to get the CRC16 value from.
L	The number of bytes to check. This can be a numeric constant , byte or word variable. The maximum size to check is 65535.

CRC16 is used in communication protocols to check if there are no transmission errors.

The 1wire for example returns a CRC byte as the last byte from it's ID. Use CRC8 for the 1wire routines.

There are a lot of different CRC16 routines. There is no real standard since the polynomial will vary from manufacture to manufacture.

The equivalent code in VB is shown below. There are multiple ways to implement it in VB. This is one of them.

VB CRC16 Sample

Private Sub Command1_Click()

Dim ar(10) As Byte Dim b As Byte Dim J As Integer

ar(1) = 1ar(2) = 2ar(3) = 3

b = Docrc8(ar(), 3) ' call funciton
Print b
'calculate value which is 216

J = CRC16(ar(), 3) ' call function Print J

End Sub

```
Function Docrc8(ar() As Byte, bts As Byte) As Byte
Dim J As Byte
Dim k As Byte
Dim crc8 As Byte
crc8 = 0
For m = 1 To bts
 x = ar(m)
 For k = 0 To 7
 J = 1 And (x Xor crc8)
 crc8 = Fix(crc8 / 2) And &HFF
 x = Fix(x / 2) And &HFF
 If J <> 0 Then
   crc8 = crc8 Xor & H8C
 End If
 Next k
Next
Docrc8 = crc8
End Function
Public Function CRC16(buf() As Byte, lbuf As Integer) As Integer
Dim CRC1 As Long
Dim b As Boolean
CRC1 = 0 ' init CRC
For i = 1 To lbuf ' for each byte
 CRC MSB = CRC1 \setminus 256
 crc_LSB = CRC1 And 255
 CRC_MSB = CRC_MSB Xor buf(i)
 CRC1 = (CRC\_MSB * 256) + crc\_LSB
 For J = 0 To 7 Step 1 ' for each bit
  CRC1 = shl(CRC1, b)
  If b Then CRC1 = CRC1 Xor &H1021
Next J
Next i
CRC16 = CRC1
End Function
'Shift Left function
Function shl(n As Long, ByRef b As Boolean) As Long
 Dim L As Long
 L = n
 L = L * 2
 If (L > \&HFFFF\&) Then
  b = True
 Else
  b = False
 End If
 shI = L And &HFFFF&
End Function
```

See also

CHECKSUM 427, CRC8 607, CRC16UNI 605, CRC32 607, TCPCHECKSUM 920

ASM

The following routine is called from mcs.lib : _CRC16 The routine must be called with X pointing to the data. The soft stack –Y must contain the number of bytes to scan. On return, R16 and R17 contain the CRC16 value. The used registers are : R16-R19, R25.

;##### X = Crc16(ar(1), 7)

Ldi R24,\$07 ; number of bytes St -y, R24 Ldi R26,\$64 ; address of ar(1) Ldi R27,\$00 ; load constant in register Rcall _Crc16 ; call routine Ldi R26,\$60 ; address of X St X+,R16 ; store crc16 LSB St X , R17 ; store CRC16 MSB

Example

\$regfile = "m48def.dat" ' specify the used micro **\$crystal** = 8000000 ' used crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

```
Dim Ar(10) As Byte
Dim J As Byte
Dim W As Word
Dim L As Long
Ar(1) = 1
Ar(2) = 2
Ar(3) = 3
J = Crc8(ar(1) , 3)
value which is 216
W = Crc16(ar(1) , 3)
L = Crc32(ar(1) , 3)
End
```

'calculate

'24881 '494976085

6.183 CRC16UNI

Action

Returns the CRC16 value of a variable or array.

Syntax

Var = CRC16UNI(source , length , initial, polynomial, refin, refout)

Remarks

var	The variable that is assigned with the CRC16 of variable source. Should be a word or integer variable.
source	The source variable or first element of the array to get the CRC16 value from.
length	The number of bytes to check.
initial	The initial value of the CRC. This is usual 0 or &HFFFF.
polynomia I	The polynomial value to use.
refin	Reflect the data input bits. Use 0 to disable this option. Use a non-zero value to enable this option.
refout	Reflect the data output. Use 0 to disable this option. Use a non-zero value to enable this option.

CRC16 is used in communication protocols to check if there are no transmission errors.

The 1wire for example returns a CRC byte as the last byte from it's ID. Use CRC8 for the 1wire routines.

There are a lot of different CRC16 routines. There is no real standard since the polynomial will vary from manufacture to manufacture.

At <u>http://www.ross.net/crc/download/crc_v3.txt</u> you can find a great document about CRC calculation from Ross N. Williams. At the end you will find an example that is good for dealing with most CRC variations. The BASCOM CRC16UNI function is a conversion of this example.

There is a difference however : The CRC16UNI function does not XOR the output bytes. This because most CRC functions XOR with 0.

The example will show some of the most used combinations.

See also

CHECKSUM [427], CRC8 [607], CRC16 [603], CRC32 [607], TCPCHECKSUM [920]

Example

'name 'copyright 'purpose 'micro 'suited for demo 'commercial addon needed	<pre>: crc8-16-32.bas : (c) 1995-2008, MCS Electronics : demonstrates CRC : Mega48 : yes : no</pre>	
<pre>\$regfile = "m48def.dat" \$crystal = 8000000</pre>		' specify the used mic:

\$crystal = 8000000
\$baud = 19200
\$hwstack = 32
\$swstack = 10

' specify the used mich
' used crystal frequend
' use baud rate
' default use 32 for th
' default use 10 for th

BASCOM Language Reference 607

```
$framesize = 40
                                                             ' default use 40 for th
Dim Ar(10) As Byte
Dim J As Byte
Dim W As Word
Dim L As Long
Dim S As String * 16
S = "123456789"
Ar(1) = 1
Ar(2) = 2
Ar(3) = 3
J = Crc8(ar(1), 3)
                                                             'calculate value which
W = Crc16(ar(1), 3)
                                                             '24881
L = Crc32(ar(1), 3)
                                                             494976085
                   data , length, intial value , Poly, reflect input, reflect output
Print Hex(crc16uni(s , 9 , 0 , &H1021 , 0 , 0))
                                                             'CRC-CCITT (0x0000)
Print Hex(crc16uni(s , 9 , &HFFFF , &H1021 , 0 , 0))
                                                           'CRC-CCITT (0xFFFF)
Print Hex(crc16uni(s , 9 , &H1D0F , &H1021 , 0 , 0))
                                                            'CRC-CCITT (0x1D0F)
                                                                                   H
Print Hex(crc16uni(s , 9 , 0 , &H8005 , 1 , 1))
                                                            'crc16
                                                                                   I
Print Hex(crc16uni(s , 9 , &HFFFF , &H8005 , 1 , 1))
                                                            'crc16-modbus
```

End

6.184 CRC32

Action

Returns the CRC32 value of a variable.

Syntax

Var = CRC32(source , L)

Remarks

Var	The LONG variable that is assigned with the CRC32 of variable source.
	The source variable or first element of the array to get the CRC 32 value from.
L	The number of bytes to check. This can be a word variable.

CRC32 is used in communication protocols to check if there are no transmission errors.

See also

CHECKSUM 427, CRC8 607, CRC16 603, CRC16UNI 605, TCPCHECKSUM 920

Example

608 BASCOM-AVR

```
$regfile = "m48def.dat"
                                                               ' specify
the used micro
                                                               ' used
$crystal = 8000000
crystal frequency
$baud = 19200
                                                               ' use baud
rate
$hwstack = 32
                                                               ' default
use 32 for the hardware stack
\$swstack = 10
                                                               ' default
use 10 for the SW stack
$framesize = 40
                                                               ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim Ar(10) As Byte
Dim J As Byte
Dim W As Word
Dim L As Long
Ar(1) = 1
Ar(2) = 2
Ar(3) = 3
J = Crc8(ar(1), 3)
                                                               'calculate
value which is 216
W = Crc16(ar(1), 3)
                                                               24881
L = Crc32(ar(1), 3)
                                                               494976085
End
```

6.185 CRYSTAL

Action

Special byte variable that can be used with software UART routine to change the baud rate during runtime.

Syntax

CRYSTAL = var (old option do not use !!)

___CRYSTAL1 = var BAUD #1, 2400

Remarks

With the software UART you can generate good baud rates. But chips such as the ATtiny22 have an internal 1 MHz clock. The clock frequency can change during runtime by influence of temperature or voltage.

The crystal variable can be changed during runtime to change the baud rate.

The above has been changed in version 1.11 Now you still can change the baud rate with the crystal variable. But you don't need to dimension it. And the name has been changed:

__CRYSTALx where x is the channel number.

When you opened the channel with #1, the variable will be named ____CRYSTAL1

But a better way is provided now to change the baud rate of the software uart at run time. You can use the BAUD option now:

Baud #1, 2400 'change baud rate to 2400 for channel 1

When you use the baud # option, you must specify the baud rate before you print or use input on the channel. This will dimension the ____CRYSTALx variable and load it with the right value.

When you don't use the BAUD # option the value will be loaded from code and it will not use 2 bytes of your SRAM.

The ____CRYSTALx variable is hidden in the report file because it is a system variable. But you may assign a value to it after BAUD #x, zzzz has dimensioned it.

The old CRYSTAL variable does not exist anymore.

Some values for 1 MHz internal clock : 66 for 2400 baud 31 for 4800 baud 14 for 9600 baud

See also

OPEN 7947, CLOSE 7947

Example

Dim B as byte Open "comd.1:9600,8,n,1,inverted" For Output As #1 Print #1 , B Print #1 ,"serial output" baud #1, 4800 'use 4800 baud now Print #1,"serial output" ____CRYSTAL1 = 255 Close#1 End

6.186 CURSOR

Action

Set the LCD Cursor State.

Syntax

CURSOR ON / OFF BLINK / NOBLINK

Remarks

You can use both the ON or OFF and BLINK or NOBLINK parameters. At power up the cursor state is ON and NOBLINK.

See also

DISPLAY 66计, LCD 75社, SHIFTLCD 88社, SHIFTCURSOR 87社

Example

```
_____
                                      _____
                         : lcd.bas
'name
                         : (c) 1995-2005, MCS Electronics
'copyright
                         : demo: LCD, CLS, LOWERLINE, SHIFTLCD,
'purpose
SHIFTCURSOR, HOME
                           CURSOR, DISPLAY
                        : Mega8515
'micro
'micro : Mega
'suited for demo : yes
'commercial addon needed : no
·_____
                                 _____
_____
$regfile = "m8515.dat"
                                                           ' specify
the used micro
$crystal = 4000000
                                                           ' used
crystal frequency
$baud = 19200
                                                           ' use baud
rate
\$hwstack = 32
                                                           ' default
use 32 for the hardware stack
Sswstack = 10
                                                           ' default
use 10 for the SW stack
$framesize = 40
                                                           ' default
use 40 for the frame space
$sim
'REMOVE the above command for the real program !!
'$sim is used for faster simulation
'note : tested in PIN mode with 4-bit
'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 ,
Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 ,
Db7 = Porta.7, E = Portc.7, Rs = Portc.6
'These settings are for the STK200 in PIN mode
'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-
D7
'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of
the LCD connector
'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector
Rem with the config lcdpin statement you can override the compiler
settings
Dim A As Byte
Config Lcd = 16 \times 2
                                                           'configure
lcd screen
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a 'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines
```

<pre>'\$LCD = address will turn LCD into 8-bit databus mode use this with uP with external RAM and/or ROM because it aint need the port pins !</pre>	
Cls	'clear the
LCD display Lcd "Hello world." this at the top line Wait 1	'display
Lowerline lower line	'select the
Wait 1 Lcd "Shift this." this at the lower line Wait 1	'display
<pre>For A = 1 To 10 Shiftlcd Right text to the right</pre>	'shift the
Wait 1 moment Next	'wait a
<pre>For A = 1 To 10 Shiftlcd Left text to the left</pre>	'shift the
Wait 1 moment Next	'wait a
Locate 2 , 1 position	'set cursor
Lcd "*" this	'display
Wait 1 moment	'wait a
Shiftcursor Right cursor	'shift the
Lcd "@" this	'display
Wait 1 moment	'wait a
Home Upper 1 and return home	'select line
Lcd "Replaced." text	'replace the
Wait 1 moment	'wait a
Cursor Off Noblink Wait 1 moment	'hide cursor 'wait a
Cursor On Blink Wait 1 moment	'show cursor 'wait a
Display Off	'turn
display off Wait 1 moment	'wait a
Display On display on	'turn
'NEW support for 4-line LCD	

```
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third
                                                              'goto home
on line three
Home Fourth
                                                              'first
Home F
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1
'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line
Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240
replace ? with number (0-7)
Cls
                                                              'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0); Chr(1)
                                                              'print the
special character
1 _ _ _ _ _ _ _ _ _
         ----- Now use an internal routine ------
_{temp1} = 1
                                                              'value into
ACC
!rCall _write_lcd
                                                              'put it on
LCD
End
```

6.187 DATA

Action

Specifies constant values to be read by subsequent READ statements.

Syntax

DATA var [, varn]

Remarks

Var Numeric or string constant.

The DATA related statements use the internal registers pair R8 and R9 to store the data pointer.

To store a " sign on the data line, you can use : DATA \$34

The \$-sign tells the compiler that the ASCII value will follow. You can use this also to store special characters that can't be written by the editor such as chr(7)

Another way to include special ASCII characters in your string constant is to use $\{XXX\}$. You need to include exactly 3 digits representing the ASCII character. For

example 65 is the ASCII number for the character A.

DATA "TEST{065}"

Will be read as TESTA.

While : DATA "TEST{65}" will be read as :

TEST{65}. This because only 2 digits were included instead of 3.

 $\{xxx\}$ works only for string constants. It will also work in a normal string assignment :

 $s = "{065}"$. This will assign A to the string s.

Because the DATA statements allow you to generate an EEP file to store in EEPROM, the DATA and EEPROM and EEPROM directives have been added. Read the description of these directives to learn more about the DATA statement.

The DATA statements must not be accessed by the flow of your program because the DATA statements are converted to the byte representation of the DATA.

When your program flow enters the DATA lines, unpredictable results will occur. So as in QB, the DATA statement is best be placed at the end of your program or in a place that program flow will no enter.

For example this is fine:

Print "Hello" Goto jump DATA "test"

Jump: 'because we jump over the data lines there is no problem.

The following example will case some problems: Dim S As String * 10 Print "Hello" Restore lbl Read S

DATA "test"

Print S

When the END statement is used it must be placed BEFORE the DATA lines.

Difference with QB

Integer and Word constants must end with the %-sign. Long constants must end with the &-sign. Single constants must end with the !-sign. Double constants must end with the #-sign.

See also

READ [828], RESTORE [841], \$DATA [316], \$EEPROM [321], LOOKUP [774], LOOKUPSTR [775], LOOKDOWN [772]

Example

,______ _____ : readdata.bas 'name : (c) 1995-2005, MCS Electronics 'copyright 'purpose : demo : READ, RESTORE : Mega48 'micro 'suited for demo 'commercial addon needed : no ·_____ _____ \$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack \$framesize = 40 ' default use 40 for the frame space Dim A As Integer , B1 As Byte , Count As Byte Dim S As String * 15 Dim L As Long Restore Dtal 'point to stored data For Count = 1 To 3 'for number of data items Read B1 : Print Count ; " ; B1 Next **Restore** Dta2 'point to stored data For Count = 1 To 2 'for number of data items Read A : Print Count ; " " ; A Next **Restore** Dta3 **Read** S : **Print** S Read S : Print S **Restore** Dta4 Read L : Print L 'long type 'demonstration of readlabel Dim W As Iram Word At 8 Overlay ' location is used by restore pointer 'note that W does not use any RAM it is an overlayed pointer to the data pointer

```
615
```

```
' loadlabel
```

```
W = Loadlabel(dta1)
expects the labelname
Read B1
Print B1
End
Dta1:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%
Dta3:
Data "Hello" , "World"
'Note that integer values (>255 or <0) must end with the -sign
'also note that the data type must match the variable type that is
'used for the READ statement
Dta4:
Data 123456789&
'Note that LONG values must end with the &-sign
'Also note that the data type must match the variable type that is used
'for the READ statement
```

6.188 DAYOFWEEK

Action

Returns the Day of the Week of a Date.

Syntax

```
Target = DayOfWeek()
Target = DayOfWeek(bDayMonthYear)
Target = DayOfWeek(strDate)
Target = DayOfWeek(wSysDay)
Target = DayOfWeek(ISysSec)
```

Remarks

Target	A Byte – variable, that is assigned with the day of the week	
BDayMonthYea	A Byte – variable, which holds the Day-value followed by Month (Byte) and Year (Byte)	
	A String, which holds a Date-String in the format specified in the CONFIG DATE statement	
WSysDay	A Word – variable, which holds the System Day (SysDay)	
LSysSec	A Long – variable, which holds the System Second (SysSec)	

The Function can be used with five different kind of Input:

- 1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
- 2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Week can be calculated of every date.
- 3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement
- 4. With a System Day Number.

5. With a System Second - Number

The Return-Value is in the range of 0 to 6, Monday starts with 0.

The Function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See Also

Date and Time routines (अभी), CONFIG DATE (यही), CONFIG CLOCK (यही), SYSDAY (आर्ग), SYSSEC (यही)

Example

	<pre>: datetime_test1,bas : (c) 1995-2005, MCS Electronics : show how to use the Date-Time ro : Mega103 : no : no</pre>	out	tines from
<pre>\$regfile = "m103def.dat"</pre>		,	specify
the used micro \$crystal = 4000000		,	used
crystal frequency \$baud = 19200		,	use baud
rate \$hwstack = 32		,	default
<pre>use 32 for the hardware st \$swstack = 10</pre>	ack	,	default
use 10 for the SW stack \$framesize = 40 use 40 for the frame space		1	default
Const Clockmode = 1 'use i2c for the clock			
<pre>#if Clockmode = 1 Config Clock = Soft build in clock Disable Interrupts</pre>		1	we use
<pre>#else Config Clock = User for the clock 'configure the scl and s Config Sda = Portd.6 Config Scl = Portd.5</pre>	da pins	1	we use I2C
<pre>'address of ds1307 Const Ds1307w = &HD0 of Ds1307 clock Const Ds1307r = &HD1 #endif</pre>		1	Addresses
<pre>'configure the date format Config Date = Ymd , Separa</pre>		ı	ANSI-

```
Format
'This sample does not have the clock started so interrupts are not
enabled
' Enable Interrupts
'dim the used variables
Dim Lvar1 As Long
Dim Mday As Byte
Dim Bweekday As Byte , Strweekday As String * 10
Dim Strdate As String * 8
Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte , Bhour As Byte
Dim Bday As Byte , Bmonth As Byte , Byear As Byte
Dim Lsecofday As Long
Dim Wsysday As Word
Dim Lsyssec As Long
Dim Wdayofyear As Word
' ======= DayOfWeek
_____
' Example 1 with internal RTC-Clock
_day = 4 : _month = 11 : _year = 2
                                                           ' Load RTC-
Clock for example - testing
Bweekday = Dayofweek()
Strweekday = Lookupstr(bweekday, Weekdays)
Print "Weekday-Number of " ; Date$ ; " is " ; Bweekday ; " = " ;
Strweekday
' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 26 : Bmonth = 11 : Byear = 2
Bweekday = Dayofweek(bday)
Strweekday = Lookupstr(bweekday, Weekdays)
Strdate = Date(bday)
Print "Weekday-Number of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " is " ; Bweekday ; " (" ; Date(bday) ; ") = " ; Strweekday
' Example 3 with System Day
Wsysday = 2000
                                                           ' that is
2005-06-23
Bweekday = Dayofweek(wsysday)
Strweekday = Lookupstr(bweekday, Weekdays)
Print "Weekday-Number of System Day " ; Wsysday ; " (" ; Date(wsysday) ;
 ") is "; Bweekday; " = "; Strweekday
' Example 4 with System Second
Lsyssec = 123456789
                                                           ' that is
2003-11-29 at 21:33:09
Bweekday = Dayofweek(lsyssec)
Strweekday = Lookupstr(bweekday, Weekdays)
Print "Weekday-Number of System Second " ; Lsyssec ; " (" ; Date(lsyssec
) ; ") is " ; Bweekday ; " = " ; Strweekday
```

' Example 5 with Date-String

```
618 BASCOM-AVR
```

```
' we have
Strdate = "04-11-02"
configured Date in ANSI
Bweekday = Dayofweek(strdate)
Strweekday = Lookupstr(bweekday, Weekdays)
Print "Weekday-Number of " ; Strdate ; " is " ; Bweekday ; " = " ;
Strweekday
' =========== Second of Day
_____
' Example 1 with internal RTC-Clock
_sec = 12 : _min = 30 : _hour = 18
                                                         ' Load RTC-
Clock for example - testing
Lsecofday = Secofday()
Print "Second of Day of " ; Time$ ; " is " ; Lsecofday
' Example 2 with defined Clock - Bytes (Second / Minute / Hour)
Bsec = 20 : Bmin = 1 : Bhour = 7
Lsecofday = Secofday(bsec)
Print "Second of Day of Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour
; " ("; Time(bsec); ") is "; Lsecofday
' Example 3 with System Second
Lsyssec = 1234456789
Lsecofday = Secofday(lsyssec)
Print "Second of Day of System Second " ; Lsyssec ; "(" ; Time(lsyssec)
; ") is "; Lsecofday
' Example 4 with Time - String
Strtime = "04:58:37"
Lsecofday = Secofday(strtime)
Print "Second of Day of " ; Strtime ; " is " ; Lsecofday
' ========== System Second
_____
' Example 1 with internal RTC-Clock
                         ' Load RTC-Clock for example - testing
_sec = 17 : _min = 35 : _hour = 8 : _day = 16 : _month = 4 : _year = 3
Lsyssec = Syssec()
Print "System Second of " ; Time$ ; " at " ; Date$ ; " is " ; Lsyssec
' Example 2 with with defined Clock - Bytes (Second, Minute, Hour, Day /
Month / Year)
Bsec = 20 : Bmin = 1 : Bhour = 7 : Bday = 22 : Bmonth = 12 : Byear = 1
Lsyssec = Syssec(bsec)
Strtime = Time(bsec)
Strdate = Date(bday)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec
```

' Example 3 with System Day

```
Wsysday = 2000
Lsyssec = Syssec(wsysday)
Print "System Second of System Day " ; Wsysday ; " (" ; Date(wsysday) ;
" 00:00:00) is " ; Lsyssec
' Example 4 with Time and Date String
Strtime = "10:23:50"
Strdate = "02-11-29"
                                                        ' ANSI-Date
Lsyssec = Syssec(strtime , Strdate)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec ' 91880630
' ====== Day Of Year
------
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2
                                                        ' Load RTC-
Clock for example - testing
Wdayofyear = Dayofyear()
Print "Day Of Year of " ; Date$ ; " is " ; Wdayofyear
' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wdayofyear = Dayofyear(bday)
Print "Day Of Year of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " (" ; Date(bday) ; ") is " ; Wdayofyear
' Example 3 with Date - String
Strdate = "04-10-29"
                                                        ' we have
configured ANSI Format
Wdayofyear = Dayofyear(strdate)
Print "Day Of Year of " ; Strdate ; " is " ; Wdayofyear
' Example 4 with System Second
Lsyssec = 123456789
Wdayofyear = Dayofyear(lsyssec)
Print "Day Of Year of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ;
 ") is " ; Wdayofyear
' Example 5 with System Day
Wsysday = 3000
Wdayofyear = Dayofyear(wsysday)
Print "Day Of Year of System Day " ; Wsysday ; " (" ; Date(wsysday) ; ")
is " ; Wdayofyear
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2
                                                        ' Load RTC-
Clock for example - testing
Wsysday = Sysday()
Print "System Day of " ; Date$ ; " is " ; Wsysday
' Example 2 with defined Clock - Bytes (Day / Month / Year)
```

```
Bday = 24: Bmonth = 5: Byear = 8
Wsysday = Sysday(bday)
Print "System Day of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " (" ; Date(bday) ; ") is " ; Wsysday
' Example 3 with Date - String
Strdate = "04-10-29"
Wsysday = Sysday(strdate)
Print "System Day of " ; Strdate ; " is " ; Wsysday
' Example 4 with System Second
Lsyssec = 123456789
Wsysday = Sysday(lsyssec)
Print "System Day of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ;
") is " ; Wsysday
' ====== Time
_____
' Example 1: Converting defined Clock - Bytes (Second / Minute / Hour)
to Time - String
Bsec = 20 : Bmin = 1 : Bhour = 7
Strtime = Time(bsec)
Print "Time values: Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour ; "
converted to string " ; Strtime
' Example 2: Converting System Second to Time - String
Lsyssec = 123456789
Strtime = Time(lsyssec)
Print "Time of Systemsecond " ; Lsyssec ; " is " ; Strtime
' Example 3: Converting Second of Day to Time - String
Lsecofday = 12345
Strtime = Time(lsecofday)
Print "Time of Second of Day " ; Lsecofday ; " is " ; Strtime
' Example 4: Converting System Second to defined Clock - Bytes (Second /
Minute / Hour)
Lsyssec = 123456789
Bsec = Time(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Sec=" ; Bsec ; " Min="
; Bmin ; " Hour=" ; Bhour ; " (" ; Time(lsyssec) ; ")"
' Example 5: Converting Second of Day to defined Clock - Bytes (Second /
Minute / Hour)
Lsecofday = 12345
Bsec = Time(lsecofday)
Print "Second of Day " ; Lsecofday ; " converted to Sec=" ; Bsec ; "
Min=" ; Bmin ; " Hour=" ; Bhour ; " (" ; Time(lsecofday) ; ")"
' Example 6: Converting Time-string to defined Clock - Bytes (Second /
Minute / Hour)
Strtime = "07:33:12"
Bsec = Time(strtime)
Print "Time "; Strtime; " converted to Sec="; Bsec; " Min="; Bmin;
 " Hour=" ; Bhour
```

© 2011 MCS Electronics

```
' ======= Date
_____
' Example 1: Converting defined Clock - Bytes (Day / Month / Year) to
Date - String
Bday = 29 : Bmonth = 4 : Byear = 12
Strdate = Date(bday)
Print "Dat values: Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ; Byear
; " converted to string " ; Strdate
' Example 2: Converting from System Day to Date - String
Wsysday = 1234
Strdate = Date(wsysday)
Print "System Day " ; Wsysday ; " is " ; Strdate
' Example 3: Converting from System Second to Date String
Lsyssec = 123456789
Strdate = Date(lsyssec)
Print "System Second " ; Lsyssec ; " is " ; Strdate
' Example 4: Converting SystemDay to defined Clock - Bytes (Day /
Month / Year)
Wsysday = 2000
Bday = Date(wsysday)
Print "System Day " ; Wsysday ; " converted to Day=" ; Bday ; " Month="
; Bmonth ; " Year=" ; Byear ; " (" ; Date(wsysday) ; ")"
' Example 5: Converting Date - String to defined Clock - Bytes (Day /
Month / Year)
Strdate = "04-08-31"
Bday = Date(strdate)
Print "Date " ; Strdate ; " converted to Day=" ; Bday ; " Month=" ;
Bmonth ; " Year=" ; Byear
' Example 6: Converting System Second to defined Clock - Bytes (Day /
Month / Year)
Lsyssec = 123456789
Bday = Date(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Day=" ; Bday ; "
Month=" i Bmonth ; " Year=" i Byear ; " (" i Date(lsyssec) ; ")"
' ========= Second of Day elapsed
Lsecofday = Secofday()
hour = hour + 1
Lvar1 = Secelapsed(lsecofday)
Print Lvar1
Lsyssec = Syssec()
_day = _day + 1
Lvar1 = Syssecelapsed(lsyssec)
Print Lvar1
```

```
Looptest:
' Initialising for testing
_day = 1
_month = 1
_year = 1
_sec = 12
_min = 13
hour = 14
Do
   If _year > 50 Then
     Exit Do
  End If
  _sec = _sec + 7
  If _sec > 59 Then
    Incr _min
     _{sec} = _{sec} - 60
  End If
  \min = \min + 2
  If _min > 59 Then
     Incr _hour
     _min = _min - 60
  End If
  hour = hour + 1
  If _hour > 23 Then
     Incr _day
     hour = hour - 24
  End If
  _day = _day + 1
  If _day > 28 Then
     Select Case _month
        Case 1
           Mday = 31
        Case 2
           Mday = _year And &H03
           If Mday = 0 Then
              Mday = 29
           Else
              Mday = 28
           End If
        Case 3
          Mday = 31
        Case 4
          Mday = 30
        Case 5
          Mday = 31
        Case 6
          Mday = 30
        Case 7
          Mday = 31
        Case 8
          Mday = 31
        Case 9
           Mday = 30
```

```
Case 10
           Mday = 31
        Case 11
           Mday = 30
        Case 12
           Mday = 31
     End Select
     If _day > Mday Then
        _day = _day - Mday
Incr _month
        If _month > 12 Then
           _month = 1
           Incr _year
        End If
     End If
  End If
  If _year > 99 Then
     Exit Do
  End If
Lsecofday = Secofday()
Lsyssec = Syssec()
Bweekday = Dayofweek()
Wdayofyear = Dayofyear()
Wsysday = Sysday()
Print Time$ ; " " ; Date$ ; " " ; Lsecofday ; " " ; Lsyssec ; " " ;
Bweekday ; " " ; Wdayofyear ; " " ; Wsysday
Loop
End
'only when we use I2C for the clock we need to set the clock date time
#if Clockmode = 0
'called from datetime.lib
Dim Weekday As Byte
Getdatetime:
  I2cstart
                                                               ' Generate
start code
  I2cwbyte Ds1307w
                                                               ' send
address
  I2cwbyte 0
                                                               ' start
address in 1307
  I2cstart
                                                               ' Generate
start code
  I2cwbyte Ds1307r
                                                               ' send
address
  I2crbyte _sec , Ack
                                                               ' MINUTES
  I2crbyte _min , Ack
  I2crbyte _hour , Ack
                                                               ' Hours
                                                               ' Day of
  I2crbyte Weekday , Ack
Week
  I2crbyte _day , Ack
                                                               ' Day of
Month
                                                               ' Month of
  I2crbyte _month , Ack
Year
  I2crbyte _year , Nack
                                                               ' Year
  I2cstop
  _sec = Makedec(_sec) : _min = Makedec(_min) : _hour = Makedec(_hour)
```

```
_day = Makedec(_day) : _month = Makedec(_month) : _year = Makedec(
_year)
Return
Setdate:
 _day = Makebcd(_day) : _month = Makebcd(_month) : _year = Makebcd(
_year)
 I2cstart
                                                              ' Generate
start code
  I2cwbyte Ds1307w
                                                              ' send
address
                                                              ' starting
  I2cwbyte 4
address in 1307
                                                              ' Send Data
 I2cwbyte _day
to SECONDS
  I2cwbyte _month
                                                              ' MINUTES
  I2cwbyte _year
                                                              ' Hours
  I2cstop
Return
Settime:
  _sec = Makebcd(_sec) : _min = Makebcd(_min) : _hour = Makebcd(_hour)
  I2cstart
                                                               ' Generate
start code
  I2cwbyte Ds1307w
                                                              ' send
address
  I2cwbyte 0
                                                              ' starting
address in 1307
 I2cwbyte _sec
                                                              ' Send Data
to SECONDS
 I2cwbyte _min
                                                              ' MINUTES
  12cwbyte hour
                                                              ' Hours
  I2cstop
Return
#endif
```

```
Weekdays:
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" ,
"Saturday" , "Sunday"
```

6.189 DAYOFYEAR

Action

Returns the Day of the Year of a Date

Syntax

```
Target = DayOfYear()
Target = DayOfYear(bDayMonthYear)
Target = DayOfYear(strDate)
Target = DayOfYear(wSysDay)
```

Target = **DayOfYear**(ISysSec)

Remarks

Target	A Integer, that is assigned with the Day of the Year
BDayMonthYea	A Byte, which holds the Day-value followed by Month(Byte) and Year
r	(Byte)

624

	A String, which holds a Date-String in the format specified in the CONFIG DATE statement
WSysDay	A Variable (Word) which holds a System Day (SysDay)
LsysSec	A Variable (Long) which holds a System Second (SysSec)

The Function can be used with five different kind of Input:

- 1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
- 2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Year can be calculated of every date.
- 3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement.
- 4. With a System Day Number (WORD)
- 5. With a System Second Number (LONG)

The Return-Value is in the Range of 0 to 364 (365 in a leap year). January the first starts with 0.

The function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See also

Date and Time Routines (993), SysSec (915), SysDay (917)

Example

See DayOfWeek 615

6.190 DATE\$

Action

Internal variable that holds the date.

Syntax

DATE\$ = "mm/dd/yy" var = **DATE\$**

Remarks

The DATE\$ variable is used in combination with the CONFIG CLOCK directive.

The CONFIG CLOCK statement will use the TIMER0 or TIMER2 in asynchrone mode to create an interrupt that occurs every second. In this interrupt routine the *_Sec*, *_Min* and *_Hour* variables are updated. The *_dat*, *_month* and *_year* variables are also updated. The date format is in the same format as in VB.

When you assign DATE\$ to a string variable these variables are assigned to the DATE\$ variable.

When you assign the DATE\$ variable with a constant or other variable, the _day, _month and _year variables will be changed to the new date.

The only difference with VB is that all data must be provided when assigning the date. This is done for minimal code. You can change this behavior of course.

The async timer is only available in the M103, 90S8535, M163 and M32(3), Mega128, Mega64, Mega8. For other chips it will not work.

As new chips are launched by Atmel, and support is added by MCS, the list above might not be complete. It is intended to serve as an example for chips with a timer that can be used in asynchrone mode. So when your micro has a timer that can be used in asynchrone mode, it should work.

bo not confuse DATE\$ with the DATE function.

ASM

The following ASM routines are called. When assigning DATE\$: _set_date (calls _str2byte) When reading DATE\$: _make_dt (calls _byte2str)

See also

TIME\$ 930, CONFIG CLOCK 461, DATE 627

Example

```
_____
'name
                        : megaclock.bas
'copyright
                        : (c) 1995-2005, MCS Electronics
'purpose
                        : shows the new TIME$ and DATE$ reserved
variables
                        : Mega103
'micro
'micro : Meg.
'suited for demo : yes
'commercial addon needed : no
·_____
                                             _____
------
$regfile = "m103def.dat"
                                                          ' specify
the used micro
$crystal = 4000000
                                                          ' used
crystal frequency
$baud = 19200
                                                          ' use baud
rate
\$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 10
                                                          ' default
use 10 for the SW stack
$framesize = 40
                                                          ' default
use 40 for the frame space
'With the 8535 and timer2 or the Megal03 and TIMER0 you can
'easily implement a clock by attaching a 32768 Hz xtal to the timer
'And of course some BASCOM code
'This example is written for the STK300 with M103
Enable Interrupts
'[configure LCD]
$1cd = &HC000
                                                          'address for
```

```
E and RS
$lcdrs = &H8000
                                                            'address for
only E
Config Lcd = 20 \times 4
                                                            'nice
display from bg micro
Config Lcdbus = 4
                                                            'we run it
in bus mode and I hooked up only db4-db7
Config Lcdmode = Bus
                                                            'tell about
the bus mode
'[now init the clock]
Config Date = Mdy , Separator = /
                                                            ' ANSI-
Format
Config Clock = Soft
                                                            'this is how
simple it is
'The above statement will bind in an ISR so you can not use the TIMER
anymore!
'For the M103 in this case it means that TIMER0 can not be used by the
user anymore
'assign the date to the reserved date$
'The format is MM/DD/YY
Date$ = "11/11/00"
'assign the time, format in hh:mm:ss military format(24 hours)
'You may not use 1:2:3 !! adding support for this would mean overhead
'But of course you can alter the library routines used
Time$ = "02:20:00"
                 _____
'clear the LCD display
Cls
Do
                                                            'cursor home
 Home
 Lcd Date$ ; " " ; Time$
                                                            'show the
date and time
LOOD
'The clock routine does use the following internal variables:
'_day , _month, _year , _sec, _hour, _min
'These are all bytes. You can assign or use them directly
day = 1
'For the _year variable only the year is stored, not the century
End
```

6.191 DATE

Action

Returns a date-value (String or 3 Bytes for Day, Month and Year) depending of the Type of the Target

Syntax

```
bDayMonthYear = Date(ISysSec)
bDayMonthYear = Date(ISysDay)
bDayMonthYear = Date(strDate)
```

strDate = Date(ISysSec)
strDate = Date(ISysDay)
strDate = Date(bDayMonthYear)

Remarks

	A Date-String in the format specified in the CONFIG DATE statement
LsysSec	A LONG – variable which holds the System Second (SysSec = TimeStamp)
LsysDay	A WORD – variable, which holds then System Day (SysDay)
BDayMonthYea r	A BYTE – variable, which holds Days, followed by Month (Byte) and Year (Byte)

Converting to String:

The target string must have a length of at least 8 Bytes, otherwise SRAM after the target-string will be overwritten.

Converting to Soft clock date format (3 Bytes for Day, Month and Year):

Three Bytes for Day, Month and Year must follow each other in SRAM. The variablename of the first Byte, the one for Day must be passed to the function.

See also

Date and Time Routines [993], DAYOFYEAR [624], SYSDAY [917]

Example

· ·	
'name	: datetime_test1,bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: show how to use the Date-Time routines from
the DateTime.Lib	
'micro	: Mega103
'suited for demo	: no
'commercial addon needed	: no

\$regfile = "m103def.dat"
the used micro
\$crystal = 4000000
crystal frequency
\$baud = 19200
rate
\$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
use 10 for the SW stack
\$framesize = 40
use 40 for the frame space
Const Clockmode = 1

' specify

' use baud

' default

' default

' default

' used

```
'use i2c for the clock
#if Clockmode = 1
 Config Clock = Soft
                                                           ' we use
build in clock
 Disable Interrupts
#else
 Config Clock = User
                                                           ' we use I2C
for the clock
  'configure the scl and sda pins
 Config Sda = Portd.6
 Config Scl = Portd.5
  'address of ds1307
 Const Ds1307w = &HD0
                                                           ' Addresses
of Ds1307 clock
 Const Ds1307r = &HD1
#endif
'configure the date format
                                                           ' ANST-
Config Date = Ymd , Separator = -
Format
'This sample does not have the clock started so interrupts are not
enabled
Enable Interrupts
'dim the used variables
Dim Lvar1 As Long
Dim Mday As Byte
Dim Bweekday As Byte , Strweekday As String * 10
Dim Strdate As String * 8
Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte , Bhour As Byte
Dim Bday As Byte , Bmonth As Byte , Byear As Byte
Dim Lsecofday As Long
Dim Wsysday As Word
Dim Lsyssec As Long
Dim Wdayofyear As Word
' ======= DayOfWeek
------
' Example 1 with internal RTC-Clock
_day = 4 : _month = 11 : _year = 2
                                                           ' Load RTC-
Clock for example - testing
Bweekday = Dayofweek()
Strweekday = Lookupstr(bweekday, Weekdays)
Print "Weekday-Number of " ; Date$ ; " is " ; Bweekday ; " = " ;
Strweekday
' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 26 : Bmonth = 11 : Byear = 2
Bweekday = Dayofweek(bday)
Strweekday = Lookupstr(bweekday, Weekdays)
Strdate = Date(bday)
Print "Weekday-Number of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " is " ; Bweekday ; " (" ; Date(bday) ; ") = " ; Strweekday
```

```
' Example 3 with System Day
                                                          ' that is
Wsysday = 2000
2005-06-23
Bweekday = Dayofweek(wsysday)
Strweekday = Lookupstr(bweekday , Weekdays)
Print "Weekday-Number of System Day " ; Wsysday ; " (" ; Date(wsysday) ;
 ") is "; Bweekday; " = "; Strweekday
' Example 4 with System Second
Lsyssec = 123456789
                                                           ' that is
2003-11-29 at 21:33:09
Bweekday = Dayofweek(lsyssec)
Strweekday = Lookupstr(bweekday, Weekdays)
Print "Weekday-Number of System Second " ; Lsyssec ; " (" ; Date(lsyssec
); ") is "; Bweekday; " = "; Strweekday
' Example 5 with Date-String
Strdate = "04-11-02"
                                                           ' we have
configured Date in ANSI
Bweekday = Dayofweek(strdate)
Strweekday = Lookupstr(bweekday, Weekdays)
Print "Weekday-Number of " ; Strdate ; " is " ; Bweekday ; " = " ;
Strweekdav
' ========== Second of Day
-----
' Example 1 with internal RTC-Clock
_sec = 12 : _min = 30 : _hour = 18
                                                          ' Load RTC-
Clock for example - testing
Lsecofday = Secofday()
Print "Second of Day of " ; Time$ ; " is " ; Lsecofday
' Example 2 with defined Clock - Bytes (Second / Minute / Hour)
Bsec = 20 : Bmin = 1 : Bhour = 7
Lsecofday = Secofday(bsec)
Print "Second of Day of Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour
; " ("; Time(bsec); ") is "; Lsecofday
' Example 3 with System Second
Lsyssec = 1234456789
Lsecofday = Secofday(lsyssec)
Print "Second of Day of System Second " ; Lsyssec ; "(" ; Time(lsyssec)
; ") is "; Lsecofday
' Example 4 with Time - String
Strtime = "04:58:37"
Lsecofday = Secofday(strtime)
Print "Second of Day of " ; Strtime ; " is " ; Lsecofday
```

```
' ========= System Second
______
' Example 1 with internal RTC-Clock
                          ' Load RTC-Clock for example - testing
_sec = 17 : _min = 35 : _hour = 8 : _day = 16 : _month = 4 : _year = 3
Lsyssec = Syssec()
Print "System Second of "; Time$; " at "; Date$; " is "; Lsyssec
' Example 2 with with defined Clock - Bytes (Second, Minute, Hour, Day /
Month / Year)
Bsec = 20 : Bmin = 1 : Bhour = 7 : Bday = 22 : Bmonth = 12 : Byear = 1
Lsyssec = Syssec(bsec)
Strtime = Time(bsec)
Strdate = Date(bday)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec
' Example 3 with System Day
Wsysday = 2000
Lsyssec = Syssec(wsysday)
Print "System Second of System Day " ; Wsysday ; " (" ; Date(wsysday) ;
" 00:00:00) is " ; Lsyssec
' Example 4 with Time and Date String
Strtime = "10:23:50"
Strdate = "02-11-29"
                                                            ' ANSI-Date
Lsyssec = Syssec(strtime , Strdate)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec
          ' 91880630
' ====== Day Of Year
_____
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2
                                                            ' Load RTC-
Clock for example - testing
Wdayofyear = Dayofyear()
Print "Day Of Year of " ; Date$ ; " is " ; Wdayofyear
' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wdayofyear = Dayofyear(bday)
Print "Day Of Year of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " (" ; Date(bday) ; ") is " ; Wdayofyear
' Example 3 with Date - String
Strdate = "04-10-29"
                                                            ' we have
configured ANSI Format
Wdayofyear = Dayofyear(strdate)
Print "Day Of Year of " ; Strdate ; " is " ; Wdayofyear
' Example 4 with System Second
```

```
Lsyssec = 123456789
Wdayofyear = Dayofyear(lsyssec)
Print "Day Of Year of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ;
") is " ; Wdayofyear
' Example 5 with System Day
Wsysday = 3000
Wdayofyear = Dayofyear(wsysday)
Print "Day Of Year of System Day " ; Wsysday ; " (" ; Date(wsysday) ; ")
is " ; Wdayofyear
' Example 1 with internal RTC-Clock
_day = 20 : _month = 11 : _year = 2
                                                        ' Load RTC-
Clock for example - testing
Wsysday = Sysday()
Print "System Day of " ; Date$ ; " is " ; Wsysday
' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24: Bmonth = 5: Byear = 8
Wsysday = Sysday(bday)
Print "System Day of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " (" ; Date(bday) ; ") is " ; Wsysday
' Example 3 with Date - String
Strdate = "04-10-29"
Wsysday = Sysday(strdate)
Print "System Day of " ; Strdate ; " is " ; Wsysday
' Example 4 with System Second
Lsyssec = 123456789
Wsysday = Sysday(lsyssec)
Print "System Day of System Second " ; Lsyssec ; " (" ; Date(lsyssec) ;
") is " ; Wsysday
' ====== Time
-----
' Example 1: Converting defined Clock - Bytes (Second / Minute / Hour)
to Time - String
Bsec = 20 : Bmin = 1 : Bhour = 7
Strtime = Time(bsec)
Print "Time values: Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour ; "
converted to string " ; Strtime
' Example 2: Converting System Second to Time - String
Lsyssec = 123456789
Strtime = Time(lsyssec)
Print "Time of Systemsecond " ; Lsyssec ; " is " ; Strtime
' Example 3: Converting Second of Day to Time - String
Lsecofday = 12345
Strtime = Time(lsecofday)
Print "Time of Second of Day " ; Lsecofday ; " is " ; Strtime
```

```
' Example 4: Converting System Second to defined Clock - Bytes (Second /
Minute / Hour)
Lsyssec = 123456789
Bsec = Time(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Sec=" ; Bsec ; " Min="
 ; Bmin ; " Hour=" ; Bhour ; " (" ; Time(lsyssec) ; ")"
' Example 5: Converting Second of Day to defined Clock - Bytes (Second /
Minute / Hour)
Lsecofdav = 12345
Bsec = Time(lsecofday)
Print "Second of Day " ; Lsecofday ; " converted to Sec=" ; Bsec ; "
Min=" ; Bmin ; " Hour=" ; Bhour ; " (" ; Time(lsecofday) ; ")"
' Example 6: Converting Time-string to defined Clock - Bytes (Second /
Minute / Hour)
Strtime = "07:33:12"
Bsec = Time(strtime)
Print "Time "; Strtime; " converted to Sec="; Bsec; " Min="; Bmin;
  ' Hour=" ; Bhour
' ======= Date
-----
' Example 1: Converting defined Clock - Bytes (Day / Month / Year) to
Date - String
Bday = 29 : Bmonth = 4 : Byear = 12
Strdate = Date(bday)
Print "Dat values: Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ; Byear
; " converted to string " ; Strdate
' Example 2: Converting from System Day to Date - String
Wsysday = 1234
Strdate = Date(wsysday)
Print "System Day " ; Wsysday ; " is " ; Strdate
' Example 3: Converting from System Second to Date String
Lsyssec = 123456789
Strdate = Date(lsyssec)
Print "System Second " ; Lsyssec ; " is " ; Strdate
' Example 4: Converting SystemDay to defined Clock - Bytes (Day /
Month / Year)
Wsysday = 2000
Bday = Date(wsysday)
Print "System Day " ; Wsysday ; " converted to Day=" ; Bday ; " Month="
; Bmonth ; " Year=" ; Byear ; " (" ; Date(wsysday) ; ")"
' Example 5: Converting Date - String to defined Clock - Bytes (Day /
Month / Year)
Strdate = "04-08-31"
Bday = Date(strdate)
Print "Date " ; Strdate ; " converted to Day=" ; Bday ; " Month=" ;
```

```
Bmonth ; " Year=" ; Byear
' Example 6: Converting System Second to defined Clock - Bytes (Day /
Month / Year)
Lsyssec = 123456789
Bday = Date(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Day=" ; Bday ; "
Month=" ; Bmonth ; " Year=" ; Byear ; " (" ; Date(lsyssec) ; ")"
' ========== Second of Day elapsed
Lsecofday = Secofday()
hour = hour + 1
Lvar1 = Secelapsed(lsecofday)
Print Lvar1
Lsyssec = Syssec()
_day = _day + 1
Lvar1 = Syssecelapsed(lsyssec)
Print Lvar1
Looptest:
' Initialising for testing
_day = 1
_month = 1
_year = 1
sec = 12
_min = 13
hour = 14
Do
   If _year > 50 Then
      Exit Do
   End If
  _sec = _sec + 7
If _sec > 59 Then
     Incr _min
     _sec = _sec - 60
  End If
  min = min + 2
  If _min > 59 Then
    Incr _hour
     \min = \min - 60
  End If
  hour = hour + 1
  If _hour > 23 Then
     Incr _day
     hour = hour - 24
  End If
  _day = _day + 1
  If day > 28 Then
```

```
Select Case _month
        Case 1
           Mday = 31
        Case 2
           Mday = _year And &H03
           If Mday = 0 Then
             Mday = 29
           Else
             Mday = 28
           End If
        Case 3
           Mday = 31
        Case 4
          Mday = 30
        Case 5
          Mday = 31
        Case 6
          Mday = 30
        Case 7
          Mday = 31
        Case 8
          Mday = 31
        Case 9
          Mday = 30
        Case 10
          Mday = 31
        Case 11
          Mday = 30
        Case 12
          Mday = 31
     End Select
     If _day > Mday Then
        _day = _day - Mday
        Incr _month
        If _month > 12 Then
          _month = 1
           Incr _year
        End If
    End If
 End If
  If _year > 99 Then
    Exit Do
 End If
Lsecofday = Secofday()
Lsyssec = Syssec()
Bweekday = Dayofweek()
Wdayofyear = Dayofyear()
Wsysday = Sysday()
Print Time$ ; " " ; Date$ ; " " ; Lsecofday ; " " ; Lsyssec ; " " ;
Bweekday ; " " ; Wdayofyear ; " " ; Wsysday
Loop
End
'only when we use I2C for the clock we need to set the clock date time
#if Clockmode = 0
'called from datetime.lib
Dim Weekday As Byte
```

```
Getdatetime:
  I2cstart
                                                              ' Generate
start code
  I2cwbyte Ds1307w
                                                              ' send
address
  I2cwbyte 0
                                                              ' start
address in 1307
  I2cstart
                                                              ' Generate
start code
  I2cwbyte Ds1307r
                                                              ' send
address
  I2crbyte _sec , Ack
                                                              ' MINUTES
  I2crbyte _min , Ack
  I2crbyte _hour , Ack
                                                              ' Hours
  I2crbyte Weekday , Ack
                                                              ' Day of
Week
                                                              ' Day of
  I2crbyte _day , Ack
Month
                                                              ' Month of
  I2crbyte _month , Ack
Year
  I2crbyte _year , Nack
                                                              ' Year
  I2cstop
  _sec = Makedec(_sec) : _min = Makedec(_min) : _hour = Makedec(_hour)
  _day = Makedec(_day) : _month = Makedec(_month) : _year = Makedec(
year)
Return
Setdate:
 _day = Makebcd(_day) : _month = Makebcd(_month) : _year = Makebcd(
_year)
 I2cstart
                                                              ' Generate
start code
  I2cwbyte Ds1307w
                                                              ' send
address
  I2cwbyte 4
                                                              ' starting
address in 1307
  I2cwbyte _day
                                                              ' Send Data
to SECONDS
  I2cwbyte _month
                                                              ' MINUTES
                                                              ' Hours
  I2cwbyte _year
  I2cstop
Return
Settime:
  sec = Makebcd( sec) : min = Makebcd( min) : hour = Makebcd( hour)
  I2cstart
                                                              ' Generate
start code
  I2cwbyte Ds1307w
                                                              ' send
address
 I2cwbyte 0
                                                              ' starting
address in 1307
 I2cwbyte _sec
                                                              ' Send Data
to SECONDS
  I2cwbyte _min
                                                              ' MINUTES
                                                              ' Hours
  I2cwbyte _hour
  I2cstop
Return
#endif
Weekdays:
```

```
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" ,
```

"Saturday" , "Sunday"

6.192 DBG

Action

Prints debug info to the hardware UART

Syntax

DBG

Remarks

See <u>\$DBG</u> [318] for more information

6.193 DCF77TIMEZONE

Action

This function will return the offset to Greenwich Time.

Syntax

res = DCF77TimeZone()

Remarks

Res	The target variable that is assigned with the result. The result will be:
	- 0: when there is no valid DCF77 data yet
	- 1: when in "Middle Europe Normal Time"
	- 2: when in "Middle Europe daylight saving Time"

In Middle Europe, daylight saving is used to make better use of the day light in the summer.

The last Sunday in March at 02:00 AM the Daylight Saving will start. All clocks are set from 2:00 to 3:00.

Your weekend, is one hour shorter then.

But the last Sunday of October is better : at 03:00 AM, the Daylight Saving will end and all clocks are set from 03:00 to 02:00.

When you have a lot of clocks in your house, you can understand why DCF77 synchronized clocks are so popular.

See also

CONFIG DCF77 475

Example

Print = DCF77TimeZone()

6.194 **DEBUG**

638

Action

Instruct compiler to start or stop debugging, or print variable to serial port

Syntax

DEBUG ON | OFF | var

Remarks

ON	Enable debugging
OFF	Disable debugging
var	A variable which values must be printed to the serial port

During development of your program a common issue is that you need to know the value of a variable.

You can use PRINT to print the value but then it will be in the application as well. You can use conditional compilation such as :

CONST TEST=1 #IF TEST print var #ENDIF

But that will result in a lot of typing work. The DEBUG option is a combination of conditional compilation and PRINT. Whenever you activate DEBUG with the ON parameter, all 'DEBUG var' statements will be compiled. When you turn DEBUG OFF, all 'DEBUG var' statements will not be compiled.

You can not nest the ON and OFF. The last statements wins. Typical you will have only one DEBUG ON statement. And you set it to OFF when your program is working.

An example showing nesting is NOT supported: DEBUG ON DEBUG ON ' it is still ON DEBUG OFF' it is OFF now

An example showing multiple DEBUG: DEBUG ON DEBUG var ' this is printed DEBUG var2 ' this is also printed

DEBUG OFF DEBUG var3 'this is NOT printed DEBUG var4 ' this is not printed

DEBUG ON 'turn DEBUG ON If A = 2 Then DEBUG A 'this is printed End If

See also

DBG

ASM

NONE

Example

DEBUG ON Dim A As Byte DEBUG A End

6.195 DEBOUNCE

Action

Debounce a port pin connected to a switch.

Syntax

DEBOUNCE Px.y , state , label [, SUB]

Remarks

Px.y	A port pin like PINB.0 , to examine.
State	0 for jumping when PINx.y is low , 1 for jumping when PINx.y is high
Label	The label to GOTO when the specified state is detected
SUB	The label to GOSUB when the specified state is detected

When you specify the optional parameter SUB, a GOSUB to label is performed instead of a GOTO.

The DEBOUNCE statement tests the condition of the specified pin and if true there will be a delay for 25 mS and the condition will be checked again. (eliminating bounce of a switch)

When the condition is still true and there was no branch before, it branches to specified the label.

When the condition is not true, or the logic level on the pin is not of the specified level, the code on the next line will be executed.

When DEBOUNCE is executed again, the state of the switch must have gone back in the original position before it can perform another branch. So if you are waiting for a pin to go low, and the pin goes low, the pin must change to high, before a new low level will result in another branch.

Each DEBOUNCE statement, which uses a different port, uses 1 BIT of the internal memory to hold its state. And as the bits are stored in SRAM, it means that even while you use only 1 pin/bit, a byte is used for storage of the bit.

DEBOUNCE will not wait for the input value to met the specified condition. You need to use BITWAIT if you want to wait until a bit will have a certain value.

So DEBOUNCE will not halt your program while a BITWAIT can halt your program if the bit will never have the specified value. You can combine BITWAIT and DEBOUNCE statements by preceding a DEBOUNCE with a BITWAIT statement.

See also

CONFIG DEBOUNCE 4827, BITWAIT 414

```
Example
                 _____
_____
                       : deboun.bas
'name
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : demonstrates DEBOUNCE
                       : Mega48
'micro
'suited for demo : yes
'commercial addon needed : no
·_____
_____
$regfile = "m48def.dat"
                                                      ' specify
the used micro
$crystal = 4000000
                                                       ' used
crystal frequency
$baud = 19200
                                                       ' use baud
rate
$hwstack = 32
                                                       ' default
use 32 for the hardware stack
\$swstack = 10
                                                       ' default
use 10 for the SW stack
                                                      ' default
$framesize = 40
use 40 for the frame space
Config Debounce = 30
                                                      'when the
config statement is not used a default of 25mS will be used but we
override to use 30 mS
 'Debounce Pind.0 , 1 , Pr 'try this for branching when high(1) Debounce Pind.0 , 0 , Pr , Sub
 Debounce Pind.0 , 0 , Pr , Sub

' ^----- label to branch to
               ^----- Branch when P1.0 goes low(0)
           ^----- Examine P1.0
 'When Pind.0 goes low jump to subroutine Pr
 'Pind.0 must go high again before it jumps again
 'to the label Pr when Pind.0 is low
 Debounce Pind.0 , 1 , Pr
                                                      'no branch
 Debounce Pind.0 , 1 , Pr
                                                      'will result
in a return without gosub
End
Pr:
 Print "PIND.0 was/is low"
Return
```

6.196 DECR

Action

Decrements a variable by one.

Syntax DECR var

Remarks

There are often situations where you want a number to be decreased by 1. It is simpler to write : DECR var compared to : var = var - 1

See also

INCR 736

Example

· · · · · · · · · · · · · · · · · · ·		
'copyright 'purpose 'micro 'suited for demo 'commercial addon needed		
<pre>\$regfile = "m48def.dat"</pre>		' specify
the used micro \$crystal = 4000000		' used
crystal frequency \$baud = 19200		' use baud
rate		
<pre>\$hwstack = 32 use 32 for the hardware st</pre>	ack	' default
\$swstack = 10 use 10 for the SW stack		' default
\$framesize = 40 use 40 for the frame space	2	' default
Dim A As Byte , I As Integ	Jer	
A = 5		'assign
value to a Decr A		'decrease
(by one) Print A		'print it
I = 1000 Decr I Print I		

End

6.197 DECLARE FUNCTION

Action

642

Declares a user function.

Syntax

DECLARE FUNCTION TEST[([BYREF/BYVAL] var as type)] As type

Remarks

test	Name of the function.
Var	Name of the variable(s).
Туре	Type of the variable(s) and of the result. Byte,Word, Dword, Integer, Long, Single, Double or String. Bits are not supported.

When BYREF or BYVAL is not provided, the parameter will be passed by reference. Use BYREF to pass a variable by reference with its address. Use BYVAL to pass a copy of the variable. Use BYLABEL to pass the address of a label.

See the CALL 422 and DECLARE SUB 644 statements for more details.

You must declare each function before writing the function or calling the function. And the declaration must match the function. Bits are global and can not be passed to functions or subs.

When you want to pass a string, you pass it with it's name : string. So the size is not important. For example :

Declare function Test(s as string, byval z as string) as byte

上 When you set the function result, you need to take care that no other code is executed after this. So a good way to set the result would be this :

Function Myfunc(b as byte) as Byte local bDummy as byte 'some code here Myfunc=3 ' assign result ' no other code is executed End Function

Also good would be:

```
Function Myfunc(b as byte) as Byte
  local bDummy as byte
  'some code here
  Myfunc=1 ' assign default result
  Print "this is a test "; b
  Myfunc=4 ' now again the result is the last code
  ' no other code is executed
End Function
```

If you execute other code after you assigned the function result, registers will be

trashed. This is no problem if you assigned the function result to a variable. But when you use a function without assigning it to a variable, some temporarily registers are used which might be trashed.

Thus this special attention is only needed when you use the function like : If Myfunc()=3 then 'myfunc is not assigned to a variable but the result is needed for the test

When you use : myvar=Myfunc() Then you will not trash the registers. So in such a case there is no problem to run code after the function assignment.

To keep it safe, assign the result just before you exit the function.

See also

CALL 4227, SUB 9147, CONFIG SUBMODE 5557

Example

```
-----
                      : function.bas
'name
                      : (c) 1995-2005, MCS Electronics
'copyright
                      : demonstration of user function
'purpose
'micro : Mega48
'suited for demo : yes
'commercial addon needed : no
1_____
                                          _____
------
$regfile = "m48def.dat"
                                                     ' specify
the used micro
$crystal = 4000000
                                                     ' used
crystal frequency
                                                     ' use baud
$baud = 19200
rate
$hwstack = 32
                                                     ' default
use 32 for the hardware stack
\$swstack = 10
                                                     ' default
use 10 for the SW stack
$framesize = 40
                                                     ' default
use 40 for the frame space
'A user function must be declare before it can be used.
'A function must return a type
Declare Function Myfunction(byval I As Integer , S As String) As Integer
'The byval paramter will pass the parameter by value so the original
value
'will not be changed by the function
Dim K As Integer
Dim Z As String * 10
Dim T As Integer
'assign the values
K = 5
Z = "123"
T = Myfunction(k, Z)
```

```
Print T
End
Function Myfunction(byval I As Integer , S As String) As Integer
  'you can use local variables in subs and functions
 Local P As Integer
  P = I
  'because I is passed by value, altering will not change the original
  'variable named k
  I = 10
  P = Val(s) + I
  'finally assign result
  'Note that the same data type must be used !
  'So when declared as an Integer function, the result can only be
  'assigned with an Integer in this case.
 Myfunction = P
End Function
```

6.198 DECLARE SUB

Action

Declares a subroutine.

Syntax

DECLARE SUB TEST[([BYREF/BYVAL/BYLABEL] var as type)]

Remarks

test	Name of the procedure.
Var	Name of the variable(s).
	Type of the variable(s). Byte, Word, Dword, Integer, Long, Single, Double or String.

When BYREF or BYVAL is not provided, the parameter will be passed by reference. Use BYREF to pass a variable by reference with its address. When using the referenced address, you work on the original variable. So a change of the variable inside the sub routine, will change the variable outside the routine as well.

Use BYVAL to pass a copy of the variable. Passing a copy of the variable allows to alter the copy in the sub routine while leaving the original variable unaltered. BYVAL will not change the original passed variable but it requires more code since a copy of the parameter must be created.

Use BYLABEL to pass the address of a label. BYLABEL will pass the word address. It will not work for processors with multiple 64 KB pages.

Using BYLABEL on the EEPROM is possible but the EEPROM image must proceed the call with the label name. See also <u>READEEPROM</u> and <u>LOADLABEL</u> 765.

If you pass a string you may specify the length of the string. This length will be the maximum length the string may grow. This is important when you pass a string BYVAL.

When you for example pass a string like "ABC" to a subroutine or function BYVAL, the

compiler will create a copy with a length of 3. This is sufficient to pass it to the sub routine.

But if the sub routine adds data to the string, it will not fit since the string is too short. In such a case you can specify the length. **s as string * 10**, will create a string with a size of 10.

See the \underline{CALL} statement for more details.

You must declare each function before writing the function or calling the function. And the declaration must match the function. Bits are global and can not be passed with functions or subs.

See also

CALL 422, SUB 914, FUNCTION 642, CONFIG SUBMODE 555

Example

\$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 19200 ' use baud rate \$hwstack = 32 ' default use 32 for the hardware stack \$swstack = 10 ' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space ' First the SUB programs must be declared 'Try a SUB without parameters Declare Sub Test2 'SUB with variable that can not be changed(A) and 'a variable that can be changed(B1), by the sub program 'When BYVAL is specified, the value is passed to the subprogram 'When BYREF is specified or nothing is specified, the address is passed to 'the subprogram

Declare Sub Test(byval A As Byte , B1 As Byte)
Declare Sub Testarray(byval A As Byte , B1 As Byte)
'All variable types that can be passed

```
'Notice that BIT variables can not be passed.
'BIT variables are GLOBAL to the application
Declare Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S
As String)
'passing string arrays needs a different syntax because the length of
the strings must be passed by the compiler
'the empty () indicated that an array will be passed
Declare Sub Teststr(b As Byte , Dl() As String)
Dim Bb As Byte , I As Integer , W As Word , L As Long , S As String * 10
      'dim used variables
Dim Ar(10) As Byte
Dim Sar(10) As String * 8
                                                             'strng array
For Bb = 1 To 10
  Sar(bb) = Str(bb)
                                                             'fill the
arrav
Next
Bb = 1
'now call the sub and notice that we always must pass the first address
with index 1
Call Teststr(bb , Sar(1))
Call Test2
                                                             'call sub
Test2
                                                             'or use
without CALL
'Note that when calling a sub without the statement CALL, the enclosing
parentheses must be left out
Bb = 1
Call Test(1 , Bb)
                                                             'call sub
with parameters
Print Bb
                                                             'print value
that is changed
'now test all the variable types
Call Testvar(bb , I , W , L , S )
Print Bb ; I ; W ; L ; S
'now pass an array
'note that it must be passed by reference
Testarray 2 , Ar(1)
Print "ar(1) = " ; Ar(1)
Print "ar(3) = "; Ar(3)
$notypecheck
                                                             ' turn off
type checking
Testvar Bb , I , I , I , S
'you can turn off type checking when you want to pass a block of memory
$typecheck
                                                             'turn it
back on
End
'End your code with the subprograms
'Note that the same variables and names must be used as the declared
ones
Sub Test(byval A As Byte , B1 As Byte)
                                                             'start sub
    Print A ; " " ; B1
                                                             'print
passed variables
    B1 = 3
                                                             'change
value
    'You can change A, but since a copy is passed to the SUB,
```

```
'the change will not reflect to the calling variable
End Sub
                                                               'sub without
Sub Test2
parameters
    Print "No parameters"
End Sub
Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S As
String)
    Local X As Byte
    X = 5
                                                               'assign
local
    в = Х
    I = -1
    W = 40000
    L = 20000
    S = "test"
End Sub
Sub Testarray(byval A As Byte , B1 As Byte)
                                                               'start sub
    Print A ; " " ; B1
                                                               'print
passed variables
   B1 = 3
                                                               'change
value of element with index 1
                                                               'specify the
   B1(1) = 3
index which does the same as the line above
    B1(3) = 3
                                                               'modify
other element of array
'You can change A, but since a copy is passed to the SUB,
    'the change will not reflect to the calling variable
End Sub
'notice the empty() to indicate that a string array is passed
Sub Teststr(b As Byte , Dl() As String)
 Dl(b) = Dl(b) + "add"
End Sub
```

Example BYLABEL

```
$regfile = "m88def.dat"
$hwstack = 40
$swstack = 80
$framesize = 80
```

Dim B As Byte , W As Word

```
Declare Sub Somesub(bylabel Mylabel As Word)
Somesub Alabel
End
```

```
Sub Somesub(bylabel Mylabel As Word)
W = Mylabel
data
lds _dptrl,{W }
LDS _dptrh,{W+1}
Read B : Print B
End Sub
```

```
Alabel:
Data 1 , 2 , 3
```

6.199 DEFxxx

648

Action

Declares all variables that are not dimensioned of the DefXXX type.

Syntax	
DEFBIT b	Define BIT
DEFBYTE c	Define BYTE
DEFINT I	Define INTEGER
DEFWORD X	Define WORD
DEFLNG	Define LONG
DEFSNG s	Define SINGLE
DEFDBL z	Define DOUBLE

Remarks

While you can DIM each individual variable you use, you can also let the compiler handle it for you.

All variables that start with a certain letter will then be dimmed as the specified type.

Example

Defbit b : DefInt c 'default type for bit and integers

Set b1 'set bit to 1

c = 10 'let c = 10

6.200 DEFLCDCHAR

Action

Define a custom LCD character.

Syntax

DEFLCDCHAR char,r1,r2,r3,r4,r5,r6,r7,r8

Remarks

char	Constant representing the character (0-7).
r1-r8	The row values for the character.

You can use the <u>LCD designer</u> 79 to build the characters.

It is important that a CLS follows the DEFLCDCHAR statement(s). So make sure you use the DEFLCDCHAR before your CLS statement.

Special characters can be printed with the $\underline{Chr}_{428}()$ function.

LCD Text displays have a 64 byte memory that can be used to show your own custom

649

characters. Each character uses 8 bytes as the character is an array from 8x8 pixels. You can create a maximum of 8 characters this way. Or better said : you can show a maximum of 8 custom characters at the same time. You can redefine characters in your program but with the previous mentioned restriction.

A custom character can be used to show characters that are not available in the LCD font table. For example a \hat{U}_{\cdot}

You can also use custom characters to create a bar graph or a music note.

See also

Tools LCD designer 79

Partial Example

```
Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228 '
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240 '
replace ? with number (0-7)
Cls 'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0) ; Chr(1) 'print the
special character
```

6.201 DEG2RAD

Action

Converts an angle in to radians.

Syntax

var = DEG2RAD(Source)

Remarks

	A numeric variable that is assigned with the degrees of variable Source.
Source	The single or double variable to get the degrees of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG 822

```
'copyright : (c) 1995-2005, MCS Electronics
'micro : Mega48
'suited for demo : yes
'commercial addon needed : no
'purpose : demonstrates DEG2RAD function
```

```
Dim S As Single
S = 90
S = Deg2Rad(s)
Print S
S = Rad2deg(s)
Print S
End
```

6.202 DELAY

Action

Delay program execution for a short time.

_ _ _ _ _ _ _

Syntax

DELAY

Remarks

Use DELAY to wait for a short time. The delay time is ca. 1000 microseconds.

Interrupts that occur frequently and/or take a long time to process, will let the delay last longer.

When you need a very accurate delay, you need to use a timer.

See also

WAIT 948 , WAITMS 950

· ·		
'commercial addon needed	: delay.bas : (c) 1995-2005, MCS Electronics : demo: DELAY, WAIT, WAITMS : Mega48 : yes : no	
<pre>\$regfile = "m48def.dat" the used micro \$crystal = 4000000 crystal frequency \$baud = 19200 rate \$hwstack = 32 use 32 for the hardware s \$swstack = 10 use 10 for the SW stack \$framesize = 40 use 40 for the frame space</pre>		' specify ' used ' use baud ' default ' default ' default

```
Ddrb = &HFF
                                                              'port B as
output
Portb = 255
Print "Starting"
Delay
                                                               'lets wait
for a very short time
Print "Now wait for 3 seconds"
Portb = 0
Wait 3
Print "Ready"
Waitms 10
                                                              'wait 10
milliseconds
Portb = 255
End
```

6.203 DELCHAR

Action

Delete one character from a string.

Syntax

DELCHAR string, pos

Remarks

string	The string where the character is removed from.
pos	The position where the character must be removed from.
-	A value of 1 would remove the first character.

Do not confuse with the DELCHAR**S** statement which removes all characters based on a character value.

The DELCHAR removes one character from a string based on an index.

See also

DELCHARS [652], INSERTCHAR 746, INSTR 747, MID 787, CHARPOS 424, REPLACECHARS [840]

```
' (c) 1995-2011, MCS
' del_insert_chars.bas
' This sample demonstrates the delchar, delchars and insertchar
statements
'
$regfile="m88def.dat"
$crystal = 8000000
$hwstack = 40
$swstack = 40
$framesize = 40
```

```
dim s as string * 30
s = "This is a test string" ' create a string
delchar s, 1 ' remove the first char
print s ' print it
insertchar s,1, "t" ' put a small t back
print s
delchars s,"s" ' remove all s
print s
end
```

6.204 DELCHARS

Action

Delete all character from a string matching the provided character value.

Syntax

DELCHARS string, value

Remarks

string	The string where the characters are removed from.
	The value of the character which must be removed from the string. You can use "A" to remove all capital A characters. Or you can pass a byte with the value of 65 to remove all characters with ASCII value 65 (A)

Do not confuse with the DELCHAR statement which removes one character based on an index value.

DELCHARS removes ALL characters from a string matching value.

See also

DELCHAR [651], INSERTCHAR [746], INSTR [747], MID [787], CHARPOS [424], REPLACECHARS [840]

653

```
s = "This is a test string" ' create a string
delchar s, 1 ' remove the first char
print s ' print it
insertchar s,1, "t" ' put a small t back
print s
delchars s,"s" ' remove all s
print s
end
```

6.205 DIM

Action

Dimension a variable.

Syntax

DIM var AS [XRAM/SRAM/ERAM]type [AT location/variable] [OVERLAY]

Remarks

Var	Any valid variable name such as b1, i or longname. var can also be an array : ar(10) for example.
Туре	Bit, Byte, Word, Integer, Long, Dword, Single, Double or String
XRAM	Specify XRAM to store variable into external memory
SRAM	Specify SRAM to store variable into internal memory (default)
ERAM	Specify ERAM to store the variable into EEPROM
OVERLAY	Specify that the variable is overlaid in memory.
location	The address of name of the variable when OVERLAY is used.

A string variable needs an additional length parameter: *Dim s As XRAM String * 10*

In this case, the string can have a maximum length of 10 characters. Internally one additional byte is needed to store the end of string marker. Thus in the example above, 11 bytes will be used to store the string.

Note that BITS can only be stored in internal memory.

You may also specify IRAM. IRAM is the place in memory where the registers are located : absolute address 0 - 31. BASCOM uses most of these addresses, depending on the instructions/options you use. For a $\frac{TINY}{368}$ chip it makes sense to use IRAM since there is NO SRAM in most tiny AVR chips (TINY15 for example). You may also use to IRAM to overlay registers in memory.

SCOPE

The scope for DIM is global. So no matter where you use the DIM statements, the variable will end up as a global visible variable that is visible in all modules, procedures and functions.

When you need a LOCAL variable that is local to the procedure or function, you can

use LOCAL 768.

Since LOCAL variables are stored on the frame, it takes more code to dynamic generate and clean up these variables.

AT

The optional **AT** parameter lets you specify where in memory the variable must be stored. When the memory location already is occupied, the first free memory location will be used. You need to look in the report file to see where the variable is located in memory.

OVERLAY

The **OVERLAY** option will not use any variable space. It will create a sort of phantom variable:

Dim x as Long at \$60 'long uses 60,61,62 and 63 hex of SRAM

Dim b1 as Byte at \$60 OVERLAY Dim b2 as Byte at \$61 OVERLAY

B1 and B2 are no real variables! They refer to a place in memory. In this case to &H60 and &H61. By assigning the phantom variable B1, you will write to memory location &H60 that is used by variable X.

So to define it better, OVERLAY does create a normal usable variable, but it will be stored at the specified memory location which could be already be occupied by another OVERLAY variable, or by a normal variable.

Take care with the OVERLAY option. Use it only when you understand it.

You can also read the content of B1: Print B1 This will print the content of memory location &H60.

By using a phantom variable you can manipulate the individual bytes of real variables.

Another example

Dim L as Long at &H60 Dim W as Word at &H62 OVERLAY

W will now point to the upper two bytes of the long.

Using variable name instead of address

As variables can be moved though the program during development it is not always convenient to specify an address. You can also use the name of the variable :

DIM W as WORD Dim B as BYTE AT W OVERLAY

Now B is located at the same address as variable W.

For XRAM variables, you need additional hardware : an external RAM and address decoder chip.

ERAM

For ERAM variables, it is important to understand that these are not normal variables. ERAM variables serve as a way to simple read and write the EEPROM memory. You can use READEEPROM and WRITEEEPROM for that purpose too.

To write to an ERAM variable you have to use an SRAM variable as the source : eramVAR= sramVAR

To read from an ERAM variable you have to use an SRAM variable as the targer : sramVAR=eramVAR

Both variables need to be of the same data type. So when writing to an ERAM double, the source variable need to be of the double type too.

ERAM can be assigned with a numeric value too : eramVAR= 123

You can not use an ERAM variable as you would use a normal variable.

Also keep in mind that when you write to ERAM, you write to EEPROM, and that after 100.000 times, the EEPROM will not erase properly.

Dim b as byte, bx as ERAM byte B= 1 Bx=b ' write to EEPROM B=bx ' read from EEPROM

Xmega

The XMEGA need an additional configuration command : <u>CONFIG EEPROM</u> = MAPPED, in order to use ERAM.

See Also

CONST 5947, LOCAL 7687

Example

'	
'name	: dim.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: demo: DIM
'micro	: Mega48
'suited for demo	: yes
'commercial addon needed	: no
· ·	

\$regfile = "m48def.dat" ' specify the used micro ' used \$crystal = 4000000crystal frequency ' use baud **Sbaud = 19200** rate ' default **\$hwstack = 32** use 32 for the hardware stack \$swstack = 10 ' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Dim B1 As Bit 'bit can be

```
0 or 1
Dim A As Byte
                                                              'byte range
from 0-255
Dim C As Integer
                                                              'integer
range from -32767 - +32768
Dim L As Long
Dim W As Word
Dim S As String * 11
                                                               'length can
be up to 11 characters
'new feature : you can specify the address of the variable
Dim K As Integer At &H120
'the next dimensioned variable will be placed after variable s
Dim Kk As Integer
'Assign bits
B1 = 1
                                                              'or
Set B1
                                                              'use set
'Assign bytes
A = 12
A = A + 1
'Assign integer
C = -12
C = C + 100
Print C
W = 50000
Print W
'Assign long
L = 12345678
Print L
'Assign string
S = "Hello world"
Print S
End
```

6.206 DIR

Action

Returns the filename that matches the specified file mask.

Syntax

sFile = **DIR**(mask) sFile = **DIR**()

Remarks

SFile	A string variable that is assigned with the filename.
Mask	A file mask with a valid DOS file mask like *.TXT
	Use *.* to select all files.

The first function call needs a file mask. All other calls do not need the file mask. In fact when you want to get the next filename from the directory, you must not provide

a mask after the first call.

Dir() returns an empty string when there are no more files or when no file name is found that matches the mask.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [695], FILEATTR [684], SEEK [680], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], DISKSIZE [667], GET [697], PUT [819], FILELEN [687], FILEDATE [686], FILETIME [687], FILEDATETIME [686], WRITE [953], INPUT [744]

ASM

Calls	_Dir ; with file mask	_Dir0 ; without file mask
· ·	X : points to the string with the mask	Z : points to the target variable
Output		

Partial Example

'Lets have a look at the file we created Print "Dir function demo" S = Dir("*.*") 'The first call to the DIR() function must contain a file mask ' The * means everything.

While Len(s)> 0 ' if there was a file found Print S ;" ";Filedate();" ";Filetime();" ";Filelen() ' print file , the date the fime was created/changed , the time and the size of the file S = Dir()' get next Wend

6.207 DISABLE

Action

Disable specified interrupt.

Syntax

DISABLE interrupt [device]

Remarks

Interrupt	Description
INTO	External Interrupt 0
INT1	External Interrupt 1
OVF0,TIMER0, COUNTER0	TIMER0 overflow interrupt
OVF1,TIMER1,	TIMER1 overflow interrupt
COUNTER1	
CAPTURE1, ICP1	INPUT CAPTURE TIMER1 interrupt

COMPARE1A,OC1A	TIMER1 OUTPUT COMPARE A interrupt
COMPARE1B,OC1B	TIMER1 OUTPUT COMPARE B interrupt
SPI	SPI interrupt
URXC	Serial RX complete interrupt
UDRE	Serial data register empty interrupt
UTXC	Serial TX complete interrupt
SERIAL	Disables URXC, UDRE and UTXC
ACI	Analog comparator interrupt
ADC	A/D converter interrupt

By default all interrupts are disabled.

To disable all interrupts specify INTERRUPTS.

To enable the enabling and disabling of individual interrupts use ENABLE INTERRUPTS.

The ENABLE INTERRUPTS serves as a master switch. It must be enabled/set in order for the individual interrupts to work.

The interrupts that are available will depend on the used microprocessor. The available interrupts are shown automatically in the editor.

To disable the JTAG you can use DISABLED JTAG. The JTAG is not an interrupt but a device.

See also

ENABLE 676

Example

```
'name
                    : serint.bas
                    : (c) 1995-2005, MCS Electronics
'copyright
                    : serial interrupt example for AVR
'purpose
                    : 90S8535
'micro
'suited for demo
                    : yes
'commercial addon needed : no
------
                       _____
 _____
$regfile = "8535def.dat"
                                                ' specify
the used micro
$crystal = 4000000
                                                ' used
crystal frequency
$baud = 19200
                                                ' use baud
rate
$hwstack = 32
                                                ' default
use 32 for the hardware stack
                                                ' default
$swstack = 10
use 10 for the SW stack
$framesize = 40
                                                ' default
use 40 for the frame space
Const Cmaxchar = 20
                                                'number of
characters
```

© 2011 MCS Electronics

659

Dim B As Bit	'a flag for
signalling a received character Dim Bc As Byte	'byte
counter Dim Buf As String * Cmaxchar	'serial
buffer Dim D As Byte	
'Buf = Space(20)	
'unremark line above for the MID() function in the 'we need to fill the buffer with spaces otherwise i garbage	
Print "Start"	
On Urxc Rec_isr serial receive ISR	'define
Enable Urxc receive isr	'enable
Enable Interrupts interrupts to occur	'enable
Do If B = 1 Then	'we received
something Disable Serial	
Print Buf buffer	'print
Print Bc character counter	'print
'now check for buffer full If Bc = Cmaxchar Then	'buffer full
Buf = "" Bc = 0	'clear 'rest
character counter End If	
Reset B	'reset
receive flag Enable Serial	
End If Loop	
Rec_isr:	
Print "*" If Bc < Cmaxchar Then	'does it fit
into the buffer? Incr Bc	'increase
buffer counter	
If Udr = 13 Then	'return?
Buf = Buf + Chr (0) Bc = Cmaxchar	
Else Buf = Buf + Chr(udr)	'add to
buffer End If	

```
' Mid(buf , Bc , 1) = Udr
'unremark line above and remark the line with Chr() to place
'the character into a certain position
'B = 1 'set flag
End If
B = 1 'set flag
Return
```

6.208 DISKFREE

Action

Returns the free size of the Disk in KB.

Syntax

IFreeSize = DISKFREE()

Remarks

IFreeSize A Long Variable, which is assigned with the available Bytes on the Disk in Kilo Bytes.

This functions returns the free size of the disk in KB. With the support of FAT32, the return value was changed from byte into KB.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [698], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKSIZE [667], GET [697], PUT [819], FILEDATE [688], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_GetDiskFreeSize
Input	none
Output	r16-r19: Long-Value of free Bytes

Partial Example

Dim Gbtemp1 As Byte ' scratch byte
Gbtemp1 =Initfilesystem(1) ' we must init the filesystem once
If Gbtemp1 > 0 Then
 Print#1 ,"Error "; Gbtemp1
Else
 Print#1 ," OK"
Print "Disksize : ";Disksize() ' show disk size in bytes
Print "Disk free: ";Diskfree() ' show free space too
End If

6.209 DISKSIZE

Action

Returns the size of the Disk in KB.

Syntax

ISize = DISKSIZE()

Remarks

lSize	A Long Variable, which is assigned with the capacity of the disk in Kilo
	Bytes

This functions returns the capacity of the disk in KB. With the support of FAT32, the return value was changed from byte into KB.

See also

INITFILESYSTEM [737), OPEN [794), CLOSE [437), FLUSH [689), PRINT [810), LINE INPUT [762), LOC [766], LOF [767], EOF [687], FREEFILE [698], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [751], DISKFREE [660], GET [697], PUT [819], FILEDATE [688], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_GetDiskSize
Input	none
Output	16-r19: Long-Value of capacity in Bytes

Partial Example

Dim Gbtemp1 As Byte' scratch byte
Gbtemp1 = Initfilesystem(1)' we must init the filesystem once
If Gbtemp1 > 0 Then
 Print#1 ,"Error "; Gbtemp1
Else
 Print#1 ," OK"
Print "Disksize : "; Disksize()' show disk size in bytes
Print "Disk free: "; Diskfree()' show free space too
End If

6.210 DISPLAY

Action

Turn LCD display ON or OFF.

Syntax

DISPLAY ON | OFF DISPLAY ON | OFF , CURSOR | NOCURSOR , BLINK | NOBLINK

Remarks

The display is turned on at power up.

When you use DISPLAY with a single parameter, the compiler will maintain a variable to hold the status of the display. In some cases this can lead to an unexpected result. This depends on the order of how the commands are called.

If you experience this problem, you can use the alternative syntax which demands that all 3 parameters are specified.

This does not use any state variable and will update the LCD command register. The second syntax is advised to be used.

See also

LCD 752

Example

_____ _____ : lcd.bas 'name : (c) 1995-2005, MCS Electronics 'copyright : demo: LCD, CLS, LOWERLINE, SHIFTLCD, 'purpose SHIFTCURSOR, HOME CURSOR, DISPLAY 'micro : Mega8515 'suited for demo : yes 'commercial addon needed : no ·_____ _____ _____ \$regfile = "m8515.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 19200 ' use baud rate ' default \$hwstack = 32use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Śsim 'REMOVE the above command for the real program !! '\$sim is used for faster simulation 'note : tested in PIN mode with 4-bit 'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 , Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 , Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6 'These settings are for the STK200 in PIN mode 'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-D7 'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of the LCD connector 'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector

```
Rem with the config lcdpin statement you can override the compiler
settings
Dim A As Byte
Config Lcd = 16 * 2
                                                                 'configure
lcd screen
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a
'When you dont include this option 16 * 2 is assumed
'16 * la is intended for 16 character displays with split addresses over
2 lines
'$LCD = address will turn LCD into 8-bit databus mode
    use this with uP with external RAM and/or ROM
τ.
        because it aint need the port pins !
Cls
                                                                  'clear the
LCD display
Lcd "Hello world."
                                                                  'display
this at the top line
Wait 1
Lowerline
                                                                 'select the
lower line
Wait 1
Lcd "Shift this."
                                                                 'display
this at the lower line
Wait 1
For A = 1 To 10
  Shiftlcd Right
                                                                 'shift the
text to the right
   Wait 1
                                                                 'wait a
moment
Next
For A = 1 To 10
                                                                 'shift the
   Shiftlcd Left
text to the left
   Wait 1
                                                                 'wait a
moment
Next
Locate 2 , 1
                                                                 'set cursor
position
Lcd "*"
                                                                 'display
this
Wait 1
                                                                 'wait a
moment
                                                                  'shift the
Shiftcursor Right
cursor
Lcd "@"
                                                                 'display
this
Wait 1
                                                                 'wait a
moment
Home Upper
                                                                 'select line
1 and return home
Lcd "Replaced."
                                                                 'replace the
text
Wait 1
                                                                 'wait a
moment
```

Cursor Off Noblink 'hide cursor Wait 1 'wait a moment Cursor On Blink 'show cursor Wait 1 'wait a moment Display Off 'turn display off Wait 1 'wait a moment 'turn Display On display on '-----NEW support for 4-line LCD-----Thirdline Lcd "Line 3" Fourthline Lcd "Line 4" Home Third 'goto home on line three Home Fourth Home F 'first letteer also works Locate 4 , 1 : Lcd "Line 4" Wait 1 'Now lets build a special character 'the first number is the characternumber (0-7)'The other numbers are the rowvalues 'Use the LCD tool to insert this line **Deflcdchar** 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228 replace ? with number (0-7)**Deflcdchar** 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240 1 replace ? with number (0-7)Cls 'select data RAM Rem it is important that a CLS is following the deflcdchar statements because it will set the controller back in datamode Lcd Chr(0); Chr(1)'print the special character '----- Now use an internal routine ------ $_temp1 = 1$ 'value into ACC !rCall _write_lcd 'put it on LCD End

6.211 DO-LOOP

Action

Repeat a block of statements until condition is true.

Syntax DO statements LOOP [UNTIL expression]

Remarks

You can exit a DO..LOOP with the EXIT DO statement. The DO-LOOP is always performed at least once.

The main part of your code can best be executed within a DO.. LOOP. You could use a GOTO also but it is not as clear as the DO LOOP. Main: ' code GOTO Main

Do 'Code Loop

Of course in the example above, it is simple to see what happens, but when the code consist of a lot of lines of code, it is not so clear anymore what the GOTO Main does.

See also

EXIT 6827, WHILE-WEND 9527, FOR-NEXT 697

```
/_____
'name
                     : do_loop.bas
                     : (c) 1995-2005, MCS Electronics
'copyright
                     : demo: DO, LOOP
'purpose
                     : Mega48
'micro
'suited for demo
                     : yes
'commercial addon needed : no
·_____
  _____
$regfile = "m48def.dat"
                                                  ' specify
the used micro
                                                  ' used
$crystal = 4000000
crystal frequency
$baud = 19200
                                                  ' use baud
rate
\$hwstack = 32
                                                  ' default
use 32 for the hardware stack
$swstack = 10
                                                  ' default
use 10 for the SW stack
$framesize = 40
                                                  ' default
use 40 for the frame space
Dim A As Byte
A = 1
                                                  'assign a
var
                                                  'begin a
Do
do..loop
 Print A
                                                  'print var
 Incr A
                                                  'increase by
one
Loop Until A = 10
                                                  'do until
a=10
End
'You can write a never-ending loop with the following code
```

```
666
```

```
Do
'Your code goes here
Loop
```

6.212 DriveCheck

Action

Checks the Drive, if it is ready for use

Syntax

bErrorCode = **DRIVECHECK**()

Remarks

bErrorCode	A Byte Variable, which is assigned with the return value of the
	function

This function checks the drive, if it is ready for use (for example, whether a compact flash card is inserted). The functions returns 0 if the drive can be used, otherwise an error code is returned. For Error code see section Error codes.

See also

<u>DriveReset</u> هما , <u>DriveInit</u> من <u>DriveGetIdentity</u> من <u>DriveWriteSector</u> من , <u>DriveReadSector</u> من ,

ASM

Calls	_DriveCheck	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

Dim bError as Byte bError = DriveCheck()

6.213 DriveGetIdentity

Action

Returns the Parameter information from the Card/Drive

Syntax

bErrorCode = **DRIVEGETIDENTIFY**(wSRAMPointer)

Remarks

BErrorCode	A Byte Variable, which is assigned with the error code of the function
wSRAMPoint	A Word Variable, which contains the SRAM address (pointer) , to which
er	the information of the Drive should be written

The Identify Drive Function returns the parameter information (512 Bytes) from the Compact Flash Memory Card/Drive and writes it to SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing. This information are for example number of sectors of the card, serial number and so on. Refer to the Card/Drive manual for further information. The functions returns 0 if no error occurred. For Error code see section Error codes.

Note: For meaning of wSRAMPointer see Note in DriveReadSector

See also

DriveCheck (and), DriveReset (and), DriveInit (and), DriveWriteSector (and), DriveReadSector (and)

ASM

Calls	_DriveGetIdentity	
Input		Z: SRAM-Address of buffer *)
Output	r25: Errorcode	C-Flag: Set on Error

*) Please note: This is not the address of wSRAMPointer, it is its content, which is the starting-address of the buffer.

Partial Example

Dim bError as Byte Dim aBuffer(512) as Byte' Hold Sector to and from CF-Card Dim wSRAMPointer as Word' Address-Pointer for write

' give Address of first Byte of the 512 Byte Buffer to Word-Variable wSRAMPointer =VarPtr(aBuffer(1))

' Now read the parameter Information from CF-Card bError = DriveGetIdentity(wSRAMPointer)

6.214 Drivelnit

Action

Sets the AVR-Hardware (PORTs, PINs) attached to the Drive and resets the Drive.

Syntax

bErrorCode = **DRIVEINIT**()

Remarks

	A Byte Variable, which is assigned with the error code of the function
--	--

Set the Ports and Pins attaching the Drive for Input/Output and give initial values to the output-pins. After that the Drive is reset. Which action is done in this function

depends of the drive and its kind of connection to the AVR. The functions returns 0 if no error occurred. For Error code see section Error codes.

See also

<u>DriveCheck</u> همه , <u>DriveReset</u> همه , <u>DriveGetIdentity</u> همه , <u>DriveWriteSector</u> همه , <u>DriveReadSector</u>

ASM

Calls	_DriveInit	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

Dim bError as Byte bError = DriveInit()

6.215 DriveReset

Action

Resets the Drive.

Syntax

bErrorCode = **DRIVERESET**()

Remarks

BErrorCode A Byte Variable, which is assigned with the error code of the function

This function resets the drive and brings it to an initial state. The functions returns 0 if no error occurred. For Error code see section Error codes.

See also

<u>DriveCheck</u> همه <u>DriveInit</u> مراجع <u>DriveGetIdentity</u> مراجع <u>DriveWriteSector</u> مراجع <u>DriveReadSector</u> مراجع

ASM

Calls	_DriveReset	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

Dim bError as Byte bError = DriveReset()

6.216 DriveReadSector

Action

Read a Sector (512 Bytes) from the (Compact Flashcard) Drive

Syntax

bErrorCode = **DRIVEREADSECTOR**(wSRAMPointer, ISectorNumber)

Remarks

	A Byte Variable, which is assigned with the error code of the function
	A Word Variable, which contains the SRAM address (pointer) , to which the Sector from the Drive should be written
ISectorNumber	A Long Variable, which give the sector number on the drive be transfer.

Reads a Sector (512 Bytes) from the Drive and write it to SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing. The functions returns 0 if no error occurred. For Error code see section Error codes.

Note: wSRAMPointer is not the variable, to which the content of the desired drivesector should be written, it is the Word-Variable/Value which contains the SRAM address of the range, to which 512 Bytes should be written from the Drive. This gives you the flexibility to read and write every SRAM-Range to and from the drive, even it is not declared as variable. If you know the SRAM-Address (from the compiler report) of a buffer you can pass this value directly, otherwise you can get the address with the BASCOM-function VARPTR (see example).

See also

DriveCheck 6667, DriveReset 6687, DriveInit 6677, DriveGetIdentity 6667, DriveWriteSector 677

ASM

Calls	_DriveReadSector	
Input	Z: SRAM-Address of buffer *)	X: Address of Long-variable with sectornumber
Output	r25: Errorcode	C-Flag: Set on Error

This is not the address of wSRAMPointer, it is its content, which is the startingaddress of the buffer.

Partial Example

Dim bError as Byte Dim aBuffer(512)as Byte' Hold Sector to and from CF-Card Dim wSRAMPointer as Word' Address-Pointer for write Dim ISectorNumber as Long' Sector Number ' give Address of first Byte of the 512 Byte Buffer to Word-Variable wSRAMPointer =VarPtr(aBuffer(1))

' Set Sectornumber, sector 32 normally holds the Boot record sector of first partition ISectorNumber = 32

' Now read in sector 32 from CF-Card

bError = DriveReadSector(wSRAMPointer , ISectorNumber)

' Now Sector number 32 is in Byte-Array bBuffer

6.217 DriveWriteSector

Action

Write a Sector (512 Bytes) to the (Compact Flashcard) Drive

Syntax

bErrorCode = **DRIVEWRITESECTOR**(wSRAMPointer, lSectorNumber)

Remarks

bErrorCode	A Byte Variable, which is assigned with the error code of the function
	A Word Variable, which contains the SRAM address (pointer), from
r	which the Sector to the Drive should be written
ISectorNumb	A Long Variable, which give the sector number on the drive to
er	transfer.

Writes a Sector (512 Bytes) from SRAM starting at the address, to which the content of the variable wSRAMPointer is pointing to the Drive to sector number ISectornumber. The functions returns 0 if no error occurred. For Error code see section Error codes.

For the meaning of wSRAMPointer see Note in DriveReadSector

See also

DriveCheck (and), DriveReset (and), DriveInit (and), DriveGetIdentity (and), DriveReadSector

ASM

Calls	_DriveWriteSector	
Input	Z: SRAM-Address of buffer *)	X: Address of Long-variable with sectornumber
Output	r25: Errorcode	C-Flag: Set on Error

This is not the address of wSRAMPointer, it is its content, which is the startingaddress of the buffer.

Partial Example

Dim bError as Byte Dim aBuffer(512) as Byte' Hold Sector to and from CF-Card Dim wSRAMPointer as Word' Address-Pointer for read Dim ISectorNumber as Long' Sector Number

' give Address of first Byte of the 512 Byte Buffer to Word-Variable wSRAMPointer =VarPtr(aBuffer(1))

' Set Sectornumber

ISectorNumber = 3

' Now Write in sector 3 from CF-Card bError = DriveWriteSector(wSRAMPointer , ISectorNumber)

6.218 DTMFOUT

Action

Sends a DTMF tone to the compare1 output pin of timer 1.

Syntax

DTMFOUT number, duration **DTMFOUT** string , duration

Remarks

	A variable or numeric constant that is equivalent with the number of your phone keypad.
Duration	Time in mS the tone will be generated.
string	A string variable that holds the digits to be dialed.

The DTMFOUT statement is based on an Atmel application note (314).

It uses TIMER1 to generate the dual tones. As a consequence, timer1 can not be used in interrupt mode by your application. You may use it for other tasks.

Since the TIMER1 is used in interrupt mode you must enable global interrupts with the statement ENABLE INTERRUPTS [676]. The compiler could do this automatic but when you use other interrupts as well it makes more sense that you enable them at the point where you want them to be enabled.

The working range is from 4 MHz to 10 MHz system clock(xtal).

The DTMF output is available on the TIMER1 OCA1 pin. For a 2313 this is PORTB.3.

Take precautions when connecting the output to your telephone line.

Ring voltage can be dangerous!

System Resources used

TIMER1 in interrupt mode

See also

NONE

ASM

The following routine is called from mcs.lib : _DTMFOUT R16 holds the number of the tone to generate, R24-R25 hold the duration time in mS. Uses R9,R10,R16-R23

The DTMF table is remarked in the source and shown for completeness, it is generated by the compiler however with taking the used crystal in consideration.

```
Example
 _____
                       : dtmfout.bas
'name
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : demonstrates DTMFOUT statement based on AN
314 from Atmel
'micro : Mega48
'suited for demo : yes
'commercial addon needed : no
_____
$regfile = "m48def.dat"
                                                        ' specify
the used micro
$crystal = 8000000
                                                        ' used
crystal frequency
$baud = 19200
                                                        ' use baud
rate
$hwstack = 32
                                                        ' default
use 32 for the hardware stack
$swstack = 10
                                                        ' default
use 10 for the SW stack
$framesize = 40
                                                        ' default
use 40 for the frame space
'since the DTMFOUT statement uses the TIMER1 interrupt you must enable
'global interrupts
'This is not done by the compiler in case you have more ISRs
Enable Interrupts
'the first sample does dtmfout in a loop
Dim Btmp As Byte , Sdtmf As String * 10
Sdtmf = "12345678"
                                                        ' number to
dial
Do
Dtmfout Sdtmf , 50
                                                       ' lets dial a
number
               ^ duration is 50 mS for each digit
Waitms 1000
                                                       ' wait for
one second
```

```
' As an alternative you can send single digits
' there are 16 dtmf tones
 For Btmp = 0 To 15
                                                                  ' dtmf out
  Dtmfout Btmp , 50
on PORTB.3 for the 2313 for 500 \ensuremath{\,\mathrm{mS}}
    'output is on the OC1A output pin
    Waitms 500
                                                                  ' wait 500
msec
Next
Loop
End
'the keypad of most phones looks like this :
'1 2 3
            optional are A
'4 5 6
                           В
'7 8 9
                           С
۰ *
    0
      #
                           D
'the DTMFOUT translates a numeric value from 0-15 into :
' numeric value phone key
    0
                      0
    1
                      1
    2
                      2
÷
    3
                      3
÷
  etc.
   9
                      9
   10
                       *
   11
                       #
  12
                      Α
  13
                      В
  14
                      С
1
   15
                      D
```

6.219 ECHO

Action

Turns the ECHO on or off while asking for serial INPUT.

Syntax

ECHO value

Remarks

Value ON to enable ECHO and OFF to disable ECHO.
--

When you use INPUT to retrieve values for variables, all info you type can be echoed back. In this case you will see each character you enter. When ECHO is OFF, you will not see the characters you enter.

In versions 1.11.6.2 and earlier the ECHO options were controlled by an additional parameter on the INPUT statement line like : INPUT "Hello " , var NOECHO

This would suppress the ECHO of the typed data. The new syntax works by setting ECHO ON and OFF. For backwards compatibility, using NOECHO on the INPUT statement line will also work. In effect it will turn echo off and on automatic.

By default, ECHO is always ON.

See also

INPUT 744

ASM

The called routines from mcs.lib are _ECHO_ON and _ECHO_OFF

The following ASM is generated when you turn ECHO OFF. Rcall Echo_Off This will set bit 3 in R6 that holds the ECHO state.

When you turn the echo ON the following code will be generated Rcall $\mathsf{Echo}_\mathsf{On}$

Example

_____ 'name : input.bas : (c) 1995-2005, MCS Electronics 'copyright 'purpose : demo: INPUT, INPUTHEX 'micro : Mega48 'suited for demo : yes 'commercial addon needed : no 1_____ _____ _____ \$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10 ' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Dim V As Byte , B1 As Byte Dim C As Integer , D As Byte Dim S As String * 15 **Input** "Use this to ask a question ", V Input B1 'leave out for no question Input "Enter integer " , C Print C Inputhex "Enter hex number (4 bytes) " , C Print C Inputhex "Enter hex byte (2 bytes) " , D Print D Input "More variables " , C , D

```
Print C ; " " ; D
Input C Noecho 'supress
echo
Input "Enter your name " , S
Print "Hello " ; S
Input S Noecho 'without
echo
Print S
End
```

6.220 ELSE

Action

Executed if the IF-THEN expression is false.

Syntax ELSE

ELSE

Remarks

You don't have to use the ELSE statement in an IF THEN \dots END IF structure. You can use the ELSEIF statement to test for another condition.

IF a = 1 THEN ... ELSEIF a = 2 THEN ... ELSEIF b1 > a THEN ... ELSE ... END IF

See also

IF 7357, END IF 7357, SELECT-CASE 8527

Example

```
_____
                    : if_then.bas
'name
                    : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                    : demo: IF, THEN, ELSE
                    : Mega48
'micro
'suited for demo
                    : yes
'commercial addon needed : no
! _____
                                     _____
 ------
$regfile = "m48def.dat"
                                                 ' specify
the used micro
$crystal = 4000000
                                                 ' used
crystal frequency
$baud = 19200
                                                 ' use baud
```

```
rate
                                                              ' default
$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Dim A As Byte , B1 As Byte
Input "Number " , A
                                                              'ask for
number
If A = 1 Then
                                                              'test number
  Print "You got it!"
End If
If A = 0 Then
                                                              'test again
  Print "Wrong"
                                                              'thats wrong
Else
                                                              'print this
if a is not 0
  Print "Almost?"
End If
Rem You Can Nest If Then Statements Like This
B1 = 0
If A = 1 Then
  If B1 = 0 Then
     Print "B1=0"
  End If
Else
 Print "A is not 0"
End If
Input "Number " , A
                                                              .
If A = 1 Then
 Print "Ok"
Elseif A = 2 Then
                                                              'use elseif
for more tests
 Print "2" : A = 3
Elseif A = 3 Then
 Print "3"
End If
If A.1 = 1 Then Print "Bit 1 set"
                                                              'test for a
bit
End
```

6.221 ENABLE

Action

Enable specified interrupt.

Syntax

ENABLE interrupt [, prio]

Remarks

Interrupt	Description
INTO	External Interrupt 0

677

INT1	External Interrupt 1	
OVF0,TIMER0, COUNTER0	TIMER0 overflow interrupt	
OVF1,TIMER1,	TIMER1 overflow interrupt	
COUNTER1		
CAPTURE1, ICP1	INPUT CAPTURE TIMER1 interrupt	
COMPARE1A,OC1A or	TIMER1 OUTPUT COMPARE A interrupt	
COMPARE1, OC1	In case of only one compare interrupt	
COMPARE1B,OC1B	TIMER1 OUTPUT COMPARE B interrupt	
SPI	SPI interrupt	
URXC	Serial RX complete interrupt	
UDRE	Serial data register empty interrupt	
UTXC	Serial TX complete interrupt	
SERIAL	Disables URXC, UDRE and UTXC	
ACI	Analog comparator interrupt	
ADC	A/D converter interrupt	
XMEGA ONLY		
prio	The priority you want to assign to the interrupt. Specify Lo, Hi or Med. In the Xmega you must provide the priority of the interrupts. Lo=Low priority. Hi=High priority and Med=Medium priority. If you do not specify a priority, LO will be used.	

By default all interrupts are disabled.

The global interrupts master switch is also disabled by default.

If you enable an interrupt, it will only fire if the master interrupt switch is enabled. You enable this master switch with ENABLE INTERRUPTS. You can disable it with DISABLE INTERRUPTS.

If an interrupt is executed, the global master switch will be disabled automatically by the hardware.

This is to prevent other interrupts to occur.

When the interrupt routine returns, the processor hardware will automatically enable the master switch so new interrupts may occur.

It depends on the processor how many and which interrupts it has. If you type ENABLE in the editor, you will get a pop up with a list of interrupts you can chose from.

XMEGA

In normal AVR chips the priority is determined by the interrupts address. The lower the address, the higher the priority.

In the DAT file you can find a list with interrupts and their address. For example , taken from the <code>m1280def.dat</code> file "

[INTLIST] count=56 INTname1=INT0,\$002,EIMSK.INT0,EIFR.INTF0 INTname2=INT1,\$004,EIMSK.INT1,EIFR.INTF1 INTname3=INT2,\$006,EIMSK.INT2,EIFR.INTF2 INTname4=INT3,\$008,EIMSK.INT3,EIFR.INTF3 INTname5=INT4,\$00a,EIMSK.INT4,EIFR.INTF4 INTname6=INT5,\$00c,EIMSK.INT5,EIFR.INTF5

INTO has the highest priority since it has the lowest address (address 2)

The XMEGA has a priority system. You can specify if an interrupt as a low, medium or high priority.

But you MUST enable these priorities with <u>CONFIG PRIORITY</u> [533] If you use LO and MED priority, they have to be enabled.

See also

DISABLE 657, ON 789, CONFIG PRIORITY 533

Partial Example

Enable Interrupts'allow interrupts to be setEnable Timer1'enables the TIMER1 interrupt

6.222 ENCODER

Action

Reads pulses from a rotary encoder.

Syntax

Var = **ENCODER**(pin1, pin2, LeftLabel, RightLabel , wait)

Remarks

Var	The target variable that is assigned with the result
Pin1 and pin2	These are the names of the PIN registers to which the output of the encoder is connected. Both pins must be on the same PIN register. So Pinb.0 and Pinb.7 is valid while PinB.0 and PinA.0 is not.
LeftLabel	The name of the label that will be called/executed when a transition to the left is encoded.
RightLabel	The name of the label that will be called/executed when a transition to the right is encountered.
wait	A value of 0 will only check for a rotation/pulse. While a value of 1 will wait until a user actual turns the encoder. A value of 1 will thus halt your program.

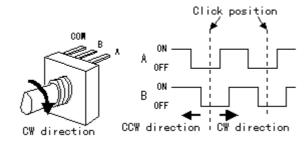
There are some conditions you need to fulfill :

- The label that is called by the encoder must be terminated by a RETURN statement.
- The pin must work in the input mode. By default all pins work in input mode.
- The pull up resistors must be activated by writing a logic 1 to the port registers as the examples shows.

Rotary encoders come in many flavors. Some encoders also have a build in switch.

A sample of an encoder





Since the microprocessor has internal pull up resistors, you do not need external pull up resistors for most encoders.

```
_____
  _____
         _____
'name
                       : encoder.bas
'copyright
                       : (c) 1995-2005, MCS Electronics
'purpose
                        : demonstration of encoder function
'micro
                        : Mega128
'suited for demo
                        : yes
'commercial addon needed : no
'An encoder has 2 outputs and a ground
'We connect the outputs to pinb.0 and pinb.1
'You may choose different pins as long as they are at the same PORT
'The pins must be configured to work as input pins
'This function works for all PIN registers
_____
$regfile = "m128def.dat"
                                                        ' specify
the used micro
$crystal = 4000000
                                                         ' used
crystal frequency
                                                        ' use baud
$baud = 19200
rate
$hwstack = 32
                                                        ' default
use 32 for the hardware stack
$swstack = 10
                                                        ' default
use 10 for the SW stack
$framesize = 40
                                                        ' default
use 40 for the frame space
Print "Encoder test"
Dim B As Byte
```

680

```
'we have dimmed a byte because we need to maintain the state of the
encoder
Portb = &B11
                                                        ' activate
pull up registers
Do
  B = Encoder(pinb.0, Pinb.1, Links, Rechts, 1)
                                               ^--- 1 means wait for
change which blocks programflow
                                ^----- labels which are
called
                ^----- port PINs
  Print B
 Waitms 10
Loop
End
'so while you can choose PINB0 and PINB7, they must be both member of
PINB
'this works on all PIN registers
Links:
 Print "left rotation"
Return
Rechts:
 Print "right rotation"
Return
End
```

6.223 END

Action

Terminate program execution.

Syntax END

Remarks

STOP can also be used to terminate a program.

When an END statement is encountered, all interrupts are disabled and a neverending loop is generated.

When a STOP is encountered the interrupts will not be disabled. Only a never ending loop will be created.

In an embedded application you probably do not want to end the application. But there are cases where you do want to end the application. For example when you control some motors, and you determine a failure, you do not want to use a Watchdog reset because then the failure will occur again. In that case you want to display an error, and wait for service personal to fix the failure.

It is important to notice that without the END statement, your program can behave strange in certain cases. For example : Print "Hello"

Note that there is no END statement. So what will happen? The program will print "Hello". But as the compiler places the library code behind the program code, the micro will execute the library code ! But without being called. As most library code are assembler sub routines that end with a RET, your program will most likely crash, or reset and repeat for ever.

See also

STOP 910

Example

Print "Hello"	'print this
End	'end program execution and disable all interrupts

6.224 EOF

Action

Returns the End of File Status.

Syntax

bFileEOFStatus = **EOF**(#bFileNumber)

Remarks

bFileEOFStatus	(Byte) A Byte Variable, which assigned with the EOF Status
bFileNumber	(Byte) Number of the opened file

This functions returns information about the End of File Status

Return value	Status
0	NOT EOF
255	EOF

In case of an error (invalid file number) 255 (EOF) is returned too.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], FREEFILE [695], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], DISKSIZE [661], GET [697], PUT [819], FILEDATE [685], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_FileEOF	
Input	r24: Filenumber	
Output	r24: EOF Status	r25: Error code
	C-Flag: Set on Error	

Partial Example

Ff =Freefile()' get file handle
Open "test.txt" For Input As #ff ' we can use a constant for the file too
Print Lof(#ff); " length of file"
Print Fileattr(#ff); " file mode"' should be 1 for input
Do
LineInput #ff , S ' read a line
' line input is used to read a line of text from a file
Print S ' print on terminal emulator
Loop Until Eof(#ff)<> 0
'The EOF() function returns a non-zero number when the end of the file is reached
'This way we know that there is no more data we can read
Close #ff

6.225 EXIT

Action

Exit a FOR..NEXT, DO..LOOP , WHILE ..WEND, SUB..END SUB or FUNCTION..END FUNCTION.

Syntax

EXIT FOR EXIT DO EXIT WHILE EXIT SUB EXIT FUNCTION

Remarks

With the EXIT statement you can exit a structure at any time.

· ·		
'name 'copyright 'purpose 'micro 'suited for demo 'commercial addon needed		
<pre>\$regfile = "m48def.dat" the used micro \$crystal = 4000000 crystal frequency \$baud = 19200 rate \$hwstack = 32 use 32 for the hardware s \$swstack = 10 use 10 for the SW stack \$framesize = 40</pre>	stack	' specify ' used ' use baud ' default ' default ' default

```
use 40 for the frame space
Dim B1 As Byte , A As Byte
B1 = 50
                                                              'assign var
For A = 1 To 100
                                                              'for next
loop
   If A = B1 Then
                                                              'decision
     Exit For
                                                              'exit loop
   End If
Next
Print "Exit the FOR..NEXT when A was " ; A
A = 1
Do
  Incr A
  If A = 10 Then
    Exit Do
 End If
Loop
Print "Loop terminated"
End
```

6.226 EXP

Action

Returns e(the base of the natural logarithm) to the power of a single or double variable.

Syntax

Target = **EXP**(source)

Remarks

Target	The single or double that is assigned with the Exp() of the target.
Source	The source to get the Exp of.

See also

LOG 772 , LOG10 772

Example

```
_____
_____
'copyright
                 : (c) 1995-2005, MCS Electronics
'micro
                 : Mega88
                 : no, but without the DOUBLE, it works for
'suited for demo
DEMO too in M48
'commercial addon needed : no
'purpose
                 : demonstrates EXP function
·_____
_____
$regfile = "m88def.dat"
                                         ' specify
```

the used micro **\$crystal =** 8000000 crystal frequency

' used

```
$baud = 19200 ' use baud
rate
$hwstack = 32 ' default
use 32 for the hardware stack
$swstack = 40 ' default
use 10 for the SW stack
$framesize = 40 ' default
use 40 for the frame space
```

Dim X As Single

X = Exp(1.1)
Print X
'prints 3.004166124
X = 1.1
X = Exp(x)
Print X
'prints 3.004164931

Dim D As Double

```
D = Exp(1.1)
Print D
'prints 3.00416602394643
D = 1.1
D = Exp(d)
Print D
'prints 3.00416602394638
End
```

6.227 FILEATTR

Action

Returns the file open mode.

Syntax

bFileAttribut = **FILEATTR**(bFileNumber)

Remarks

bFileAttribut	(Byte) File open mode, See table
bFileNumber	(Byte) Number of the opened file

This functions returns information about the File open mode

Return value	Open mode
1	INPUT
2	OUTPUT
8	APPEND
32	BINARY

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC

 [766], LOF [767], EOF [687], FREEFILE [695], SEEK [850], BSAVE [420], BLOAD [416], KILL [757],

 DISKFREE [660], DISKSIZE [667], GET [697], PUT [819], FILEDATE [683], FILETIME [687],

 FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_FileAttr	
Input	r24: Filenumber	
Output	24: File open mode	r25: Errorcode
	C-Flag: Set on Error	

Partial Example

'open the file in BINARY mode Open "test.biN" For Binary As #2 Print Fileattr(#2); " file mode"' should be 32 for binary Put #2 , Sn ' write a single Put #2 , Stxt ' write a string Close #2

6.228 FILEDATE

Action

Returns the date of a file

Syntax

sDate = FILEDATE ()
sDate = FILEDATE (file)

Remarks

Sdate	A string variable that is assigned with the date.
File	The name of the file to get the date of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [695], FILEATTR [684], SEEK [680], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], DISKSIZE [667], GET [697], PUT [819], FILELEN [687], FILETIME [687], FILEDATETIME [686], DIR [656], WRITE [953], INPUT [744]

ASM

Calls	_FileDateS ; with filename	_FileDateS0;for current file from DIR ()
	X : points to the string with the mask	Z : points to the target variable
Output		

Partial Example

Print "File demo" Print Filelen("josef.img");" length" ' length of file Print Filetime("josef.img");" time" ' time file was changed Print Filedate("josef.img");" date" ' file date

6.229 FILEDATETIME

Action

Returns the file date and time of a file

Syntax

Var = FILEDATETIME () Var = **FILEDATETIME** (file)

Remarks

	A string variable or byte array that is assigned with the file date and time of the specified file
File	The name of the file to get the date time of.

When the target variable is a string, it must be dimensioned with a length of at least 17 bytes.

When the target variable is a byte array, the array size must be at least 6 bytes.

When you use a numeric variable, the internal file date and time format will be used.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC

 Treeh, LOF Treeh, EOF Gesh, FREEFILE Gesh, FILEATTR Gesh, SEEK [850h, BSAVE [420h, BLOAD [416h, KILL Treh, DISKFREE Geoh, GET Gesh, PUT [819h, FILELEN Gesh, FILEDATE Gesh, FILETIME Gesh, SEEK [850h, SEE DIR 656, WRITE 953, INPUT 744

ASM

Calls	_FileDateTimeS	_FileDateTimeS0
Input		
Output		

Calls	_FileDateTimeB	_FileDateTimeB0
Input		
Output		

Example

See fs subfunc decl lib.bas in the samples dir.

6.230 FILELEN

Action

Returns the size of a file

Syntax

ISize = FILELEN ()
ISize = FILELEN (file)

Remarks

lSize	A Long Variable, which is assigned with the file size in bytes of the file.
File	A string or string constant to get the file length of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [698], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], GET [697], PUT [819], FILEDATE [688], FILETIME [687], FILEDATETIME [686], DIR [656], WRITE [953], INPUT [744]

ASM

Calls	_FileLen	
Input		
Output		

Partial Example

Print "File demo" Print Filelen("josef.img");" length" ' length of file Print Filetime("josef.img");" time" ' time file was changed Print Filedate("josef.img");" date" ' file date

6.231 FILETIME

Action

Returns the time of a file

Syntax

sTime = **FILETIME** () sTime = **FILETIME** (file)

Remarks

Stime	A string variable that is assigned with the file time.
File	The name of the file to get the time of.

688 BASCOM-AVR

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also

INITFILESYSTEM [737), OPEN [794], CLOSE [437), FLUSH [889], PRINT [810], LINE INPUT [762), LOC [766], LOF [767], EOF [687], FREEFILE [695], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [751], DISKFREE [660], GET [697], PUT [819], FILELEN [687], FILEDATE [685], FILEDATETIME [686], DIR [666], WRITE [693], INPUT [744]

ASM

Calls	_FileTimeS ; with file param	_FileTimeS0 ; current file
	X : points to the string with the mask	Z : points to the target variable
Output		

Example

Print "File demo" Print Filelen("josef.img");" length" ' length of file Print Filetime("josef.img");" time" ' time file was changed Print Filedate("josef.img");" date" ' file date

6.232 FIX

Action

Returns for values greater then zero the next lower value, for values less then zero the next upper value.

Syntax

var = **FIX**(x)

Remarks

Var	A single variable that is assigned with the FIX of variable x.
Х	The single to get the FIX of.

See Also

INT 748, ROUND 847, SGN 874

Example

!		
'name 'copyright 'purpose	:	round_fix_int.bas (c) 1995-2005, MCS Electronics demo : ROUND,FIX
'micro	:	Mega48
'suited for demo 'commercial addon needed		yes no
'		

```
$regfile = "m48def.dat"
```

' specify the used mich

```
$crystal = 4000000
$baud = 19200
$baud = 19200
$hwstack = 32
$hwstack = 32
$swstack = 10
$framesize = 40
Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next
End
```

6.233 FLUSH

Action

Write current buffer of File to Card and updates Directory

Syntax

FLUSH #bFileNumber FLUSH

Remarks

BFileNumber Filenumber, which identifies an opened file such as #1 or #ff

This function writes all information of an open file, which is not saved yet to the Disk. Normally the Card is updated, if a file will be closed or changed to another sector.

When no file number is specified, all open files will be flushed.

See also

```
INITFILESYSTEM [737], OPEN [794], CLOSE [437], PRINT [816], LINE INPUT [762], LOC [766], LOF [767],
EOF [687], FREEFILE [695], FILEATTR [684], SEEK [656], BSAVE [426], BLOAD [416], KILL [757],
DISKFREE [666], DISKSIZE [667], GET [697], PUT [819], FILEDATE [685], FILETIME [687],
FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]
```

ASM

Calls	_FileFlush	_FilesAllFlush
Input	r24: filenumber	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

\$include "startup.inc"

'open the file in BINARY mode Open "test.biN" For Binary As #2 Put #2 , B ' write a byte Put #2 , W ' write a word Put #2 , L ' write a long Ltemp = Loc(#2) + 1 ' get the position of the next byte Print Ltemp ;" LOC"' store the location of the file pointer Print Lof(#2);" length of file" Print Fileattr(#2);" file mode"' should be 32 for binary Put #2 , Sn ' write a single Put #2 , Stxt ' write a string

Flush #2 ' flush to disk Close #2

6.234 FORMAT

Action

Formats a numeric string.

Syntax

target = FORMAT(source, "mask")

Remarks

target	The string that is assigned with the formatted string.	
source	The source string that holds the number.	
mask	The mask for formatting the string.	
	When spaces are in the mask, leading spaces will be added when the length of the mask is longer than the source string. " " '8 spaces when source is "123" it will be " 123". When a + is in the mask (after the spaces) a leading + will be assigned when the number does not start with the - sign. "+" with number "123" will be "+123". When zero's are provided in the mask, the string will be filled with leading zero;s. " +00000" with 123 will be " +00123" An optional decimal point can be inserted too: "000.00" will format the number 123 to "001.23" Combinations can be made but the order must be : spaces, + , 0 an optional point and zero's.	

When you do not want to use the overhead of the single or double, you can use the LONG. You can scale the value by a factor 100.

Then use FORMAT to show the value.

```
For example : Dim L as Long, X as Long , Res as Long
L = 1
X = 2
Res = L / X
Now this would result in 0 because an integer or Long does not support floating point.
But when you scale L with a factor 100, you get :
L= 100
X = 2
Res = L / X
```

Now Res will be 50. To show it the proper way we can use FORMAT. Format works with strings so the variables need to be converted to string first.

Dim S1 as string * 16 : s1 = Str(Res) Print Format(s1,"000.00")

See also

FUSING 696

Example

```
_____
_____
'name
                     : format.bas
                     : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                      : demo : FORMAT
                      : Mega48
'micro
'suited for demo : yes
'commercial addon needed : no
·_____
                                 _____
_____
$regfile = "m48def.dat"
                                                   ' specify
the used micro
                                                   ' used
$crystal = 4000000
crystal frequency
$baud = 19200
                                                   ' use baud
rate
$hwstack = 32
                                                   ' default
use 32 for the hardware stack
$swstack = 10
                                                   ' default
use 10 for the SW stack
$framesize = 40
                                                   ' default
use 40 for the frame space
Dim S As String * 10
Dim I As Integer
S = "12345"
S = Format(s, "+")
Print S
S = "123"
S = Format(s , "00000")
Print S
S = "12345"
S = Format(s , "000.00")
Print S
S = "12345"
S = Format(s, "+000.00")
Print S
End
```

6.235 FOR-NEXT

Action

Execute a block of statements a number of times.

Syntax

FOR var = start TO end [STEP value]

Remarks

var	The variable counter to use
start	The starting value of the variable var
end	The ending value of the variable var
	The value var is increased/decreased with each time NEXT is encountered.

- For incremental loops, you must use TO.
- For decremental loops, you must use a negative step size.
- You must end a FOR structure with the NEXT statement.
- The use of STEP is optional. By default, a value of 1 is used.

When you know in advance how many times a block of code must be executed, the FOR..NEXT loop is convenient to use. You can exit a FOR .. NEXT loop with the EXIT FOR statement.

It is important that the if you use variables for START and END, that these are of the

same data type. So for example:

Dim x, as byte, st as byte, ed as byte

FOR x = st TO ED ' this is ok since all variables are of the same data type

Dim x as Byte, st as Word, Ed as Long FOR x = st TO ED ' this is NOT ok since all variables are of different data type.

The reason is that when the condition is evaluated, it will create a compare on 2 bytes, while you actually want to have a word since the end variable is a word.

A for next loop with an integer has an upper limit of 32766 and not 32767, the maximum value that fits into an integer.

This is done in order to save code space. Checking an overflow from 32767 to -32768 would cost extra code.

There are also other alternatives. You can use a Do.. Loop for example :

Dim Var As Byte Do 'code Incr Var Loop Until Var = 10

There are various way to get the result you need.

See also

EXIT FOR 682

Example

'name 'copyright 'purpose : for_next.bas : (c) 1995-2005, MCS Electronics : demo: FOR, NEXT

```
'micro
                         : Mega48
'suited for demo
                         : yes
'commercial addon needed : no
                                                 _____
·_____
$regfile = "m48def.dat"
                                                           ' specify
the used micro
$crystal = 4000000
                                                           ' used
crystal frequency
$baud = 19200
                                                           ' use baud
rate
$hwstack = 32
                                                           ' default
use 32 for the hardware stack
$swstack = 10
                                                           ' default
use 10 for the SW stack
$framesize = 40
                                                           ' default
use 40 for the frame space
Dim A As Byte , B1 As Byte , C As Integer
For A = 1 To 10 Step 2
  Print "This is A " ; A
Next A
Print "Now lets count down"
For C = 10 To -5 Step -1
 Print "This is C " ; C
Next
Print "You can also nest FOR..NEXT statements."
For A = 1 To 10
 Print "This is A " ; A
 For B1 = 1 To 10
   Print "This is B1 " ; B1
 Next
                                                           ' note that
you do not have to specify the parameter
Next A
End
```

6.236 FOURTHLINE

Action

Set LCD cursor to the start of the fourth line.

Syntax FOURTHLINE

Remarks

Only valid for LCD displays with 4 lines.

See also

HOME [724], UPPERLINE [945], LOWERLINE [776], THIRDLINE [930], LOCATE [777]

Example

Thirdline Lcd "Line 3" Fourthline Lcd "Line 4" Home Third on line three Home Fourth Home F

'goto home

'first letter also works

6.237 FRAC

Action

Returns the fraction of a single.

Syntax

var = FRAC(single)

Remarks

	A numeric single variable that is assigned with the fraction of variable single.
single	The single variable to get the fraction of.

The fraction is the right side after the decimal point of a single.

See Also

<u>INT</u> 748

Example

```
'copyright : (c) 1995-2005, MCS Electronics
'micro : Mega48
'suited for demo : yes
'commercial addon needed : no
'purpose : demonstrates FRAC function
'------
```

```
$regfile = "m48def.dat"
                                                               ' specify
the used micro
$crystal = 8000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
                                                              ' default
\$swstack = 40
use 10 for the SW stack
                                                              ' default
$framesize = 40
use 40 for the frame space
```

```
Dim X As Single
X = 1.123456
Print X
Print Frac(x)
End
```

6.238 FREEFILE

Action

Returns a free Filenumber.

Syntax

bFileNumber = **FREEFILE()**

Remarks

bFileNumber A byte variable , which can be used for opening next file

This function gives you a free file number, which can be used for file – opening statements. In contrast to VB this file numbers start with 128 and goes up to 255. Use range 1 to 127 for user defined file numbers to avoid file number conflicts with the system numbers from FreeFile()

This function is implemented for compatility with VB.

See also

INITFILESYSTEM [737), OPEN [794), CLOSE [437), FLUSH [689), PRINT [810), LINE INPUT [762), LOC [766], LOF [767], EOF [687], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], DISKSIZE [667], GET [697], PUT [819], FILEDATE [685], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_GetFreeFileNumber	
Input	none	
Output	r24: Filenumber	r25: Errorcode
	C-Flag: Set on Error	

Partial Example

Ff =Freefile() ' get file handle
Open"test.txt" For Input As #ff ' we can use a constant for the file too
Print Lof(#ff);" length of file"
Print Fileattr(#ff);" file mode" ' should be 1 for input
Do
 LineInput #ff , S ' read a line
 ' line input is used to read a line of text from a file
 Print S ' print on terminal emulator
Loop UntilEof(ff)<> 0
'The EOF() function returns a non-zero number when the end of the file is reached

696 BASCOM-AVR

'This way we know that there is no more data we can read Close $\#\mathrm{ff}$

6.239 FUSING

Action

FUSING returns a formatted string of a single value.

Syntax

target = FUSING(source, "mask")

Remarl	KS	
target	The string that is assigned with the formatted string.	
source	The source variable of the type SINGLE that will be converted	
mask	The mask for formatting the string.	
	The mask is a string constant that always must start with #. After the decimal point you can provide the number of digits you want the string to have: #.### will give a result like 123.456. Rounding is used when you use the # sign. So 123.4567 will be converted into 123.457	
	When no rounding must be performed, you can use the & sign instead of the # sign. But only after the DP. #.&&& will result in 123.456 when the single has the value 123.4567	

When the single is zero, 0.0 will be returned, no matter how the mask is set up.

See also

FORMAT 690 , STR 911

Example

'name 'copyright 'purpose 'micro 'suited for demo 'commercial addon needed '	: fusing.bas : (c) 1995-2005, MCS Electronics : demo : FUSING : Mega48 : yes : no	
<pre>\$regfile = "m48def.dat" the used micro</pre>		' specify
\$crystal = 4000000		' used
crystal frequency \$baud = 19200		' use baud
<pre>rate \$hwstack = 32 use 32 for the hardware s</pre>	stack	' default
\$swstack = 10	SLACK	' default
use 10 for the SW stack \$framesize = 40		' default

```
use 40 for the frame space
Dim S As Single , Z As String * 10
'now assign a value to the single
S = 123.45678
'when using str() you can convert a numeric value into a string
Z = Str(s)
Print Z
                                                              'prints
123.456779477
Z = Fusing(s, "#.##")
'now use some formatting with 2 digits behind the decimal point with
rounding
Print Fusing(s , "#.##")
                                                              'prints
123.46
'now use some formatting with 2 digits behind the decimal point without
rounding
Print Fusing(s , "#.&&")
                                                              'prints
123.45
'The mask must start with #.
'It must have at least one # or & after the point.
'You may not mix & and # after the point.
End
```

6.240 GET

Action

Reads a byte from the hardware or software UART. Reads data from a file opened in BINARY mode.

Syntax

GET #channel, var
GET #channel, var , [pos] [, length]

Remarks

GET in combination with the software/hardware UART reads one byte from the UART. GET in combination with the AVR-DOS file system is very flexible and versatile. It works on files opened in BINARY mode and you can reads all data types.

A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
The variable or variable array that will be assigned with the data from the file
This is an optional parameter that may be used to specify the position where the reading must start from. This must be a long variable.
This is an optional parameter that may be used to specify how many bytes must be read from the file.

By default you only need to provide the variable name. When the variable is a byte, 1 byte will be read. When the variable is a word or integer, 2 bytes will be read. When the variable is a long or single, 4 bytes will be read. When the variable is a string, the number of bytes that will be read is equal to the dimensioned size of the string. DIM

698 BASCOM-AVR

S as string * 10 , would read 10 bytes.

Note that when you specify the length for a string, the maximum length is 254. The maximum length for a non-string array is 65535.

Partial Example :

GET #1, var ,,2 ' read 2 bytes, start at current position GET #1, var , PS ' start at position stored in long PS GET #1, var , PS, 2 ' start at position stored in long PS and read 2 bytes

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [688], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [698], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], DISKSIZE [667], PUT [819], FILEDATE [688], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

current position	goto new position first
Byte:	
_FileGetRange_1	_FileGetRange_1
Input:	Input:
r24: File number	r24: File number
X: Pointer to variable	X: Pointer to variable
T-Flag cleared	r16-19 (A): New position (1-based)
	T-Flag Set
Word/Integer:	
_FileGetRange_2	_FileGetRange_2
Input:	Input:
r24: File number	r24: File number
X: Pointer to variable	X: Pointer to variable
T-Flag cleared	r16-19 (A): New position (1-based)
	T-Flag Set
Long/Single:	
_FileGetRange_4	_FileGetRange_4
Input:	Input:
r24: File number	r24: File number
X: Pointer to variable	X: Pointer to variable
T-Flag cleared	r16-19 (A): New position (1-based)

BASCOM Language Reference

699

' get the

' store the

' should be

' write a

	T-Flag Set
String (<= 255 Bytes) with fixed length	
_FileGetRange_Bytes	_FileGetRange_Bytes
Input:	Input:
r24: File number	r24: File number
r20: Count of Bytes	r20: Count of bytes
X: Pointer to variable	X: Pointer to variable
T-Flag cleared	r16-19 (A): New position (1-based)
	T-Flag Set
Array (> 255 Bytes) with fixed length	
_FileGetRange	_FileGetRange
Input:	Input:
r24: File number	r24: File number
r20/21: Count of Bytes	r20/21: Count of bytes
X: Pointer to variable	X: Pointer to variable
T-Flag cleared	r16-19 (A): New position (1-based)
	T-Flag Set

Output from all kind of usage: r25: Error Code C-Flag on Error X: requested info

Partial Example

'for the binary file demo we need some variables of different types
Dim B As Byte , W As Word , L As Long , Sn As Single , Ltemp As Long
Dim Stxt As String * 10
B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt = "test"

'open the file in BINARY mode Open "test.biN"for Binary As #2 Put#2 , B ' write a byte Put#2 , W ' write a word Put#2 , L ' write a long Ltemp = Loc(#2) + 1 position of the next byte Print Ltemp ; " LOC" location of the file pointer Print Seek(#2) ; " = LOC+1" Print Seek(#2) ; " length of file" Print Fileattr(#2) ; " file mode" 32 for binary Put #2 , Sn single

700 BASCOM-AVR

```
Put #2 , Stxt
                                                                       ' write a
string
Flush #2
                                                                       ' flush to
disk
Close #2
'now open the file again and write only the single
Open "test.bin" For Binary As #2
L = 1 'specify the file position
                                                                       ' reset is
B = Seek(#2 , L)
the same as using SEEK #2,L
Get#2 , B ' get the byte
Get#2 , W ' get the word
Get#2 , L ' get the long
Get#2 , Sn ' get the single
Get#2 , Stxt ' get the string
Close #2
```

6.241 GETADC

Action

Retrieves the analog value from the specified channel.

Syntax

var = GETADC(channel [,offset])

Syntax Xmega

var = GETADC(ADC , channel [,MUX])

Remarks AVR

Var	The variable that is assigned with the A/D value. This should be a Word or other 16 bit variable.
Channel	The channel to measure. Might be higher then 7 on some chips. The Mega2560 has 16 channels. So the range is 0-15 on a Mega2560.
	An optional numeric variable of constant that specifies gain or mode. This option has effect on newer AVR micro's only. The offset will be added by the channel value and inserted into the ADMUX register. This way you can control gain.

Remarks XMEGA

var	The variable that is assigned with the A/D value. This should be a Word or other 16 bit variable.
ADC	The ADC to use. This is either ADCA or ADCB.
Channel	The channel to use. There are 4 channels in the range from 0-3.
MUX	An optional numeric variable or constant that specifies the MUX value thus which input pin is used for the measurement. The MUX number is coded with negative and positive input pin info. The positive pins are have an offset of 8. So PINO in single ended mode would need a value of 8. When you do not supply the mux value, the value used by the CONFIG

ADC command will be used. If you supply it, it will change the MUX
register of the corresponding channel.

The GETADC() function only will work on microprocessors that have an A/D converter. The pins of the A/D converter input can be used for digital I/O too. But it is important that no I/O switching is done while using the A/D converter.

Make sure you turn on the AD converter with the <u>START</u> and ADC statement or by setting the proper bit in the ADC configuration register.

Some micro's have more then 7 channels. This is supported as well. The ADCSRB register contains a bit named MUX5 that must be set when a channel higher then 7 is used. The compiler (lib routine) will handle this automatic. This is true for new chips like Mega1280, Mega2560 and probably other new chips with 100 pins.

An example on how to read singled ended input on a Mega1280:

W = Getadc(0, 64) ' from data sheet : 100000 ADC8

W = Getadc(1, 64) ' from data sheet : 100001 ADC9

This will read channel 0 and 1. The offset is 64 in order to use singled ended input. ADC8 is portK.0

GetADC() returns a word variable since the A/D converter data registers consist of 2 registers. The resolution depends on the chip.

The variable ADCD can be used to access the data register directly. The compiler will handle access to the byte registers automatically.

See also

<u>CONFIG ADC</u> 45ने

Example

_____ : adc.bas 'name : (c) 1995-2005, MCS Electronics 'copyright : demonstration of GETADC() function for 8535 'purpose or M163 micro 'micro : Mega163 'suited for demo : yes 'commercial addon needed : no 'use in simulator : possible ' Getadc() will also work for other AVR chips that have an ADC converter \$regfile = "m163def.dat" ' we use the M163 \$crystal = 4000000\$hwstack = 32' default use 32 for the hardware stack \$swstack = 10'default use 10 for the SW stack **\$framesize** = 40 'default use 40 for the frame space

'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function

```
'The prescaler divides the internal clock by 2,4,8,16,32,64 or 128
'Because the ADC needs a clock from 50-200 KHz
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc
'With STOP ADC, you can remove the power from the chip
'Stop Adc
Dim W As Word , Channel As Byte
Channel = 0
'now read A/D value from channel 0
Do
 W = Getadc(channel)
 Print "Channel " ; Channel ; " value " ; W
  Incr Channel
  If Channel > 7 Then Channel = 0
LOOD
End
'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
' OFF
      : AREF, internal reference turned off
'AVCC
         : AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin
```

'Using the additional param on chip that do not have the internal reference will have no effect.

Example Xmega

```
(c) 1995-2010, MCS
                    xm128-ADC.bas
' This sample demonstrates the Xmega128A1 ADC
!_____.
                                                 _____
$regfile = "xm128a1def.dat"
$crystal = 3200000
\$hwstack = 64
\$swstack = 64
$framesize = 64
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external xmegafix rol r1014
'First Enable The Osc Of Your Choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
```

```
Print "ADC test"
'setup the ADC-A converter
Config Adca = Single , Convmode = Unsigned , Resolution = 12bit , Dma =
Off , Reference = Intlv , Event_mode = None , Prescaler = 32 , Ch0_gain
= 1 , Ch0_inp = Single_ended , Mux0 = &B000_00 _
 Chl_gain = 1 , Chl_inp = Single_ended , Mux1 = &B1_000 , Ch2_gain = 1 ,
Ch2_inp = Single_ended , Mux2 = &B10_000 , Ch3_gain = 1 , Ch3_inp =
Single_ended , Mux3 = \&B11_000
Dim W As Word , I As Byte , Mux As Byte
Do
  Mux = I * 8
                          ' or you can use shift left,3 to get the
proper offset
   W = Getadc(adca , 0 , Mux)
   W = Getadc(adca , 0) 'when not using the MUX parameter the last
value of the MUX will be used!
   ' use ADCA , use channel 0, and use the pinA.0-pinA.3 \,
   Print "RES:" ; I ; "-" ; W
   Incr I
   If I > 3 Then I = 0
  Waitms 500
Loop Until Inkey(#1) = 27
```

6.242 GETATKBD

Action

Reads a key from a PC AT keyboard.

Syntax

var = **GETATKBD**()

Remarks

var	The variable that is assigned with the key read from the keyboard.
	It may be a byte or a string variable. When no key is pressed a 0 will be returned.

The GETAKBD() function needs 2 input pins and a translation table for the keys. You can read more about this at the <u>CONFIG KEYBOARD</u> [513] compiler directive.

The Getatkbd function will wait for a pressed key. When you want to escape from the waiting loop you can set the ERR bit from an interrupt routine for example.

Getatkbd is using 2 bits from register R6 : bit 4 and 5 are used to hold the shift and control key status.

AT KEYBOARD SCANCODES

Table reprinted with permission of Adam Chapweske

http://panda.cs.ndsu.nodak.edu/~achapwes

KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1C	F0,1C	9	46	F0,46	[54	FO,54

704

BASCOM-AVR	
------------	--

В	32	F0,32		0E	F0,0E	INSERT	E0,70	EO, F0,70
С	21	F0,21	-	4E	F0,4E	HOME	E0,6C	E0, F0,6C
D	23	F0,23	=	55	FO,55	PG UP	E0,7D	
E	24	F0,24	N	5D	F0,5D	DELETE	E0,71	
F	2B	F0,2B	BKSP	66	F0,66	END	E0,69	
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	
Н	33	F0,33	TAB	0D	F0,0D	U ARROW	E0,75	
I	43	F0,43	CAPS	58	F0,58	L ARROW	E0,6B	
J	3B	F0,3B	L SHFT	12	FO,12	D ARROW	E0,72	
К	42	F0,42	L CTRL	14	FO,14	R ARROW	E0,74	
L	4B	F0,4B	L GUI	E0.1F	E0,F0,1F	NUM	77	F0,77
M	3A	F0,3A	L ALT	11	F0,11	KP /	E0,4A	-
N	31	F0,31	R SHFT	59	F0,59	KP *	7C	, F0,7C
0	44	F0,44	R CTRL	E0,14	E0,F0,14	KP -	7B	F0,7B
Р	4D	F0,4D	R GUI	E0,27	E0,F0,27	KP +	79	F0,79
Q	15	F0,15	R ALT	E0,11	E0,F0,11	KP EN	E0,5A	E0, F0,5A
R	2D	F0,2D	APPS	E0,2F	E0,F0,2F	KP.	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70
Т	2C	F0,2C	ESC	76	F0,76	KP 1	69	F0,69
U	3C	F0,3C	F1	05	F0,05	KP 2	72	F0,72
V	2A	F0,2A	F2	06	F0,06	KP 3	7A	F0,7A
W	1D	F0,1D	F3	04	F0,04	KP 4	6B	F0,6B
Х	22	F0,22	F4	0C	F0,0C	KP 5	73	F0,73
Y	35	F0,35	F5	03	F0,03	KP 6	74	F0,74
Z	1A	F0,1A	F6	0B	F0,0B	KP 7	6C	F0,6C
0	45	F0,45	F7	83	F0,83	KP 8	75	F0,75
1	16	F0,16	F8	0A	F0,0A	KP 9	7D	F0,7D
2	1E	F0,1E	F9	01	F0,01]	5B	F0,5B
3	26	F0,26	F10	09	F0,09	;	4C	F0,4C
4	25	F0,25	F11	78	F0,78	1	52	F0,52
5	2E	F0,2E	F12	07	F0,07	,	41	F0,41
6	36	F0,36	PRNT SCRN	,	E0,F0, 7C,E0, F0,12		49	F0,49
7	3D	F0,3D	SCROLL		F0,7E	/	4A	F0,4A
8	3E	F0,3E	PAUSE		-NONE-			

		E1, F0,14			
		, F0,77			

These are the usable scan codes from the keyboard. If you want to implement F1 , you look at the generated scan code : 05 hex. So in the table, at position 5+1=6, you write the value for F1.

In the sample program below, you can find the value 200. When you now press F1, the value form the table will be used so 200 will be returned.

See also

CONFIG KEYBOARD 513, GETATKBDRAW 707

Example

_____ 'name : getatkbd.bas 'copyright : (c) 1995-2005, MCS Electronics 'purpose : PC AT-KEYBOARD Sample 'micro : Mega48 'suited for demo : yes 'commercial addon needed : no ·____. _____ \$regfile = "8535def.dat" ' specify the used micro ' used \$crystal = 4000000crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space 'For this example : 'connect PC AT keyboard clock to PIND.2 on the 8535 'connect PC AT keyboard data to PIND.4 on the 8535 'The GetATKBD() function does not use an interrupt. 'But it waits until a key was pressed! 'configure the pins to use for the clock and data 'can be any pin that can serve as an input 'Keydata is the label of the key translation table **Config** Keyboard = Pind.2 , **Data** = Pind.4 , Keydata = Keydata 'Dim some used variables Dim S As String * 12 Dim B As Byte 'In this example we use SERIAL(COM) INPUT redirection \$serialinput = Kbdinput

```
'Show the program is running
Print "hello"
Do
  'The following code is remarked but show how to use the GetATKBD()
function
  ' B = Getatkbd()
                      'get a byte and store it into byte variable
  'When no real key is pressed the result is 0
  'So test if the result was > 0
  ' If B > 0 Then
  1.1
      Print B ; Chr(b)
  ' End If
  'The purpose of this sample was how to use a PC AT keyboard
  'The input that normally comes from the serial port is redirected to
the
  'external keyboard so you use it to type
  Input "Name " , S
  'and show the result
  Print S
  'now wait for the F1 key , we defined the number 200 for F1 in the
table
  Do
   B = Getatkbd()
  Loop Until B <> 0
  Print B
Loop
End
'Since we do a redirection we call the routine from the redirection
routine
Kbdinput:
'we come here when input is required from the COM port
'So we pass the key into R24 with the GetATkbd function
' We need some ASM code to save the registers used by the function
$asm
push r16
                  ; save used register
push r25
push r26
push r27
Kbdinput1:
rCall _getatkbd
                   ; call the function
tst r24
                   ; check for zero
breq Kbdinputl
                   ; yes so try again
                   ; we got a valid key so restore registers
pop r27
pop r26
pop r25
pop r16
$end Asm
'just return
Return
'The tricky part is that you MUST include a normal call to the routine
'otherwise you get an error
'This is no clean solution and will be changed
B = Getatkbd()
'This is the key translation table
Keydata:
'normal keys lower case
```

BASCOM Language Reference 707

Data 0 , 0 , 0 , 0 , 0 , 113 , 49 , 0 , 0 , 0 , 122 , 115 , 97 , 119 , 50,0 Data 0 , 99 , 120 , 100 , 101 , 52 , 51 , 0 , 0 , 32 , 118 , 102 , 116 , 114 , 53 , 0 Data 0 , 110 , 98 , 104 , 103 , 121 , 54 , 7 , 8 , 44 , 109 , 106 , 117 , 55 , 56 , 0 Data 0 , 44 , 107 , 105 , 111 , 48 , 57 , 0 , 0 , 46 , 45 , 108 , 48 , 112 , 43 , 0 Data 0 , 0 , 0 , 0 , 0 , 92 , 0 , 0 , 0 , 0 , 13 , 0 , 0 , 92 , 0 , 0 Data 0 , 60 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0 Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 , 0,0 'shifted keys UPPER case Data 0 , 0 , 0 , 0 , 0 , 81 , 33 , 0 , 0 , 0 , 90 , 83 , 65 , 87 , 34 , 0 Data 0 , 67 , 88 , 68 , 69 , 0 , 35 , 0 , 0 , 32 , 86 , 70 , 84 , 82 , 37 , 0 Data 0 , 78 , 66 , 72 , 71 , 89 , 38 , 0 , 0 , 76 , 77 , 74 , 85 , 47 , 40,0 Data 0 , 59 , 75 , 73 , 79 , 61 , 41 , 0 , 0 , 58 , 95 , 76 , 48 , 80 , 63,0 Data 0, 0, 0, 0, 0, 96, 0, 0, 0, 0, 13, 94, 0, 42, 0, 0 Data 0 , 62 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0 , 0 Data 48, 44, 50, 53, 54, 56, 0, 0, 0, 43, 51, 45, 42, 57, 0,0

6.243 GETATKBDRAW

Action

Reads a key from a PC AT keyboard.

Syntax

var = **GETATKBDRAW**()

Remarks

The variable that is assigned with the key read from the keyboard.
It may be a byte or a string variable. When no key is pressed a 0 will be returned.

The GETATKBDRAW() function needs 2 input pins and a translation table for the keys. You can read more about this at the <u>CONFIG KEYBOARD</u> 513 compiler directive.

The GetatkbdRAW function will return RAW data from a PS/2 keyboard or Mouse.

While GetatKBD is intended to wait for pressed keys, GetATkbdRAW just returns raw PS/2 data so you can use your own code to process the data.

See Also

<u>GETATKBD</u> 703), <u>CONFIG KEYBOARD</u> 513

Example

See GETATKBD.BAS

6.244 GETDSTIP

Action

Returns the IP address of the peer.

Syntax

Result = **GETDSTIP**(socket)

Remarks

Result	A LONG variable that will be assigned with the IP address of the peer or destination IP address.
Socket	The socket number (0-3)

When you are in server mode, it might be desirable to detect the IP address of the connecting client.

You can use this for logging, security, etc.

The IP number MSB, is stored in the LS byte of the variable.

See also

CONFIG TCPIP 559, GETSOCKET 717), SOCKETCONNECT 888, SOCKETSTAT 890), TCPWRITE 924), TCPWRITESTR 925), CLOSESOCKET 439), SOCKETLISTEN 889), GETDSTPORT 708

Partial Example

Dim L as Long L = GetdstIP(i) ' store current IP number of socket i

6.245 GETDSTPORT

Action

Returns the port number of the peer.

Syntax

Result = **GETDSTPort**(socket)

Remarks

	A WORD variable that is assigned with the port number of the peer or destination port number.
Socket	The socket number.

When you are in server mode, it might be desirable to detect the port number of the connecting client.

You can use this for logging, security, etc.

See also

<u>CONFIG TCPIP</u> 55की, <u>GETSOCKET</u> 717ी, <u>SOCKETCONNECT</u> 68की, <u>SOCKETSTAT</u> 680ी, <u>TCPWRITE</u> 5924ी, <u>TCPWRITESTR</u> 526ी, <u>CLOSESOCKET</u> 438ी, <u>SOCKETLISTEN</u> 686ी, <u>GETDSTIP</u> 708ी

Partial Example

Dim P as Word P = GetdstPORT(i)' store current port number of socket i

6.246 GETKBD

Action

Scans a 4x4 matrix keyboard and return the value of the key pressed.

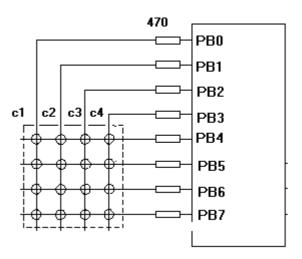
Syntax

var = GETKBD()

Remarks

Var	The numeric variable that is assigned with the value read
	from the keyboard

The GETKBD() function can be attached to a port of the uP. You can define the port with the CONFIG KBD statement. A schematic for PORTB is shown below



Note that the port pins can be used for other tasks as well. But you might need to set the port direction of those pins after you have used getkbd(). For example the LCD pins are set to output at the start of your program. A call to getkbd() would set the pins to input.

By setting DDR.x register you can set the pins to the proper state again. As an alternative you can use CONFIG PIN or CONFIG PORT. When no key is pressed 16 will be returned.

When using the 2 additional rows, 24 will be returned when no key is pressed.

On the STK200 this might not work since other hardware is connected too that interferes.

You can use the <u>Lookup()</u> $\overline{m_{+}}$ function to convert the byte into another value. This because the GetKBD() function does not return the same value as the key pressed. It will depend on which keyboard you use.

Sometimes it can happen that it looks like a key is pressed while you do not press a key. This is caused by the scanning of the pins which happens at a very high frequency.

It will depend on the used keyboard. You can add series resistors with a value of 470-1K $\,$

The routine will wait for 100 mS by default after the code is retrieved. With CONFIG KBD you can set this delay.

See also

CONFIG KBD 511

Example

```
_____
   _____
'name
                      : getkbd.bas
                     : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                     : demo : GETKBD
                     : Mega48
'micro
'suited for demo
                     : yes
'commercial addon needed : no
·_____
                                              _____
 _____
$regfile = "m48def.dat"
                                                    ' specify
the used micro
$crystal = 4000000
                                                    ' used
crystal frequency
$baud = 19200
                                                    ' use baud
rate
$hwstack = 32
                                                    ' default
use 32 for the hardware stack
\$swstack = 10
                                                    ' default
use 10 for the SW stack
                                                    ' default
$framesize = 40
use 40 for the frame space
'specify which port must be used
'all 8 pins of the port are used
Config Kbd = Portb
'dimension a variable that receives the value of the pressed key
Dim B As Byte
'loop for ever
```

```
Do
   B = Getkbd()
   'look in the help file on how to connect the matrix keyboard
   'when you simulate the getkbd() it is important that you press/click
the keyboard button
   ' before running the getkbd() line !!!
   Print B
   'when no key is pressed 16 will be returned
   'use the Lookup() function to translate the value to another one
   ' this because the returned value does not match the number on the
keyboad
Loop
End
```

6.247 GETRC

Action

Retrieves the value of a resistor or a capacitor.

Syntax

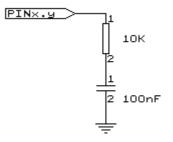
var = GETRC(pin , number)

Remarks

Var	The word variable that is assigned with the value.
Pin	The PIN name for the R/C is connection.
Number	The port pin for the R/C is connection.

The name of the input port (PIND for example) must be passed even when all the other pins are configured for output. The pin number must also be passed. This may be a constant or a variable.

A circuit is shown below:



The capacitor is charged and the time it takes to discharge it is measured and stored in the variable. Now when you vary either the resistor or the capacitor, different values will be returned. This function is intended to return a relative position of a resistor wiper, not to return the value of the resistor. But with some calculations it can be retrieved.

See also

NONE

712

Example

_____ ____ _____ 'name : getrc.bas 'copyright : (c) 1995-2005, MCS Electronics 'purpose : demonstrates how to get the value of a resistor 'micro : AT90S8535 'suited for demo : yes 'commercial addon needed : no ' The library also shows how to pass a variable for use with individual port ' pins. This is only possible in the AVR architecture and not in the 8051 _____ \$regfile = "8535def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space 'The function works by charging a capacitor and uncharge it little by little 'A word counter counts until the capacitor is uncharged. 'So the result is an indication of the position of a pot meter not the actual 'resistor value 'This example used the 8535 and a 10K ohm variable resistor connected to PIND.4 'The other side of the resistor is connected to a capacitor of 100nF. 'The other side of the capacitor is connected to ground. 'This is different than BASCOM-8051 GETRC! This because the architecture is different. 'The result of getrc() is a word so DIM one Dim W As Word Do 'the first parameter is the PIN register. 'the second parameter is the pin number the resistor/capacitor is connected to 'it could also be a variable! W = Getrc(pind, 4)Print W Wait 1 Loop

6.248 GETRC5

Action

Retrieves the RC5 remote code from a IR transmitter.

Syntax

GETRC5(address, command)

Uses

TIMER0

Remarks

address	The RC5 address
command	The RC5 command.

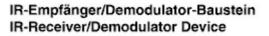
This statement is based on the AVR 410 application note. Since a timer is needed for accurate delays and background processing TIMER0 is used by this statement.

The interrupt of TIMER0 is also used by this statement. TIMER0 can be used by your application since the values are preserved by the statement but a delay can occur. The interrupt can not be reused.

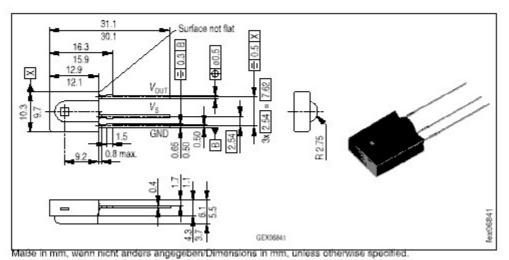
You may use any pin that can work as an input pin. Use the CONFIG RC5 statement to specify which pin is connected to the IR receiver.

GETRC5 supports extended RC5 code reception.

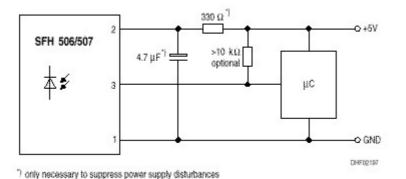
The SFH506-36 is used from Siemens. Other types can be used as well. The TSOP1736 has been tested with success.



SFH 506



For a good operation use the following values for the filter.



Most audio and video systems are equipped with an infra-red remote control.

The RC5 code is a 14-bit word bi-phase coded signal. The two first bits are start bits, always having the value 1. The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remote control transmitter.

Five system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code).

For extended RC5 code, the extended bit is bit 6 of the command. The toggle bit is stored in bit 7 of the command.

Xmega

The Xmega will use timer TCC0 instead of TIMER0.

You MUST enable the low priority interrupts since TCC0 is used in this mode. You can do this with this command :

Config Priority = Static , Vector = Application , Lo = Enabled

See also

CONFIG RC5 538 , RC5SEND 822, RC6SEND 826

Example

'name	: rc5.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: based on Atmel AVR410 application note
'micro	: 90S2313
'suited for demo	: yes
'commercial addon needed	: no

_____ \$regfile = "2313def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud** = 19200 ' use baud rate ' default \$hwstack = 32use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **Sframesize** = 40 ' default use 40 for the frame space 'use byte library for smaller code **\$lib** "mcsbyte.lbx" 'This example shows how to decode RC5 remote control signals 'with a SFH506-35 IR receiver. 'Connect to input to PIND.2 for this example 'The GETRC5 function uses TIMER0 and the TIMER0 interrupt. 'The TIMER0 settings are restored however so only the interrupt can not 'be used anymore for other tasks 'tell the compiler which pin we want to use for the receiver input **Config** Rc5 = Pind.2 'the interrupt routine is inserted automatic but we need to make it occur 'so enable the interrupts Enable Interrupts 'reserve space for variables Dim Address As Byte , Command As Byte Print "Waiting for RC5..." Do 'now check if a key on the remote is pressed 'Note that at startup all pins are set for INPUT 'so we dont set the direction here 'If the pins is used for other input just unremark the next line 'Config Pind.2 = Input Getrc5(address , Command) 'we check for the TV address and that is 0 If Address = 0 Then 'clear the toggle bit 'the toggle bit toggles on each new received command 'toggle bit is bit 7. Extended RC5 bit is in bit 6 Command = Command And &B01111111 Print Address ; " " ; Command End If LOOD End

6.249 GETREG

716

Action

Reads a byte from an internal register.

Syntax

var = GETREG(Reg)

Remarks

Most AVR chips have 32 registers named R0-R31. The GetReg function will return the value of the specified register.

Reg	The register name : R0-R31 or a register definition.
	The name of a variable that will be assigned with the content of the register.
	register.

PEEK and POKE work with an address. And will return a HW register on the Xmega since Xmega has a different address map.

GetReg and SetReg will read/write registers on all AVR processors.

See also

SETREG 858, PEEK 803, POKE 804

Example

6.250 GETTCPREGS

Action

Read a register value from the W3100A

Syntax

var = **GETTCPREGS**(address, bytes)

Remarks

Address	The address of the W3100A register.
bytes	The number of bytes to read.

Most W3100A options are implemented with BASCOM statements or functions. When there is a need to read from the W3100A register you can use the GETTCPREGS function. It can read multiple bytes. It is important that you specify the highest address. This because the registers must be read starting with the highest address.

See also

SETTCPREGS 860

ASM

NONE

Example

See SETTCPREGS 860

6.251 GETSOCKET

Action

Creates a socket for TCP/IP communication.

Syntax

Result = **GETSOCKET**(socket, mode, port, param)

Remarks

Result	A byte that is assigned with the socket number you requested. When the operation fails, it will return 255.
Mode	The socket mode. Use sock_stream(1), sock_dgrm(2), sock_ipl_raw (3), sock) or macl_raw(4). The modes are defined with constants.
	For TCP/IP communication you need to specify sock_stream or the equivalent value 1.
	For UDP communication you need to specify sock_dgrm or the equivalent value 2.
Port	This is the local port that will be used for the communication. You may specify any value you like but each socket must have it's own local port number.
	When you use 0, the value of LOCAL_PORT will be used.
	LOCAL_PORT is assigned with CONFIG TCPIP.
	After the assignment, LOCAL_PORT will be increased by 1. So the simplest way is to setup a local port with CONFIG TCPIP, and then use 0 for port.
Param	Optional parameter. Use 0 for default.
	 128 : send/receive broadcast message in UDP 64 : use register value with designated timeout value 32 : when not using no delayed ack 16: when not using silly window syndrome
	Consult the W3100A documentation for more information.

After the socket has been initialized you can use SocketConnect to connect to a client, or SocketListen to act as a server.

See also

CONFIG TCPIP 559, SOCKETCONNECT 1886, SOCKETSTAT 1890, TCPWRITE 1924,

TCPWRITESTR [925], TCPREAD [924], CLOSESOCKET [439], SOCKETLISTEN [889]

Partial Example

I = Getsocket(0 , Sock_stream , 5000 , 0)' get a new socket

6.252 GLCDCMD

Action

Sends a command byte to the SED graphical LCD display.

Syntax

GLCDCMD byte

Remarks

byte A variable or numeric constant to send to the display.

With GLCDCMD you can write command bytes to the display. This is convenient to control the display when there is no specific statement available.

You need to include the glibSED library with : \$LIB "glibsed.lbx"

See also

CONFIG GRAPHLCD जिति , LCDAT 775ते, GLCDDATA 718

Example

NONE

6.253 GLCDDATA

Action

Sends a data byte to the SED graphical LCD display.

Syntax

GLCDDATA byte

Remarks

byte A variable or numeric constant to send to the display.

With GLCDDATA you can write data bytes to the display. This is convenient to control the display when there is no specific statement available. You need to include the glibSED library with :

\$LIB "glibsed.lbx"

See also

<u>CONFIG GRAPHLCD</u> 51की, <u>LCDAT</u> 755ी, <u>GLCDCMD</u> 71की

Example

NONE

6.254 GOSUB

Action

Branch to and execute subroutine.

Syntax

GOSUB label

Remarks

Label The name of the label where to branch to.

With GOSUB, your program jumps to the specified label, and continues execution at that label.

When it encounters a RETURN statement, program execution will continue after the GOSUB statement.

See also

GOTO 7207, CALL 4227, RETURN 8437

Example

' specify

' use baud

' default

' default

' default

' used

\$regfile = "m48def.dat"
the used micro
\$crystal = 4000000
crystal frequency
\$baud = 19200
rate
\$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
use 10 for the SW stack
\$framesize = 40
use 40 for the frame space

© 2011 MCS Electronics

720 BASCOM-AVR

Goto Continue Print "This code will not be executed"

```
Continue:
with a colon
Print "We will start execution here"
Gosub Routine
Print "Back from Routine"
End
```

Routine: subroutine **Print** "This will be executed" **Return** subroutine 'start a

'return from

'end a label

6.255 GOTO

Action

Jump to the specified label.

Syntax

GOTO label

Remarks

Labels can be up to 32 characters long. When you use duplicate labels, the compiler will give you a warning.

See also

GOSUB 719

Example

Dim A As Byte Start: 'a label must end with a colon A = A + 1 'increment a If A < 10 Then 'is it less than 10? Goto Start 'do it again End If 'close IF Print "Ready" 'that is it

6.256 GRAY2BIN

Action

Returns the numeric value of a Gray code.

Syntax

var1 = GRAY2BIN(var2)

Remarks

var1	Variable that will be assigned with the binary value of the Grey code.
var2	A variable in Grey format that will be converted.

Gray code is used for rotary encoders. Gray2bin() works for byte, integer, word and long variables.

See also

BIN2GRAY 413

ASM

Depending on the data type of the target variable the following routine will be called from mcs.lbx:

_Bin2grey for bytes , _Bin2Grey2 for integer/word and _Bin2grey4 for longs.

Example

```
-----
 _____
                      : graycode.bas
'name
                      : (c) 1995-2005, MCS Electronics
'copyright
                       : show the Bin2Gray and Gray2Bin functions
'purpose
'micro
                       : Mega48
'suited for demo
                       : yes
'commercial addon needed : no
·-----
                                 _____
_____
$regfile = "m48def.dat"
                                                      ' specify
the used micro
$crystal = 4000000
                                                      ' used
crystal frequency
$baud = 19200
                                                      ' use baud
rate
$hwstack = 32
                                                      ' default
use 32 for the hardware stack
\$swstack = 10
                                                      ' default
use 10 for the SW stack
$framesize = 40
                                                      ' default
use 40 for the frame space
'Bin2Gray() converts a byte, integer, word or long into grey code.
'Gray2Bin() converts a gray code into a binary value
                                                      ' could be
Dim B As Byte
word, integer or long too
Print "BIN" ; Spc(8) ; "GREY"
For B = 0 To 15
 Print B ; Spc(10) ; Bin2gray(b)
Next
Print "GREY" ; Spc(8) ; "BIN"
For B = 0 To 15
 Print B ; Spc(10) ; Gray2bin(b)
Next
End
```

6.257 HEX

722

Action

Returns a string representation of a hexadecimal number.

Syntax

var = HEX(x)

Remarks

var	A string variable.
	A numeric variable of data type Byte, Integer, Word, Long, Single or Double.

See also

HEXVAL [722], VAL 945], STR 911, BIN 411, BINVAL 412

Example

<pre>\$regfile = "m48def.dat"</pre>	' specify
the used micro	
\$crystal = 8000000	' used
crystal frequency	
\$baud = 19200	' use baud
rate	
<pre>\$hwstack = 32</pre>	' default
use 32 for the hardware stack	
<pre>\$swstack = 10</pre>	' default
use 10 for the SW stack	
\$framesize = 40	' default
use 40 for the frame space	
<pre>Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopb</pre>	its = 1 ,

Databits = 8 , Clockpol = 0

```
Dim B As Byte , J As Integer , W As Word , L As Long
B = 1 : J = &HF001
W = &HF001
L = W
Print B ; Spc(3) ; Hex(b)
Print J ; Spc(3) ; Hex(j)
Print W ; Spc(3) ; Hex(y)
Print L ; Spc(3) ; Hex(l)
End
```

6.258 HEXVAL

Action

Convert string representing a hexadecimal number into a numeric variable.

Syntax

var = **HEXVAL**(x)

Remarks

Var	The numeric variable that must be assigned.
Х	The hexadecimal string that must be converted.

In VB you can use the VAL() function to convert hexadecimal strings. But since that would require an extra test for the leading &H signs that are required in VB, a separate function was designed.

See also

HEX [722], VAL 945], STR 911], BIN 411], BINVAL 412]

Example

\$regfile = "m48def.dat" ' specify the used micro **\$crystal** = 8000000 ' used crystal frequency ' use baud **\$baud** = 19200 rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 , Clockpol = 0

```
Dim L As Long
```

```
Dim S As String * 8
Do
    Input "Hex value " , S
    L = Hexval(s)
    Print L ; Spc(3) ; Hex(l)
Loop
```

6.259 HIGH

Action

Retrieves the most significant byte of a variable.

Syntax

var = HIGH(s)

Remarks

Var	The variable that is assigned with the MSB of var S.
S	The source variable to get the MSB from.

See also

LOW 776 , HIGHW 724

Example

```
Dim I As Integer , Z As Byte
I = &H1001
Z = High(i)
or 16 dec
End
```

' is 10 hex

6.260 HIGHW

Action

Retrieves the most significant word of a long variable.

Syntax

var = **HIGHW**(s)

Remarks

Var	The variable that is assigned with the MS word of var S.
S	The source variable to get the MSB from.

There is no LowW() function. This because when you assign a Long to a word or integer, only the lower part is assigned. For this reason you do not need a Loww() function. W=L will do the same.

See also

LOW 776 , HIGH 723

Example

```
Dim X As Word , L As Long
L = &H12345678
X = Highw(1)
Print Hex(x)
```

6.261 HOME

Action

Place the cursor at the specified line at location 1.

Syntax

HOME UPPER | LOWER | THIRD | FOURTH

Remarks

If only HOME is used than the cursor will be set to the upper line. You may also specify the first letter of the line like: HOME U

See also

<u>CLS</u> 433 , <u>LOCATE</u> 771

For a complete example see LCD 752

Partial Example

Locate 2 , 1 position Lcd "*" Home Upper 1 and return home

6.262 I2CINIT

Action

Initializes the SCL and SDA pins.

Syntax

I2CINIT I2CINIT #const

Remarks

By default the SCL and SDA pins are in the right state when you reset the chip. Both the PORT and the DDR bits are set to 0 in that case. When you need to change the DDR and/or PORT bits you can use I2CINIT to bring the pins in the proper state again.

For the XMEGA which has multiple TWI interfaces you can use a channel to specify the TWI interface otherwise the default TWIC will be used.

ASM

The I2C routines are located in i2c.lib. _i2c_init is called.

See also

I2CSEND [727], I2CSTART [728], I2CSTOP [728], I2CRBYTE [728], I2CWBYTE [728], I2C TWI Library for using TWI [966]

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
I2cinit
Dim X As Byte , Slave As Byte
X = 0
variable
Slave = &H40
address of a PCF 8574 I/O IC
I2creceive Slave , X
```

'set cursor

'display this 'select line

'reset 'slave

'get the

value **Print** X

'print it

' or use

'CONFIG TWI

Example XMEGA

Open "twic" For Binary As #4
TWID,TWIE oR TWIF
Config TwiC = 100000
will ENABLE the TWI master interface
I2cinit #4

6.263 I2CRECEIVE

Action

Receives data from an I2C serial slave device.

Syntax

I2CRECEIVE slave, var I2CRECEIVE slave, var , b2W, b2R

Syntax Xmega

I2CRECEIVE slave, var , #const I2CRECEIVE slave, var , b2W, b2R , #const

Remarks

Slave	A byte, Word/Integer variable or constant with the slave address from the I2C-device.
	 I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C 7-bit address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the data sheets which are in the same format in most cases. So if you work with 7 bit address, you need to multiply the address by 2.
Var	A byte or integer/word variable that will receive the information from the I2C-device.
b2W	The number of bytes to write. Be cautious not to specify too many bytes!
b2R	The number of bytes to receive. Be cautious not to specify too many bytes!
#const	For the Xmega, a channel constant that was specified with OPEN.

You must specify the base address of the slave chip because the read/write bit is set/ reset by the software.

When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.

The I2CRECEIVE statement combines the i2cstart,i2cwbyte, i2crbyte and i2cstop statements.

For the xmega you can optional specify the channel. Without it, SPIC will be used.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

12CSEND [727], 12CSTART [728], 12CSTOP [728], 12CRBYTE [728], 12CWBYTE [728]

Example

Config Sda = Portb.5	
Config Scl = Portb.7	
Dim X As Byte , Slave As Byte X = 0	Imagat
x = 0 variable	'reset
Slave = $\&$ H40	'slave
address of a PCF 8574 I/O IC	
I2creceive Slave , X value	'get the
Print X	'print it
Dim Buf(10)as Byte	
Buf(1) = 1 : Buf(2) = 2	

```
      I2creceive Slave , Buf(1) , 2 , 1
      'send two

      bytes and receive one byte
      'print Buf(1)

      Print Buf(1)
      'print the

      received byte
      End
```

6.264 I2CSEND

Action

Send data to an I2C-device.

Syntax

I2CSEND slave, var I2CSEND slave, var , bytes

Syntax Xmega

I2CSEND slave, var , #const I2CSEND slave, var , bytes , #const

Remarks

Slave	The slave address off the I2C-device.
	I2C uses a 7 bit address from bit 1 to bit 7. Bit 0 is used to specify a read/write operation. In BASCOM the byte transmission address is used for I2C. This means that an I2C 7-bit address of 1 becomes &B10 = 2. And we say the address is 2. This is done so you can copy the address from the

	data sheets which are in the same format in most cases. So if you work with 7 bit address, you need to multiply the address by 2.
Var	A byte, integer/word or numbers that holds the value, which will be, send to the I2C-device.
Bytes	The number of bytes to send.
#const	For the Xmega, a channel constant that was specified with OPEN.

When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.

The I2CSEND statement combines the i2cstart,i2cwbyte and i2cstop statements. For the xmega you can optional specify the channel. Without it, SPIC will be used.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

I2CRECEIVE 728), I2CSTART 728), I2CSTOP 728), I2CRBYTE 728), I2CWBYTE 728)

Example

Config Sda = Portb.5Config Scl = Portb.7Dim X As Byte , A As Byte , Bytes As ByteX = 5variable to 5Dim Ax(10)as ByteConst Slave = &H40address of a PCF 8574 I/O ICI2csend Slave , Xvalue or

```
For A = 1 To 10
   Ax(a) = A
dataspace
Next
Bytes = 10
I2csend Slave , Ax(1) , Bytes
End
```

'Fill

6.265 I2START, I2CSTOP, I2CRBYTE, I2CWBYTE, I2CREPSTART

Action

I2CSTART generates an I2C start condition. I2CREPSTART generates an I2C repeated start condition. I2CSTOP generates an I2C stop condition. I2CRBYTE receives one byte from an I2C-device. I2CWBYTE sends one byte to an I2C-device.

Syntax I2CSTART I2CREPSTART I2CSTOP I2CRBYTE var, ack/nack I2CWBYTE val

Syntax Xmega

I2CSTART #const I2CREPSTART #const I2CSTOP #const I2CRBYTE var, ack/nack , #const I2CWBYTE val , #const

Remarks

Var	A variable that receives the value from the I2C-device.
ack/nack	Specify ACK if there are more bytes to read.
	Specify NACK if it is the last byte to read.
Val	A variable or constant to write to the I2C-device.
#const	For the Xmega, a channel constant that was specified with OPEN.

These statements are provided as an addition to the I2CSEND and I2CRECEIVE statements.

While I2CSEND and I2CRECEIVE are well suited for most tasks, a slave chip might need a special sequence that is not possible with the I2C routines.

When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.

The Xmega has multiple TWI interfaces. You can use a channel to specify the TWI interface.

When using a repeated start, you must use I2CREPSTART on the XMega. Normal AVR processors may use either I2CSTART or I2CREPSTART.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files. For the XMega, the routines are located in the xmega.lib file.

See also

 I2CSEND
 7271
 I2CRECEIVE
 7261
 I2CSTART
 7281
 I2CSTOP
 7281
 I2CRBYTE
 7281
 I2CWBYTE

 17281
 I2CRECEIVE
 7261
 I2CSTART
 12CSTOP
 7281
 I2CRBYTE
 7281
 I2CWBYTE

Example

```
$regfile = "m48def.dat"
                                                              ' specify
the used micro
                                                              ' used
$crystal = 4000000
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Config Scl = Portb.4
Config Sda = Portb.5
Declare Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
Declare Sub Read_eeprom(byval Adres As Byte , Value As Byte)
Const Addressw = 174
                                                             'slave write
address
Const Addressr = 175
                                                              'slave read
address
Dim B1 As Byte , Adres As Byte , Value As Byte
                                                             'dim byte
Call Write eeprom(1, 3)
                                                              'write value
of three to address 1 of EEPROM
Call Read_eeprom(1 , Value) : Print Value
                                                              'read it
back
Call Read eeprom(5 , Value) : Print Value
                                                             'again for
address 5
'---- now write to a PCF8474 I/O expander ------
12csend &H40 , 255
                                                             'all outputs
high
I2creceive &H40 , B1
                                                              'retrieve
input
Print "Received data " ; B1
                                                              'print it
End
Rem Note That The Slaveaddress Is Adjusted Automaticly With I2csend &
I2creceive
Rem This Means You Can Specify The Baseaddress Of The Chip.
'sample of writing a byte to EEPROM AT2404
Sub Write_eeprom(byval Adres As Byte , Byval Value As Byte)
    I2cstart
                                                              'start
condition
    12cwbyte Addressw
                                                              'slave
address
    I2cwbyte Adres
                                                              'asdress of
EEPROM
    12cwbyte Value
                                                              'value to
write
    I2cstop
                                                              'stop
condition
    Waitms 10
                                                              'wait for 10
```

© 2011 MCS Electronics

```
milliseconds
End Sub
'sample of reading a byte from EEPROM AT2404
Sub Read_eeprom(byval Adres As Byte , Value As Byte)
   I2cstart
                                                               'generate
start
   12cwbyte Addressw
                                                               'slave
adsress
                                                              'address of
   I2cwbyte Adres
EEPROM
                                                               'repeated
   I2cstart
start
   12cwbyte Addressr
                                                              'slave
address (read)
   I2crbyte Value , Nack
                                                              'read byte
   I2cstop
                                                              'generate
stop
End Sub
' when you want to control a chip with a larger memory like the 24c64 it
requires an additional byte
' to be sent (consult the datasheet):
' Wires from the I2C address that are not connected will default to 0 in
most cases!
1
    I2cstart
                                                              'start
condition
   I2cwbyte &B1010_0000
                                                               'slave
address
    I2cwbyte H
                                                              'high
1
address
   I2cwbyte L
                                                               'low address
1
   I2cwbyte Value
                                                               'value to
write
   I2cstop
                                                              'stop
condition
  Waitms 10
```

Xmega Example

```
_____
                  (c) 1995-2010, MCS
                     xm128-TWI.bas
1
 This sample demonstrates the Xmega128A1 TWI
$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
\$swstack = 40
$framesize = 40
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
Dim S As String * 20
'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
```

```
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Dim N As String * 16 , B As Byte
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Config Input1 = Cr , Echo = Crlf
                                                              ' CR is used
for input, we echo back CR and LF
Open "COM1:" For Binary As #1
        ^^^^ change from COM1-COM8
Print #1 , "Xmega revision:" ; Mcu_revid
                                                             ' make sure
it is 7 or higher !!! lower revs have many flaws
Const Usechannel = 1
Dim B1 As Byte , B2 As Byte
Dim W As Word At B1 Overlay
Open "twic" For Binary As #4
                                                              ' or use
TWID, TWIE OR TWIF
Config Twi = 100000
                                                              'CONFIG TWI
will ENABLE the TWI master interface
'you can also use TWIC, TWID, TWIE of TWIF
#if Usechannel = 1
   I2cinit #4
#else
  I2cinit
#endif
Do
  I2cstart
  Waitms 20
  12cwbyte &H70
                                                              ' slave
address write
  Waitms 20
  12cwbyte & B10101010
                                                              ' write
command
 Waitms 20
  I2cwbyte 2
  Waitms 20
  I2cstop
  Print "Error : " ; Err
                                                              ' show error
status
 'waitms 50
 Print "start"
  I2cstart
  Print "Error : " ; Err
                                                              ' show error
  I2cwbyte &H71
  Print "Error : " ; Err
                                                              ' show error
  I2crbyte B1 , Ack
  Print "Error : " ; Err
                                                              ' show error
  I2crbyte B2 , Nack
  Print "Error : " ; Err
                                                              ' show error
```

```
I2cstop
 Print "received A/D : " ; W ; "-" ; B1 ; "-" ; B2
 Waitms 500
                                                               'wait a bit
Loop
Dim J As Byte , C As Byte , K As Byte
Dim Twi_start As Byte
                                                               ' you MUST
dim this variable since it is used by the lib
'determine if we have an i2c slave on the bus
For J = 0 To 200 Step 2
 Print J
  #if Usechannel = 1
    I2cstart #4
  #else
    I2cstart
  #endif
   I2cwbyte J
   If Err = 0 Then
                                                               ' no errors
      Print "FOUND : " ; Hex(j)
      'write some value to the pcf8574A
      #if Usechannel = 1
         I2cwbyte &B1100_0101 , #4
      #else
         12cwbyte &B1100_0101
      #endif
      Print Err
      Exit For
   End If
   #if Usechannel = 1
       I2cstop #4
   #else
       I2cstop
   #endif
Next
#if Usechannel = 1
    I2cstop #4
#else
    I2cstop
#endif
#if Usechannel = 1
 I2cstart #4
 12cwbyte &H71 , #4
                                                               'read
address
 I2crbyte J , Ack , #4
 Print Bin(j) ; " err:" ; Err
 I2crbyte J , Ack , #4
 Print Bin(j) ; " err:" ; Err
 <code>I2crbyte J</code> , <code>Nack</code> , #4
 Print Bin(j) ; " err:" ; Err
 I2cstop #4
#else
 I2cstart
 I2cwbyte &H71
                                                               'read
address
  I2crbyte J , Ack
 Print Bin(j) ; " err:" ; Err
 I2crbyte J , Ack
 Print Bin(j) ; " err:" ; Err
```

734

```
I2crbyte J , Nack
  Print Bin(j) ; " err:" ; Err
  I2cstop
#endif
'try a transaction
#if Usechannel = 1
  12csend &H70 , 255 , #4
                                                              ' all 1
  Waitms 1000
                                                              'all O
  12csend & H70 , 0 , #4
#else
  12csend &H70 , 255
  Waitms 1000
  12csend & H70 , 0
#endif
Print Err
'read transaction
Dim Var As Byte
Var = &B11111111
#if Usechannel = 1
  12creceive &H70 , Var , 1 , 1 , #4
                                                              ' send and
receive
 Print Bin(var) ; "-" ; Err
  I2creceive &H70 , Var , 0 , 1 , #4
                                                              ' just
receive
 Print Bin(var) ; "-" ; Err
#else
  I2creceive &H70 , Var , 1 , 1
                                                               ' send and
receive
  Print Bin(var) ; "-" ; Err
  I2creceive &H70 , Var , 0 , 1
                                                              ' just
receive
 Print Bin(var) ; "-" ; Err
#endif
```

End

6.266 IDLE

Action

Put the processor into the idle mode.

Syntax

Remarks

In the idle mode, the system clock is removed from the CPU but not from the interrupt logic, the serial port or the timers/counters.

The idle mode is terminated either when an interrupt is received(from the watchdog, timers, external level triggered or ADC) or upon system reset through the RESET pin.

Most new chips have many options for Power down/Idle. It is advised to consult the data sheet to see if a better mode is available.

You should use the new POWER IDLE statement.

See also

POWERDOWN (a), POWERSAVE (a), POWER mode (a)

Example

IDLE

6.267 IF-THEN-ELSE-END IF

Action

Allows conditional execution or branching, based on the evaluation of a Boolean expression.

Syntax

IF expression THEN

[ELSEIF expression THEN]

[**ELSE**]

END IF

Remarks

Expression Any expression that evaluates to true or false.

The one line version of IF can be used : IF expression THEN statement [ELSE statement] The use of [ELSE] is optional.

Tests like IF THEN can also be used with bits and bit indexes. IF var.bit = 1 THEN ^--- bit is a variable or numeric constant in the range from 0-255

You can use OR or AND to test on multiple conditions. The conditions are evaluated from left to right. IF A=1 OR A=2 OR A=3 OR B>10 THEN IF A=1 AND A>3 THEN

Dim Var As Byte, Idx As Byte Var = 255 Idx = 1 If Var.idx = 1 Then Print "Bit 1 is 1" EndIf

See also

ELSE 675

Example

```
Dim A As Integer
A = 10
If A = 10 Then
                                                             'test
expression
 Print "This part is executed."
                                                             'this will
be printed
Else
 Print "This will never be executed."
                                                             'this not
End If
If A = 10 Then Print "New in BASCOM"
If A = 10 Then Goto Label1 Else print "A<>10"
Label1:
Rem The following example shows enhanced use of IF THEN
If A.15 = 1 Then
                                                             'test for
bit
 Print "BIT 15 IS SET"
EndIf
Rem the following example shows the 1 line use of IF THEN [ELSE]
If A.15 = 0 Then Print "BIT 15 is cleared" Else Print "BIT 15 is set"
```

6.268 INCR

Action

Increments a variable by one.

Syntax

INCR var

Remarks

Var Any numeric variable.

See also

DECR 640

Example

```
_____
! _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
     _____
'name
                  : incr.bas
                  : (c) 1995-2005, MCS Electronics
'copyright
                  : demo: INCR
'purpose
'micro
                  : Mega48
             : yes
'suited for demo
'commercial addon needed : no
                         _____
_____
```

```
$regfile = "m48def.dat"
the used micro
$crystal = 4000000
crystal frequency
```

' specify

' used

BASCOM Language Reference

737

\$baud = 19200 rate	' use baud
<pre>\$hwstack = 32 use 32 for the hardware stack</pre>	' default
<pre>\$\$ \$2 for the hardware stack \$\$ swstack = 10 use 10 for the SW stack</pre>	' default
\$framesize = 40 use 40 for the frame space	' default
Dim A As Byte	
A = 5 value to a	'assign
Incr A	'inc (by
one) Print A	'print it

6.269 INITFILESYSTEM

Action

Initialize the file system

Syntax

bErrorCode = **INITFILESYSTEM** (bPartitionNumber)

Remarks

bErrorCode	(Byte) Error Result from Routine, Returns 0 if no Error
bPartitionNumber	(Byte) Partition number on the Flashcard Drive (normally 1)

Reads the Master boot record and the partition boot record (Sector) from the flash card and initializes the file system.

This function must be called before any other file-system function is used.

See also

OPEN 7947, CLOSE 4377, FLUSH 6897, PRINT 6107, LINE INPUT 7627, LOC 7667, LOF 7677, EOF 6877, FREEFILE 6987, FILEATTR 6847, SEEK 6807, BSAVE 4207, BLOAD 41677, KILL 7577, DISKFREE 6607 , DISKSIZE 66日, GET 697, PUT 819, FILEDATE 688日, FILETIME 687, FILEDATETIME 688日, DIR 656, FILELEN 687, WRITE 953, INPUT 744

ASM

Calls	_GetFileSystem	
Input	r24: partitionnumber (1-based)	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

Dim bErrorCode as Byte bErrorCode = InitFileSystem(1) If bErrorCode > 0 then Print "Error: "; bErrorCode

Else

Print "Filesystem successfully initialized" End If

6.270 INITLCD

Action

Initializes the LCD display.

Syntax INITLCD

Remarks

The LCD display is initialized automatic at start up when LCD statements are used by your code.

This is done by a call to _LCD_INIT.

If you include the INITLCD statement in your code, the automatic call is disabled and the _LCD_INIT is called at the place in your code where you put the INITLCD statement. (initlcd is translated into a call to _init_lcd).

Why is this useful?

• In an environments with static electricity, the display can give strange output. You can initialize the display then once in a while. When the display is initialized, the display content is cleared also.

- The LCD routines depend on the fact that the WR pin of the LCD is connected to ground. But when you connect it to a port pin, you must first set the logic level to 0 and after that you can initialize the display by using INITLCD
- Xmega chips need a stable oscillator. This is done with some CONFIG statements. The INITLCD should be placed after these commands.
- So in short you have more control when the LCD is initialized.



The <u>CONFIG LCDPIN</u> [523] has an option to use the WR pin, and use the busy flag of the display. If you have enough pins, this is the best mode.

ASM

The generated ASM code : Rcall _Init_LCD

See also

LCD 3327, CONFIG LCDPIN 523

Example

NONE

6.271 INKEY

Action

Returns the ASCII value of the first character in the serial input buffer.

Syntax

var = INKEY()
var = INKEY(#channel)

Remarks

Var	Byte, Integer, Word, Long or String variable.
	A constant number that identifies the opened channel if software UART mode

If there is no character waiting, a zero will be returned. Use the IsCharWaiting() function to check if there is a byte waiting.

The INKEY routine can be used when you have a RS-232 interface on your uP. The RS-232 interface can be connected to a comport of your computer.

As zero(0) will be returned when no character is waiting, the usage is limited when the value of 0 is used in the serial transmission. You can not make a difference between a byte with the value 0 and the case where no data is available. In that case you can use IsCharwaiting to deterimine if there is a byte waiting.

See also

WAITKEY [949], ISCHARWAITING [750]

Example

'purpose		
'		
<pre>\$regfile = "m48def.dat" the used micro \$crystal = 4000000 crystal frequency \$baud = 19200 rate \$hwstack = 32 use 32 for the hardware st \$swstack = 10 use 10 for the SW stack \$framesize = 40 use 40 for the frame space</pre>		' specify ' used ' use baud ' default ' default ' default
Dim A As Byte , S As Strin	ng * 2	
<pre>Do A = Inkey() value from serial port 's = Inkey()</pre>		'get ascii
If A > 0 Then something		'we got

```
Print "ASCII code " ; A ; " from serial"
   End If
Loop Until A = 27
                                                                 'until ESC
is pressed
A = Waitkey()
                                                                 'wait for a
key
's = waitkey()
Print Chr(a)
'wait until ESC is pressed
Do
Loop Until Inkey() = 27
'When you need to receive binary data and the bibary value \ensuremath{\mathsf{0}} ,
'you can use the IScharwaiting() function.
'This will return 1 when there is a char waiting and 0 if there is no
char waiting.
'You can get the char with inkey or waitkey then.
End
```

6.272 INP

Action

Returns a byte read from a hardware port or any internal or external memory location.

Syntax

var = INP(address)

Remarks

var	Numeric variable that receives the value.
address	The address where to read the value from. (0- &HFFFF)

The PEEK() function will read only the lowest 32 memory locations (registers). The INP() function can read from any memory location since the AVR has a linear memory model.

When you want to read from XRAM memory you must enable external memory access in the <u>Compiler Chip Options</u> [90⁻].

See also

OUT 8027, PEEK 8037, POKE 8047

Example

```
'name : peek.bas
'copyright : (c) 1995-2005, MCS Electronics
'purpose : demonstrates PEEk, POKE, CPEEK, INP and OUT
'micro : Mega162
'suited for demo : yes
'commercial addon needed : no
```

```
_____
$regfile = "m162def.dat"
                                                               ' specify
the used micro
                                                              ' used
$crystal = 4000000
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
$swstack = 10
                                                              ' default
use 10 for the SW stack
                                                              ' default
$framesize = 40
use 40 for the frame space
Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31
                                                              'only 32
registers in AVR
  B1 = Peek(i)
                                                              'get byte
from internal memory
  Print Hex(b1) ; " ";
  'Poke I , 1
                                          'write a value into memory
Next
Print
                                                             'new line
'be careful when writing into internal memory !!
'now dump a part of the code-memory(program)
For I = 0 To 255
 B1 = Cpeek(i)
                                                             'get byte
from internal memory
  Print Hex(b1) ; " ";
Next
'note that you can not write into codememory !!
Out &H8000 , 1
                                                              'write 1
into XRAM at address 8000
B1 = Inp(\&H8000)
                                                              'return
value from XRAM
Print B1
End
```

6.273 INPUTBIN

Action

Read binary data from the serial port.

Syntax

INPUTBIN var1 [,var2] **INPUTBIN** #channel , var1 [,var2]

Remarks

var1	The variable that is assigned with the characters from the serial port.
	An optional second (or more) variable that is assigned with the data from the serial input stream.

The channel is for use with the software UART routine and must be used with OPEN

794 and CLOSE. 794

The number of bytes to read depends on the variable you use. When you use a byte variable, 1 character is read from the serial port. An integer will wait for 2 characters and an array will wait until the whole array is filled.

Note that the INPUTBIN statement doesn't wait for a <RETURN> but just for the number of bytes.

You may also specify an additional numeric parameter that specifies how many bytes will be read. This is convenient when you are filling an array.

Inputbin ar(1), 4 ' will fill 4 bytes starting at index 1.

See also

PRINTBIN 812

Example

```
Dim A As Byte , C As Integer
Inputbin A , C 'wait for 3 characters
End
```

6.274 INPUTHEX

Action

Allows hexadecimal input from the keyboard during program execution.

Syntax

INPUTHEX [" prompt"] , var[, varn]

Remarks

prompt	An optional string constant printed before the prompt character.
Var,varn	A numeric variable to accept the input value.

The INPUTHEX routine can be used when you have a RS-232 interface on your uP. The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator and the keyboard as input device. You can also use the build in terminal emulator. The input entered may be in lower or upper case (0-9 and A-F)

If var is a byte then the input can be maximum 2 characters long. If var is an integer/word then the input can be maximum 4 characters long. If var is a long then the input can be maximum 8 characters long.

In VB you can specify **&H** with INPUT so VB will recognize that a hexadecimal string is being used.

BASCOM implements a new statement: INPUTHEX. This is only to save code as otherwise also code would be needed for decimal conversion.

See also

INPUT 744 , ECHO 673 , INPUTBIN 741

```
Example
                 _____
_____
                      : input.bas
'name
                      : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                      : demo: INPUT, INPUTHEX
                      : Mega48
'micro
'suited for demo
                      : yes
'commercial addon needed : no
·-----
                                 _____
_____
$regfile = "m48def.dat"
                                                     ' specify
the used micro
$crystal = 4000000
                                                     ' used
crystal frequency
$baud = 19200
                                                     ' use baud
rate
$hwstack = 32
                                                     ' default
use 32 for the hardware stack
$swstack = 10
                                                     ' default
use 10 for the SW stack
$framesize = 40
                                                     ' default
use 40 for the frame space
Dim V As Byte , B1 As Byte
Dim C As Integer , D As Byte
Dim S As String * 15
Input "Use this to ask a question " , V
Input B1
                                                     'leave out
for no question
Input "Enter integer " , C
Print C
Inputhex "Enter hex number (4 bytes) ", C
Print C
Inputhex "Enter hex byte (2 bytes) ", D
Print D
Input "More variables " , C , D
Print C ; " " ; D
Input C Noecho
                                                     'supress
echo
Input "Enter your name " , S
Print "Hello " ; S
Input S Noecho
                                                     'without
echo
Print S
End
```

6.275 INPUT

Action

Allows input from the keyboard, file or SPI during program execution.

Syntax

INPUT [" prompt"] , var[, varn]
INPUT #ch, var[, varn]

Remarks

Prompt	An optional string constant printed before the prompt character.
Var,varn	A variable to accept the input value or a string.
	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.

The INPUT routine can be used when you have an RS-232 interface on your uP. The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator and the keyboard as an input device. You can also use the built-in terminal emulator.

For usage with the AVR-DOS file system, you can read variables from an opened file. Since these variables are stored in ASCII format, the data is converted to the proper format automatically.

When you use INPUT with a file, the prompt is not supported.

Difference with VB

In VB you can specify **&H** with INPUT so VB will recognize that a hexadecimal string is being used.

BASCOM implements a new statement : INPUTHEX.

Xmega-SPI

When receiving data from the SPI interface, you need to activate the SS pin. Some chips might need an active low, others might need an active high. This will depends on the slave chip.

When you use the SS=AUTO option, the level of SS will be changed automatic. Thus SS is made low, then the data bytes are received, and finally , SS is made high again.

Receiving data works by sending a data byte and returning the data that is shifted out. The data that will be sent is a 0. You can alter this in the library, _inputspivar routine.

You can not sent constants using the INPUT with SPI. So INPUT #10, "SPI", var is not supported.

INPUT used with SPI will not wait for a return either. It will wait for the number of bytes that fits into the variable. See <u>CONFIG SPIx</u> [550] for an example.

Number of Bytes

The compiler will receive 1 byte for a variable which was dimensioned as a BYTE. It will receive 2 bytes for a WORD/INTEGER, 4 bytes for a LONG/SINGLE and 8 bytes for a DOUBLE.

As with all routines in BASCOM, the least significant Byte will be received first.

If you specify an array, it will receive one element. With an optional parameter you can provide how many bytes must be received. You must use a semicolon (;) to specify this parameter. This because the comma (,) is used to receive multiple variables.

Sample

Dim Tmparray(5) As Byte , Spi_send_byte As Byte , W as Word Input #12 , Spi_receive_byte ; 1 'READ 1 byte Input #12 , Tmparray(1) ; 1 , Tmparray(2) ; B 'read 1 byte and 'b' bytes starting at element 2

See also

INPUTHEX [742], PRINT [810], ECHO [673], WRITE [953], INPUTBIN [741]

Example

-----_____ : input.bas 'name 'copyright : (c) 1995-2005, MCS Electronics 'purpose : demo: INPUT, INPUTHEX : Mega48 'micro 'suited for demo : yes 'commercial addon needed : no _____ !_____. _____ \$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10 ' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Dim V As Byte , B1 As Byte Dim C As Integer , D As Byte Dim S As String * 15 Input "Use this to ask a question " , V Input B1 'leave out for no question Input "Enter integer " , C Print C Inputhex "Enter hex number (4 bytes) ", C Print C

```
Inputhex "Enter hex byte (2 bytes) " , D
Print D
Input "More variables " , C , D
Print C ; " " ; D
Input C Noecho
echo
Input "Enter your name " , S
Print "Hello " ; S
Input S Noecho
echo
Print S
End
```

6.276 INSERTCHAR

Action

Inserts one character into a string.

Syntax

INSERTCHAR string, pos, char

Remarks

string	The string where the character is inserted to.
pos	The position where the character is inserted to. A value of 1 would make the character the first character of the string.
char	A byte or string or string constant with the character that need to be inserted. For example you can use "A" to insert an "A", or use a byte with the value 65 to insert an "A". Or use a string. In case of a string, only the first character will be used.

See also

DELCHAR 657, DELCHARS 6527, INSTR 7477, MID 7877, CHARPOS 4247, REPLACECHARS 6407

Example

'supress

'without

```
dim s as string * 30
s = "This is a test string" ' create a string
delchar s, 1 ' remove the first char
print s ' print it
insertchar s,1, "t" ' put a small t back
print s
delchars s,"s" ' remove all s
print s
end
```

6.277 INSTR

Action

Returns the position of a sub string in a string.

Syntax

var = INSTR(start , string , substr)
var = INSTR(string , substr)

Remarks

Var	Numeric variable that will be assigned with the position of the sub string in the string. Returns 0 when the sub string is not found.
Start	An optional numeric parameter that can be assigned with the first position where must be searched in the string. By default (when not used) the whole string is searched starting from position 1.
String	The string to search.
Substr	The search string.

No constant can be used for *string* it must be a string variable. Only *substr* can be either a string or a constant.

See also

SPLIT 901, CHARPOS 424

Example

```
_____
                  : instr.bas
'name
'copyright
'purpose
'micro
                   : (c) 1995-2005, MCS Electronics
                    : INSTR function demo
'suited for demo : ves
'commercial addon needed : no
! _____
                            _____
_____
$regfile = "m48def.dat"
                                                ' specify
the used micro
$crystal = 4000000
                                                ' used
crystal frequency
```

·_____

```
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
                                                              ' default
\$swstack = 10
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
'dimension variables
Dim Pos As Byte
Dim S As String * 8 , Z As String * 8
'assign string to search
S = "abcdeab"
                                                              ' Z = "ab"
'assign search string
Z = "ab"
'return first position in pos
Pos = Instr(s , Z)
'must return 1
'now start searching in the string at location 2
Pos = Instr(2, S, Z)
'must return 6
Pos = Instr(s , "xx")
'xx is not in the string so return 0
End
```

6.278 INT

Action

Returns the integer part of a single or double.

Syntax

var = INT(source)

Remarks

	A numeric variable that is assigned with the integer of variable source.
Source	The source variable to get the integer of.

The fraction is the right side after the decimal point of a single. The integer is the left side before the decimal point.

1234.567 1234 is the integer part, .567 is the fraction

See Also

FRAC 694 , FIX 688 , ROUND 847

Example

749

```
_____
_____
'name
                       : round_fix_int.bas
'copyright
                       : (c) 1995-2005, MCS Electronics
'purpose
                        : demo : ROUND, FIX
'micro
                       : Mega48
'suited for demo : yes
'commercial addon needed : no
!_____
_____
$regfile = "m48def.dat"
                                                        ' specify
the used micro
$crystal = 4000000
                                                        ' used
crystal frequency
$baud = 19200
                                                        ' use baud
rate
\$hwstack = 32
                                                        ' default
use 32 for the hardware stack
$swstack = 10
                                                        ' default
use 10 for the SW stack
$framesize = 40
                                                       ' default
use 40 for the frame space
Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
 Print S ; Spc(3) ; Round(s) ; Spc(3) ; Fix(s) ; Spc(3) ; Int(s)
Next
End
```

6.279 IP2STR

Action

Convert an IP number into it's string representation.

Syntax

Var = **IP2STR**(num)

Remarks

An IP number is represented with dots like 192.168.0.1. The IP2STR function converts an IP number into a string. This function is intended to be used in combination with the BASCOM TCP/IP routines.

Var	The string variable that is assigned with the IP number
Num	A variable that contains the ip number is numeric format.

See also

CONFIG TCPIP 559

6.280 ISCHARWAITING

Action

Returns one(1) when a character is waiting in the hardware UART buffer.

Syntax

var = **ISCHARWAITING**()

var = ISCHARWAITING(#channel)

Remarks

Var	Byte, Integer, Word or Long variable.
Channel	A constant number that identifies the opened channel.

If there is no character waiting, a zero will be returned. If there is a character waiting, a one (1) will be returned. The character is not retrieved or altered by the function.

While the Inkey() will get the character from the HW UART when there is a character in the buffer, it will return a zero when the character is zero. This makes it unusable to work with binary data that might contain the value 0.

With IsCharWaiting() you can first check for the presence of a character and when the function returns 1, you can retrieve the character with Inkey or Waitkey.

See also

WAITKEY 949 , INKEY 738

Example

<pre>\$regfile = "m48def.dat"</pre>	' specify
the used micro	
\$crystal = 4000000	' used
crystal frequency	
\$baud = 19200	' use baud
rate	
<pre>\$hwstack = 32</pre>	' default
use 32 for the hardware stack	
<pre>\$swstack = 10</pre>	' default
use 10 for the SW stack	
\$framesize = 40	' default
use 40 for the frame space	
Dim A As Byte , S As String * 2	
Do	
A = Ischarwaiting()	
If $A = 1$ Then	'we got
something	
A = Waitkey()	'get it
Print "ASCII code " ; A ; " from serial"	
End If	
Loop Until A = 27	'until ESC
is pressed	

6.281 KILL

Action

Delete a file from the Disk

Syntax

KILL sFileName

Remarks

sFileName A String variable or string expression, which denotes the file to delete

This function deletes a file from the disk. A file in use can't be deleted. WildCards in Filename are not supported. Check the DOS-Error in variable gDOSError.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [688], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [698], FILEATTR [684], SEEK [856], BSAVE [426], BLOAD [416], DISKFREE [666], DISKSIZE [667], GET [697], PUT [819], FILEDATE [688], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_DeleteFile	
Input	X: Pointer to string with filename	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

'We can use the KILL statement to delete a file. 'A file mask is not supported Print "Kill (delete) file demo" Kill "test.txt"

6.282 LCASE

Action

Converts a string in to all lower case characters.

Syntax

Target = **LCASE**(source)

Remarks

Target	The string that is assigned with the lower case string of string target.
Source	The source string.

See also

UCASE 936

ASM

The following ASM routines are called from MCS.LIB : _LCASE The generated ASM code : (can be different depending on the micro used) ;##### Z = Lcase(s) Ldi R30,\$60 Ldi R31,\$00 ; load constant in register Ldi R26,\$6D Rcall _Lcase

Example

<pre>\$regfile = "m48def.dat"</pre>	1	specify
the used micro \$crystal = 4000000	,	used
crystal frequency \$baud = 19200	,	use baud
rate Shwstack = 32		default
use 32 for the hardware stack		
<pre>\$swstack = 10 use 10 for the SW stack</pre>	'	default
\$framesize = 40 use 40 for the frame space	, i	default
use to tot the traile space		

```
Dim S As String * 12 , Z As String * 12
S = "Hello World"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z
End
```

6.283 LCD

Action

Send constant or variable to LCD display.

Syntax

LCD x

Remarks

X Variable or constant to display.

More variables can be displayed separated by the ; -sign

LCD a ; b1 ; "constant"

The LCD statement behaves just like the $\frac{PRINT}{B10}$ statement. So $\frac{SPC}{B10}$ () can be used too.

The only difference with PRINT is that no CR+LF is added when you send data to the

LCD.

See also

<u>\$LCD</u>[332], <u>\$LCDRS</u>[337], <u>CONFIG LCD</u>[516], <u>SPC</u>[698], <u>CLS</u>[433], <u>INITLCD</u>[738], <u>SHIFTLCD</u>[882] , <u>SHIFTCURSOR</u>[877], <u>CURSOR</u>[609]

Example

_____ _____ _____ 'name : lcd.bas : (c) 1995-2005, MCS Electronics 'copyright 'purpose : demo: LCD, CLS, LOWERLINE, SHIFTLCD, SHIFTCURSOR, HOME CURSOR, DISPLAY 'micro : Mega8515 'suited for demo : yes 'commercial addon needed : no ·_____ _____ \$regfile = "m8515.dat" ' specify the used micro ' used \$crystal = 4000000crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack \$framesize = 40 ' default use 40 for the frame space \$sim 'REMOVE the above command for the real program !! '\$sim is used for faster simulation 'note : tested in PIN mode with 4-bit 'Config Lcdpin = Pin , Db4 = Portb.1 , Db5 = Portb.2 , Db6 = Portb.3 , Db7 = Portb.4 , E = Portb.5 , Rs = Portb.6 Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 , Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6 'These settings are for the STK200 in PIN mode 'Connect only DB4 to DB7 of the LCD to the LCD connector of the STK D4-D7 'Connect the E-line of the LCD to A15 (PORTC.7) and NOT to the E line of the LCD connector 'Connect the RS, V0, GND and =5V of the LCD to the STK LCD connector Rem with the config lcdpin statement you can override the compiler settings Dim A As Byte Config Lcd = 16×2 'configure

lcd screen

```
'other options are 16 * 4 and 20 * 4, 20 * 2 , 16 * 1a 'When you dont include this option 16 * 2 is assumed
'16 * 1a is intended for 16 character displays with split addresses over
2 lines
'$LCD = address will turn LCD into 8-bit databus mode
        use this with uP with external RAM and/or ROM
.
        because it aint need the port pins !
Cls
                                                                  'clear the
LCD display
Lcd "Hello world."
                                                                  'display
this at the top line
Wait 1
Lowerline
                                                                  'select the
lower line
Wait 1
Lcd "Shift this."
                                                                  'display
this at the lower line
Wait 1
For A = 1 To 10
                                                                  'shift the
  Shiftlcd Right
text to the right
  Wait 1
                                                                  'wait a
moment
Next
For A = 1 To 10
  Shiftlcd Left
                                                                  'shift the
text to the left
   Wait 1
                                                                  'wait a
moment
Next
Locate 2 , 1
                                                                  'set cursor
position
Lcd "*"
                                                                  'display
this
Wait 1
                                                                  'wait a
moment
Shiftcursor Right
                                                                  'shift the
cursor
Lcd "@"
                                                                  'display
this
Wait 1
                                                                  'wait a
moment
Home Upper
                                                                  'select line
1 and return home
Lcd "Replaced."
                                                                  'replace the
text
Wait 1
                                                                  'wait a
moment
Cursor Off Noblink
                                                                  'hide cursor
Wait 1
                                                                  'wait a
moment
Cursor On Blink
                                                                  'show cursor
Wait 1
                                                                  'wait a
moment
Display Off
                                                                  'turn
display off
Wait 1
                                                                  'wait a
```

```
moment
Display On
                                                             'turn
display on
        -----NEW support for 4-line LCD-----
. _ _ _ _ _ _ _
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third
                                                             'goto home
on line three
Home Fourth
                                                             'first
Home F
letteer also works
Locate 4 , 1 : Lcd "Line 4"
Wait 1
'Now lets build a special character
'the first number is the characternumber (0-7)
'The other numbers are the rowvalues
'Use the LCD tool to insert this line
Deflcdchar 1 , 225 , 227 , 226 , 226 , 226 , 242 , 234 , 228
replace ? with number (0-7)
Deflcdchar 0 , 240 , 224 , 224 , 255 , 254 , 252 , 248 , 240
replace ? with number (0-7)
Cls
                                                            'select data
RAM
Rem it is important that a CLS is following the deflcdchar statements
because it will set the controller back in datamode
Lcd Chr(0); Chr(1)
                                                            'print the
special character
'----- Now use an internal routine ------
\_temp1 = 1
                                                            'value into
ACC
!rCall _write_lcd
                                                             'put it on
LCD
End
```

6.284 LCDAT

Action

Send constant or variable to a SED or other graphical display.

Syntax

LCDAT y , x , var [, inv] LCDAT y , x , var [, FG, BG]

Remarks

х	X location. In the range from 0-63. The SED displays columns are 1 pixel width. Other displays might have a bigger range such as 132 or 255.
Y	Y location. The row in pixels. The maximum value depends on the display.
Var	The constant or variable to display
inv	Optional number. Value 0 will show the data normal. Any other value will invert the data.

	For COLOR DISPLAYS
FG	Foreground color
BG	Background color

You need to include the glibSED library with : \$LIB "glibsed.lbx"

Other libraries must be included with a different directive.

See also

<u>CONFIG GRAPHLCD</u> जिंही, <u>SETFONT</u> किंही, <u>GLCDCMD</u> 7िंही, <u>GLCDDATA</u> 7िंही

Example

·		
<pre>'name 'copyright 'purpose display support 'micro 'suited for demo 'commercial addon needed '</pre>	: no	
<pre>\$regfile = "m48def.dat" the used micro \$crystal = 7372800 crystal frequency \$baud = 115200 rate \$hwstack = 32 use 32 for the hardware s \$swstack = 10 use 10 for the SW stack \$framesize = 40 use 40 for the frame spac 'I used a Staver to test</pre>		' specify ' used ' use baud ' default ' default ' default

'some routines to control the display are in the glcdSED.lib file 'IMPORTANT : since the SED1520 uses 2 chips, the columns are split into 2 of 60. 'This means that data after column 60 will not print correct. You need to locate the data on the second halve 'For example when you want to display a line of text that is more then 8 chars long, (8x8=64), byte 8 will not draw correctly 'Frankly i find the KS0108 displays a much better choice.

\$lib "glcdSED1520.lbx"

'First we define that we use a graphic LCD

Config Graphlcd = 120 * 64sed , Dataport = Porta , Controlport = Portd , Ce = 5 , Ce2 = 7 , Cd = 3 , Rd = 4

'The dataport is the portname that is connected to the data lines of the LCD 'The controlport is the portname which pins are used to control the lcd

```
'CE =CS Chip Enable/ Chip select
'CE2= Chip select / chip enable of chip 2
'CD=A0 Data direction
'RD=Read
'Dim variables (y not used)
Dim X As Byte , Y As Byte
'clear the screen
Cls
Wait 2
'specify the font we want to use
Setfont Font8x8
'You can use locate but the columns have a range from 1-132
'When you want to show somthing on the LCD, use the LDAT command
'LCDAT Y , COL, value
Lcdat 1 , 1 , "1231231"
Lcdat 3 , 80 , "11"
'lcdat accepts an additional param for inversing the text
'lcdat 1,1,"123" , 1 ' will inverse the text
Wait 2
Line(0, 0) - (30, 30), 1
Wait 2
Showpic 0 , 0 , Plaatje
                                                               'show a
comnpressed picture
End
                                                               'end program
'we need to include the font files
$include "font8x8.font"
'$include "font16x16.font"
```

Plaatje:
'include the picture data
\$bgf "smile.bgf"

6.285 LCDAUTODIM

Action

Dims the 20x4vfd LCD.

Syntax

LCDAUTODIM x

Remarks

A variable or constant in the range from 0-54 0 will turn auto dim off. A value between 1-54 dims the brightness after the given
number of seconds. The value is stored permanent.

This statement works only with the 20x4vfd display from "Electronic Design Bitzer"

Available in the MCS Shop.

See also

NONE

Example

NONE

6.286 LCDCONTRAST

Action

Set the contrast of a TEXT LCD.

Syntax

LCDCONTRAST X

Remarks

A variable or constant in the range from 0-3.

Some LCD text displays support changing the contrast. Noritake displays have this option for example.

See also

NONE

X

Example

NONE

6.287 LEFT

Action

Return the specified number of leftmost characters in a string.

Syntax

var = **LEFT**(var1, n)

Remarks

Var	The string that is assigned.	
Var1	The source string.	
n	The number of characters to get from the source string.	

See also

RIGHT 844 , MID 787

Partial Example

```
Dim S As String * 15 , Z As String * 15
S = "ABCDEFG"
Z = Left(s , 5)
Print Z
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

'ABCDE

6.288 LEN

Action

Returns the length of a string.

Syntax

var = **LEN**(string)

Remarks

var	A numeric variable that is assigned with the length of string.	
string	The string to calculate the length of.	

Strings can be maximum 254 bytes long.

See Also

<u>VAL</u> 945

Partial Example

Dim S As String * 15 , Z As String * 15 S = "ABCDEFG" Print Len(s)

6.289 LINE

Action

Draws a line on a graphic display.

Syntax

LINE(x0,y0) - (x1,y1), color

Remarks

X0	Starting horizontal location of the line.	
Y0	Starting vertical location of the line.	
X1	Horizontal end location of the line	
Y1	Vertical end location of the line.	
color	The color to use. Use 0 or a non zero value.	

See Also

LINE 759 , CONFIG GRAPHLCD 500 , BOX 417 , BOXFILL 419

Example

_____ _____ : t6963_240_128.bas 'name 'copyright : (c) 1995-2005, MCS Electronics : T6963C graphic display support demo 240 * 'purpose 128 'micro : Mega8535 'suited for demo : yes 'commercial addon needed : no 1.1.1 _____ _____ \$regfile = "m8535.dat" ' specify the used micro **\$crystal** = 8000000 ' used crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32 ' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space *_____ (c) 2001-2003 MCS Electronics T6963C graphic display support demo 240 * 128 ·_____ 'The connections of the LCD used in this demo 'LCD pin connected to ' 1 GND ' 2 GND ' 3 +5V GND GND +5V -9V /WR ' 4 -9V potmeter ' 5 PORTC.0 /RD /CE '6 PORTC.1 '7 PORTC.2 C/D PORTC.3 ' 8
 '9
 NC

 '10
 RESET

 '11-18
 D0-D7

 '19
 FS

 '20
 NC
 not conneted PORTC.4 PA PORTC.5 not connected 'First we define that we use a graphic LCD ' Only 240*64 supported yet Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc , Ce = 2 , Cd = 3 , Wr = 0 , Rd = 1 , Reset = 4 , Fs = 5 , Mode = 8 'The dataport is the portname that is connected to the data lines of the LCD 'The controlport is the portname which pins are used to control the lcd 'CE, CD etc. are the pin number of the CONTROLPORT. ' For example CE =2 because it is connected to PORTC.2 'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns

```
Dim X As Byte , Y As Byte
'Clear the screen will both clear text and graph display
Cls.
'Other options are :
' CLS TEXT
            to clear only the text display
' CLS GRAPH to clear only the graphical part
Cursor Off
Wait 1
'locate works like the normal LCD locate statement
' LOCATE LINE, COLUMN LINE can be 1-8 and column 0-30
Locate 1 , 1
'Show some text
Lcd "MCS Electronics"
'And some othe text on line 2
Locate 2 , 1 : Lcd "T6963c support"
Locate 3 , 1 : Lcd "1234567890123456789012345678901234567890"
Locate 16 , 1 : Lcd "write this to the lower line"
Wait 2
Cls Text
'use the new LINE statement to create a box
'LINE(X0,Y0) - (X1,Y1), on/off
Line(0 , 0) -(239 , 127) , 255
                                                             ' diagonal
line
Line(0, 127) - (239, 0), 255
                                                             ' diagonal
line
Line(0 , 0) -(240 , 0) , 255
                                                             ' horizontal
upper line
Line(0, 127) - (239, 127), 255
                                                             'horizontal
lower line
Line(0, 0) -(0, 127), 255
                                                             ' vertical
left line
Line(239, 0) -(239, 127), 255
                                                             ' vertical
right line
Wait 2
' draw a line using PSET X,Y, ON/OFF
' PSET on.off param is 0 to clear a pixel and any other value to turn it
on
For X = 0 To 140
                                                             ' set the
  Pset X , 20 , 255
pixel
Next
For X = 0 To 140
  Pset X , 127 , 255
                                                             ' set the
pixel
Next
Wait 2
'circle time
```

'Dim variables (y not used)

BASCOM-AVR

762

```
'circle(X,Y), radius, color
'X,y is the middle of the circle,color must be 255 to show a pixel and 0
to clear a pixel
For X = 1 To 10
  Circle(20 , 20) , X , 255
                                                              ' show
circle
  Wait 1
  Circle(20 , 20) , X , 0
                                                              'remove
circle
  Wait 1
Next
Wait 2
For X = 1 To 10
  Circle(20 , 20) , X , 255
                                                              ' show
circle
  Waitms 200
Next
Wait 2
'Now it is time to show a picture
'SHOWPIC X,Y,label
'The label points to a label that holds the image data
Test:
Showpic 0 , 0 , Plaatje
Showpic 0 , 64 , Plaatje
                                                              ' show 2
since we have a big display
Wait 2
Cls Text
                                                              ' clear the
text
End
'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bqf"
```

```
'You could insert other picture data here
```

6.290 LINEINPUT

Action

Read a Line from an opened File.

Syntax

LINEINPUT #bFileNumber, sLineText

Remarks

BfileNumber	(Byte) File number, which identifies an opened file
SlineText	(String) A string, which is assigned with the next line from the file.

Only valid for files opened in mode INPUT. Line INPUT works only with strings. It is great for working on text files.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LOC [766], LOF [767],

EOF [68th], <u>FREEFILE</u> [69th], <u>FILEATTR</u> [68th], <u>SEEK</u> [85th], <u>BSAVE</u> [42th], <u>BLOAD</u> [41th], <u>KILL</u> [75th], <u>DISKFREE</u> [66th], <u>DISKSIZE</u> [66th], <u>GET</u> [69th], <u>PUT</u> [81th], <u>FILEDATE</u> [68th], <u>FILETIME</u> [68th], <u>FILEDATETIME</u> [68th], <u>DIR</u> [68th], <u>FILELEN</u> [68th], <u>WRITE</u> [95th], <u>INPUT</u> [74th]

ASM

Calls	_FileLineInput	
Input	r24: filenumber	X: Pointer to String to be written from file
	r25: Stringlength	
Output	r25: Errorcode	C-Flag: Set on Error

Example

'Ok we want to check if the file contains the written lines Ff = Freefile()' get file handle Open "test.txt" For Input As #ff ' we can use a constant for the file too Print Lof(#ff); " length of file" Print Fileattr(#ff); " file mode"' should be 1 for input Do LineInput #ff , S ' read a line ' line input is used to read a line of text from a file Print S ' print on terminal emulator Loop Until Eof(ff)<> 0 'The EOF() function returns a non-zero number when the end of the file is reached 'This way we know that there is no more data we can read Close #ff

6.291 LTRIM

Action

Returns a copy of a string with leading blanks removed

Syntax

var = LTRIM(org)

Remarks

Var	String that receives the result.
Org	The string to remove the leading spaces from

See also

RTRIM 848 , TRIM 935

ASM

NONE

Partial Example

```
Dim S As String * 6
S =" AB "
Print Ltrim(s)
```

764 BASCOM-AVR

```
Print Rtrim(s)
Print Trim(s)
End
```

6.292 LOAD

Action

Load specified TIMER with a reload value.

Syntax

LOAD TIMER , value

Remarks

TIMER	TIMER0 , TIMER1 or TIMER2(or valid timer name)	
Value	The variable or value to load.	

The TIMER0 does not have a reload mode. But when you want the timer to generate an interrupt after 10 ticks for example, you can use the LOAD statement.

It will do the calculation : (256-value)

So LOAD TIMER0, 10 will load the TIMER0 with a value of 246 so that it will overflow after 10 ticks.

TIMER1 is a 16 bit counter so it will be loaded with the value of 65536-value.

See Also

NONE

Example

NONE

6.293 LOADADR

Action

Loads the address of a variable into a register pair.

Syntax

LOADADR var , reg

Remarks

var A variable which address must be loaded into the register pair X, Y or Z.reg The register X, Y or Z.

The LOADADR statement serves as an assembly helper routine.

Example

Dim S As String * 12

Dim A As Byte

\$ASM
loadadr S , X ; load address into R26 and R27
ld _temp1, X ; load value of location R26/R27 into R24(_temp1)
\$END ASM

6.294 LOADLABEL

Action

Assigns a word variable with the address of a label.

Syntax

Var = LOADLABEL(label)

Remarks

var	The variable that is assigned with the address of the label.	
lbl	The name of the label	

In some cases you might need to know the address of a point in your program. To perform a Cpeek() for example.

You can place a label at that point and use LoadLabel to assign the address of the label to a variable.

Loadlabel will only work for processors with <= 64KB memory.

If you use Loadlabel on an EEPROM label (a label used in the \$EEPROM data area) , these labels must precede the Loadlabel function.

This would be ok :

\$eeprom 'eeprom image label1: data 1,2,3,4,5 label2: data 6,7,8,9,10 \$data ' back to normal mode

dim w as word w=loadlabel(label2)

This code will work since the loadlabel is used after the EEPROM data labels.

6.295 LOADWORDADR

Action

Loads the Z-register and sets RAMPZ if available.

Syntax

© 2011 MCS Electronics

LOADWORDADR label

Remarks

label	The name of the label which address will be loaded into R30-R31 which	
	form the Z-register.	

The code that will be generated : LDI R30,Low(label * 2) LDI R31,High(label * 2) LDI R24,1 or CLR R24 STS RAMPZ, R24

As the AVR uses a word address, to find a byte address we multiply the address with 2. RAMPZ forms together with pointer **Z** an address register. As the LS bit of Z is used to identify the lower or the upper BYTE of the address, it is extended with the RAMPZ to address more then 15 bits. For example the Mega128 has 128KB of space and needs the RAMPZ register set to the right value in order to address the upper or lower 64KB of space.

See also

LOADLABEL 7657, LOADADR 764

Example LOADWORDADR label

6.296 LOC

Action

Returns the position of last read or written Byte of the file

Syntax

ILastReadWritten = LOC (#bFileNumber)

Remarks

bFileNumber	(Byte) File number, which identifies an opened file
	(Long) Variable, assigned with the Position of last read or written Byte (1-based)

This function returns the position of the last read or written Byte. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError. If the file position pointer is changed with the command SEEK, this function can not be used till the next read/ write operation.

This function differs from VB. In VB the byte position is divided by 128.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOF [767], EOF [687], FREEFILE [693], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], DISKSIZE [667], GET [697], PUT [819], FILEDATE [685], FILETIME [687],

```
FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]
```

<u>ASM</u>

Calls	_FileLoc	
Input	r24: filenumber	X: Pointer to Long-variable, which gets th result
Outpu	r25: Errorcode	C-Flag: Set on Error
t		

Example

'open the file in BINARY mode Open "test.biN" For Binary As #2 Put #2 , B ' write a byte Put #2 , W ' write a word Put #2 , L ' write a long Ltemp = Loc(#2)+ 1 ' get the position of the next byte Print Ltemp ;" LOC"' store the location of the file pointer Print Lof(#2);" length of file" Print Fileattr(#2);" file mode"' should be 32 for binary Put #2 , Sn ' write a single Put #2 , Stxt ' write a string

Flush #2 ' flush to disk Close #2

6.297 LOF

Action

Returns the length of the File in Bytes

Syntax

IFileLength = LOF (#bFileNumber)

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
-	(Long) Variable, which assigned with the Length of the file (1- based)

This function returns the length of an opened file. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], EOF [681], FREEFILE [695], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [75], DISKFREE [660], DISKSIZE [661], GET [697], PUT [819], FILEDATE [685], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Calls	_FileLOF	
Input	r24: filenumber	X: Pointer to Long-variable, which gets th result
Output	r25: Errorcode	C-Flag: Set on Error

Example

'open the file in BINARY mode Open "test.biN" For Binary As #2 Put #2 , B ' write a byte Put #2 , W ' write a word Put #2 , L ' write a long Ltemp = Loc(#2)+ 1 ' get the position of the next byte Print Ltemp ;" LOC"' store the location of the file pointer Print Lof(#2);" length of file" Print Fileattr(#2);" file mode"' should be 32 for binary Put #2 , Sn ' write a single Put #2 , Stxt ' write a string

Flush #2 ' flush to disk Close #2

6.298 LOCAL

Action

Dimensions a variable LOCAL to the function or sub program.

Syntax

LOCAL var As Type

Remarks

Var	The name of the variable
Туре	The data type of the variable.

There can be only LOCAL variables of the type BYTE, INTEGER, WORD, DWORD, LONG, SINGLE, DOUBLE or STRING.

A LOCAL variable is a temporary variable that is stored on the frame.

When the SUB or FUNCTION is terminated, the memory will be released back to the frame.

BIT variables are not possible because they are GLOBAL to the system.

The AT , ERAM, SRAM, XRAM directives can not be used with a local DIM statement. Also local arrays are not possible.

Notice that a LOCAL variable is not initialized. It will contain a value that will depend on the value of the FRAME data. So you can not assume the variable is 0. If you like it to be 0, you need to assign it.

A normal DIM-med variable is also not initialized to 0. The reason all variables are 0 (and strings are ""), is that the RAM memory is cleared. With the $\frac{NORAMCLEAR}{35}$ option you can turn this behaviour off.

So to conclude, a LOCAL variable will behave the same as a normal variable with the \$NORAMCLEAR option enabled.

While it would be simple to initialize the LOCAL variables to 0, in most/all cases, you will assign a value to it anyway, so it would be a waste of code space.

See also

DIM 653

ASM

NONE

Example

_____ _____ 'name : declare.bas : (c) 1995-2005, MCS Electronics 'copyright 'purpose : demonstrate using declare 'micro : Mega48 'suited for demo : yes 'commercial add on needed : no ' Note that the usage of SUBS works different in BASCOM-8051 _____ \$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32 ' default use 32 for the hardware stack \$swstack = 10 ' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space ' First the SUB programs must be declared 'Try a SUB without parameters Declare Sub Test2 'SUB with variable that can not be changed(A) and 'a variable that can be changed(B1), by the sub program 'When BYVAL is specified, the value is passed to the subprogram 'When BYREF is specified or nothing is specified, the address is passed to 'the subprogram Declare Sub Test(byval A As Byte , B1 As Byte) Declare Sub Testarray(byval A As Byte , B1 As Byte) 'All variable types that can be passed 'Notice that BIT variables can not be passed. 'BIT variables are GLOBAL to the application Declare Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S As String) 'passing string arrays needs a different syntax because the length of the strings must be passed by the compiler 'the empty () indicated that an array will be passed

```
770 BASCOM-AVR
```

```
Declare Sub Teststr(b As Byte , Dl() As String)
Dim Bb As Byte , I As Integer , W As Word , L As Long , S As String * 10
      'dim used variables
Dim Ar(10) As Byte
Dim Sar(10) As String * 8
                                                              'strng array
For Bb = 1 To 10
                                                              'fill the
  Sar(bb) = Str(bb)
arrav
Next
Bb = 1
'now call the sub and notice that we always must pass the first address
with index 1
Call Teststr(bb , Sar(1))
Call Test2
                                                              'call sub
Test2
                                                              'or use
without CALL
'Note that when calling a sub without the statement CALL, the enclosing
parentheses must be left out
Bb = 1
Call Test(1 , Bb)
                                                              'call sub
with parameters
Print Bb
                                                              'print value
that is changed
'now test all the variable types
Call Testvar(bb , I , W , L , S )
Print Bb ; I ; W ; L ; S
'now pass an array
'note that it must be passed by reference
Testarray 2 , Ar(1)
Print "ar(1) = " ; Ar(1)
Print "ar(3) = " ; Ar(3)
$notypecheck
                                                              ' turn off
type checking
Testvar Bb , I , I , I , S
'you can turn off type checking when you want to pass a block of memory
$typecheck
                                                              'turn it
back on
End
'End your code with the subprograms
'Note that the same variables and names must be used as the declared
ones
Sub Test(byval A As Byte , B1 As Byte)
                                                              'start sub
    Print A ; " " ; B1
                                                              'print
passed variables
   B1 = 3
                                                              'change
value
    'You can change A, but since a copy is passed to the SUB,
    'the change will not reflect to the calling variable
End Sub
                                                              'sub without
Sub Test2
parameters
    Print "No parameters"
End Sub
```

```
771
```

```
Sub Testvar(b As Byte , I As Integer , W As Word , L As Long , S As
String)
   Local X As Byte
   X = 5
                                                              'assign
local
   в = Х
    I = -1
   W = 40000
   L = 20000
   S = "test"
End Sub
Sub Testarray(byval A As Byte , B1 As Byte)
                                                             'start sub
   Print A ; " " ; B1
                                                             'print
passed variables
                                                             'change
   B1 = 3
value of element with index 1
   B1(1) = 3
                                                             'specify the
index which does the same as the line above
                                                             'modify
   B1(3) = 3
other element of array
    'You can change A, but since a copy is passed to the SUB,
    'the change will not reflect to the calling variable
End Sub
'notice the empty() to indicate that a string array is passed
Sub Teststr(b As Byte , Dl() As String)
 Dl(b) = Dl(b) + "add"
End Sub
```

6.299 LOCATE

Action

Moves the LCD cursor to the specified position.

Syntax

LOCATE y , x

Remarks

Х	Constant or variable with the position. (1-64*)
Y	Constant or variable with the line $(1 - 4^*)$

* Depending on the used display

See also

CONFIG LCD [516], LCD [752], HOME [724], CLS [433]

Partial Example

LCD "Hello" Locate 1,10 LCD "*"

772 BASCOM-AVR

6.300 LOG

Action

Returns the natural logarithm of a single variable.

Syntax

Target = **LOG**(source)

Remarks

Target	The single or double that is assigned with the LOG() of single target.
Source	The source single or doubler to get the LOG of.

See also

EXP 683 , LOG10 772

Example

Show sample 983

6.301 LOG10

Action

Returns the base 10 logarithm of a single variable.

Syntax

Target = **LOG10**(source)

Remarks

	The single or double that is assigned with the base 10 logarithm of single/ double target.
Source	The source single or double to get the base 10 LOG of.

See also

EXP 683 , LOG 772

Example

Show sample 983

6.302 LOOKDOWN

Action

Returns the index of a series of data.

Syntax

var = **LOOKDOWN**(value, label, entries)

Remarks

Var	The returned index value
Value	The value to search for
Label	The label where the data starts
entries	The number of entries that must be searched

When you want to look in BYTE series the VALUE variable must be dimensioned as a BYTE. When you want to look in INTEGER or WORD series the VALUE variable must be dimensioned as an INTEGER.

The LookDown function is the counterpart of the LookUp function. Lookdown will search the data for a value and will return the index when the value is found. It will return -1 when the data is not found.

See also

LOOKUPSTR 775, LOOKUP 774

'copyright 'purpose	: no		
<pre>\$regfile = "m48def.dat" the used micro</pre>		ı	specify
\$crystal = 4000000		1	used
crystal frequency \$baud = 19200 rate		,	use baud
\$hwstack = 32		,	default
<pre>use 32 for the hardware sta \$swstack = 10 use 10 for the SW stack</pre>	ack	ı	default
\$framesize = 40 use 40 for the frame space		I	default
Dim Idx As Integer , Search	h As Byte , Entries As Byte		
'we want to search for the Search = 3 'there are 5 entries in the Entries = 5			
<pre>'lookup and return the inde Idx = Lookdown(search , Lai Print Idx</pre>			

774 BASCOM-AVR

```
Search = 1
Idx = Lookdown(search , Label , Entries)
Print Idx
Search = 100
Idx = Lookdown(search , Label , Entries)
                                                             ' return -1
Print Idx
if not found
'looking for integer or word data requires that the search variable is
'of the type integer !
Dim Isearch As Integer
Isearch = 400
Idx = Lookdown(isearch , Label2 , Entries)
Print Idx
                                                             ' return 3
End
Label:
Data 1 , 2 , 3 , 4 , 5
Label2:
Data 1000% , 200% , 400% , 300%
```

6.303 LOOKUP

Action

Returns a value from a table.

Syntax

var = LOOKUP(value, label)

Remarks

Var	The returned value
Value	A value with the index of the table
Label	The label where the data starts

The value can be up to 65535. 0 will return the first entry.

See also

LOOKUPSTR 775, DATA 612, LOOKDOWN 772

<pre>\$regfile = "m48def.dat"</pre>	1	specify
the used micro		
\$crystal = 4000000		used
crystal frequency \$baud = 19200		use baud
rate		use baud
<pre>\$hwstack = 32</pre>	,	default
use 32 for the hardware stack \$swstack = 10	,	default

use 10 for the SW stack
\$framesize = 40
use 40 for the frame space
Dim Bl As Byte , I As Integer
Bl = Lookup(2 , Dta)
Print Bl
(zero based)
I = Lookup(0 , Dta2)
Print I
End
Dta:

Dta: Data 1 , 2 , 3 , 4 , 5 Dta2: Data 1000% , 2000%

6.304 LOOKUPSTR

Action

Returns a string from a table.

Syntax

var = LOOKUPSTR(value, label)

Remarks

Var	The string returned
	A value with the index of the table. The index is zero-based. That is, 0 will return the first element of the table.
Label	The label where the data starts

The index value can have a maximum value of 255.

See also

LOOKUP 774, LOOKDOWN 772, DATA 612

```
$regfile = "m48def.dat"
                                                              ' specify
the used micro
$crystal = 4000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Dim S As String * 4 , Idx As Byte
Idx = 0 : S = Lookupstr(idx , Sdata)
```

```
Print S
'This'
End
Sdata:
Data "This" , "is" , "a test"
```

6.305 LOW

Action

Retrieves the least significant byte of a variable.

Syntax

var = LOW(s)

Remarks

Var	The variable that is assigned with the LSB of var S.
S	The source variable to get the LSB from.

You can also assign a byte to retrieve the LSB of a Word or Long. For example :

 $\mathsf{B}=\mathsf{L}\,$, where B is a byte and L is a Long.

See also

HIGH 723 , HIGHW 724

Example

<pre>\$regfile = "m48def.dat"</pre>	' specify
the used micro	
\$crystal = 4000000	' used
crystal frequency	
\$baud = 19200	' use baud
rate	
<pre>\$hwstack = 32</pre>	' default
use 32 for the hardware stack	
<pre>\$swstack = 10</pre>	' default
use 10 for the SW stack	
\$framesize = 40	' default
use 40 for the frame space	
Dim I As Integer , Z As Byte	
I = &H1001	
Z = Low(i)	' is 1
End	

6.306 LOWERLINE

Action

Reset the LCD cursor to the lower line.

'will print

Syntax LOWERLINE

Remarks

NONE

See also

UPPERLINE 945 , THIRDLINE 930 , FOURTHLINE 693 , HOME 724

Partial Example

```
Lcd "Test"
Lowerline
Lcd "Hello"
End
```

6.307 MACRO

Action

This statement allow you to define a Macro.

Syntax

MACRO name macrodef END MACRO

Remarks

name	The name of the macro. Each macro need to have a unique name.
macrodef	The code you want to have inserted when you use the macro.

Macro's must be defined before they can be used. When a macro is defined but not used in your code, it will not be compiled. You can use \$INCLUDE to include a large number of macro's.

When the compiler encounters the name of a defined macro, it will insert the defined code at that place. While it looks similar to a sub routine, there are differences. A sub routine for example is called and has a RETURN(RET).

See also

SUB 914 , GOSUB 719

Example

Macro Usb_reset_data_toggle Ueconx.rstdt = 1 End Macro

Macro Usb_disable_stall_handshake
Ueconx.stallrqc = 1

End Macro

```
Macro Set_power_down_mode
Smcr = 0
Smcr = Bits(se , Sm1)
sleep
End Macro
```

Usb_reset_data_toggle ' this will insert UECONRX.RSTD=1 Set_power_down_mode ' this will insert the following code : Smcr = 0 Smcr = **Bits**(se , Sm1) sleep

6.308 MAKEBCD

Action

Convert a variable into its BCD value.

Syntax

var1 = MAKEBCD(var2)

Remarks

var1	Variable that will be assigned with the converted value.
Var2	Variable that holds the decimal value.

When you want to use an I2C clock device, which stores its values as BCD values you can use this function to convert variables from decimal to BCD. For printing the BCD value of a variable, you can use the BCD() function which converts a BCD number into a BCD string.

See also

MAKEDEC 779, BCD 409, MAKEINT 778

Example

```
Dim A As Byte
A = 65
Lcd A
Lowerline
Lcd Bcd(a)
A = Makebcd(a)
Lcd " " ; A
End
```

6.309 MAKEINT

Action

Compact two bytes into a word or integer.

Syntax

varn = **MAKEINT**(LSB , MSB)

Remarks

Varn	Variable that will be assigned with the converted value.
LSB	Variable or constant with the LS Byte.
MSB	Variable or constant with the MS Byte.

The equivalent code is: varn = (256 * MSB) + LSB

See also

LOW [776], HIGH [723], MAKEBCD [778], MAKEDEC [779]

Example

```
Dim A As Integer , I As Integer
A = 2
I = Makeint(a , 1)
256) + 2 = 258
End
```

'I = (1 *

6.310 MAKEDEC

Action

Convert a BCD byte or Integer/Word variable to its DECIMAL value.

Syntax

var1 = MAKEDEC(var2)

Remarks

var1	Variable that will be assigned with the converted value.
var2	Variable that holds the BCD value.

When you want to use an I2C clock device, which stores its values as BCD values you can use this function to convert variables from BCD to decimal.

See also

MAKEBCD 7781, MAKEBCD 7781, MAKEINT 7781

```
Dim A As Byte
A = 65
Print A
Print Bcd(a)
A = Makedec(a)
Print Spc(3) ; A
End
```

6.311 MAKEMODBUS

Action

780

Creates a MODBUS master/client frame.

Syntax

PRINT [#x,] **MAKEMODBUS**(slave, function, address, varbts)

Remarks slave The slave to address. This is a variable or constant with a valid MODBUS slave to address. function The function number. This must be a constant. At the moment the following functions are supported : 03 : read register(s) 06 : write single register 16 : write multiple registers address The starting address of the register For a function that sends data like function 6 and 16, this must be a varbts variable. For function 06 which can only write a single register, this can be a byte or integer or word. For function 16 it may be a long, single or double. For function 6 and 16 the address of the variable is passed to the function. For function 3 you may also specify the number of bytes to receive. Or you can use a variable. When you specify a byte, a word will be used anyway since a word (2 bytes) is the minimum in MODBUS protocol. But when sending data, you can send content of a byte. For the MSB the value 0 will be sent in that case.

The MAKEMODBUS function need to be used in combination with the PRINT statement. It can only be used with the hardware UART(1-4).

The MODBUS protocol is an industry standard. The protocol can be used with RS-232, RS-485 or TCP/IP or CAN.

The current BASCOM implementation only works with RS-232 or RS485.

In MODBUS we use client/master and server/slave. You may see it as a web server and a web browser. The web server is the client/slave that reacts on the master/web browser.

A slave will only respond when it is addressed. All other slaves just keep listening till they are addressed.

An addressed slave will process the data and send a response.

In MODBUS the data is sent with MSB first and LSB last. The special CRC16 checksum is sent LSB first and MSB last.

When multiple registers are sent with function 16, the data is split up into words, and for each word, the MSB-LSB order is used.

For example a LONG is 4 bytes. LSB, NSB1, NSB2, MSB. It would be sent as : NSB1, LSB, MSB, NSB2.

In order to use the MODBUS functionality, you need to include the MODBUS.LBX with the \$LIB directive.

Notice that BASCOM only supports the MODBUS master. A MODBUS server that supports the above functions will be available from MCS.

See also

PRINT 810

```
_____
'name
                          : rs485-modbus-master.bas
                          : (c) 1995-2008, MCS Electronics
'copyright
'purpose
                          : demo file for MAKEMODBUS
'micro
                          : Mega162
'suited for demo
                          : yes
'commercial addon needed : no
    _____
$regfile = "m162def.dat"
                                                             ' specify the used mich
$crystal = 8000000
$baud = 19200
                                                              ' use baud rate
                                                              ' default use 42 for the
$hwstack = 42
                                                              ' default use 40 for th
\$swstack = 40
                                                              ' default use 40 for th
$framesize = 40
$lib "modbus.lbx"
                                                              ' specify the additiona
Config Print1 = Portb.1 , Mode = Set
                                                              ' specify RS-485 and d:
                                                              'make an alias
Rs485dir Alias Portb.1
Config Rs485dir = Output
                                                              'set direction register
Rs485dir = 0
                                                              ' set the pin to 0 for
Portc.0 = 1
                                                              ' a pin is used with a
'The circuit from the help is used. See Using MAX485
         TX RX
' COM0 PD.1 PD.0 rs232 used for debugging
' COM1 PB.3 PB.2 rs485 used for MODBUS halve duplex
' PB.1 data direction rs485
'configure the first UART for RS232
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8
'configure the second UAR for RS485/MODBUS. Make sure all slaves/servers use the sa
Config Com2 = 9600, Synchrone = 0, Parity = Even, Stopbits = 1, Databits = 8,
'use OPEN/CLOSE for using the second UART
Open "COM2:" For Binary As #1
'dimension some variables
Dim B As Byte
Dim W As Word
Dim L As Long
W = &H4567
                                                              'assign a value
                                                              'assign a value
L = &H12345678
Print "RS-485 MODBUS master"
Do
   If Pinc.0 = 0 Then
                                                              ' test switch
      Waitms 500
                                                              ' delay
      Print "send request to slave/server"
     ' Send one of the following three messages
```

```
BASCOM-AVR
```

```
' Print #1 , Makemodbus(2 , 3 , 8 , 2);
' Print #1 , Makemodbus(2 , 6 , 8 , W);
Print #1 , Makemodbus(2 , 16 , 8 , L);
End If
If Ischarwaiting(#1) <> 0 Then
B = Waitkey(#1)
Print Hex(b) ; ",";
End If
Loop
```

' slave 2, function 3, ' slave 2, function 6, ' slave 2, function 16, 'was something returned 'then get it

'print the info

End

782

6.312 MAKETCP

Action

Creates a TCP/IP formatted long variable.

Syntax

var = **MAKETCP**(b1,b2,b3,b4 [opt]) var = **MAKETCP**(num)

Remarks

var	The target variable of the type LONG that is assigned with the IP number
b1-b4	Four variables of numeric constants that form the IP number. b1 is the MSB of the IP/long b4 is the LSB of the IP/long example var = MakeTCP(192,168,0, varx). We can also use reverse order with the optional parameter : example var = MakeTCP(var3,0,168, 192, 1). A value of 1 will use reverse order while a value of 0 will result in normal order.
	When you use a constant, provide only one parameter : example var = MakeTCP(192.168.0.2). Notice the dots !

MakeTCP is a helper routine for the TCP/IP library.

See also

CONFIG TCPIP 559 , IP2STR 749

Example

NONE

6.313 MAX

Action

Returns the maximum value of a byte or word array.

Syntax

var1 = **MAX**(var2)

MAX(ar(1), m , idx)

Remarks

var1	Variable that will be assigned with the maximum value.	
var2	The first address of the array.	
	The MAX statement can return the index too	
Ar(1)	Starting element to get the maximum value and index of.	
М	Returns the maximum value of the array.	
Idx	Return the index of the array that contains the maximum value. Returns 0 if there is no maximum value.	

The MIN() and MAX() functions work on BYTE and WORD arrays only.

See also

<u>MIN</u> 785

```
_____
                 _____
                               _____
_____
'name
                      : minmax.bas
                      : (c) 1995-2005, MCS Electronics
'copyright
                      : show the MIN and MAX functions
'purpose
'micro
                      : Mega48
'suited for demo
                      : yes
'commercial addon needed : no
· _____
                                 _____
_____
$regfile = "m48def.dat"
                                                     ' specify
the used micro
$crystal = 4000000
                                                     ' used
crystal frequency
$baud = 19200
                                                     ' use baud
rate
\$hwstack = 32
                                                     ' default
use 32 for the hardware stack
\$swstack = 10
                                                     ' default
use 10 for the SW stack
$framesize = 40
                                                     ' default
use 40 for the frame space
' These functions only works on BYTE and WORD arrays at the moment !!!!!
'Dim some variables
Dim Wb As Byte , B As Byte
Dim W(10) As Word
                                                     ' or use a
BYTE array
'fill the word array with values from 1 to 10
For B = 1 To 10
 W(b) = B
Next
Print "Max number " ; Max(w(1))
```

784 BASCOM-AVR

```
Print "Min number " ; Min(w(1))
Dim Idx As Word , M1 As Word
Min(w(1) , M1 , Idx)
Print "Min number " ; M1 ; " index " ; Idx
Max(w(1) , M1 , Idx)
Print "Max number " ; M1 ; " index " ; Idx
End
```

6.314 MEMCOPY

Action

Copies a block of memory

Syntax

bts = MEMCOPY(source, target , bytes[, option])

Remarks

bts	The total number of bytes copied. This must be a word or integer
source	The first address of the source variable that will be copied.
target	The first address of the target variable that will be copied to.
bytes	The number of bytes to copy from "source" to "target"
option	An optional numeric constant with one of the following values : 1 - only the source address will be increased after each copied byte 2 - only the target address will be increased after each copied byte 3 - both the source and target address will be copied after each copied byte

By default, option 3 is used as this will copy a block of memory from one memory location to another location. But it also possible to fill an entire array of memory block with the value of 1 memory location. For example to clear a whole block or preset it with a value.

And with option 2, you can for example get a number of samples from a register like PINB and store it into an array.

See also

NONE

ASM

NONE

'name	: MEMCOPY.BAS
'copyright	: (c) 1995-2006, MCS Electronics
'purpose	: show memory copy function
'suited for demo	: yes
'commercial addon needed	: no
'use in simulator	: possible
· ·	
<pre>\$regfile = "m88def.dat"</pre>	' specify

```
100
```

```
the used micro
$crystal = 8000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
\$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 16
                                                              ' default
use 10 for the SW stack
$framesize = 40
Dim Ars(10) As Byte
                                                              'source
bytes
Dim Art(10) As Byte
                                                              'target
bytes
                                                              'index
Dim J As Byte
For J = 1 To 10
                                                              'fill array
 Ars(j) = J
Next
J = Memcopy(ars(1), Art(1), 4)
                                                              'copy 4
bytes
Print J ; " bytes copied"
For J = 1 To 10
 Print Art(j)
Next
J = Memcopy(ars(1), Art(1), 10, 2)
                                                              'assign them
all with element 1
Print J ; " bytes copied"
For J = 1 To 10
 Print Art(j)
Next
Dim W As Word , L As Long
W = 65511
J = Memcopy(w, L, 2)
                                                              'copy 2
bytes from word to long
```

6.315 MIN

End

Action

Returns the minimum value of a byte or word array.

Syntax

var1 = MIN(var2)
MIN(ar(1), m , idx)

Remarks

var1	Variable that will be assigned with the minimum value.
var2	The first address of the array.
	The MIN statement can return the index too

Ar(1)	Starting element to get the minimum value and index of
М	Returns the minimum value of the array
	Return the index of the array that contains the minimum value. Returns 0 if there is no minimum value.

The MIN() ans MAX() functions work on BYTE and WORD arrays only.

See also

MAX 782

'copyright 'purpose 'micro 'suited for demo 'commercial addon needed		
<pre>\$regfile = "m48def.dat"</pre>		' specify
the used micro \$crystal = 4000000		' used
crystal frequency		' use baud
\$baud = 19200 rate		use baud
\$hwstack = 32		' default
<pre>use 32 for the hardware st \$swstack = 10</pre>	LACK	' default
use 10 for the SW stack \$framesize = 40		' default
use 40 for the frame space	2	deraure
' These functions only wor 'Dim some variables Dim Wb As Byte , B As Byte Dim W(10) As Word BYTE array	rks on BYTE and WORD arrays at the	moment !!!!! ' or use a
<pre>'fill the word array with For B = 1 To 10 W(b) = B Next</pre>	values from 1 to 10	
Print "Max number " ; Max Print "Min number " ; Min		
Dim Idx As Word , M1 As Wo Min(w(1) , M1 , Idx) Print "Min number " ; M1 ,		

6.316 MID

Action

The MID function returns part of a string (a sub string). The MID statement replaces part of a string variable with another string.

Syntax

var = MID(var1 ,st [, l])
MID(var ,st [, l]) = var1

Remarks

var	The string that is assigned.
Var1	The source string.
st	The starting position.
1	The number of characters to get/set.

See also

LEFT 758 , RIGHT 844

Example

```
Dim S As String * 15 , Z As String * 15
S = "ABCDEFG"
Z = Left(s , 5)
Print Z
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

'ABCDE

6.317 MOD

Action

Calculates the remainder of a division.

Syntax

var1 = var2 MOD var3

Remarks

var1	Variable that will be assigned with the modules of var2 and var3.	
var2	A numeric variable to take the modules from	
var3	The modulus	

The MOD operation is similar to the division operation(/). But while a division returns the number of times a number can be divided, the MOD returns the remainder.

For example : 21 MOD 3 will result in 0 since 7x2=21. There will be no remainder. But 22 MOD 3 will result in 1 since 22-(7x3)=1

In BASCOM, the variable you assign determine which kind of math will be used. When

788 BASCOM-AVR

you have 2 word variables you want to get the modulus from, you have to assign a word variable too.

When you assign a byte, byte math will be used.

See also

Language Fundamentals 274

Example

6.318 NBITS

Action

Set all except the specified bits to 1.

Syntax

Var = **NBITS**(b1 [,bn])

Remarks

Var	The BYTE/PORT variable that is assigned with the constant.
B1,bn	A list of bit numbers that NOT must be set to 1.

While it is simple to assign a value to a byte, and there is special Boolean notation **&B** for assigning bits, the Bits() and NBits() function makes it simple to assign a few bits.

B = &B01111101: how many zero's are there? This would make it more readable: B = NBits(1, 7)You can read from the code that bit 1 and bit 7 are NOT set to 1. It does not save code space as the effect is the same.

The NBITS() function will set all bits to 1 except for the specified bits. It can only be used on bytes and port registers. Valid bits are in range from 0 to 7.

See Also

BITS 415

'	
'name	: bits-nbits.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: demo for Bits() AND Nbits()
'micro	: Mega48
'suited for demo	: yes
'commercial addon needed	: no
'use in simulator	: possible
	-

' specify

' use baud

' default

' default

' default

' used

\$regfile = "m48def.dat"
the used micro
\$crystal = 4000000
crystal frequency
\$baud = 19200
rate
\$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
use 10 for the SW stack
\$framesize = 40
use 40 for the frame space

Dim B As Byte

```
'while you can use &B notation for setting bits, like B = &B1000_0111
'there is also an alternative by specifying the bits to set
B = Bits(0, 1, 2, 7) 'set only
bit 0,1,2 and 7
Print B
'and while bits() will set all bits specified to 1, there is also Nbits
```

6.319 ON INTERRUPT

Action

Execute subroutine when the specified interrupt occurs.

Syntax

ON interrupt label [NOSAVE]

Remarks

Interrupt	INTO, INT1, INT2, INT3, INT4,INT5, TIMER0,TIMER1, TIMER2, ADC, EEPROM, CAPTURE1, COMPARE1A, COMPARE1B,COMPARE1. Or you can use the AVR name convention:
	OC2, OVF2, ICP1, OC1A, OC1B, OVF1, OVF0, SPI, URXC, UDRE, UTXC, ADCC, ERDY and ACI.
Label	The label to jump to if the interrupt occurs.
NOSAVE	When you specify NOSAVE, no registers are saved and restored in the interrupt routine. So when you use this option make sure to save and restore all used registers.
	When you omit NOSAVE all used registers will be saved. These are SREG , R31 to R16 and R11 to R0 with exception of R6,R8 and R9 .
	R12 – R15 are not saved. When you use floating point math in the ISR (not recommended) you must save and restore R12-R15 yourself in the ISR.

My_Isr: Push R12 ' save registers Push R13 Push R14 Push R15
Single = single + 1 ' we use FP
Pop R15 'restore registers Pop R14 Pop R13 Pop R12 RETURN
When the AVR has extended IO-space (for example ATMega48, 88 or 168, see datasheet at the end: Registersummary), the compiler uses R23 for a number of operations. So Push and Pop R23 as well when using the NOSAVE-option when using these AVR's with extended IO-space.

You must return from the interrupt routine with the RETURN statement.

The first RETURN statement that is encountered that is outside a condition will generate a RETI instruction. You may have only one such RETURN statement in your interrupt routine because the compiler restores the registers and generates a RETI instruction when it encounters a RETURN statement in the ISR. All other RETURN statements are converted to a RET instruction.

The possible interrupt names can be looked up in the selected microprocessor register file. 2313def.dat for example shows that for the compare interrupt the name is COMPARE1. (look at the bottom of the file)

What are interrupts good for?

An interrupt will halt your program and will jump to a specific part of your program. You can make a DO .. LOOP and poll the status of a pin for example to execute some code when the input on a pin changes.

But with an interrupt you can perform other tasks and when then pin input changes a special part of your program will be executed. When you use INPUT "Name ", v for example to get a user name via the RS-232 interface it will wait until a RETURN is received. When you have an interrupt routine and the interrupt occurs it will branch to the interrupt code and will execute the interrupt code. When it is finished it will return to the Input statement, waiting until a RETURN is entered.

Maybe a better example is writing a clock program. You could update a variable in your program that updates a second counter. But a better way is to use a TIMER interrupt and update a seconds variable in the TIMER interrupt handler.

There are multiple interrupt sources and it depends on the used chip which are available.

To allow the use of interrupts you must set the global interrupt switch with a ENABLE INTERRUPTS statement. This only allows that interrupts can be used. You must also set the individual interrupt switches on!

ENABLE TIMER0 for example allows the TIMER0 interrupt to occur.

With the DISABLE statement you turn off the switches.

When the processor must handle an interrupt it will branch to an address at the start of flash memory. These addresses can be found in the DAT files.

The compiler normally generates a RETI instruction on these addresses so that in the event that an interrupt occurs, it will return immediately.

When you use the ON ... LABEL statement, the compiler will generate code that jumps to the specified label. The SREG and other registers are saved at the LABEL location and when the RETURN is found the compiler restores the registers and generates the RETI so that the program will continue where it was at the time the interrupt occurred.

When an interrupt is services no other interrupts can occur because the processor(not the compiler) will disable all interrupts by clearing the master interrupt enable bit. When the interrupt is services the interrupt is also cleared so that it can occur again when the conditions are met that sets the interrupt.

It is not possible to give interrupts a priority. The interrupt with the lowest address has the highest interrupt!

Finally some tips :

* when you use a timer interrupt that occurs each 10 uS for example, be sure that the interrupt code can execute in 10 uS. Otherwise you would loose time.

* it is best to set just a simple flag in the interrupt routine and to determine it's status in the main program. This allows you to use the NOSAVE option that saves stack space and program space. You only have to Save and Restore R24 and SREG in that case.

* Since you can not PUSH a hardware register, you need to load it first:

PUSH R24 ; since we are going to use R24 we better save it

IN r24, SREG ; get content of SREG into R24 PUSH R24 ; we can save a register

;here goes your asm code POP R24 ;get content of SREG

OUT SREG, R24 ; save into SREG POP R24 ; get r24 back

See Also

On VALUE 792

Partial Example

Enable Interrupts Enable Int0 interrupt On Int0 Label2 Nosave label2 on INT0

'enable the

'jump to

Do'endless loop nop	
Loop	
End	
Label2:	
Dim A As Byte If A > 1 Then	
Return	'qenerates a
RET because it is inside a condition	generaces a
End If	
Return	'generates a
RETI because it is the first RETURN	
Return	'generates a
RET because it is the second RETURN	

6.320 ON VALUE

Action

Branch to one of several specified labels, depending on the value of a variable.

Syntax

ON var [GOTO] [GOSUB] label1 [, label2] [,CHECK]

Remarks

	The numeric variable to test. This can also be a SFR such as PORTB.
label1, label2	The labels to jump to depending on the value of var.
CHECK	An optional check for the number of provided labels.

Note that the value is zero based. So when var is 0, the first specified label is jumped/branched.

It is important that each possible value has an associated label. When there are not enough labels, the stack will get corrupted. For example : ON value label1, label2

And value = 2, there is no associated label.

You can use the optional CHECK so the compiler will check the value against the number of provided labels. When there are not enough labels for the value, there will be no GOTO or GOSUB and the next line will be executed.

See Also

ON INTERRUPT 789

ASM

The following code will be generated for a non-MEGA micro with ON value GOTO. Ldi R26,\$60 ; load address of variable Ldi R27,\$00 ; load constant in register Ld R24,X Clr R25

Ldi R30, Low(ON_1_ * 1) ; load Z with address of the label

Ldi R31, High(ON_1_ * 1)

Add zl,r24 ; add value to Z Adc zh,r25

Ijmp ; jump to address stored in Z

ON_1_:

Rjmp lbl1 ; jump table Rjmp lbl2 Rjmp lbl3

The following code will be generated for a non-MEGA micro with ON value GOSUB.

;##### On X Gosub L1 , L2 Ldi R30,Low(ON_1_EXIT * 1) Ldi R31,High(ON_1_EXIT * 1) Push R30 ;push return address Push R31 Ldi R30,Low(ON_1_ * 1) ;load table address Ldi R31,High(ON_1_ * 1) Ldi R26,\$60 Ld R24,X Clr R25

Add zl,r24 ; add to address of jump table Adc zh,r25 Ijmp ; jump !!!

ON_1_: Rjmp L1 Rjmp L2 ON_1_EXIT:

As you can see a jump is used to call the routine. Therefore the return address is first saved on the stack.

Example

```
_____
_____
'name
                  : ongosub.bas
                  : (c) 1995-2005, MCS Electronics
'copyright
                  : demo : ON .. GOSUB/GOTO
'purpose
'micro
                  : Mega48
'suited for demo
                   : yes
'commercial addon needed : no
_____
$regfile = "m48def.dat"
                                             ' specify
the used micro
$crystal = 4000000
                                             ' used
crystal frequency
$baud = 19200
                                             ' use baud
rate
$hwstack = 32
                                             ' default
```

794

```
use 32 for the hardware stack
                                                              ' default
\$swstack = 10
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Dim A As Byte
Input "Enter value 0-2 " , A
                                                              'ask for
input
Rem Note That The Starting Value Begins With 0
On A Gosub L0 , L1 , L2
Print "Returned"
If Portb < 2 Then
                                                              'you can
also use the portvalue
 On Portb Goto G0 , G1
End If
End_prog:
End
L0:
 Print "0 entered"
Return
L1:
 Print "1 entered"
Return
L2:
 Print "2 entered"
Return
G0:
 Print "P1 = 0"
 Goto End_prog
G1:
  Print "P1 = 1"
  Goto End_prog
```

6.321 OPEN

Action

Opens a device.

Syntax

OPEN "device" for MODE As #channel **OPEN** file FOR MODE as #channel

Remarks

Device	The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device.
	With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you

© 2011 MCS Electronics

0	
	want to use. COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stop bits.
	The format for COM1 and COM2 is : COM1: or COM2:
	There is no speed/baud rate parameter since the default baud rate will be used that is specified with \$BAUD or \$BAUD1
	The format for the software UART is: COMpin:speed,8,N,stopbits[, INVERTED] Where pin is the name of the PORT-pin. Speed must be specified and stop bits can be 1 or 2. 7 bit data or 8 bit data may be used. For parity N, O or E can be used.
	An optional parameter ,INVERTED can be specified to use inverted RS- 232.
	Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232.
	For the AVR-DOS file system, Device can also be a string or filename constant like "readme.txt" or sFileName
	For the Xmega, you can also open SPIC, SPID, SPIE and SPIF for SPI communication. Or for TWI you can use TWIC, TWID, TWIE or TWIF.
MODE	You can use BINARY or RANDOM for COM1 and COM2, but for the software UART pins, you must specify INPUT or OUTPUT.
	For the AVR-DOS file system, MODE may be INPUT, OUTPUT, APPEND or BINARY.
Channel	The number of the channel to open. Must be a positive constant >0 .
	For the AVR-DOS file system, the channel may be a positive constant or a numeric variable. Note that the AVD-DOS file system uses real file handles. The software UART does not use real file handles.
	For the Xmega UART, you may use a variable that starts with BUART. This need to be a numeric variable like a byte. Using a variable allows you to use the UART dynamic.

UART

The statements that support the device are $\underline{PRINT}_{[810]}$, $\underline{INPUT}_{[744]}$, $\underline{INPUTHEX}_{[742]}$, $\underline{INKEY}_{[738]}$ and $\underline{WAITKEY}_{[949]}$

Every opened device must be closed using the CLOSE #channel statement. Of course, you must use the same channel number.

In DOS the #number is a DOS file number that is passed to low level routines. In BASCOM the channel number is only used to identify the channel but there are no file handles. So opening a channel, will not use a channel. Closing a channel is not needed for UARTS. When you do so, it is ignored. If you OPEN the channel again, you

will get an error message.

So use OPEN in the begin of your program, and if you use CLOSE, use it at the end of your program.

What is the difference?

In VB you can close the channel in a subroutine like this:

OPEN "com1:" for binary as #1 Call test Close #1 End

Sub test Print #1, "test" End Sub

This will work since the file number is a real variable in the OS. In BASCOM it will not work : the CLOSE must come after the last I/O statement:

OPEN "com1:" for binary as #1 Call test End

Sub test Print #1, "test" End Sub Close #1

The INPUT statement in combination with the software UART, will not echo characters back because there is no default associated pin for this.

AVR-DOS

The AVR-DOS file system uses real file handles. This means that the CLOSE statement can be used at any place in your program just as with VB.

There are a few file mode, all inherited from VB/QB. They work exactly the same.

File mode	Description
OUTPUT	Use OUTPUT to create a file, and to write ASCII data to the file. A readme. txt file on your PC is an example of an ASCII file. ASCII files have a trailing CR+LF for each line you print. The PRINT statement is used in combination with OUTPUT mode.
INPUT	This mode is intended to OPEN an ASCII file and to read data only. You can not write data in this mode. The file need to exist, and must contain ASCII data. LINEINPUT can be used to read data from the file.
APPEND	APPEND mode is used on ASCII files and will not erase the file, but will append data to the end of the file. This is useful when you want to log data to a file. Opening in OUTPUT mode would erase the file if it existed.
BINARY	In BINARY mode you have full read and write access to all data in the file. You can open a text file to get binary access, or you can open a binary file

such as an image file. GET and PUT can be used with binary files.

The following information from the author is for advanced users only.

GET/PUT is not supposed to work with INPUT/OUTPUT due to the rules in VB/QBASIC. In the file CONFIG_AVR-DOS.bas (nearly at the of the file) you will find the constants ' permission Masks for file access routine regarding to the file open mode Const cFileWrite_Mode = &B00101010 ' Binary, Append, Output Const cFileRead_Mode = &B00100001 ' Binary, Input Const cFileSeekSet_Mode = &B00100000 ' Binary Const cFileInputLine = &B00100001 ' Binary, Input Const cFilePut_Mode = &B00100000 ' Binary Const cFileGet_Mode = &B00100000 , Binary

Where you can control, which routines can used in each file open mode. There you can see, that in standard usage GET and PUT is only allowed in BINARY. Some time ago I wrote the Bootloader with AVR-DOS and I had the problem to keep Flash usage as low as possible. In the Bootloader I had to work with GET to read in the bytes, because the content is no ASCII text. On the other side, if you open a file in INPUT mode, you need less code. So I tested to open the File in input mode and allow to use GET in Input Mode.

I changed:

Const cFileGet_Mode = &B00100001 So GET can work in INPUT too in the BOOTLOADER.

If you switch in the constants cFileGet_Mode the last 0 to a 1, you can use GET in INPUT Open mode to. With the bootloader.bas I changed the Config_AVR-DOS.bas too. With this changed Config_AVR-DOS.bas GET can used in INPUT, with the standard CONFIG_AVR-DOS not.

This change makes no problem in code, but I think this is only something for experienced AVR-DOS user.

Whether he can use GET in INPUT mode depends only on this last bit in the constant cFileGET_Mode in the file Config_AVR-DOS.bas. This bit controls, what can be used in INPUT mode.

Xmega-SPI

The Xmega has 4 SPI interfaces. The channel is used to communicate with the different devices.

And just as with the Xmega UART, you can use the SPI dynamic. When the channel variable starts with BSPI, you can pass a variable channel.

An example you will find at <u>CONFIG SPIx</u> 550

You can OPEN a SPI device only in BINARY mode.

Xmega-TWI

The Xmega has 4 TWI interfaces. The channel is used to communicate with the different devices.

You can OPEN a TWI device only in BINARY mode. Only constants are allowed for the channel.

See also

CLOSE [437], CRYSTAL [608], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [681]

Example

```
' 'name : open.bas
'copyright : (c) 1995-2005, MCS Electronics
'purpose : demonstrates software UART
'micro : Mega48
'suited for demo : yes
'commercial addon needed : no
```

\$regfile = "m48def.dat" ' specify the used micro **\$crystal** = 10000000 ' used crystal frequency **\$baud =** 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space

Dim B As Byte

'Optional you can fine tune the calculated bit delay 'Why would you want to do that? 'Because chips that have an internal oscillator may not 'run at the speed specified. This depends on the voltage, temp etc. 'You can either change \$CRYSTAL or you can use 'BAUD #1,9610 'In this example file we use the DT006 from www.simmstick.com 'This allows easy testing with the existing serial port 'The MAX232 is fitted for this example. 'Because we use the hardware UART pins we MAY NOT use the hardware UART 'The hardware UART is used when you use PRINT, INPUT or other related statements 'We will use the software UART. Waitms 100 'open channel for output **Open** "comd.1:19200,8,n,1" **For Output As #1** Print #1 , "serial output" 'Now open a pin for input Open "comd.0:19200,8,n,1" For Input As #2 'since there is no relation between the input and output pin 'there is NO ECHO while keys are typed Print #1 , "Number" 'get a number **Input** #2 , B 'print the number **Print** #1 , B

```
'now loop until ESC is pressed
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
   'store in byte
   B = Inkey(#2)
   'when the value > 0 we got something
   If B > 0 Then
      Print #1 , Chr(b)
                                                             'print the
character
  End If
Loop Until B = 27
Close #2
Close #1
'OPTIONAL you may use the HARDWARE UART
'The software UART will not work on the hardware UART pins
'so you must choose other pins
'use normal hardware UART for printing
'Print B
'When you dont want to use a level inverter such as the MAX-232
'You can specify , INVERTED :
'Open "comd.0:300,8,n,1, inverted" For Input As #2
'Now the logic is inverted and there is no need for a level converter
'But the distance of the wires must be shorter with this
End
```

Example XMEGA TWI

```
(c) 1995-2010, MCS
                      xm128-TWI.bas
  This sample demonstrates the Xmega128A1 TWI
        ------
$regfile = "xm128a1def.dat"
$crystal = 3200000
$hwstack = 64
\$swstack = 40
$framesize = 40
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
Dim S As String * 20
'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Dim N As String * 16 , B As Byte
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
```

```
800 BASCOM-AVR
```

```
1, Databits = 8
Config Input1 = Cr , Echo = Crlf
                                                               ' CR is used
for input, we echo back CR and LF
Open "COM1:" For Binary As #1
        ^^^^ change from COM1-COM8
Print #1 , "Xmega revision:" ; Mcu_revid
                                                               ' make sure
it is 7 or higher !!! lower revs have many flaws
Const Usechannel = 1
Dim B1 As Byte , B2 As Byte
Dim W As Word At B1 Overlay
Open "twic" For Binary As #4
                                                               ' or use
TWID, TWIE OR TWIF
Config Twi = 100000
                                                               'CONFIG TWI
will ENABLE the TWI master interface
'you can also use TWIC, TWID, TWIE of TWIF
#if Usechannel = 1
   I2cinit #4
#else
  I2cinit
#endif
Do
  I2cstart
  Waitms 20
  12cwbyte &H70
                                                               ' slave
address write
  Waitms 20
  I2cwbyte & B10101010
                                                               ' write
command
  Waitms 20
  I2cwbyte 2
 Waitms 20
  I2cstop
 Print "Error : " ; Err
                                                               ' show error
status
 'waitms 50
  Print "start"
  I2cstart
  Print "Error : " ; Err
                                                               ' show error
  12cwbyte &H71
  Print "Error : " ; Err
                                                               ' show error
  I2crbyte B1 , Ack
  Print "Error : " ; Err
I2crbyte B2 , Nack
                                                               ' show error
  Print "Error : " ; Err
                                                               ' show error
  I2cstop
  Print "received A/D : " ; W ; "-" ; B1 ; "-" ; B2
                                                               'wait a bit
  Waitms 500
Loop
```

```
Dim J As Byte , C As Byte , K As Byte
Dim Twi_start As Byte
                                                              ' you MUST
dim this variable since it is used by the lib
'determine if we have an i2c slave on the bus
For J = 0 To 200 Step 2
  Print J
  #if Usechannel = 1
    I2cstart #4
  #else
    I2cstart
  #endif
   I2cwbyte J
   If Err = 0 Then
                                                              ' no errors
      Print "FOUND : " ; Hex(j)
      'write some value to the pcf8574A
      #if Usechannel = 1
         12cwbyte & B1100_0101 , #4
      #else
         12cwbyte &B1100_0101
      #endif
      Print Err
      Exit For
   End If
   #if Usechannel = 1
       I2cstop #4
   #else
       I2cstop
   #endif
Next
#if Usechannel = 1
    I2cstop #4
#else
    I2cstop
#endif
#if Usechannel = 1
  I2cstart #4
  12cwbyte &H71 , #4
                                                              'read
address
  I2crbyte J , Ack , #4
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Ack , #4
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Nack , #4
  Print Bin(j) ; " err:" ; Err
  I2cstop #4
#else
  I2cstart
  I2cwbyte &H71
                                                              'read
address
  I2crbyte J , Ack
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Ack
  Print Bin(j) ; " err:" ; Err
  I2crbyte J , Nack
  Print Bin(j) ; " err:" ; Err
  I2cstop
#endif
```

```
'try a transaction
#if Usechannel = 1
                                                               ' all 1
 12csend & H70 , 255 , #4
 Waitms 1000
 I2csend & H70 , 0 , #4
                                                              'all O
#else
 12csend & H70 , 255
 Waitms 1000
 12csend & H70 , 0
#endif
Print Err
'read transaction
Dim Var As Byte
Var = &B11111111
#if Usechannel = 1
 I2creceive &H70 , Var , 1 , 1 , #4
                                                               ' send and
receive
 Print Bin(var) ; "-" ; Err
 I2creceive &H70 , Var , 0 , 1 , #4
                                                               ' just
receive
 Print Bin(var) ; "-" ; Err
#else
                                                               ' send and
 I2creceive & H70 , Var , 1 , 1
receive
 Print Bin(var) ; "-" ; Err
 I2creceive &H70 , Var , 0 , 1
                                                               ' just
receive
 Print Bin(var) ; "-" ; Err
#endif
End
```

6.322 OUT

Action

Sends a byte to a hardware port or internal or external memory address.

Syntax

OUT address, value

Remarks

	The address where to send the byte to in the range of 0-FFFF hex.	
Value	The variable or value to output.	

The OUT statement can write a value to any AVR memory location.

It is advised to use Words for the address. An integer might have a negative value and will write of course to a word address. So it will be 32767 higher as supposed. This because an integer has it's most significant bit set when it is negative.

To write to XRAM locations you must enable the External RAM access in the <u>Compiler Chip Options</u> .

802

You do not need to use OUT when setting a port variable. Port variables and other registers of the micro can be set like this : PORTB = value, where PORTB is the name of the register.

Take special care when using register variables. The address-part of the OUT statement, expects a numeric variable or constant. When you use a hardware register like for example PORTB, what will happen is that the value of PORTB will be used. Just as when you use a variable, it will use the variable value. So when the goal is to just write to a hardware register, you need to use the normal assignment : PORTB=3

See also

INP 7407 , PEEK 8037 , POKE 8047

Example

Out &H8000 , 1 'send 1 to the databus(d0-d7) at hex address 8000 End

6.323 PEEK

Action

Returns the content of a register.

Syntax

var = **PEEK**(address)

Remarks

	Numeric variable that is assigned with the content of the memory location address	
Address	dress Numeric variable or constant with the address location.(0-31)	

Peek() will read the content of a register. Inp() can read any memory location

See also

POKE [804], CPEEK [598], INP [740], OUT [802], SETREG [858], GETREG [716]

Example

'	
'name	: peek.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: demonstrates PEEk, POKE, CPEEK, INP and OUT
'micro	: Mega48
'suited for demo	: yes
'commercial addon needed	: no

804

```
$regfile = "m162def.dat"
                                                              ' specify
the used micro
$crystal = 4000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
                                                              ' default
$hwstack = 32
use 32 for the hardware stack
$swstack = 10
                                                              ' default
use 10 for the SW stack
                                                              ' default
$framesize = 40
use 40 for the frame space
Dim I As Integer , B1 As Byte
'dump internal memory
For I = 0 To 31
                                                              'only 32
registers in AVR
  B1 = Peek(i)
                                                              'get byte
from internal memory
  Print Hex(b1) ; " ";
                                          'write a value into memory
  'Poke I , 1
Next
Print
                                                              'new line
'be careful when writing into internal memory !!
'now dump a part of the code-memory(program)
For I = 0 To 255
 B1 = Cpeek(i)
                                                              'get byte
from internal memory
  Print Hex(b1) ; " ";
Next
'note that you can not write into codememory !!
Out &H8000 , 1
                                                              'write 1
into XRAM at address 8000
B1 = Inp(\&H8000)
                                                              'return
value from XRAM
Print B1
End
```

6.324 POKE

Action

Write a byte to an internal register.

Syntax

POKE address , value

Remarks

Address	Numeric variable with the address of the memory location to set. (0-31)
Value	Value to assign. (0-255)

See also

PEEK 8037, CPEEK 5987, INP 7407, OUT 8027, SETREG 8587, GETREG 7167

Example

Poke 1, 1 'write 1 to R1 End

6.325 POPALL

Action

Restores all registers that might be used by BASCOM.

Syntax

POPALL

Remarks

When you are writing your own ASM routines and mix them with BASIC you are unable to tell which registers are used by BASCOM because it depends on the used statements and interrupt routines that can run on the background.

That is why Pushall saves all used registers and POPALL restores all registers.

The SREG register is also saved/restored. The SREG register contains the processor flags and it is important to save these.

If the micro has a RAMPZ register, the RAMPZ register is saved/restored also. RAMPZ is used to address multiple pages in flash and SRAM memory.

See also

PUSHALL 818

6.326 **POWER**

Action

Returns the power of a single or double variable and its argument

Syntax

var = POWER(source, raise)

Remarks

	A numeric variable that is assigned with the power of variable source ^ raise.
Source The single or double variable to get the power of.	

The POWER function works for positive floating point variables only. When you use a b , the sign will be preserved.

While Excel does not allow raising a negative single, QB does allow it. The Power functions uses less code compared with the code that is generated when you use ^ for floating point values.

It is important that you use single variables for both single and raise. Constants are not accepted.

In version 1.11.9.2 the power function is improved so that it returns the same result as Excel. Previously it returned the same number as QB/VB. For example : $-2 \land 2$ would be returned as -4, but $-2 \land 3$ would be returned as -8 which is wring since $-2 \land 3 = -2 \times -2 \times -2 = 4 \times -2 = -8$. Minus times a minutes makes a positive number. So it depends on the sign of the base and if the number of raise if even or odd.

Base	Raise	Result
0	0	NAN
NAN	х	NAN
х	NAN	NAN
Infinity	х	NAN
х	Infinity	NAN
0	x<0	Infinity
0	x>0	0
х	0	1
x<0	x<>int(x)	NAN

The exception handling was also improved.

See Also

EXP 683 ,LOG 7721, LOG10 7721 , SQR 9031

Example

Show sample 983

Example for Double Exceptions

\$regfile = "m128def.dat"
\$crystal = 4000000

Dim D1 As Double , D2 As Double , D3 As Double Dim dInf as Double, dNAN as Double

d1 = -1: dNAN = log(d1)d1 = 1: d2 = 0: dInf = D1 / D2

D1 = 0: D2 = 0: GoSub ShowPowerTest

D1 = dNAN: D2 = 3: GoSub ShowPowerTest

D1 = 3: D2 = dNAN: GoSub ShowPowerTest

D1 = dInf: D2 = 4: GoSub ShowPowerTest

- D1 = 4: D2 = dInf: GoSub ShowPowerTest
- D1 = 0: D2 = -2: GoSub ShowPowerTest
- D1 = 0: D2 = 3: GoSub ShowPowerTest
- D1 = 5: D2 = 0: GoSub ShowPowerTest
- D1 = -2: D2 = -3.5: GoSub ShowPowerTest
- D1 = -2: D2 = 3.5: GoSub ShowPowerTest
- D1 = -2: D2 = -3: GoSub ShowPowerTest
- D1 = -2: D2 = -4: GoSub ShowPowerTest
- D1 = -2: D2 = -5: GoSub ShowPowerTest
- D1 = -2: D2 = 3: GoSub ShowPowerTest
- D1 = -2: D2 = 4: GoSub ShowPowerTest
- D1 = -2: D2 = 5: GoSub ShowPowerTest

end

ShowPowerTest:

D3 = POWER(D1, D2)

Print "POWER("; D1; ", "; D2; ") = "; D3

Return

-----Simulator Output ------POWER() - Test

POWER(0, 0) = NAN

- POWER(NAN, 3) = NAN
- POWER(3, NAN) = NAN
- POWER(Infinity, 4) = NAN
- POWER(4, Infinity) = NAN
- POWER(0, -2) = Infinity
- POWER(0, 3) = 0

POWER(5, 0) = 1 POWER(-2, -3.5) = NAN POWER(-2, 3.5) = NAN POWER(-2, -3) = -125E-3POWER(-2, -4) = 62.5E-3POWER(-2, -5) = -31.25E-3POWER(-2, 3) = -8POWER(-2, 4) = 16POWER(-2, 5) = -32

6.327 POWER MODE

Action

Put the micro processor in one of the supported power reserving modes.

Syntax

POWER mode

Remarks

The mode depends on the micro processor. Some valid options are :

- IDLE
- POWERDOWN
- STANDBY
- ADCNOISE
- POWERSAVE

So for standby you would use : POWER STANDBY It is also possible to use POWERDOWN, IDLE or POWERSAVE. These modes were/are supported by most processors. It is recommended to use the new POWER command because it allows to use more modes.

POWER has nothing to do with the **POWER** () function.

See also IDLE [734], POWERDOWN (809), POWERSAVE (809)

Example POWER IDLE

6.328 POWERDOWN

Action

Put processor into power down mode.

Syntax POWERDOWN

Remarks

In the power down mode, the external oscillator is stopped. The user can use the WATCHDOG to power up the processor when the watchdog timeout expires. Other possibilities to wake up the processor is to give an external reset or to generate an external level triggered interrupt.

You should use the new POWER POWERDOWN statement.

See also IDLE [734], POWERSAVE 809, POWER mode 808

Example

Powerdown

6.329 POWERSAVE

Action

Put processor into power save mode.

Syntax POWERSAVE

Remarks

The POWERSAVE mode is only available in the 8535, Mega8, Mega163.

Most new chips have many options for Power down/Idle. It is advised to consult the data sheet to see if a better mode is available.

You should use the new POWER POWERSAVE statement.

See also IDLE [734], POWERDOWN [809], POWER mode [808]

Example

Powersave

6.330 PRINT

Action

Send output to the RS-232 port. Writes a string to a file. Writes data to a device.

Syntax

PRINT [#channel ,] var ; " constant"

Remarks

Var The variable or constant to print.

You can use a semicolon (;) to print more than one variable at one line. When you end a line with a semicolon, no linefeed and carriage return will be added.

The PRINT routine can be used when you have a RS-232 interface on your uP. The RS-232 interface can be connected to a serial communication port of your computer.

This way you can use a terminal emulator as an output device.

You can also use the build in terminal emulator.

AVR-DOS

The AVR-DOS file system also supports PRINT. But in that case, only strings can be written to disk.

When you need to print to the second hardware UART, or to a software UART, you need to specify a channel : PRINT #1, "test"

The channel must be opened first before you can print to it. Look at OPEN and CLOSE for more details about the optional channel. For the first hardware UART, there is no need to use channels.

PRINT " test" will always use the first hardware UART.

Xmega-SPI

When sending data to the SPI interface, you need to activate the SS pin. Some chips might need an active low, others might need an active high. This will depends on the slave chip.

When you use the SS=AUTO option, the level of SS will be changed automatic. Thus SS is made low, then the data bytes are sent, and finally, SS is made high again.

For SPI, no CRLF will be sent. Thus a trailing ; is not needed.

Number of Bytes

The compiler will send 1 byte for variable which was dimensioned as a BYTE. It will send 2 bytes for a WORD/INTEGER, 4 bytes for a LONG/SINGLE and 8 bytes for a DOUBLE.

As with all routines in BASCOM, the least significant Byte will be send first.

When you send a numeric constant, the binary value will be sent : 123 will be send a 1 byte with the value of 123.

If you send an array, it will send one element. With an optional parameter you can provide how many bytes must be sent. You must use a comma (,) to specify this parameter. This because the semi colon (;) is used to send multiple variables.

Sample

Dim Tmparray(5) As Byte , Spi_send_byte As Byte , W as Word Config Spid = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk32 , Data_order = Msb , Ss = Auto Open "SPID" For Binary As #12 Print #12 , Spi_send_byte ; W ' send ONE BYTE and 2 bytes of W Print #12 , Tmparray(1) , 2 ' send 2 bytes of tmparray, starting at element 1 Print #12 , Tmparray(1) ' send 1 byte Print #12 , Tmparray(3) , 2 ' send 2 bytes starting at index 3 Print #12 , 123 ; 1000 ; Tmparray(1) , B ' send byte with value 123, 2 bytes with value 1000, and 'b' bytes of array

See also

INPUT 744, OPEN 7947, CLOSE 4377, SPC 8987

Example

!_____ _____ _____ 'name : print.bas 'copyright : (c) 1995-2005, MCS Electronics 'purpose : demo: PRINT, HEX : Mega48 'suited for demo : yes 'commercial addon needed : no 1_____ _____ \$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space Dim A As Byte , B1 As Byte , C As Integer , S As String * 4 $\Delta = 1$ Print "print variable a " ; A 'new line Print Print "Text to print." 'constant to print B1 = 10 Print Hex(b1) 'print in hexa notation C = & HA000'assign value to c%

Print Hex(C) 'print in hex notation Print C 'print in decimal notation C = -32000Print C Print Hex(C) Rem Note That Integers Range From -32767 To 32768 Print "You can also use multiple" _ ; "lines using _" Print "use it for long lines" 'From version 1.11.6.4 : A = &B1010 0111 Print Bin(a) S = "1001"A = Binval(s) '9 dec Print A End

6.331 PRINTBIN

Action

Print binary content of a variable to the serial port.

Syntax

PRINTBIN var [; varn]
PRINTBIN #channel, var [; varn]

Remarks

Var	The variable which value is send to the serial port.	
varn	Optional variables to send.	

The channel is optional and for use with OPEN 794 and CLOSE 437 statements.

PRINTBIN is equivalent to PRINT CHR(var); When you use a Long for example, 4 bytes are printed.

Multiple variables may be sent. They must be separated by the ; sign.

The number of bytes to send can be specified by an additional numeric parameter. This is convenient when sending the content of an array.

Printbin ar(1); 3 ' will send 3 bytes from array ar(). Printbin ar(1); 2; ar(2); 4 ' will send 2 bytes from array ar() starting at index 1, then 4 bytes from array ar() starting at index 4.

When you use Printbin ar(1), the whole array will be printed. When you need to print the content of a big array(array with more then 255 elements) you need to use the CONFIG PRINTBIN option.

See also

INPUTBIN 747, CONFIG PRINTBIN 535

Example

```
Dim A(10) As Byte, C As Byte
For C = 1 To 10
    A(c)= c 'fill array
Next
Printbin A(1) 'print content of a(1). Note that the whole array will be
sent!
End
```

6.332 PSET

Action

Sets or resets a single pixel.

Syntax

PSET X , Y, value

Remarks

Х	The X location of the pixel. In range from 0-239.
Y	The Y location of the pixel. In range from 0-63.
value	The value for the pixel. 0 will clear the pixel. 1 Will set the pixel.

The PSET is handy to create a simple data logger or oscilloscope.

See also

SHOWPIC (883), CONFIG GRAPHLCD [506], LINE [759]

Example

```
-----
_____
                     : t6963_240_128.bas
'name
                      : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                      : T6963C graphic display support demo 240 *
128
                     : Mega8535
'micro
'micro : Meg
'suited for demo : yes
'commercial addon needed : no
·_____
                                  _____
_____
$regfile = "m8535.dat"
                                                   ' specify
the used micro
$crystal = 8000000
                                                   ' used
crystal frequency
                                                   ' use baud
$baud = 19200
rate
$hwstack = 32
                                                   ' default
use 32 for the hardware stack
\$swstack = 10
                                                   ' default
use 10 for the SW stack
$framesize = 40
                                                   ' default
use 40 for the frame space
```

```
814
      BASCOM-AVR
                                 _____
                        (c) 2001-2003 MCS Electronics
                    T6963C graphic display support demo 240 * 128
    'The connections of the LCD used in this demo
    'LCD pin
                            connected to
    ' 1
             GND
                            GND
    ' 2
            GND
                           GND
    ' 3
            +5V
                          +5V
    ' 4
             -9V
                           -9V potmeter
    ' 5
             /WR
                          PORTC.0
    '6
             /RD
                           PORTC.1
    '7
             /CE
                           PORTC.2
    ' 8
             C/D
                           PORTC.3
    '9
            NC
                           not conneted
    '10
                          PORTC.4
            RESET
    '11-18 D0-D7
                           PA
    '19
           FS
                          PORTC.5
    '20
            NC
                          not connected
    'First we define that we use a graphic LCD
    ' Only 240*64 supported yet
    Config Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc ,
    LCD
    'The controlport is the portname which pins are used to control the lcd
    'CE, CD etc. are the pin number of the CONTROLPORT.
    ' For example CE =2 because it is connected to PORTC.2
    'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns
    'Dim variables (y not used)
    Dim X As Byte , Y As Byte
    'Clear the screen will both clear text and graph display
    Cls
    'Other options are :
    ' CLS TEXT to clear only the text display
    ' CLS GRAPH to clear only the graphical part
    Cursor Off
    Wait 1
    'locate works like the normal LCD locate statement
    ' LOCATE LINE, COLUMN LINE can be 1-8 and column 0-30
    Locate 1 , 1
    'Show some text
    Lcd "MCS Electronics"
    'And some othe text on line 2
    Locate 2 , 1 : Lcd "T6963c support"
    Locate 3 , 1 : Lcd "1234567890123456789012345678901234567890"
    Locate 16 , 1 : Lcd "write this to the lower line"
    Wait 2
    Cls Text
    'use the new LINE statement to create a box
```

'LINE(X0,Y0) - (X1,Y1), on/off Line(0, 0) - (239, 127), 255 ' diagonal line Line(0 , 127) - (239 , 0) , 255 ' diagonal line **Line**(0 , 0) -(240 , 0) , 255 ' horizontal upper line Line(0, 127) - (239, 127), 255 'horizontal lower line **Line**(0 , 0) -(0 , 127) , 255 ' vertical left line Line(239, 0) - (239, 127), 255 ' vertical right line Wait 2 ' draw a line using PSET X,Y, ON/OFF ' PSET on.off param is 0 to clear a pixel and any other value to turn it on **For** X = 0 **To** 140 **Pset** X , 20 , 255 ' set the pixel Next **For** X = 0 **To** 140 **Pset** X , 127 , 255 ' set the pixel Next Wait 2 'circle time 'circle(X,Y), radius, color 'X,y is the middle of the circle,color must be 255 to show a pixel and 0 to clear a pixel **For** X = 1 **To** 10 Circle(20 , 20) , X , 255 ' show circle Wait 1 **Circle**(20 , 20) , X , 0 'remove circle Wait 1 Next Wait 2 **For** X = 1 **To** 10 **Circle**(20, 20), X, 255 ' show circle Waitms 200 Next Wait 2 'Now it is time to show a picture 'SHOWPIC X,Y,label 'The label points to a label that holds the image data Test: Showpic 0 , 0 , Plaatje Showpic 0 , 64 , Plaatje ' show 2 since we have a big display Wait 2 Cls Text ' clear the text End

```
'This label holds the mage data
Plaatje:
'$BGF will put the bitmap into the program at this location
$bgf "mcs.bgf"
'You could insert other picture data here
```

6.333 PS2MOUSEXY

Action

Sends mouse movement and button information to the PC.

Syntax

PS2MOUSEXY X , Y, button

Remarks

Х	The X-movement relative to the current position.
	The range is −255 to 255.
Y	The Y-movement relative to the current position.
	The range is −255 to 255.
Button	A variable or constant that represents the button state.
	0 – no buttons pressed 1- left button pressed 2- right button pressed 4- middle button pressed
	You can combine these values by adding them. For example, 6 would emulate that the right and middle buttons are pressed.
	To send a mouse click, you need to send two ps2mouseXY statements. The first must indicate that the button is pressed, and the second must release the button.
	Ps2mouseXY 0,0,1 ' left mouse pressed
	PsmouseXY 0,0,0 ' left mouse released

The SENDSCAN statement could also be used.

See also

SENDSCAN 8627, CONFIG PS2EMU 535

6.334 PULSEIN

Action

Returns the number of units between two occurrences of an edge of a pulse.

Syntax

PULSEIN var , PINX , PIN , STATE

Remarks		
var	A word variable that is assigned with the result.	
PINX	A PIN register like PIND	
PIN	The pin number(0-7) to get the pulse time of.	
STATE	May be 0 or 1.	
	0 means sample 0 to 1 transition. 1 means sample 1 to 0 transition.	

ERR variable will be set to 1 in case of a time out. A time out will occur after 65535 unit counts. With 10 uS units this will be after 655.35 mS.

You can add a <u>bitwait</u> statement to be sure that the PULSEIN statement will wait for the start condition. But when using the BITWAIT statement and the start condition will never occur, your program will stay in a loop.

The PULSIN statement will wait for the specified edge.

When state 0 is used, the routine will wait until the level on the specified input pin is 0. Then a counter is started and stopped until the input level gets 1.

No hardware timer is used. A 16 bit counter is used. It will increase in 10 uS units. But this depends on the XTAL. You can change the library routine to adjust the units.

See also

PULSEOUT 817

ASM

The following ASM routine is called from mcs.lib _pulse_in (calls _adjust_pin)

On entry ZL points to the PINx register , R16 holds the state, R24 holds the pin number to sample. On return XL + XH hold the 16 bit value.

Example

Dim w As Word pulsein w , PIND , 1 , 0 'detect time from 0 to 1 print w End

6.335 PULSEOUT

Action

Generates a pulse on a pin of a PORT of specified period in 1uS units for 4 MHz.

Syntax

PULSEOUT PORT , PIN , PERIOD

Remarks

PORT	Name of the PORT. PORTB for example
PIN	Variable or constant with the pin number (0-7).
PERIOD	Number of periods the pulse will last. The periods are in uS when an XTAL of 4 MHz is used.

The pulse is generated by toggling the pin twice, thus the initial state of the pin determines the polarity.

The PIN must be configured as an output pin before this statement can be used.

See also

PULSEIN 816

Example

Dim A As Byte Config Portb = Output output pins Portb = 0	'PORTB all
	'all pins O
Do	
For $A = 0$ To 7	
Pulseout Portb , A , 60000	'generate
pulse	
Waitms 250	'wait a bit
Next	
Loop	'loop for
ever	

6.336 PUSHALL

Action

Saves all registers that might be used by BASCOM.

Syntax PUSHALL

Remarks

When you are writing your own ASM routines and mix them with BASIC you are unable to tell which registers are used by BASCOM because it depends on the used statements and interrupt routines that can run on the background.

That is why Pushall saves all used registers. Use POPALL to restore the registers.

The saved registers are : R0-R5, R7, R10, R11 and R16-R31

The SREG register is also saved. The SREG register contains the processor flags and it is important to save these.

If the micro has a RAMPZ register, the RAMPZ register is saved too. RAMPZ is used to address multiple pages in flash and SRAM memory.

See also

POPALL 805

6.337 PUT

Action

Writes a byte to the hardware or software UART. Writes data to a file opened in BINARY mode.

Syntax

PUT #channel, var
PUT #channel, var ,[pos] [,length]

Remarks

PUT in combination with the software/hardware UART is provided for compatibility with BASCOM-8051. It writes one byte

PUT in combination with the AVR-DOS file system is very flexible and versatile. It works on files opened in BINARY mode and you can write all data types.

	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Var	The variable or variable array that will be written to the file
Pos	This is an optional parameter that may be used to specify the position where the data must be written. This must be a long variable.
	This is an optional parameter that may be used to specify how many bytes must be written to the file.

By default you only need to provide the variable name. When the variable is a byte, 1 byte will be written. When the variable is a word or integer, 2 bytes will be written. When the variable is a long or single, 4 bytes will be written. When the variable is a string, the number of bytes that will be written is equal to the dimensioned size of the string. DIM S as string * 10, would write 10 bytes.

Note that when you specify the length for a string, the maximum length is 255. The maximum length for a non-string array is 65535.

Example

PUT #1, var PUT #1, var , , 2 ' write 2 bytes at default position PUT #1, var ,PS, 2 ' write 2 bytes at location storied in variable PS

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [695], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], DISKSIZE [667], GET [697], FILEDATE [685], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]



current position	Goto new position first
Byte:	
_FilePutRange_1 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_1 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Word/Integer:	
_FilePutRange_2 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_2 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Long/Single:	
_FilePutRange_4 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_4 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
String (<= 255 Bytes) with fixed length	
_FilePutRange_Bytes Input: r24: File number r20: Count of Bytes X: Pointer to variable T-Flag cleared	_FilePutRange_Bytes Input: r24: File number r20: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Array (> 255 Bytes) with fixed length	
_FilePutRange Input: r24: File number r20/21: Count of Bytes X: Pointer to variable T-Flag cleared	_FilePutRange Input: r24: File number r20/21: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set

Output from all kind of usage: r25: Error Code C-Flag on Error

Example

'for the binary file demo we need some variables of different types Dim B AsByte, W AsWord, L AsLong, Sn AsSingle, Ltemp AsLong Dim Stxt AsString* 10 B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt = "test"

'open the file in BINARY mode Open"test.biN"ForBinaryAs#2 Put#2 , B ' write a byte Put#2 , W ' write a word Put#2 , L ' write a long Ltemp =Loc(#2)+ 1 ' get the position of the next byte Print Ltemp ;" LOC"' store the location of the file pointer Print Seek(#2);" = LOC+1"

PrintLof(#2);" length of file" PrintFileattr(#2);" file mode"' should be 32 for binary Put#2 , Sn ' write a single Put#2 , Stxt ' write a string

Flush#2 ' flush to disk Close#2

'now open the file again and write only the single Open"test.bin"ForBinaryAs#2 L = 1 'specify the file position B =Seek(#2, L)' reset is the same as using SEEK #2,L Get#2, B ' get the byte Get#2, W ' get the word Get#2, L ' get the long Get#2, Sn ' get the single Get#2, Stxt ' get the string Close#2

6.338 QUOTE

Action

The Quote function will return a string surrounded by quotes.

Syntax

var = QUOTE(Source)

Remarks

Var	A string variable that is assigned with the quoted string of variable source.
Source	The string or string constant to be quoted.

The Quote() function can be used in HTML web server pages.

See also

NONE

Example

```
Dim S as String * 20
S = "test"
S = Quote(s)
Print S ' would print "test"
End
```

6.339 RAD2DEG

822

Action

Converts a value in radians to degrees.

Syntax

var = **RAD2DEG**(Source)

Remarks

	A numeric variable that is assigned with the angle of variable source.
Source	The single or double variable to get the angle of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

DEG2RAD 649

Example

```
: (c) 1995-2005, MCS Electronics
: Mega48
'copyright
'micro
'micro : Meg
'suited for demo : yes
'commercial addon needed : no
'purpose
                    : demonstrates DEG2RAD function
*_____
_____
Dim S As Single
S = 90
S = Deg2Rad(s)
Print S
S = Rad2deg(s)
Print S
End
```

6.340 RC5SEND

Action

Sends RC5 remote code.

Syntax

RC5SEND togglebit, address, command

Uses

TIMER1

Remarks

Togglebit	Make the toggle bit 0 or 32 to set the toggle bit	
Address	The RC5 address	
Command	The RC5 command.	

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.

Look in a data sheet for the proper pin when used with a different chip.

Most audio and video systems are equipped with an infra-red remote control. The RC5 code is a 14-bit word bi-phase coded signal.

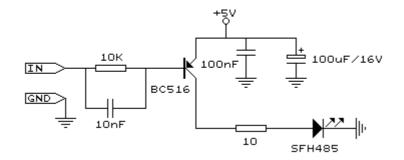
The two first bits are start bits, always having the value 1.

The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remote control transmitter.

Five system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code). An IR booster circuit is shown below:



See also

CONFIG RC5 538 , GETRC5 712 , RC6SEND 826

Example

```
'name
                       : sendrc5.bas
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : code based on application note from Ger
Langezaal
'micro
                       : AT90S2313
'suited for demo
                       : yes
'commercial addon needed : no
_ _ _ _ _ _
_____
$regfile = "2313def.dat"
                                                       ' specify
the used micro
```

```
$crystal = 4000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
\$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
    +5V <---[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC5SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3
Dim Togbit As Byte , Command As Byte , Address As Byte
Command = 12
                                                              ' power on
off
Togbit = 0
                                                              ' make it 0
or 32 to set the toggle bit
Address = 0
Do
  Waitms 500
  Rc5send Togbit , Address , Command
   'or use the extended RC5 send code. You can not use both
   'make sure that the MS bit is set to 1, so you need to send
   '&B10000000 this is the minimal requirement
   '&B11000000 this is the normal RC5 mode
   '&B10100000 here the toggle bit is set
   ' Rc5sendext &B11000000 , Address , Command
Loop
End
```

6.341 RC5SENDEXT

Action

Sends extended RC5 remote code.

Syntax

RC5SENDEXT togglebit, address, command

Uses

TIMER1

Remarks

Togglebit	Make the toggle bit 0 or 32 to set the toggle bit	
Address	The RC5 address	
Command	The RC5 command.	

Normal RC5 code uses 2 leading bits with the value '1'. After that the toggle bit follows.

With extended RC5, the second bit is used to select the bank. When you make it 1 (the default and normal RC5) the RC5 code is compatible. When you make it 0, you select bank 0 and thus use extended RC5 code.

825

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.

Look in a data sheet for the proper pin when used with a different chip.

Most audio and video systems are equipped with an infra-red remote control. The RC5 code is a 14-bit word bi-phase coded signal.

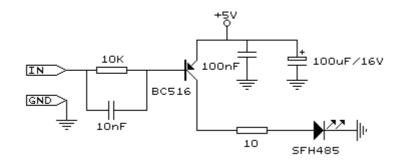
The two first bits are start bits, always having the value 1.

The next bit is a control bit or toggle bit, which is inverted every time a button is pressed on the remote control transmitter.

Five system bits hold the system address so that only the right system responds to the code.

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is six bits long, allowing up to 64 different commands per address.

The bits are transmitted in bi-phase code (also known as Manchester code). An IR booster circuit is shown below:



See also

CONFIG RC5 538, GETRC5 712, RC6SEND 826

Example

'name	: sendrc5.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: code based on application note from Ger
Langezaal	
'micro	: AT90S2313
'suited for demo	: yes
'commercial addon needed	: no
· ·	
<pre>\$regfile = "2313def.dat"</pre>	' specify
the used micro	
\$crystal = 4000000	' used
crystal frequency	
\$baud = 19200	' use baud

' default

' default

rate

\$hwstack = 32

\$swstack = 10

use 32 for the hardware stack

use 10 for the SW stack

```
826 BASCOM-AVR
```

```
' default
$framesize = 40
use 40 for the frame space
   +5V <---[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC5SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3
Dim Togbit As Byte , Command As Byte , Address As Byte
Command = 12
                                                             ' power on
off
Togbit = 0
                                                             ' make it 0
or 32 to set the toggle bit
Address = 0
Do
  Waitms 500
    ' Rc5send Togbit , Address , Command
   'or use the extended RC5 send code. You can not use both
   'make sure that the MS bit is set to 1, so you need to send
   '&B10000000 this is the minimal requirement
   '&B11000000 this is the normal RC5 mode
   '&B10100000 here the toggle bit is set
  Rc5sendExt &B11000000 , Address , Command
Loop
End
```

6.342 RC6SEND

Action

Sends RC6 remote code.

Syntax

RC6SEND togglebit, address, command

Uses

TIMER1

the code.

Remarks

Togglebit	Make the toggle bit 0 or 1 to set the toggle bit
Address	The RC6 address
Command The RC6 command.	

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.

Look in a data sheet for the proper pin when used with a different chip.

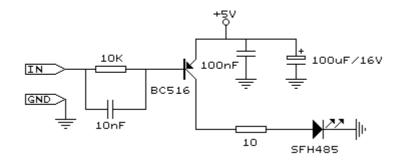
Most audio and video systems are equipped with an infrared remote control. The RC6 code is a 16-bit word bi-phase coded signal. The header is 20 bits long including the toggle bits. Eight system bits hold the system address so that only the right system responds to

Usually, TV sets have the system address 0, VCRs the address 5 and so on. The command sequence is eight bits long, allowing up to 256 different commands per

address.

The bits are transmitted in bi-phase code (also known as Manchester code).

An IR booster circuit is shown below:



Device	Address
TV	0
VCR	5
SAT	8
DVD	4

This is not a complete list.

Command	Value	Command	Value
Key 0	0	Balance right	26
Key 1	1	Balance left	27
Key 2-9	2-9	Channel search+	30
Previous program	10	Channel search -	31
Standby	12	Next	32
Mute/un-mute	13	Previous	33
Personal preference	14	External 1	56
Display	15	External 2	57
Volume up	16	TXT submode	60
Volume down	17	Standby	61
Brightness up	18	Menu on	84
Brightness down	19	Menu off	85
Saturation up	20	Help	129
Saturation down	21	Zoom -	246
Bass up	22	Zoom +	247
Bass down	23		
Treble up	24		
Treble down	25		

This list is by far not complete.

Since there is little info about RC6 on the net available, use code at your own risk!

See also CONFIG RC5 5387, GETRC5 7127, RC5SEND 8227

Example

```
_____
_____
                     : sendrc6.bas
'name
'copyright
                     : (c) 1995-2005, MCS Electronics
'purpose
                      : code based on application note from Ger
Langezaal
                     : AT90S2313
'micro
'suited for demo
                     : yes
'commercial addon needed : no
·_____
                                _____
_____
$regfile = "2313def.dat"
                                                   ' specify
the used micro
$crystal = 4000000
                                                   ' used
crystal frequency
$baud = 19200
                                                   ' use baud
rate
$hwstack = 32
                                                   ' default
use 32 for the hardware stack
\$swstack = 10
                                                   ' default
use 10 for the SW stack
                                                   ' default
$framesize = 40
use 40 for the frame space
  +5V <---[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC6SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3
Dim Togbit As Byte , Command As Byte , Address As Byte
```

```
'this controls the TV but you could use rc6send to make your DVD region
free as well :-)
'Just search the net for the codes you need to send. Do not ask me for
info please.
Command = 32
                                                              ' channel
next
                                                              ' make it 0
Togbit = 0
or 32 to set the toggle bit
Address = 0
Do
   Waitms 500
  Rc6send Togbit , Address , Command
Loop
End
```

6.343 READ

Action

Reads those values and assigns them to variables.

Syntax

READ var

Remarks

829

Var Variable that is assigned data value.

It is best to place the DATA [612] lines at the end of your program.



It is important that the variable is of the same type as the stored data.

See also

DATA 612 , RESTORE 841

· · · · · · · · · · · · · · · · · · ·		
'copyright 'purpose	: no	
<pre>\$regfile = "m48def.dat"</pre>		' specify
the used micro \$crystal = 4000000		' used
crystal frequency \$baud = 19200		' use baud
rate \$hwstack = 32		' default
use 32 for the hardware s \$swstack = 10	tack	' default
<pre>use 10 for the SW stack \$framesize = 40 use 40 for the frame space</pre>		' default
Dim A As Integer , B1 As : Dim S As String * 15 Dim L As Long Restore Dtal stored data For Count = 1 To 3 of data items Read B1 : Print Count Next	Byte , Count As Byte	'point to 'for number
Restore Dta2		'point to
<pre>stored data For Count = 1 To 2 of data items Read A : Print Count ; Next</pre>	" "; A	'for number
Restore Dta3 Read S : Print S Read S : Print S		
Restore Dta4 Read L : Print L		'long type

```
'demonstration of readlabel
                                                             ' location
Dim W As Iram Word At 8 Overlay
is used by restore pointer
'note that W does not use any RAM it is an overlayed pointer to the data
pointer
W = Loadlabel(dta1)
                                                              ' loadlabel
expects the labelname
Read B1
Print B1
End
Dtal:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%
Dta3:
Data "Hello" , "World"
'Note that integer values (>255 or <0) must end with the %-sign
'also note that the data type must match the variable type that is
'used for the READ statement
Dta4:
Data 123456789&
'Note that LONG values must end with the &-sign
'Also note that the data type must match the variable type that is used
'for the READ statement
```

6.344 READEEPROM

Action

Reads the content from the DATA EEPROM and stores it into a variable.

Syntax

READEEPROM var , address

Remarks

Var	The name of the variable that must be stored
Address	The address in the EEPROM where the data must be read from.

This statement is provided for backwards compatibility with BASCOM-8051. You can also use the ERAM variable instead of READEEPROM :

Dim V as Eram Byte 'store in EEPROM Dim B As Byte 'normal variable B = 10V = B 'store variable in EEPROM B = V 'read from EEPROM

When you use the assignment version, the data types must be equal! According to a data sheet from ATMEL, the first location in the EEPROM with address 0, can be overwritten during a reset so don't use it.

You may also use ERAM variables as indexes. Like :

Dim ar(10) as Eram Byte

When you omit the address label in consecutive reads, you must use a new READEEPROM statement. It will not work in a loop:

Readeeprom B , Label1 Print B

Do Readeeprom B Print B Loop Until B = 5

This will not work since there is no pointer maintained. The way it will work :

ReadEEprom B , Label1 ' specify label ReadEEPROM B ' read next address in EEPROM ReadEEPROM B ' read next address in EEPROM

In the XMEGA, you need to set the mode to mapped : <u>CONFIG EEPROM</u> 49^{+} = MAPPED.

See also

WRITEEEPROM 954 , <u>\$EEPROM</u> 321

ASM

NONE

```
_____
                         _____
 _____
                 : eeprom2.bas
'name
'copyright
                 : (c) 1995-2005, MCS Electronics
'purpose
                 : shows how to use labels with READEEPROM
                 : Mega48
'micro
'suited for demo
                 : yes
'commercial addon needed : no
! _____
_____
```

```
$regfile = "m48def.dat"
                                                               ' specify
the used micro
$crystal = 4000000
                                                               ' used
crystal frequency
$baud = 19200
                                                               ' use baud
rate
\$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                               ' default
use 40 for the frame space
'first dimension a variable
Dim B As Byte
Dim Yes As String * 1
'Usage for readeeprom and writeeprom :
```

832 BASCOM-AVR

```
'readeeprom var, address
'A new option is to use a label for the address of the data
'Since this data is in an external file and not in the code the eeprom
data
'should be specified first. This in contrast with the normal DATA lines
which must
'be placed at the end of your program!!
'first tell the compiler that we are using EEPROM to store the DATA
$eeprom
'the generated EEP file is a binary file.
'Use $EEPROMHEX to create an Intel Hex file usable with AVR Studio.
'$eepromhex
'specify a label
Label1:
Data 1 , 2 , 3 , 4 , 5
Label2:
Data 10 , 20 , 30 , 40 , 50
'Switch back to normal data lines in case they are used
$data
'All the code above does not generate real object code
'It only creates a file with the EEP extension
'Use the new label option
Readeeprom B , Label1
Print B
                                                              'prints 1
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B
                                                             'prints 2
Readeeprom B , Label2
Print B
                                                              'prints 10
Readeeprom B
Print B
                                                             'prints 20
'And it works for writing too :
'but since the programming can interfere we add a stop here
Input "Ready?" , Yes
B = 100
Writeeeprom B , Labell
B = 101
Writeeeprom B
'read it back
Readeeprom B , Labell
Print B
                                                             'prints 100
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B
                                                              'prints 101
End
```

6.345 READHITAG

Action

Read HITAG RFID transponder serial number.

Syntax

result = **READHITAG**(var)

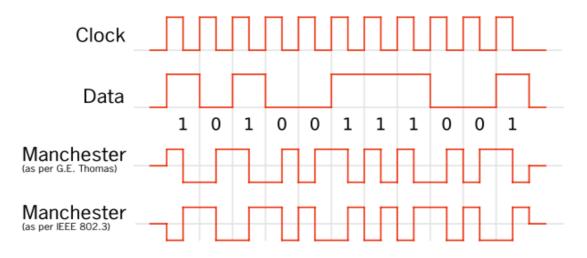
Remarks

A numeric variable that will be 0 if no serial number was read
from the transponder. It will return 1 if a valid number was
read.

RFID is used for entrance systems, anti theft, and many other applications where a wireless chip is an advantage over the conventional magnetic strip and chip-card. The HITAG series from Philips(NXP) is one of the oldest and best available. The HTRC110 chip is a simple to use chip that can read and write transponders. Each transponder chip has a 5 byte(40 bits) unique serial number. The only disadvantage of the HTRC110 is that you need to sign an NDA in order to

get the important documents and 8051 example code.

When the transponder is held before the coil of the receiver, the bits stream will be modulated with the bit values. Just like RC5, HITAG is using Manchester encoding. This is a simple and reliable method used in transmission systems. Manchester encoding is explained very well at the <u>Wiki</u> Manchester page.

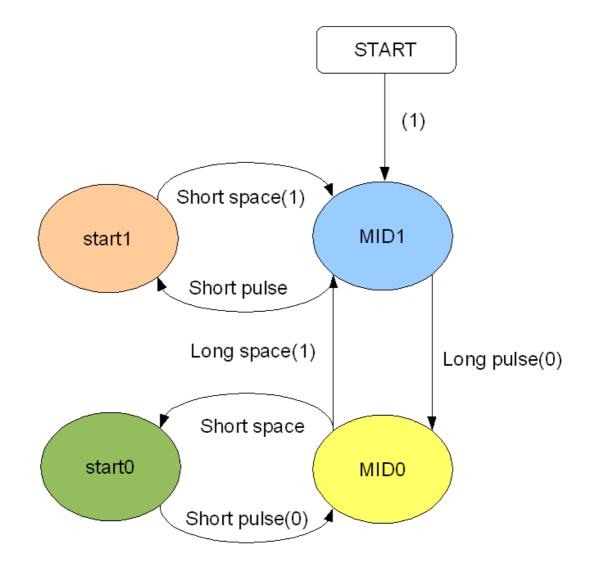


The image above is copied from the Wiki.

There are 2 methods to decode the bits. You can detect the edges of the bits and sample on 3/4 of the bit time.

Another way is to use a state machine. The state machine will check the length between the edges of the pulse. It will start with the assumption that there is a (1). Then it will enter the MID1 state. If the next pulse is a long pulse, we have received a (0). When it received a short pulse, we enter the start1 state. Now we need to receive a short space which indicated a (1), otherwise we have an invalid state. When we are in the MID0 state, we may receive a long space(1) or a short space. All others pulses are invalid and lead to a restart of the pulse state(START).

Have a look at the image above. Then see how it really works. We start with assuming a (1). We then receive a long pulse so we receive a (0). Next we receive a long space which is a (1). And again a long pulse which is a (0) again. Then we get a short space and we are in start1 state. We get a short pulse which is a (0) and we are back in MID0 state. The long space will be a (1) and we are in MID1 state again. etc.etc. When ever we receive a pulse or space which is not defined we reset the pulse state machine.



At 125 KHz, the bit time is 512 uS. A short pulse we define as halve a bit time which is 256 uS.

We use a 1/4 of the bit time as an offset since the pulses are not always exactly precise.

So a short bit is 128-384(256-128 - 256+128) uS. And a long bit is 384-640 uS (512-128 - 512+128).

We use TIMER0 which is an 8 bit timer available in all AVR's to determine the time. Since most micro's have an 8 MHz internal clock, we run the program in 8 MHz. It depends on the pre scaler value of the timer, which value are used to determine the length between the edges.

You can use 64 or 256. The generated constants are : _TAG_MIN_SHORT, _TAG_MAX_SHORT , _TAG_MIN_LONG and _TAG_MAX_LONG.

We need an interrupt to detect when an edge is received. We can use the INTx for this and configure the pin to interrupt when a logic level changes. Or we can use the PIN interrupt so we can use more pins.

The sample contains both methods.

It is important that the ReadHitag() functions needs a variable that can store 5 bytes. This would be an array.

And you need to check the _TAG constants above so that they do not exceed 255.

When you set up the interrupt, you can also use it for other tasks if needed. You only

need to call the _checkhitag routine in the subroutine. And you need to make sure that the additional code you write does not take up too much time.

When you use the PCINT interrupt it is important to realize that other pins must be masked off. The PCMSK register may have only 1 bit enabled. Otherwise there is no way to determine which pin was changed.

EM4095

The EM4095 is similar to the HTRC110. The advantage of the EM4095 is that it has a synchronized clock and needs no setup and less pins.

The EM4095 library uses the same method as the RC5 decoding : the bit is sampled on 3/4 of the bit length. The parity handling is the same. The EM4095 decoding routine is smaller then the HTRC110 decoding library.

A reference design for the EM4095 will be available from MCS.

See also

READMAGCARD [835], CONFIG HITAG [494]

Example

See <u>CONFIG HITAG</u> [494] for 2 examples.

6.346 READMAGCARD

Action

Read data from a magnetic card.

Syntax

READMAGCARD var , count , coding

Remarks

Var	A byte array the receives the data.
Count	A byte variable that returns the number of bytes read.
coding	A numeric constant that specifies if 5 or 7 bit coding is used. Valid values are 5 and 7.

There can be 3 tracks on a magnetic card.

Track 1 stores the data in 7 bit including the parity bit. This is handy to store alpha numeric data.

On track 2 and 3 the data is stored with 5 bit coding.

The ReadMagCard routine works with ISO7811-2 5 and 7 bit decoding.

The recurried multiple	is for 5 bit county are.
Returned number	ISO characterT
0	0
1	1
2	2

The returned numbers for 5 bit coding are:

3

3

4	4
5	5
6	6
7	7
8	8
9	9
10	hardware control
11	start byte
12	hardware control
13	separator
14	hardware control
15	stop byte

Example

```
-----
!_____
      _____
'name
                       : magcard.bas
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : show you how to read data from a magnetic
card
'micro
                       : Mega48
'suited for demo
                       : yes
'commercial addon needed : no
·-----
                                            _____
_____
$regfile = "m48def.dat"
                                                       ' specify
the used micro
$crystal = 4000000
                                                       ' used
crystal frequency
$baud = 19200
                                                       ' use baud
rate
$hwstack = 32
                                                       ' default
use 32 for the hardware stack
\$swstack = 10
                                                       ' default
use 10 for the SW stack
$framesize = 40
                                                       ' default
use 40 for the frame space
'[reserve some space]
Dim Ar(100) As Byte , B As Byte , A As Byte
'the magnetic card reader has 5 wires
'red - connect to +5V
        - connect to GND
'black
'yellow - Card inserted signal CS
'green - clock
'blue
        - data
'You can find out for your reader which wires you have to use by
connecting +5V
'And moving the card through the reader. CS gets low, the clock gives a
clock pulse of equal pulses
'and the data varies
'I have little knowledge about these cards and please dont contact me
about magnectic readers
'It is important however that you pull the card from the right direction
as I was doing it wrong for
```

836

```
'some time :-)
'On the DT006 remove all the jumpers that are connected to the LEDs
'[We use ALIAS to specify the pins and PIN register]
_mport Alias Pinb
                                                               'all pins
are connected to PINB
_mdata Alias 0
                                                               'data line
(blue) PORTB.0
                                                               'CS line
_mcs Alias 1
(yellow) PORTB.1
_mclock Alias 2
                                                               'clock line
(green) PORTB.2
Config Portb = Input
                                                               'we only
need bit 0,1 and 2 for input
Portb = 255
                                                               'make them
high
Do
  Print "Insert magnetic card"
                                                               'print a
message
                                                               'read the
  Readmagcard Ar(1) , B , 5
data
  Print B ; " bytes received"
  For A = 1 To B
   Print Ar(a);
                                                               'print the
bytes
  Next
  Print
Loop
```

'By specifying 7 instead of 5 you can read 7 bit data

6.347 READSIG

Action

This function reads a byte from the signature area in the XMEGA.

Syntax

var = READSIG(offset)

Remarks

Var	A byte that is assigned with the signature byte.
Offset	A byte variable or constant with an offset to the signature.

The Xmega has a number of signature bytes that are important. For example the ADC is calibrated in the factory and the calibration data need to be loaded into the ADC registers in order to achieve 12 bit resolution.

The following offset table is copied from the Xmega128A1 definition file. It should be the same for all other Xmega chips but it is best to check it.

Const NVM_PROD_SIGNATURES_RCOSC2M_offset = &H00	' RCOSC 2MHz
Calibration Value	
Const NVM_PROD_SIGNATURES_RCOSC32K_offset =&H02	' RCOSC
32kHz Calibration Value	
Const NVM_PROD_SIGNATURES_RCOSC32M_offset = &H03	' RCOSC 32MHz

Calibration Value	
Const NVM_PROD_SIGNATURES_LOTNUM0_offset = &H08	' Lot Number
Byte 0, ASCII	
Const NVM_PROD_SIGNATURES_LOTNUM1_offset = &H09	' Lot Number
Byte 1, ASCII	
Const NVM_PROD_SIGNATURES_LOTNUM2_offset = &H0A	' Lot Number
Byte 2, ASCII	
Const NVM_PROD_SIGNATURES_LOTNUM3_offset = &H0B	' Lot Number
Byte 3, ASCII	
Const NVM_PROD_SIGNATURES_LOTNUM4_offset = &H0C	' Lot Number
Byte 4, ASCII	
Const NVM_PROD_SIGNATURES_LOTNUM5_offset = &H0D	' Lot Number
Byte 5, ASCII	
Const NVM_PROD_SIGNATURES_WAFNUM_offset = &H10	' Wafer
Number	
Const NVM_PROD_SIGNATURES_COORDX0_offset = &H12	' Wafer
Coordinate X Byte 0	
Const NVM_PROD_SIGNATURES_COORDX1_offset = &H13	' Wafer
Coordinate X Byte 1	
Const NVM_PROD_SIGNATURES_COORDY0_offset = &H14	' Wafer
Coordinate Y Byte 0	
Const NVM_PROD_SIGNATURES_COORDY1_offset = &H15	' Wafer
Coordinate Y Byte 1	
Const NVM_PROD_SIGNATURES_ADCACAL0_offset = &H20	' ADCA Calibration
Byte 0	
Const NVM_PROD_SIGNATURES_ADCACAL1_offset = &H21	' ADCA Calibration
Byte 1	
Const NVM_PROD_SIGNATURES_ADCBCAL0_offset = &H24	' ADCB
Calibration Byte 0	
Const NVM_PROD_SIGNATURES_ADCBCAL1_offset = &H25	' ADCB
Calibration Byte 1	
Const NVM_PROD_SIGNATURES_TEMPSENSE0_offset = &H2E	' Temperature Sensor
Calibration Byte 0 Const NVM_PROD_SIGNATURES_TEMPSENSE1_offset = &H2F	- Tomporatura Concor
Const $NVM_PROD_SIGNATORES_TEMPSENSET_ONSET = & PZP Calibration Byte 0$	' Temperature Sensor
Const NVM_PROD_SIGNATURES_DACAOFFCAL_offset = &H30	' DACA Calibration
Byte 0	DACA Calibration
Const NVM_PROD_SIGNATURES_DACACAINCAL_offset = &H31	DACA Calibration
Byte 1	
Const NVM_PROD_SIGNATURES_DACBOFFCAL_offset = &H32	' DACB Calibration
Byte 0	
Const NVM_PROD_SIGNATURES_DACBGAINCAL_offset = &H33	' DACB Calibration
Byte 1	
- /	

See also

NONE

Example

```
(c) 1995-2010, MCS
xm128-readsig.bas
This sample demonstrates how to read signature bytes
```

\$regfile = "xm128aldef.dat"

```
scrystal = 3200000
\$hwstack = 64
\$swstack = 40
$framesize = 40
'include the following lib and code, the routines will be replaced since
they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits =
1 , Databits = 8
Dim Offset As Byte , J As Byte
For J = 0 To 32
    Offset = Readsig(j) : Print J ; " - " ; Offset
Next
```

```
6.348 REM
```

End

```
Action
```

Instruct the compiler that comment will follow.

Syntax

REM or '

Remarks

You can and should comment your program for clarity and your later sanity. You can use REM or ' followed by your comment. All statements after REM or ' are treated as comments so you cannot use statements on the same line after a REM statement.

Block comments can be used too:

'(start block comment print "This will not be compiled') end block comment

Example

Rem TEST.BAS version 1.00

Print A ' " this is comment : PRINT " Hello "

840

^ - - - This Will Not Be Executed!

6.349 REPLACECHARS

Action

Replace all occurrences of a character in a string by a different character.

Syntax

REPLACECHARS string , old, new

Remarks

string	A string variable.
	A character or byte with the ASCII value of the character to search for.
new	A character of byte with the ASCII value with the new value.

When we have a string with a content of : "abcdefabc" and we want to replace the "a" by an "A" we can use : Replacechars string , "a" , "A"

All occurrences are replaced.

See also

INSTR 747, MID 787, CHARPOS 424, DELCHAR 651, INSERTCHAR 746, DELCHARS 652

Example

\$regfile = "m644def.DAT"
\$hwstack = 24
32 for the hw stack
\$swstack = 24
use 10 for the SW stack
\$framesize = 24
use 40 for the frame

Dim Textout As String * 22 Dim Var As String * 1

Textout = "abcdefabdef"
Replacechars Textout , "a" , "A"
Print Textout

Var = "e"
Replacechars Textout , Var , "A"
Print Textout

```
End
```

6.350 RESET

Action

Reset a bit to zero.

'default use

' default

' default

Syntax RESET bit RESET var.x RESET var

Remarks

Bit	Bit or Boolean variable.
Var	A byte, integer, word or long variable.
X	Bit of variable to clear. Valid values are : 0-7 (byte, registers), 0-15 (Integer/Word) and (0-31) for a Long

You can also use the constants from the definition file to set or reset a bit. RESET PORTB.PB7 'will reset bin 7 of portB. This because PB7 is a defined constant in the definition file.

When the bit is not specified, bit 0 will be cleared.

See also

SET 853 , TOGGLE 933

Example

SEE SET 853

6.351 RESTORE

Action

Allows READ to reread values in specified DATA statements by setting data pointer to beginning of data statement.

Syntax

RESTORE label

Remarks

label The label of a DATA statement.

See also

DATA 612 , READ 828 , LOOKUP 774

Example

'name 'copyright 'purpose

: readdata.bas : (c) 1995-2005, MCS Electronics : demo : READ,RESTORE

```
842 BASCOM-AVR
```

```
'micro
                        : Mega48
'suited for demo
                        : yes
'commercial addon needed : no
·_____.
                                      _____
_____
$regfile = "m48def.dat"
                                                          ' specify
the used micro
$crystal = 4000000
                                                          ' used
crystal frequency
$baud = 19200
                                                          ' use baud
rate
$hwstack = 32
                                                          ' default
use 32 for the hardware stack
$swstack = 10
                                                          ' default
use 10 for the SW stack
$framesize = 40
                                                          ' default
use 40 for the frame space
Dim A As Integer , B1 As Byte , Count As Byte
Dim S As String * 15
Dim L As Long
Restore Dtal
                                                          'point to
stored data
For Count = 1 To 3
                                                          'for number
of data items
  Read B1 : Print Count ; " " ; B1
Next
Restore Dta2
                                                          'point to
stored data
For Count = 1 To 2
                                                          'for number
of data items
   Read A : Print Count ; " " ; A
Next
Restore Dta3
Read S : Print S
Read S : Print S
Restore Dta4
Read L : Print L
                                                          'long type
'demonstration of readlabel
Dim W As Iram Word At 8 Overlay
                                                          ' location
is used by restore pointer
'note that W does not use any RAM it is an overlayed pointer to the data
pointer
                                                          ' loadlabel
W = Loadlabel(dta1)
expects the labelname
Read B1
Print B1
End
Dtal:
Data &B10 , &HFF , 10
Dta2:
Data 1000% , -1%
Dta3:
Data "Hello" , "World"
```

```
'Note that integer values (>255 or <0) must end with the %-sign
'also note that the data type must match the variable type that is
'used for the READ statement
Dta4:
Data 123456789&
'Note that LONG values must end with the &-sign
'Also note that the data type must match the variable type that is used
'for the READ statement
```

6.352 **RETURN**

Action

Return from a subroutine.

Syntax RETURN

Remarks

Subroutines must be ended with a related RETURN statement. Interrupt subroutines must also be terminated with the Return statement.

See also

GOSUB 719

```
_____
_____
'name
                     : gosub.bas
                     : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                      : demo: GOTO, GOSUB and RETURN
'micro
                      : Mega48
                     : yes
'suited for demo
'commercial addon needed : no
·-----
                                 _____
_____
$regfile = "m48def.dat"
                                                   ' specify
the used micro
$crystal = 4000000
                                                   ' used
crystal frequency
$baud = 19200
                                                   ' use baud
rate
\$hwstack = 32
                                                   ' default
use 32 for the hardware stack
\$swstack = 10
                                                   ' default
use 10 for the SW stack
$framesize = 40
                                                   ' default
use 40 for the frame space
Goto Continue
Print "This code will not be executed"
                                                   'end a label
Continue:
with a colon
Print "We will start execution here"
```

```
Gosub Routine
Print "Back from Routine"
End
Routine:
subroutine
    Print "This will be executed"
Return
subroutine
```

'start a

'return from

6.353 RIGHT

Action

Return a specified number of rightmost characters in a string.

Syntax

var = RIGHT(var1 ,n)

Remarks

var	The string that is assigned.
Var1	The source string.
st	The number of bytes to copy from the right of the string.

See also

LEFT 758 , MID 787

Example

```
Dim S As String * 15 , Z As String * 15
S = "ABCDEFG"
Z = Left(s , 5)
Print Z
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

'ABCDE

6.354 RND

Action

Returns a random number.

Syntax

var = **RND**(limit)

Remarks

Limit	Word that limits the returned random number.
Var	The variable that is assigned with the random number.

The RND() function returns an Integer/Word and needs an internal storage of 2 bytes.

(____RSEED). Each new call to Rnd() will give a new positive random number.

Notice that it is a software based generated number. And each time you will restart your program the same sequence will be created.

You can use a different SEED value by dimensioning and assigning ____RSEED yourself: Dim ____rseed as word : ____rseed = 10234 Dim I as word : I = rnd(10)

When your application uses a timer you can assign ____RSEED with the timer value. This will give a better random number.

See also

NONE

```
_____
                       : rnd.bas
'name
'copyright
                       : (c) 1995-2005, MCS Electronics
'purpose
                       : demo : RND() function
                      : Mega48
'micro
'suited for demo
                      : yes
'commercial addon needed : no
*_____
_____
$regfile = "m48def.dat"
                                                     ' specify
the used micro
$crystal = 4000000
                                                     ' used
crystal frequency
$baud = 19200
                                                     ' use baud
rate
\$hwstack = 32
                                                     ' default
use 32 for the hardware stack
\$swstack = 10
                                                     ' default
use 10 for the SW stack
$framesize = 40
                                                     ' default
use 40 for the frame space
Dim I As Word
                                                     ' dim
variable
Do
                                                     'get random
 I = Rnd(40)
number (0-39)
 Print I
                                                     'print the
value
                                                     'wait 1
 Wait 1
second
                                                     'for ever
Loop
End
```

6.355 ROTATE

846

Action

Rotate all bits one place to the left or right.

Syntax

ROTATE var , LEFT/RIGHT[, shifts]

Remarks

Var	Byte, Integer/Word or Long variable.
Shifts	The number of shifts to perform.

The ROTATE statement rotates all the bits in the variable to the left or right. All bits are preserved so no bits will be shifted out of the variable.

This means that after rotating a byte variable with a value of 1, eight times the variable will be unchanged.

When you want to shift out the MS bit or LS bit, use the SHIFT statement.

See also

SHIFT 875, SHIFTIN 877, SHIFTOUT 881

Example

'name	: rotate.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: example for ROTATE and SHIFT statement
'micro	: Mega48
'suited for demo	: yes
'commercial addon needed	: no
· ·	

```
$regfile = "m48def.dat"
                                                               ' specify
the used micro
$crystal = 4000000
                                                               ' used
crystal frequency
$baud = 19200
                                                               ' use baud
rate
$hwstack = 32
                                                               ' default
use 32 for the hardware stack
\$swstack = 10
                                                               ' default
use 10 for the SW stack
$framesize = 40
                                                               ' default
use 40 for the frame space
'dimension some variables
Dim B As Byte , I As Integer , L As Long
```

'the shift statement shift all the bits in a variable one 'place to the left or right 'An optional paramater can be provided for the number of shifts. 'When shifting out then number 128 in a byte, the result will be 0 'because the MS bit is shifted out

```
B = 1
Shift B , Left
Print B
'B should be 2 now
B = 128
Shift B , Left
Print B
'B should be 0 now
'The ROTATE statement preserves all the bits
'so for a byte when set to 128, after a ROTATE, LEFT , the value will
'be 1
'Now lets make a nice walking light
'First we use PORTB as an output
Config Portb = Output
'Assign value to portb
Portb = 1
Do
  For I = 1 To 8
     Rotate Portb , Left
     'wait for 1 second
     Wait 1
  Next
   'and rotate the bit back to the right
  For I = 1 To 8
    Rotate Portb , Right
     Wait 1
  Next
Loop
End
```

6.356 ROUND

Action

Returns a value rounded to the nearest value.

Syntax

var = **ROUND**(x)

Remarks

	A single or double variable that is assigned with the ROUND of variable x.
Х	The single or double to get the ROUND of.

Round(2.3) = 2 , Round(2.8) = 3 Round(-2.3) = -2 , Round(-2.8) = -3

See Also

INT 748 , FIX 688 , SGN 874

Example

```
848 BASCOM-AVR
```

```
'name
                         : round fix int.bas
                        : (c) 1995-2005, MCS Electronics
'copyright
                         : demo : ROUND,FIX
'purpose
'micro
                         : Mega48
'suited for demo
                         : yes
'commercial addon needed : no
·_____.
_____
$regfile = "m48def.dat"
                                                          ' specify
the used micro
$crystal = 4000000
                                                          ' used
crystal frequency
$baud = 19200
                                                          ' use baud
rate
\$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 10
                                                          ' default
use 10 for the SW stack
                                                          ' default
$framesize = 40
use 40 for the frame space
Dim S As Single , Z As Single
For S = -10 To 10 Step 0.5
 Print S; Spc(3); Round(s); Spc(3); Fix(s); Spc(3); Int(s)
Next
```

End

6.357 RTRIM

Action

Returns a copy of a string with trailing blanks removed

Syntax

var = RTRIM(org)

Remarks

var	String that is assigned with the result.
org	The string to remove the trailing spaces from

See also

TRIM 935 , LTRIM 763

ASM

NONE

```
Dim S As String * 6
S =" AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

6.358 SECELAPSED

Action

Returns the elapsed Seconds to a former assigned time-stamp.

Syntax

Target = **SECELAPSED**(TimeStamp)

Remarks

Target	A variable (LONG), that is assigned with the elapsed Seconds
	A variable (LONG), which holds a timestamp like the output of an earlier
	called SecOfDay()

The Function works with the SOFTCLOCK variables _sec, _min and _hour and considers a jump over midnight and gives a correct result within 24 hour between two events.

The Return-Value is in the range of 0 to 86399.

See also

<u>Date and Time Routines (993)</u>, <u>SecOfDay</u> (849), <u>SysSecElapsed</u> (916)

Partial Example

```
Lsecofday = Secofday()
_hour = _hour + 1
Lvar1 = Secelapsed(lsecofday)
Print Lvar1
```

6.359 SECOFDAY

Action

Returns the Seconds of a Day.

Syntax

```
Target = SECOFDAY()
Target = SECOFDAY(bSecMinHour)
Target = SECOFDAY(strTime)
Target = SECOFDAY(ISysSec)
```

Remarks

Target	A variable (LONG), that is assigned with the Seconds of the Day	
bSecMinHour	A Byte, which holds the Second-value followed by Minute(Byte) and	
	Hour(Byte)	
strTime	A String, which holds the time in the format "hh:mm:ss"	
LSysSec	A Variable (Long) which holds the System Second	

The Function can be used with 4 different kind of inputs:

- Without any parameter. The internal Time of SOFTCLOCK (_sec, _min, _hour) is used.
- 2. With a user defined time array. It must be arranged in same way (Second, Minute, Hour) as the internal SOFTCLOCK time. The first Byte (Second) is the input by this kind of usage. So the Second of Day can be calculated of every time.
- 3. With a time-String. The time-string must be in the Format "hh:mm:ss".
- 4. With a System Second Number (LONG)

The Return-Value is in the range of 0 to 86399 from 00:00:00 to 23:59:59. No validity-check of input is made.

See also

Date and Time Routines [993], SysSec [915]

Partial Example

```
' ========= Second of Day
_____
' Example 1 with internal RTC-Clock
_sec = 12 : _min = 30 : _hour = 18
                                                         ' Load RTC-
Clock for example - testing
Lsecofday = Secofday()
Print "Second of Day of " ; Time$ ; " is " ; Lsecofday
' Example 2 with defined Clock - Bytes (Second / Minute / Hour)
Bsec = 20 : Bmin = 1 : Bhour = 7
Lsecofday = Secofday(bsec)
Print "Second of Day of Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour
; " (" ; Time(bsec); ") is "; Lsecofday
' Example 3 with System Second
Lsyssec = 1234456789
Lsecofday = Secofday(lsyssec)
Print "Second of Day of System Second " ; Lsyssec ; "(" ; Time(lsyssec)
; ") is "; Lsecofday
' Example 4 with Time - String
Strtime = "04:58:37"
Lsecofday = Secofday(strtime)
```

6.360 SEEK

Action

Function: Returns the position of the next Byte to be read or written Statement: Sets the position of the next Byte to be read or written

Print "Second of Day of "; Strtime; " is "; Lsecofday

Syntax

Function: NextReadWrite = SEEK (#bFileNumber)
Statement: SEEk #bFileNumber, NewPos

Remarks	
bFileNumber	(Byte) Filenumber, which identifies an opened file
NextReadWrit A Long Variable, which is assigned with the Position of the next B to be read or written (1-based)	
	A Long variable that holds the new position the file pointer must be set too.

This function returns the position of the next Byte to be read or written. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError.

The statement also returns an error in the gbDOSerror variable in the event that an error occurs.

You can for example not set the file position behinds the file size.

In VB the file is filled with 0 bytes when you set the file pointer behind the size of the file. For embedded systems this does not seem a good idea.

Seek and Loc seems to do the same function, but take care : the seek function will return the position of the next read/write, while the Loc function returns the position of the last read/write. You may say that Seek = Loc+1.

In QB/VB you can use seek to make the file bigger. When a file is 100 bytes long, setting the file pointer to 200 will increase the file with 0 bytes. By design this is not the case in AVR-DOS.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [681], FREEFILE [695], FILEATTR [684], BSAVE [420], BLOAD [416], KILL [751], DISKFREE [660], DISKSIZE [661], GET [697], PUT [819], FILEDATE [685], FILETIME [687], FILEDATETIME [686], DIR [656], FILELEN [687], WRITE [953], INPUT [744]

ASM

Function Calls	_FileSeek	
Input	r24: filenumber	X: Pointer to Long-variable, which gets the result
Output	r25: Errorcode	C-Flag: Set on Error
Statement Calls	_FileSeekSet	
Input	r24: filenumber	X: Pointer to Long-variable with the position
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

852 BASCOM-AVR

```
Open "test.biN" for Binary As #2
Put#2 , B
                                                                 ' write a
byte
Put#2 , W
                                                                 ' write a
word
Put#2 , L
                                                                 ' write a
long
Ltemp = Loc(#2) + 1
                                                                 ' get the
position of the next byte
Print Ltemp ; " LOC"
location of the file pointer
                                                                 ' store the
Print Seek(#2) ; " = LOC+1"
Close #2
'now open the file again and write only the single
Open "test.bin" For Binary As #2
Seek#2 , Ltemp
                                                                 ' set the
filepointer
Sn = 1.23
                                                                 ' change the
single value so we can check it better
```

```
'specify the
```

6.361 SELECT-CASE-END SELECT

Action

Put #2 , Sn = 1
file position
Close #2

Executes one of several statement blocks depending on the value of an expression.

Syntax

SELECT CASE var CASE test1 : statements [CASE test2 : statements] CASE ELSE : statements END SELECT

Remarks

Var	Variable to test the value of
Test1	Value to test for.
Test2	Value to test for.

You can test for conditions to like:

CASE IS > 2 :

Another option is to test for a range :

CASE 2 TO 5 :

See also

Example

```
_____
_____
                       : case.bas
'name
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : demonstrates SELECT CASE statement
'micro
                       : Mega48
'suited for demo
                       : yes
'commercial addon needed : no
·-----
                                         _____
_____
$regfile = "m48def.dat"
                                                       ' specify
the used micro
$crystal = 4000000
                                                       ' used
crystal frequency
$baud = 19200
                                                       ' use baud
rate
                                                       ' default
$hwstack = 32
use 32 for the hardware stack
$swstack = 10
                                                       ' default
use 10 for the SW stack
                                                       ' default
$framesize = 40
use 40 for the frame space
Dim I As Byte
                                                       'dim
variable
Dim S As String * 5 , Z As String * 5
Do
 Input "Enter value (0-255) " , I
 Select Case I
   Case 1 : Print "1"
   Case 2 : Print "2"
   Case 3 To 5 : Print "3-5"
   Case Is >= 10 : Print ">= 10"
   Case Else : Print "Not in Case statement"
 End Select
Loop
End
'note that a Boolean expression like > 3 must be preceded
'by the IS keyword
```

6.362 SET

Action

Set a bit to the value one.

Syntax

SET bit SET var.x SET var

Remarks

Bit	Bit or Boolean variable.	
Var	A byte, integer, word or long variable.	
Х	Bit of variable to set. Valid values are : 0-7 (byte, registers), 0-15 (Integer/Word) and (0-31) for a Long	

When the bit is not specified, bit 0 will be set.

Also notice that the bit range is 0-255. Using a larger value on a variable will overwrite a different variable !

When you need an array of say 128 bits you can use code like this : dim ar(32) as long

You can index these variables like : SET ar(1).127, in this case you write only to the memory of the intended variable.

See also

RESET 840, TOGGLE 933

```
_____
_____
'name
                      : boolean.bas
'copyright
                      : (c) 1995-2009, MCS Electronics
'purpose
                      : demo: AND, OR, XOR, NOT, BIT, SET, RESET and
MOD
'suited for demo
                      : yes
'commercial add on needed : no
'use in simulator : possible
·_____
_____
'This very same program example can be used in the Help-files for
     AND, OR, XOR, NOT, BIT, SET, RESET and MOD
$baud = 19200
$crystal = 16000000
$regfile = "m32def.dat"
\$hwstack = 40
\$swstack = 20
$framesize = 20
Dim A As Byte , B1 As Byte , C As Byte
Dim Aa As Bit , I As Integer
A = 5 : B1 = 3
                                                    ' assign
values
C = A And B1
                                                     ' and a with
b
Print "A And B1 = " ; C
                                                    ' print it:
result = 1
C = A Or B1
Print "A Or B1 = " ; C
                                                    ' print it:
result = 7
C = A Xor B1
Print "A Xor B1 = " ; C
                                                    ' print it:
result = 6
```

855

A = 1 C = Not A Print "c = Not A " ; C result = 254 C = C Mod 10 Print "C Mod 10 = " ; C result = 4	' print it: ' print it:
<pre>If Portb.1 = 1 Then Print "Bit set" Else Print "Bit not set" End If Bit not set</pre>	'result =
<pre>Aa = 1 Set Aa statement If Aa = 1 Then Print "Bit set (aa=1)" Else</pre>	'use this or 'use the set
<pre>Print "Bit not set(aa=0)" End If Bit set (aa=1)</pre>	'result =
<pre>Aa = 0 Reset Aa reset If Aa = 1 Then Print "Bit set (aa=1)" Else Print "Bit not set(aa=0)" End If Bit not set(aa=0)</pre>	'now try 0 'or use 'result =
<pre>C = 8 variable to &B0000_1000 Set C statement without specifying the bit Print C result = 9 ; bit0 has been set</pre>	'assign 'use the set 'print it:
B1 = 255 variable Reset B1.0 of a byte variable Print B1 result = 254 = &B1111110	'assign 'reset bit 0 'print it:
<pre>B1 = 8 variable to &B00001000 Set B1.0 Print B1 result = 9 = &B00001001 End</pre>	'assign 'set it 'print it:

6.363 SETFONT

Action

Sets the current font which can be used on some graphical displays.

Syntax

SETFONT font

Remarks

font	The name of the font that need to be used with LCDAT
	statements.

Since SED-based displays do not have their own font generator, you need to define your own fonts. You can create and modify your own fonts with the FontEditor Plugin.

SETFONT will set an internal used data pointer to the location in memory where you font is stored. The name you specify is the same name you use to define the font.

You need to include the used fonts with the \$include directive:

\$INCLUDE "font8x8.font"

The order of the font files is not important. The location in your source is however important.

The \$INCLUDE statement will include binary data and this may not be accessed by the flow of your program.

When your program flow enters into font code, unpredictable results will occur. So it is best to place the \$INCLUDE files at the end of your program behind the END statement.

You need to include the glibSED library with :

\$LIB "glibsed.lbx"

While original written for the SED1521, fonts are supported on a number of displays now including color displays.

See also

<u>CONFIG GRAPHLCD</u> 516 , <u>LCDAT</u> 755, <u>GLCDCMD</u> 718, <u>GLCDDATA</u> 718

'name 'copyright 'purpose display support 'micro 'suited for demo 'commercial addon needed '	<pre>: sed1520.bas : (c) 1995-2005, MCS Electronics : demonstrates the SED1520 based graphical : Mega48 : yes : no</pre>
<pre>\$regfile = "m48def.dat" the used micro \$crystal = 7372800 crystal frequency \$baud = 115200 rate \$hwstack = 32 use 32 for the hardware s</pre>	' specify ' used ' use baud ' default tack

BASCOM Language Reference 857

\$swstack = 10' default use 10 for the SW stack ' default **\$framesize** = 40 use 40 for the frame space 'I used a Staver to test 'some routines to control the display are in the glcdSED.lib file 'IMPORTANT : since the SED1520 uses 2 chips, the columns are split into 2 of 60. 'This means that data after column 60 will not print correct. You need to locate the data on the second halve 'For example when you want to display a line of text that is more then 8 chars long, (8x8=64) , byte 8 will not draw correctly 'Frankly i find the KS0108 displays a much better choice. \$lib "glcdSED1520.lbx" 'First we define that we use a graphic LCD **Config** Graphlcd = 120 * 64sed , Dataport = Porta , Controlport = Portd , Ce = 5, Ce2 = 7, Cd = 3, Rd = 4'The dataport is the portname that is connected to the data lines of the LCD 'The controlport is the portname which pins are used to control the lcd 'CE =CS Chip Enable/ Chip select 'CE2= Chip select / chip enable of chip 2 'CD=A0 Data direction 'RD=Read 'Dim variables (y not used) Dim X As Byte , Y As Byte 'clear the screen Cls Wait 2 'specify the font we want to use Setfont Font8x8 'You can use locate but the columns have a range from 1-132 'When you want to show somthing on the LCD, use the LDAT command 'LCDAT Y , COL, value Lcdat 1 , 1 , "1231231" Lcdat 3 , 80 , "11" 'lcdat accepts an additional param for inversing the text 'lcdat 1,1,"123" , 1 ' will inverse the text Wait 2 Line(0, 0) - (30, 30), 1Wait 2 Showpic 0 , 0 , Plaatje 'show a comnpressed picture End 'end program 'we need to include the font files \$include "font8x8.font" '\$include "font16x16.font"

BASCOM-AVR

```
Plaatje:
'include the picture data
$bgf "smile.bgf"
```

6.364 SETREG

858

Action

Writes a byte value to an internal register.

Syntax

SETREG Reg , value

Remarks

Most AVR chips have 32 registers named R0-R31. Registers R16-R31 can be assigned directly. Register R0-R15 do not accept this.

In some cases you might want to write to the internal registers. While you can include some ASM code directly, you can also use the BASIC SETREG statment.

Reg	The register name : R0-R31 or a register definition.
Value	A constant or byte value to assign to the register.

PEEK and POKE work with an address. And will return a HW register on the Xmega since Xmega has a different address map. GetReg and SetReg will read/write registers on all AVR processors.

Internally the compiler will use R24 if you write a constant to register R0-R15 : For example : Setreg R0 , 1

Compiles into: Ldi R24,\$01 Mov R0, R24

Setreg R31 , 1 Compiles into: Ldi R31,\$01

See also

GETREG [716], PEEK [803], POKE [804]

Example

6.365 SETTCP

Action

(Re) Configures the TCP/IP W3100A chip.

Syntax

SETTCP MAC , IP , SUBMASK , GATEWAY

Remarks

MAC	The MAC address you want to assign to the W3100A.
	The MAC address is a unique number that identifies your chip. You must use a different address for every W3100A chip in your network. Example : 123.00.12.34.56.78
	You need to specify 6 bytes that must be separated by dots. The bytes must be specified in decimal notation.
IP	The IP address you want to assign to the W3100A.
	The IP address must be unique for every W3100A in your network. When you have a LAN, 192.168.0.10 can be used. 192.168.0.x is used for LAN's since the address is not an assigned internet address.
SUBMASK	The submask you want to assign to the W3100A.
	The submask is in most cases 255.255.255.0
GATEWAY	This is the gateway address of the W3100A.
	The gateway address you can determine with the IPCONFIG command at the command prompt :
	C:\>ipconfig Windows 2000 IP Configuration
	Ethernet adapter Local Area Connection 2:
	Connection-specific DNS Suffix . : IP Address : 192.168.0.3 Subnet Mask : 255.255.255.0 Default Gateway : 192.168.0.1 Use 192.168.0.1 in this case.

The CONFIG TCPIP statement may be used only once.

When you want to set the TCP/IP settings dynamically for instance when the settings are stored in EEPROM, you can not use constants. For this purpose, SETTCP must be used.

SETTCP can take a variable or a constant for each parameter.

When you set the TCP/IP settings dynamically, you do not need to set them with CONFIG TCPIP. In the CONFIG TCPIP you can use the NOINIT parameter so that the MAC and IP are not initialized which saves code.

See also

<u>GETSOCKET</u> (717), <u>SOCKETCONNECT</u> (388), <u>SOCKETSTAT</u> (388), <u>TCPWRITE</u> (324), <u>TCPWRITESTR</u> (325), <u>TCPREAD</u> (324), <u>CLOSESOCKET</u> (439), <u>SOCKETLISTEN</u> (388), <u>CONFIG</u> <u>TCPIP</u> (559)

Example

See the DHCP.BAS example from the BASCOM Sample dir.

6.366 SETTCPREGS

Action

Writes to a W3100A register

Syntax

SETTCPREGS address, var , bytes

Remarks

	The address of the register W3100A register. This must be the value of the MSB. For example in location &H92 and &H93, the timeout is stored. You need to specify &H93 then.
var	The variable to write.
bytes	The number of bytes to write.

Most W3100A options are implemented with BASCOM statements or functions. When there is a need to write to the W3100A register you can use the SETTCPREGS commands. It can write multiple bytes. It is important that you specify the highest address. This because the registers must be written starting with the highest address.

See also

GETTCPREGS 716

ASM

NONE

<pre>\$regfile = "m88def.dat" ' specify</pre>	
the used micro	
<pre>\$crystal = 8000000 ' used crystal frequency \$baud = 19200 ' use bau</pre>	٩
<pre>rate \$hwstack = 80 use 32 for the hardware stack \$swstack = 128 use 10 for the SW stack</pre>	

BASCOM Language Reference 861

	Briedom Zanguage Kelerenee	
\$framesize = 80	' def	
use 40 for the frame space	der	auit
use to for the frame space		
Const Sock_stream = \$01	' Tap)
Const Sock_dgram = \$02	' Udp	
Const Sock_ipl_raw = \$03	-	Layer
Raw Sock	L	
Const Sock_macl_raw = \$04	' Mac	Layer
Raw Sock		-
Const Sel_control = 0	' Con	firm
Socket Status		
Const Sel_send = 1	' Con	firm Tx
Free Buffer Size		
Const Sel_recv = 2	' Con	firm Rx
Data Size		
'socket status		
Const Sock_closed = \$00	' Sta	tus Of
Connection Closed		
Const Sock_arp = \$01	' Sta	tus Of
Arp		OF
Const Sock_listen = \$02	- SLa	tus Of
Waiting For Tcp Connection Setup Const Sock_synsent = \$03		tua Of
Setting Up Tcp Connection	SLd	tus Of
Const Sock_synsent_ack = \$04		tus Of
Setting Up Tcp Connection	566	Cus OI
Const Sock_synrecv = \$05	' Sta	tus Of
Setting Up Tcp Connection	500	Cub OI
Const Sock_established = \$06	' Sta	tus Of
Tcp Connection Established		
Const Sock_close_wait = \$07	' Sta	tus Of
Closing Tcp Connection		
Const Sock_last_ack = \$08	' Sta	tus Of
Closing Tcp Connection		
Const Sock_fin_wait1 = \$09	' Sta	tus Of
Closing Tcp Connection		
Const Sock_fin_wait2 = \$0a	' Sta	tus Of
Closing Tcp Connection		
Const Sock_closing = \$0b	' Sta	tus Of
Closing Tcp Connection		
Const Sock_time_wait = \$0c	' Sta	tus Of
Closing Tcp Connection		
Const Sock_reset = \$0d	' Sta	tus Of
Closing Tcp Connection		tua Of
Const Sock_init = \$0e Socket Initialization	- SLa	tus Of
Const Sock_udp = \$0f		tus Of
Udp	500	CUS OI
Const Sock_raw = \$10	' Sta	tus of
IP RAW	500	cub or
'we do the usual		
Print "Init TCP"	' dis	play a
message		_
Enable Interrupts	' bef	ore we
use config tcpip , we need to enable the		
Config Tcpip = Int0 , Mac = 12.128.12.3	4.56.78 , Ip = 192.168.0.8	
Submask = 255.255.255.0 , Gateway = 192		0 , Tx
= \$55 , Rx = \$55 , Twi = &H80 , Clock =	400000	
Print "Init done"		

862 BASCOM-AVR

```
'set the IP address to 192.168.0.135
Settcp 12.128.12.24.56.78 , 192.168.0.135 , 255.255.255.0 , 192.168.0.88
Dim L As Long
'now read the IP address direct from the registers
L = Gettcpregs(\&H91, 4)
Print Ip2str(1)
Dim B4 As Byte At L Overlay
                                                             ' this byte
is the same as the LSB of L
'now make the IP address 192.168.0.136 by writing to the LSB
B4 = 136
                                                             'write
Settcpregs &H91 , L , 4
'and check if it worked
L = Gettcpregs(\&H91, 4)
Print Ip2str(1)
'while the address has the right value now the chip needs a reset in
order to use the new settings
L = &B1000001
                                                             ' set
sysinit and swrest bits
                                                             ' write 1
Settcpregs &H00 , L , 1
register
'and with PING you can check again that now it works
End
```

6.367 SENDSCAN

Action

Sends scan codes to the PC.

Syntax

SENDSCAN label

Remarks

Label	The name of the label that contains the scan codes.
-------	---

The SENDSCAN statement can send multiple scan codes to the PC. The label is used to specify the start of the scan codes. The first byte specifies the number of bytes that follow.

The following table lists all mouse scan codes.

Emulated Action	Data sent to host				
Move up one	08,00,01				
Move down one	28,00,FF				
Move right one	08,01,00				
Move left one	18,FF,00				
Press left button	09,00,00				

Release left button	08,00,00				
Press middle button	0C,00,00				
Release middle button	08,00,00				
Press right button	0A,00,00				
Release right button	08,00,00				

To emulate a left mouse click, the data line would look like this:

DATA 6 , &H09, &H00, &H00, &H08 , &H00, &H00 ^ send 6 bytes ^ left click

^ release

See also

PS2MOUSEXY and , CONFIG PS2EMU [535]

Example

_____ _____ 'name : ps2_emul.bas : (c) 1995-2005, MCS Electronics 'copyright : PS2 Mouse emulator 'purpose 'micro : 90S2313 'suited for demo : NO, commercial addon needed 'commercial addon needed : yes 1_____ _____ _____ \$regfile = "2313def.dat" ' specify the used micro \$crystal = 4000000' used crystal frequency **\$baud =** 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10 ' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space \$lib "mcsbyteint.lbx" ' use optional lib since we use only bytes 'configure PS2 pins Config Ps2emu = Int1 , Data = Pind.3 , Clock = Pinb.0 ^----- used interrupt ^----- pin connected to DATA 1 ^-- pin connected to clock 'Note that the DATA must be connected to the used interrupt pin Waitms 500 ' optional delay ' you need Enable Interrupts to turn on interrupts yourself since an INT is used

```
Print "Press u,d,l,r,b, or t"
Dim Key As Byte
Do
   Key = Waitkey()
                                                              ' get key
from terminal
    Select Case Key
      Case "u" : Ps2mousexy 0 , 10 , 0
                                                             ' up
      Case "d" : Ps2mousexy 0 , -10 , 0
                                                             ' down
      Case "1" : Ps2mousexy -10 , 0 , 0
                                                             ' left
     Case "r" : Ps2mousexy 10 , 0 , 0
                                                             ' right
     Case "b" : Ps2mousexy 0 , 0 , 1
                                                              ' left
button pressed
                 Ps2mousexy 0 , 0 , 0
                                                             ' left
button released
     Case "t" : Sendscan Mouseup
                                                             ' send a
scan code
     Case Else
   End Select
Loop
Mouseup:
```

6.368 SENDSCANKBD

Action

by 1 unit

Sends keyboard scan codes to the PC.

Data 3 , &H08 , &H00 , &H01

Syntax

SENDSCANKBD label | var

Remarks

Label	The name of the label that contains the scan codes.
var	The byte variable that will be sent to the PC.

The SENDSCANKBD statement can send multiple scan codes to the PC. The label is used to specify the start of the scan codes. The first byte specifies the number of bytes that follow.

You can also send the content of a variable. This way you can send dynamic information.

You need to make sure you send the make and break codes.

The following tables lists all scan codes.

AT KEYBOARD SCANCODES

Table reprinted with permission of Adam Chapweske

http://panda.cs.ndsu.nodak.edu/~achapwes

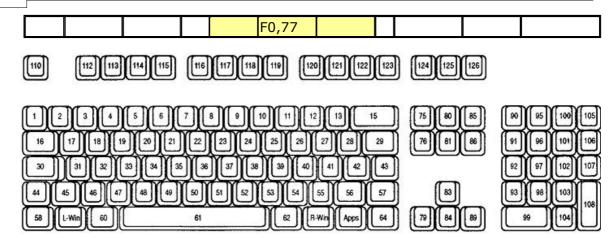
	KEY	MAKE	BREAK	KEY	MAKE	BREAK	KEY	MAKE	BREAK
A	1	1C	F0,1C	9	46	F0,46	[FO,54

' mouse up

BASCOM Language Reference

865

В	32	F0,32	`	0E	F0,0E	INSERT	E0,70	E0,F0,70
С	21	F0,21	-	4E	F0,4E	HOME	E0,6C	E0,F0,6C
D	23	F0,23	=	55	FO,55	PG UP	E0,7D	E0,F0,7D
Е	24	F0,24	١	5D	F0,5D	DELETE	E0,71	E0,F0,71
F	2B	F0,2B	BKSP	66	F0,66	END	E0,69	E0,F0,69
G	34	F0,34	SPACE	29	F0,29	PG DN	E0,7A	E0,F0,7A
Н	33	F0,33	ТАВ	0D	F0,0D	U ARROW	E0,75	E0,F0,75
I	43	F0,43	CAPS	58	F0,58	L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B	L SHFT	12	FO,12	D ARROW	E0,72	E0,F0,72
К	42	F0,42	L CTRL	14	FO,14	R ARROW	E0,74	E0,F0,74
L	4B	F0,4B	L GUI	E0,1F	E0, F0,1F	NUM	77	F0,77
М	3A	F0,3A	L ALT	11	F0,11	KP /	E0,4A	E0,F0,4A
N	31	F0,31	R SHFT	59	F0,59	KP *	7C	F0,7C
0	44	F0,44	R CTRL	E0,14	EO, F0,14	KP -	7B	F0,7B
Ρ	4D	F0,4D	R GUI	E0,27	E0, F0,27	KP +	79	F0,79
Q	15	F0,15	R ALT	E0,11	EO, F0,11	KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D	APPS	E0,2F	E0, F0,2F	KP.	71	F0,71
S	1B	F0,1B	ENTER	5A	F0,5A	KP 0	70	F0,70
Т	2C	F0,2C	ESC	76	F0,76	KP 1	69	F0,69
U	3C	F0,3C	F1	05	F0,05	KP 2	72	F0,72
V	2A	F0,2A	F2	06	F0,06	KP 3	7A	F0,7A
W	1D	F0,1D	F3	04	F0,04	KP 4	6B	F0,6B
Х	22	F0,22	F4	0C	F0,0C	KP 5	73	F0,73
Y	35	F0,35	F5	03	F0,03	KP 6	74	F0,74
Z	1A	F0,1A	F6	0B	F0,0B	KP 7	6C	F0,6C
0	45	F0,45	F7	83	F0,83	KP 8	75	F0,75
1	16	F0,16	F8	0A	F0,0A	KP 9	7D	F0,7D
2	1E	F0,1E	F9	01	F0,01]	5B	F0,5B
2 3 4	26	F0,26	F10	09	F0,09	•	4C	F0,4C
4	25	F0,25	F11	78	F0,78	-	52	F0,52
5 6	2E	F0,2E	F12	07	F0,07	/	41	F0,41
6	36	F0,36	PRNT	E0,12,	E0,F0,		49	F0,49
			SCRN	E0,7C	7C,E0, F0,12			
7	3D	F0,3D	SCROL L	7E	F0,7E	/	4A	F0,4A
8	3E	F0,3E	PAUSE	E1,14,7 7,	-NONE-			
				E1, F0,14,				



ACPI Scan Codes

Кеу	Make Code	Break Code
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

Windows Multimedia Scan Codes

Кеу	Make Code	Break Code
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20
WWW Favorites	E0, 18	E0, F0, 18

To emulate volume up, the data line would look like this:

DATA 5 , &HE0, &H32, &HE0, &HF0 , &H32 ^ send 5 bytes ^ volume up

See also

CONFIG ATEMU 455

```
_____
_____
                       : ps2_kbdemul.bas
'name
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : PS2 AT Keyboard emulator
'micro
                       : 90S2313
'suited for demo
                       : no, ADD ON NEEDED
'commercial addon needed : yes
·_____
_____
$regfile = "2313def.dat"
                                                       ' specify
the used micro
                                                       ' used
$crystal = 4000000
crystal frequency
$baud = 19200
                                                       ' use baud
rate
$hwstack = 32
                                                       ' default
use 32 for the hardware stack
\$swstack = 10
                                                       ' default
use 10 for the SW stack
$framesize = 40
                                                       ' default
use 40 for the frame space
$lib "mcsbyteint.lbx"
                                                       ' use
optional lib since we use only bytes
'configure PS2 AT pins
Enable Interrupts
                                                       ' you need
to turn on interrupts yourself since an INT is used
Config Atemu = Int1 , Data = Pind.3 , Clock = Pinb.0
                ^----- used interrupt
                            ^---- pin connected to DATA
^-- pin connected to clock
'Note that the DATA must be connected to the used interrupt pin
Waitms 500
                                                       ' optional
delay
'rcall _AT_KBD_INIT
Print "Press t for test, and set focus to the editor window"
Dim Key2 As Byte , Key As Byte
Do
   Key2 = Waitkey()
                                                       ' get key
from terminal
   Select Case Key2
     Case "t" :
     Waitms 1500
     Sendscankbd Mark
                                                       ' send a
scan code
     Case Else
   End Select
Loop
Print Hex(key)
                                                      ' send mark
Mark:
Data 12 , &H3A , &HF0 , &H3A , &H1C , &HF0 , &H1C , &H2D , &HF0 , &H2D ,
&H42 , &HF0 , &H42
```

868 BASCOM-AVR

```
' ^ send 12 bytes
' m a
```

r

6.369 SERIN

Action

Reads serial data from a dynamic software UART.

Syntax

SERIN var , bts , port , pin, baud , parity , dbits , sbits

Remarks

While the OPEN and CLOSE statements can be used for software UARTS, they do not permit to use the same pin for input and output. The settings used when opened the communication channel can also not be changed at run time.

The SERIN and SEROUT statements are dynamic software UART routines to perform input and output. You can use them on the same pin for example send some data with SEROUT and get back an answer using SERIN.

Since the SERIN and SEROUT routines can use any pin and can use different	
parameter values, the code size of these routines is larger.	

Paramete	Description
r	
Var	A variable that will be assigned with the received data.
Bts	The number of bytes to receive. String variables will wait for a return (ASCII 13). There is no check if the variable you assign is big enough to hold the result.
Port	The name of the port to use. This must be a letter like A for portA.
Pin	The pin number you want to use of the port. This must be in the range from 0-7.
Baud	The baud rate you want to use. For example 19200.
Parity	A number that codes the parity. $0 = NONE$, $1 = EVEN$, $2 = ODD$
Dbits	The number of data bits. Use 7 or 8.
Sbits	The number of stop bits. 1 to 2.

The use of SERIN will create an internal variable named ____SER_BAUD. This is a LONG variable. It is important that you specify the correct crystal value with \$CRYSTAL so the correct calculation can be made for the specified baud rate.

Note that ____SER_BAUD will not hold the passed baud rate but will hold the bit delay used internal.

Since the SW UART is dynamic you can change all the parameters at run time. For example you can store the baud rate in a variable and pass this variable to the SERIN routine.

Your code could change the baud rate under user control this way.

It is important to realize that software timing is used for the bit timing. Any interrupt that occurs during SERIN or SEROUT will delay the transmission. Disable interrupts while you use SERIN or SEROUT.

ASM

The routine called is named _serin and is stored in mcs.lib For the baud rate calculation, _calc_baud is called.

See also

SEROUT 870

```
_____
'name
                     : serin_out.bas
                     : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                      : demonstration of DYNAMIC software UART
                     : AT90S2313
'micro
'suited for demo
                      : yes
'commercial addon needed : no
·_____
_____
$regfile = "2313def.dat"
                                                    ' specify
the used micro
                                                    ' used
$crystal = 4000000
crystal frequency
$baud = 19200
                                                    ' use baud
rate
                                                    ' default
$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
                                                    ' default
use 10 for the SW stack
                                                    ' default
$framesize = 40
use 40 for the frame space
'tip : Also look at OPEN and CLOSE
'some variables we will use
Dim S As String * 10
Dim Mybaud As Long
'when you pass the baud rate with a variable, make sure you dimesion it
as a LONG
Mybaud = 19200
Do
 'first get some data
 Serin S , 0 , D , 0 , Mybaud , 0 , 8 , 1
 'now send it
 {\tt Serout}\ {\tt S} , 0 , D , 1 , Mybaud , 0 , 8 , 1
                                   ^ 1 stop bit
                               ^---- 8 data bits
                              ^----- even parity (0=N, 1 = E, 2=O)
                       ^----- baud rate
                     ----- pin number
               ^----- port so PORTA.0 and PORTA.1
are used
            ^----- for strings pass 0
         veriable
 .
 Wait 1
Loop
End
```

BASCOM-AVR

```
'because the baud rate is passed with a variable in this example, you could change it under user control 'for example check some DIP switches and change the variable mybaud
```

6.370 SEROUT

870

Action

Sends serial data through a dynamic software UART.

Syntax

SEROUT var , bts , port , pin, baud , parity , dbits , sbits

Remarks

While the OPEN and CLOSE statements can be used for software UARTS, they do not permit to use the same pin for input and output. The settings used when opened the communication channel can also not be changed at run time.

The SERIN and SEROUT statements are dynamic software UART routines to perform input and output. You can use them on the same pin for example send some data with SEROUT and get back an answer using SERIN.

Since the SERIN and SEROUT routines can use any pin and can use different parameter values, the code size of these routines is larger.

Parame	te Description
Var	A variable which content is send through the UART. A constant can NOT be used.
Bts	The number of bytes to receive. String variables will wait for a return (ASCII 13). There is no check if the variable you assign is big enough to hold the result.
Port	The name of the port to use. This must be a letter like A for portA.
Pin	The pin number you want to use of the port. This must be in the range from 0-7.
Baud	The baud rate you want to use. For example 19200.
Parity	A number that codes the parity. $0 = NONE$, $1 = EVEN$, $2 = ODD$
Dbits	The number of data bits. Use 7 or 8.
Sbits	The number of stop bits. 1 to 2.

The use of SEROUT will create an internal variable named ____SER_BAUD. This is a LONG variable. It is important that you specify the correct crystal value with \$CRYSTAL so the correct calculation can be made for the specified baud rate.

Note that ____SER_BAUD will not hold the passed baud rate but will hold the bit delay used internal.

Since the SW UART is dynamic you can change all the parameters at run time. For example you can store the baud rate in a variable and pass this variable to the SEROUT routine.

Your code could change the baud rate under user control this way.

It is important to realize that software timing is used for the bit timing. Any interrupt

that occurs during SERIN or SEROUT will delay the transmission. Disable interrupts while you use SERIN or SEROUT.

The SEROUT will use the pin in Open Collector mode. This means that you can connect several AVR chips and poll the `bus' with the SERIN statement.

ASM

The routine called is named _serout and is stored in mcs.lib For the baud rate calculation, _calc_baud is called.

See also

SERIN 868

```
_____
_____
'name
                       : serin_out.bas
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                      : demonstration of DYNAMIC software UART
                      : AT90S2313
'micro
'suited for demo : yes
'commercial addon needed : no
! _____
_____
$regfile = "2313def.dat"
                                                     ' specify
the used micro
$crystal = 4000000
                                                     ' used
crystal frequency
$baud = 19200
                                                     ' use baud
rate
$hwstack = 32
                                                     ' default
use 32 for the hardware stack
\$swstack = 10
                                                     ' default
use 10 for the SW stack
$framesize = 40
                                                     ' default
use 40 for the frame space
'tip : Also look at OPEN and CLOSE
'some variables we will use
Dim S As String * 10
Dim Mybaud As Long
'when you pass the baud rate with a variable, make sure you dimesion it
as a LONG
Mybaud = 19200
Do
 'first get some data
 Serin S , 0 , D , 0 , Mybaud , 0 , 8 , 1
 'now send it
 {\tt Serout}\ {\tt S} , 0 , D , 1 , Mybaud , 0 , 8 , 1
                                    ^ 1 stop bit
                                ^---- 8 data bits
                              ^----- even parity (0=N, 1 = E, 2=0)
                        ^----- baud rate
                  ^----- pin number
                ^----- port so PORTA.0 and PORTA.1
are used
```

```
872 BASCOM-AVR
```

```
' ^----- for strings pass 0
' ^----- variable
Wait 1
Loop
End
'because the baud rate is passed with a variable in this example, you
could change it under user control
'for example check some DIP switches and change the variable mybaud
```

6.371 SETIPPROTOCOL

Action

Configures socket RAW-mode protocol

Syntax

SETIPPROTOCOL socket, value

Remarks

Socket	The socket number. (0-3)
Value	The IP-protocol value to set.

In order to use W3100A's IPL_RAW Mode, the protocol value of the IP Layer to be used (e.g., 01 in case

of ICMP) needs to be set before socket initialization.

As in UDP, data transmission and reception is possible when the corresponding channel is initialized.

The PING example demonstrates the usage. As a first step, SETIPPROTOCOL is used : Setipprotocol Idx , **1** And second, the socket is initialized : Idx = Getsocket(idx , **3**, 5000, 0)

The W3100A data sheet does not provide much more details about the IPR register.

See also

SETTCPREGS 860, GETSOCKET 717

ASM

NONE

'							
'name	:	PING_TWI.bas	http://www	.faqs.org	g/rf	cs/rf	c792
'copyright	:	(c) 1995-2005, MCS	Electronics				
'purpose	:	Simple PING progra	ım				
'micro	:	Mega88					
'suited for demo	:	yes					
'commercial addon needed	:	no					
<pre>\$regfile = "m32def.dat"</pre>			1	specify	the	used	mic

873

```
$crystal = 8000000
                                                              ' used crystal frequence
                                                              ' use baud rate
$baud = 19200
                                                              ' default use 32 for th
$hwstack = 80
                                                              ' default use 10 for th
\$swstack = 128
                                                              ' default use 40 for th
$framesize = 80
Const Debug = 1
Const Sock_stream = $01
                                                              ' Tcp
                                                              ' Udp
Const Sock_dgram = $02
Const Sock_ipl_raw = $03
                                                              ' Ip Layer Raw Sock
                                                              ' Mac Layer Raw Sock
Const Sock_macl_raw = $04
Const Sel_control = 0
                                                              ' Confirm Socket Status
Const Sel_send = 1
                                                              ' Confirm Tx Free Buffe
Const Sel_recv = 2
                                                              ' Confirm Rx Data Size
'socket status
                                                              ' Status Of Connection
Const Sock closed = $00
Const Sock arp = $01
                                                              ' Status Of Arp
                                                              ' Status Of Waiting For
Const Sock_listen = $02
                                                              ' Status Of Setting Up
Const Sock_synsent = $03
                                                              ' Status Of Setting Up
Const Sock_synsent_ack = $04
                                                              ' Status Of Setting Up
Const Sock_synrecv = $05
                                                              ' Status Of Tcp Connect
Const Sock_established = $06
                                                              ' Status Of Closing Tcp
Const Sock_close_wait = $07
Const Sock_last_ack = $08
                                                              ' Status Of Closing Tcp
Const Sock_fin_wait1 = $09
                                                              ' Status Of Closing Tcp
Const Sock_fin_wait2 = $0a
                                                              ' Status Of Closing Tcp
Const Sock_closing = $0b
                                                              ' Status Of Closing Tcp
Const Sock_time_wait = $0c
                                                              ' Status Of Closing Tcp
                                                              ' Status Of Closing Tcp
Const Sock_reset = $0d
Const Sock_init = $0e
                                                              ' Status Of Socket Init
Const Sock_udp = $0f
                                                              ' Status Of Udp
Const Sock_raw = $10
                                                              ' Status of IP RAW
'we do the usual
Print "Init TCP"
                                                              ' display a message
Enable Interrupts
                                                              ' before we use config
Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.2
Print "Init done"
Dim Peersize As Integer , Peeraddress As Long , Peerport As Word
Dim Idx As Byte , Result As Word , J As Byte , Res As Byte
Dim Ip As Long
Dim Dta(12) As Byte , Rec(12) As Byte
Dta(1) = 8
                                                              'type is echo
Dta(2) = 0
                                                              'code
Dta(3) = 0
                                                              ' for checksum initial:
Dta(4) = 0
                                                              ' checksum
                                                              ' a signature can be an
Dta(5) = 0
Dta(6) = 1
                                                                 signature
                                                              ' sequence number - any
Dta(7) = 0
Dta(8) = 1
Dta(9) = 65
Dim W As Word At Dta + 2 Overlay
                                                              'same as dta(3) and dta
W = Tcpchecksum(dta(1) , 9)
                                                              ' calculate checksum an
#if Debug
```

```
For J = 1 To 9
   Print Dta(j)
 Next
#endif
Ip = Maketcp(192.168.0.16)
                                                             'try to check this serv
Print "Socket " ; Idx ; " " ; Idx
Setipprotocol Idx , 1
                                                             'set protocol to 1
'the protocol value must be set BEFORE the socket is openend
Idx = Getsocket(idx , 3 , 5000 , 0)
Do
   Result = Udpwrite(ip , 7 , Idx , Dta(1) , 9)
                                                             'write ping data
   Print Result
   Waitms 100
   Result = Socketstat(idx , Sel_recv)
                                                             'check for data
   Print Result
   If Result >= 11 Then
      Print "Ok"
      Res = Tcpread(idx , Rec(1) , Result)
                                                             'get data with TCPREAD
      #if Debug
                                                             .
        Print "DATA RETURNED :" ; Res
        For J = 1 To Result
         Print Rec(j) ; " " ;
        Next
        Print
      #endif
   Else
                                                             'there might be a probl
      Print "Network not available"
   End If
   Waitms 1000
Loop
```

6.372 SGN

Action

Returns the sign of a float value.

Syntax

var = **SGN**(x)

Remarks

Var	A single or double variable that is assigned with the SGNS of variable x.
Х	The single or double to get the sign of.

For values <0, -1 will be returned For 0, 0 will be returned For values >0, 1 will be returned

See Also

INT 748, FIX 688, ROUND 847

```
Example
```

```
Dim S As Single , X As Single , Y As Single
X = 2.3 : S = Sgn(x)
Print S
X = -2.3 : S = Sgn(x)
Print S
End
```

6.373 SHIFT

Action

Shift all bits one place to the left or right.

Syntax

SHIFT var , LEFT/RIGHT[, shifts] [,SIGNED]

Remarks

Var	Byte, Integer/Word, Long or Single variable.			
Shifts	The number of shifts to perform.			
signed	An option that only works with right shifts. It will preserve the sign bit which otherwise would be cleared by the first shift.			

The SHIFT statement rotates all the bits in the variable to the left or right.

When shifting LEFT the most significant bit, will be shifted out of the variable. The LS bit becomes zero. Shifting a variable to the left, multiplies the variable with a value of two.

When shifting to the RIGHT, the least significant bit will be shifted out of the variable. The MS bit becomes zero. Shifting a variable to the right, divides the variable by two. Use the SIGNED parameter to preserve the sign.

A Shift performs faster than a multiplication or division.

See also

ROTATE 8467, SHIFTIN 8777, SHIFTOUT 8877

```
_____
'name
                   : shift.bas
                   : (c) 1995-2005, MCS Electronics
'copyright
                   : example for SHIFTIN and SHIFTOUT statement
'purpose
'micro
                   : Mega48
'suited for demo
                   : yes
'commercial addon needed : no
_____
                               _____
 _____
$regfile = "m48def.dat"
                                              ' specify
the used micro
```

876 BASCOM-AVR

```
$crystal = 4000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
\$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Dim L As Long
Clock Alias Portb.0
Output Alias Portb.1
Sin Alias Pinb.2
                                                             'watch the
PIN instead of PORT
'shiftout pinout, pinclock, var, parameter [, bits , delay]
' value for parameter :
   0 - MSB first ,clock low
   1 - MSB first, clock high
   2 - LSB first, clock low
  3 - LSB first, clock high
'The bits is a new option to indicate the number of bits to shift out
'For a byte you should specify 1-8 , for an integer 1-16 and for a long
1-32
'The delay is an optional delay is uS and when used, the bits parameter
must
'be specified too!
'Now shift out 9 most significant bits of the LONG variable L
Shiftout Output , Clock , L , 0 , 9
'shiftin pinin,pinclock,var,parameter [,bits ,delay]
 0 - MSB first ,clock low (4)
1
  1 - MSB first, clock high (5)
   2 - LSB first, clock low
                             (6)
   3 - LSB first, clock high (7)
'To use an external clock, add 4 to the parameter
'The shiftin also has a new optional parameter to specify the number of
bits
'The bits is a new option to indicate the number of bits to shift out
'For a byte you should specify 1-8, for an integer 1-16 and for a long
1-32
'The delay is an optional delay is uS and when used, the bits parameter
must
'be specified too!
'Shift in 9 bits into a long
Shiftin Sin , Clock , L , 0 , 9
'use shift to shift the bits to the right place in the long
Shift L , Right , 23
End
```

6.374 SHIFTCURSOR

Action

Shift the cursor of the LCD display left or right by one position.

Syntax

SHIFTCURSOR LEFT | RIGHT

See also

SHIFTLCD 882

Partial Example

LCD "Hello" SHIFTCURSOR LEFT End

6.375 SHIFTIN

Action

Shifts a bit stream into a variable.

Syntax

SHIFTIN pin , pclock , var , option [, bits , delay]

Remarks

Pin	The port pin which serves as an input.PINB.2 for example
Pclock	The port pin which generates the clock.
Var	The variable that is assigned. The existing value is not preserved. For example when you shiftin 3 bits, the whole byte will be replaced with the 3 bits. See CONFIG SHIFTIN for other SHIFTIN behaviour.
Option	 Option can be : 0 - MSB shifted in first when clock goes low 1 - MSB shifted in first when clock goes high 2 - LSB shifted in first when clock goes low 3 - LSB shifted in first when clock goes high Adding 4 to the parameter indicates that an external clock signal is used for the clock. In this case the clock will not be generated. So using 4 will be the same a 0 (MSB shifted in first when clock goes low) but the clock must be generated by an external signal. 4 - MSB shifted in first when clock goes low with ext. clock 5 - MSB shifted in first when clock goes low with ext. clock 6 - LSB shifted in first when clock goes low with ext. clock 7 - LSB shifted in first when clock goes low with ext. clock
Bits	Optional number of bits to shift in. Maximum 255. The number of bits is automatic loaded depending on the used variable. For a long for example

	which is 4 bytes long, 32 will be loaded.
Delay	Optional delay in uS.

If you do not specify the number of bits to shift, the number of shifts will depend on the type of the variable.

When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur. For a Long and Single 32 shifts will occur.

The SHIFTIN routine can be used to interface with all kind of chips. The PIN is normally connected with the output of chip that will send information.

The PCLOCK pin can be used to clock the bits as a master, that is the clock pulses will be generated. Or it can sample a pin that generates these pulses.

The VARIABLE is a normal BASIC variable. And may be of any type except for BIT. The data read from the chip is stored in this variable.

The OPTIONS is a constant that specifies the direction of the bits. The chip that outputs the data may send the LS bit first or the MS bit first. It also controls on which edge of the clock signal the data must be stored.

When you add 4 to the constant you tell the compiler that the clock signal is not generated but that there is an external clock signal.

The number of bits may be specified. You may omit this info. In that case the number of bits of the element data type will be used.

The DELAY normally consists of 2 NOP instructions. When the clock is too fast you can specify a delay time(in uS).

SHIFTIN with option NEW

The new option <u>CONFIG SHIFTIN</u> 548 = NEW , will change the behaviour of the SHIFTIN statement.

When using this option, it will work for all SHIFTIN statements. The SHIFTIN will work more like the normal SHIFT statement. Bits are shifted from left to right or right to left.

The new SHIFTIN can preserve the value/bits when shifting in bits.

For example when the value of a word is &B101 and you shift in 3 bits with value &B111, the resulting value will be &B101**111**. When you **not** want to preserve the value, you can add a value of **8** to the parameter. When you add a value of **16**, the value will also not be preserved, but then the value will be cleared initially. You would only need this when shifting in less 8 bits then the size of the variable.

Another important difference is that the new SHIFTIN can only SHIFTIN a maximum of 8 bytes. For quick operation, register R16-R23 are used. You may specify the number of bits to shiftin. This may be a variable too. When you shiftin a value into a Word, the number of bits is automatic loaded with 16. This is true for all numeric data types.

Some of the code is stored in the MCS library. While this reduces code when SHIFTIN is used multiple times, it has the drawback that the code is written for 8 bytes and thus is not optimal for shifting in less bytes.

You can choose to generate a part of the library code instead. Add a value of 32 to the parameter to do so.

Another new option is not to set the initial pin state for the clock and input pin. By default the clock pin is made an input or output, depending on the external clock option. And the clock is set to an initial state when no external clock is used. When you want to use shiftin after a shiftout, you might not want the level to change.

In this case, add 64 to the parameter.

Pin	The port pin which serves as an input.PINB.2 for example		
Pclock	The port pin which generates the clock. An external signal can also be used for the clock. In that case, the pin is used in input mode.		
Var	The variable that is assigned. The existing value is preserved. With some additional constants which you can add to the option parameter, you can influence the behaviour : - 8 - Do NOT preserve the value. This saves code. -16 - Do not preserve value, but clear the value before shifting in the bits		
Option	A constant which can be one of the following values :		
	0 – MS bit shifted in first when clock goes low 1 – MS bit shifted in first when clock goes high 2 – LS bit shifted in first when clock goes low 3 – LS bit shifted in first when clock goes high		
	Adding 4 to the parameter indicates that an external clock signal is used for the clock. In this case the clock will not be generated. So using 4 will be the same a 0 (MSB shifted in first when clock goes low) but the clock must be generated by an external signal.		
	 4 - MSB shifted in first when clock goes high with ext. clock 5 - MSB shifted in first when clock goes low with ext. clock 6 - LSB shifted in first when clock goes high with ext. clock 7 - LSB shifted in first when clock goes low with ext. clock 		
	Add a value of 8 to the option, so the existing variable will not be preserved.		
	Add a value of 16 to the option to clear the variable first. Add a value of 32 to the option to generate code instead of using the lib code.		
	Add a value of 64 to the option when you do not want the clock and input pin data direction and state want to be set. For example, when using SHIFTIN after a SHIFTOUT statement.		
	Example : Shiftin Pind.3 , Portd.4 , W , 2 + 32 + 16 , 3		
Bits	Optional number of bits to shift in. Maximum 64. The number of bits is automatic loaded depending on the used variable. For a long for example which is 4 bytes long, 32 will be loaded. You can use a constant or variable.		
Delay	Optional delay in uS. When not specified, 2 nops are used. The delay is intended to slow down the clock frequency.		

The initial state for the clock depends on the option. For option 1 and 3, it will be low. For option 0 and 2 it will be high.

Thus for example option 2 will set the clock pin high. Then the clock is brought low and the data is sampled/stored. After this the clock is made high again. This means when ready, the clock pin will be in the same state as the initial state.

See also

SHIFTOUT 881, SHIFT 875

```
-----
_____
                        : shift.bas
'name
                        : (c) 1995-2005, MCS Electronics
'copyright
                        : example for SHIFTIN and SHIFTOUT statement
'purpose
'micro
                        : Mega48
'suited for demo
                        : yes
'commercial addon needed : no
·_____
_____
$regfile = "m48def.dat"
                                                        ' specify
the used micro
                                                        ' used
$crystal = 4000000
crystal frequency
$baud = 19200
                                                        ' use baud
rate
$hwstack = 32
                                                        ' default
use 32 for the hardware stack
\$swstack = 10
                                                        ' default
use 10 for the SW stack
$framesize = 40
                                                        ' default
use 40 for the frame space
Dim L As Long
clock Alias Portb.0
Output Alias Portb.1
sinp Alias Pinb.2
                                                         'watch the
PIN instead of PORT
'shiftout pinout, pinclock, var, parameter [, bits , delay]
' value for parameter :
  0 - MSB first ,clock low
 1 - MSB first,clock high
 2 - LSB first, clock low
' 3 - LSB first, clock high
'The bits is a new option to indicate the number of bits to shift out
'For a byte you should specify 1-8 , for an integer 1-16 and for a long
1 - 32
'The delay is an optional delay is uS and when used, the bits parameter
must
'be specified too!
'Now shift out 9 most significant bits of the LONG variable L
Shiftout Output , Clock , L , 0 , 9
'shiftin pinin,pinclock,var,parameter [,bits ,delay]
' 0 - MSB first ,clock low (4)
  1 - MSB first, clock high (5)
  2 - LSB first,clock low
                           (6)
  3 - LSB first, clock high (7)
'To use an external clock, add 4 to the parameter
'The shiftin also has a new optional parameter to specify the number of
bits
'The bits is a new option to indicate the number of bits to shift out
```

```
881
```

```
'For a byte you should specify 1-8 , for an integer 1-16 and for a long 1-32
'The delay is an optional delay is uS and when used, the bits parameter must
'be specified too!
```

```
'Shift in 9 bits into a long
Shiftin Sinp , Clock , L , 0 , 9
'use shift to shift the bits to the right place in the long
Shift L , Right , 23
End
```

6.376 SHIFTOUT

Action

Shifts a bit stream out of a variable into a port pin .

Syntax

SHIFTOUT pin , pclock , var , option [, bits , delay]

Remarks

Pin	The port pin which serves as a data output.		
Pclock	The port pin which generates the clock.		
Var	The variable that is shifted out.		
Option	Option can be :		
	0 – MSB shifted out first when clock goes low 1 – MSB shifted out first when clock goes high 2 – LSB shifted out first when clock goes low 3 – LSB shifted out first when clock goes high		
Bits	Optional number of bits to shift out.		
Delay	Optional delay in uS. When you specify the delay, the number of bits must also be specified. When the default must be used you can also use NULL for the number of bits.		

If you do not specify the number of bits to shift, the number of shifts will depend on the type of the variable.

When you use a byte, 8 shifts will occur and for an integer, 16 shifts will occur. For a Long and Single 32 shifts will occur.

The SHIFTIN routine can be used to interface with all kind of chips.

The PIN is normally connected with the input of a chip that will receive information.

The PCLOCK pin is used to clock the bits out of the chip.

The VARIABLE is a normal BASIC variable. And may be of any type except for BIT. The data that is stored in the variable is sent with PIN.

The OPTIONS is a constant that specifies the direction of the bits. The chip that reads the data may want the LS bit first or the MS bit first. It also controls on which edge of the clock signal the data is sent to PIN.

The number of bits may be specified. You may omit this info. In that case the number

882 BASCOM-AVR

of bits of the element data type will be used.

The DELAY normally consists of 2 NOP instructions. When the clock is too fast you can specify a delay time(in uS).

The clock pin is brought to a initial level before the shifts take place. For mode 0, it is made 1. This way, the first clock can go from 1 to 0. And back to 1. You could see this as another clock cycle. So check if you use the proper mode. Or put the clock pin in the right state before you use SHIFT.

See also SHIFTIN 877, SHIFT 875

Example

See <u>SHIFTIN</u> 877 sample

6.377 SHIFTLCD

Action

Shift the LCD display left or right by one position.

Syntax

SHIFTLCD LEFT / RIGHT

Remarks

NONE

See also

SHIFTCURSOR (877), SHIFTCURSOR (877), INITLCD (738), CURSOR (609)

Partial Example

Cls	'clear the
LCD display	
Lcd "Hello world."	'display
this at the top line	
Wait 1	
Lowerline	'select the
lower line	
Wait 1	
Lcd "Shift this."	'display
this at the lower line	
Wait 1	
For $A = 1$ To 10	
Shiftlcd Right	'shift the
text to the right	
Wait 1	'wait a
moment	
Next	

'shift the

'set cursor

'display

'wait a

'shift the

'display

'wait a

For A = 1 To 10
 Shiftlcd Left
text to the left
 Wait 1
moment
Next
Locate 2 , 1
position
Lcd "*"
this
Wait 1
moment
Shiftcursor Right
cursor
Lcd "@"

6.378 SHOWPIC

this

Action

Shows a BGF file on the graphic display

Syntax

SHOWPIC x, y , label

Remarks

Showpic can display a converted BMP file. The BMP must be converted into a BGF file with the Tools Graphic Converter [82].

The X and Y parameters specify where the picture must be displayed. X and Y must be 0 or a multiple of 8. The picture height and width must also be a multiple of 8.

The label tells the compiler where the graphic data is located. It points to a label where you put the graphic data with the \$BGF directive.

You can store multiple pictures when you use multiple labels and \$BGF directives,

Note that the BGF files are RLE encoded to save code space.

See also

PSET [813] , \$BGF [312] , CONFIG GRAPHLCD [506] , LINE [759] , CIRCLE [429] , SHOWPICE [883]

Example

See <u>\$BGF</u> 312 example

6.379 SHOWPICE

Action

Shows a BGF file stored in EEPROM on the graphic display

Syntax

SHOWPICE x, y , label

Remarks

Showpice can display a converted BMP file that is stored in the EEPROM of the micro processor. The BMP must be converted into a BGF file with the <u>Tools Graphic</u> Converter B2.

The X and Y parameters specify where the picture must be displayed. X and Y must be 0 or a multiple of 8. The picture height and width must also be a multiple of 8.

The label tells the compiler where the graphic data is located. It points to a label where you put the graphic data with the \$BGF directive. You can store multiple pictures when you use multiple labels and \$BGF directives,

Note that the BGF files are RLE encoded to save code space.

See also

PSET [813], \$BGF [312], CONFIG GRAPHLCD [506], LINE [759], SHOWPIC [883], CIRCLE [429]

Example

_____ : showpice.bas 'name : (c) 1995-2005, MCS Electronics 'copyright : demonstrates showing a picture from EEPROM 'purpose : AT90S8535 'micro 'suited for demo : yes 'commercial addon needed : no 1_____ _____ \$regfile = "8535def.dat" ' specify the used micro **\$crystal** = 8000000 ' used crystal frequency **\$baud** = 19200 ' use baud rate \$hwstack = 32' default use 32 for the hardware stack \$swstack = 10' default use 10 for the SW stack **\$framesize** = 40 ' default use 40 for the frame space 'First we define that we use a graphic LCD ' Only 240*64 supported yet **Config** Graphlcd = 240 * 128 , Dataport = Porta , Controlport = Portc , Ce = 2, Cd = 3, Wr = 0, Rd = 1, Reset = 4, Fs = 5, Mode = 8 'The dataport is the portname that is connected to the data lines of the LCD 'The controlport is the portname which pins are used to control the lcd 'CE, CD etc. are the pin number of the CONTROLPORT. ' For example CE =2 because it is connected to PORTC.2 'mode 8 gives 240 / 8 = 30 columns , mode=6 gives 240 / 6 = 40 columns 'we will load the picture data into EEPROM so we specify \$EEPROM 'the data must be specified before the showpicE statement.

```
$eeprom
Plaatje:
'the $BGF directive will load the data into the EEPROM or FLASH
depending on the $EEPROM or $DATA directive
$bgf "mcs.bgf"
'switch back to normal DATA (flash) mode
$data
'Clear the screen will both clear text and graph display
Cls
'showpicE is used to show a picture from EEPROM
'showpic must be used when the data is located in Flash
Showpice 0 , 0 , Plaatje
End
```

6.380 SIN

Action

Returns the sine of a float

Syntax

var = **SIN**(source)

Remarks

Var	A numeric variable that is assigned with sinus of variable source.
source	The single or double variable to get the sinus of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD 649, ATN 403, COS 596

```
$regfile = "m48def.dat"
                                                              ' specify
the used micro
                                                              ' used
$crystal = 8000000
crystal frequency
Sbaud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As Single , X As Single
S = 0.5 : X = Tan(s) : Print X
                                                              ' prints
0.546302195
S = 0.5 : X = Sin(s) : Print X
                                                              ' prints
```

```
© 2011 MCS Electronics
```

BASCOM-AVR

```
0.479419108
S = 0.5 : X = Cos(s) : Print X
0.877588389
End
```

' prints

6.381 SINH

886

Action

Returns the sinus hyperbole of a float

Syntax

var = **SINH**(source)

Remarks

	A numeric variable that is assigned with sinus hyperbole of variable source.
source	The single or double variable to get the sinus hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD [649], ATN [403], COS [596], SIN [885], TANH [929], COSH [597]

Example

Show sample 983

6.382 SOCKETCONNECT

Action

Establishes a connection to a TCP/IP server.

Syntax

Result = **SOCKETCONNECT**(socket, IP, port)

Remarks

A byte that is assigned with 0 when the connection succeeded. It will return 1 when an error occurred.
The IP number of the server you want to connect to. This may be a number like 192.168.0.2 or a LONG variable that is assigned
Note that the LSB of the LONG, must contain the MSB of the IP number.
 The port number of the server you are connecting to.

You can only connect to a server. Standardized servers have dedicated port numbers. For example, the HTTP protocol(web server) uses port 80.

After you have established a connection the server might send data. This depends entirely on the used protocol. Most servers will send some welcome text, this is called a banner.

You can send or receive data once the connection is established.

The server might close the connection after this or you can close the connection yourself. This also depends on the protocol.

See also

CONFIG TCPIP [559], GETSOCKET [717], SOCKETSTAT [890], TCPWRITE [924], TCPWRITESTR [925], TCPREAD [924], CLOSESOCKET [439], SOCKETLISTEN [889]

· '		
'name 'copyright 'purpose is programmed ' 'micro 'suited for demo 'commercial addon needed	<pre>: servertest.bas : (c) 1995-2005, MCS Electronics : start the easytcp.exe program af and create 2 connections : Mega161 : no : yes</pre>	
<pre>\$regfile = "ml6ldef.dat" the used micro \$crystal = 4000000 crystal frequency the used</pre>		' specify ' used
<pre>\$baud = 19200 rate \$hwstack = 32 use 32 for the hardware st \$swstack = 10 use 10 for the SW stack</pre>	tack	' use baud ' default ' default
\$framesize = 40 use 40 for the frame space	2	' default
<pre>Const Sock_stream = \$01 Const Sock_dgram = \$02 Const Sock_ipl_raw = \$03 Raw Sock</pre>		' Tcp ' Udp ' Ip Layer
<pre>Const Sock_macl_raw = \$04 Raw Sock Const Sel_control = 0 Const Sel_control = 0</pre>		' Mac Layer ' Confirm
Socket Status Const Sel_send = 1 Free Buffer Size Const Sel_recv = 2		' Confirm Tx ' Confirm Rx
Data Size 'socket status		L Status of
<pre>Const Sock_closed = \$00 Connection Closed Const Sock_arp = \$01 Arp</pre>		' Status Of ' Status Of
Const Sock_listen = \$02		' Status Of

BASCOM-AVR

Waiting For Tcp Connection Setup			
Const Sock_synsent = \$03	÷.	Status	Of
Setting Up Tcp Connection			
Const Sock_synsent_ack = \$04	1	Status	Of
Setting Up Tcp Connection			
Const Sock_synrecv = \$05	1	Status	Of
Setting Up Tcp Connection			
Const Sock_established = \$06	1	Status	Of
Tcp Connection Established			
Const Sock_close_wait = \$07	1	Status	Of
Closing Tcp Connection			
<pre>Const Sock_last_ack = \$08</pre>	1	Status	Of
Closing Tcp Connection			
<pre>Const Sock_fin_wait1 = \$09</pre>	1	Status	Of
Closing Tcp Connection			
Const Sock_fin_wait2 = \$0a	1	Status	Of
Closing Tcp Connection			
Const Sock_closing = \$0b	1	Status	Of
Closing Tcp Connection			
Const Sock_time_wait = \$0c	1	Status	Of
Closing Tcp Connection			
Const Sock_reset = \$0d	1	Status	Of
Closing Tcp Connection			
Const Sock_init = \$0e	1	Status	Of
Socket Initialization			
Const Sock_udp = \$0f	1	Status	Of
Udp			
Const Sock_raw = \$10	1	Status	of
IP RAW			

\$lib "topip.lbx" ' specify
the topip library
Print "Init , set IP to 192.168.0.8" ' display a
message
Enable Interrupts ' before we
use config topip , we need to enable the interrupts
Config Topip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 ,
Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx =
\$55 , Rx = \$55

'Use the line below if you have a gate way 'Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx = \$55 , Rx = \$55

Dim Bclient As Byte number Dim Idx As Byte Dim Result As Word Dim S As String * 80 Dim Flags As Byte Dim Peer As Long

Do For Idx = 0 To 3 Result = Socketstat(idx , 0) ' get status Select Case Result Case Sock_established If Flags.idx = 0 Then ' if we did not send a welcome message yet Flags.idx = 1

888

' socket

' result

889

```
Result = Tcpwrite(idx , "Hello from W3100A{013}{010}")
   ' send welcome
            End If
                                                             ' get number
            Result = Socketstat(idx , Sel_recv)
of bytes waiting
            If Result > 0 Then
               Do
                 Result = Tcpread(idx , S)
                 Print "Data from client: " ; Idx ; " " ; S
                 Peer = Getdstip(idx)
                 Print "Peer IP " ; Ip2str(peer)
                 'you could analyse the string here and send an
appropiate command
                 'only exit is recognized
                 If Lcase(s) = "exit" Then
                    Closesocket Idx
                 Elseif Lcase(s) = "time" Then
                    Result = Tcpwrite(idx , "12:00:00{013}{010}")
' you should send date$ or time$
                 End If
               Loop Until Result = 0
            End If
       Case Sock_close_wait
            Print "close_wait"
            Closesocket Idx
       Case Sock_closed
            Print "closed"
            Bclient = Getsocket(idx , Sock_stream , 5000 , 0)
get socket for server mode, specify port 5000
            Print "Socket " ; Idx ; " " ; Bclient
            Socketlisten Idx
            Print "Result " ; Result
            Flags.idx = 0
                                                             ' reset the
hello message flag
    End Select
  Next
Loop
End
```

6.383 SOCKETLISTEN

Action

Opens a socket in server(listen) mode.

Syntax

SOCKETLISTEN socket

Remarks

Socket The socket number you want to use for the server in the range of 0 -3.

The socket will listen to the port you specified with the GetSocket function. You can listen to a maximum of 4 sockets at the same time.

After the connection is closed by either the client or the server, a new connection need to be created and the SocketListen statement must be used again.

See also

CONFIG TCPIP [559], GETSOCKET [717], SOCKETCONNECT [886], SOCKETSTAT [890], TCPWRITE [924], TCPWRITESTR [925], TCPREAD [924], CLOSESOCKET [439]

Example

See <u>SOCKETCONNECT</u> and example

6.384 SOCKETSTAT

Action

Returns information of a socket.

Syntax

Result = **SOCKETSTAT**(socket , mode)

Remarks

Result	A word variable that is assigned with the result.	
Socket	The socket number you want to get information of	
Mode	A parameter that specified what kind of information you want to retrieve.	
	SEL_CONTROL or 0 : returns the status register value	
	SEL_SEND or 1 : returns the number of bytes that might be placed int the transmission buffer.	
	SEL_RECV or 2 : returns the number of bytes that are stored in the reception buffer.	

The SocketStat function contains actual 3 functions. One to get the status of the connection, one to determine how many bytes you might write to the socket, and one to determine how many bytes you can read from the buffer.

When you specify mode 0, one of the following byte values will be returned:

Value	State	Description
0	SOCK_CLOSED	Connection closed
1	SOCK_ARP	Standing by for reply after transmitting ARP request
2	SOCK_LISTEN	Standing by for connection setup to the client when acting in passive mode
3	SOCK_SYNSENT	Standing by for SYN,ACK after transmitting SYN for connecting setup when acting in active mode
4	SOCK_SYNSENT_ACK	Connection setup is complete after SYN,ACK is received and ACK is transmitted in active mode
5	SOCK_SYNRECV	SYN,ACK is being transmitted after receiving SYN from the client in listen state, passive mode
6	SOCK_ESTABLISHED	Connection setup is complete in active, passive mode

BASCOM Language Reference

891

7	SOCK_CLOSE_WAIT	Connection being terminated
8	SOCK_LAST_ACK	Connection being terminated
9	SOCK_FIN_WAIT1	Connection being terminated
10	SOCK_FIN_WAIT2	Connection being terminated
11	SOCK_CLOSING	Connection being terminated
12	SOCK_TIME_WAIT	Connection being terminated
13	SOCK_RESET	Connection being terminated after receiving reset packet from peer.
14	SOCK_INIT	Socket initializing
15	SOCK_UDP	Applicable channel is initialized in UDP mode.
16	SOCK_RAW	Applicable channel is initialized in IP layer RAW mode
17	SOCK_UDP_ARP	Standing by for reply after transmitting ARP request packet to the destination for UDP transmission
18	SOCK_UDP_DATA	Data transmission in progress in UDP RAW mode
19	SOCK_RAW_INIT	W3100A initialized in MAC layer RAW mode

The SocketStat function is also used internally by the library.

See also

CONFIG TCPIP 559, GETSOCKET 717, SOCKETCONNECT 886, TCPWRITE 924, TCPWRITESTR 925, TCPREAD 924, CLOSESOCKET 439, SOCKETLISTEN 889

Partial Example

Tempw = Socketstat(i , 0)' get status Select Case Tempw Case Sock_established Case Else End Select

6.385 SONYSEND

Action

Sends Sony remote IR code.

Syntax

SONYSEND address [, bits]

Uses

TIMER1

Remarks

Address	The address of the Sony device.		
bits	This is an optional parameter. When used, it must be 12, 15 or 20.		

Also, when you use this option, the address variable must be of the type LONG.

SONY CD Infrared Remote Control codes (RM-DX55)

Function	Hex	Bin
Power	A91	1010 1001 0001
Play	4D1	0100 1101 0001
Stop	1D1	0001 1101 0001
Pause	9D1	1001 1101 0001
Continue	B91	1011 1001 0001
Shuffle	AD1	1010 1101 0001
Program	F91	1111 1001 0001
Disc	531	0101 0011 0001
1	011	0000 0001 0001
2	811	1000 0001 0001
3	411	0100 0001 0001
4	C11	1100 0001 0001
5	211	0010 0001 0001
6	A11	1010 0001 0001
7	611	0110 0001 0001
8	E11	1110 0001 0001
9	111	0001 0001 0001
0	051	0000 0101 0001
>10	E51	1110 0101 0001
enter	D11	1101 0001 0001
clear	F11	1111 0001 0001
repeat	351	0011 0101 0001
disc -	BD1	1011 1101 0001
disc +	H7D1	0111 1101 0001
<<	0D1	0000 1101 0001
>>	8D1	1000 1101 0001
<<	CD1	1100 1101 0001
>>	2D1	0010 1101 0001
SONY Cassette	RM-J901)	
Deck A		
stop	1C1	0001 1100 0001
play >	4C1	0100 1100 0001
play <	EC1	1110 1100 0001
>>	2C1	0010 1100 0001
<<	CC1	1100 1100 0001
record	6C1	0110 1100 0001
pause	9C1	1001 1100 0001
Dec B		
stop	18E	0001 1000 1110
play >	58E	0101 1000 1110
play <	04E	0000 0100 1110

BASCOM Language Reference

893

>>	38E	0011 1000 1110
<<	D8E	1101 1000 1110
record	78E	0111 1000 1110
pause	98E	1001 1000 1110

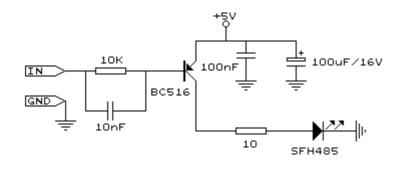
---[SONY TV Infrared Remote Control codes (RM-694)]-----

```
program + = &H090 : 0000 1001 0000
program - = &H890 : 1000 1001 0000
volume + = &H490 : 0100 1001 0000
volume - = &HC90 : 1100 1001 0000
power = &HA90 : 1010 1001 0000
sound on/off = &H290 : 0010 1001 0000
1 = \&H010 : 0000 \ 0001 \ 0000
2 = &H810 : 1000 0001 0000
3 = &H410 : 0100 0001 0000
4 = &HC10 : 1100 0001 0000
5 = &H210 : 0010 0001 0000
6 = &HA10 : 1010 0001 0000
7 = &H610 : 0110 0001 0000
8 = &HE10 : 1110 0001 0000
9 = &H110 : 0001 0001 0000
0 =  H910 : 1001 0001 0000
-/-- = &HB90 : 1011 1001 0000
```

For more SONY Remote Control info: http://www.fet.uni-hannover.de/purnhage/

The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A. Look in a data sheet for the proper pin when used with a different chip.

An IR booster circuit is shown below:



See also

CONFIG RC5 538 , GETRC5 712 , RC5SEND 822 , RC6SEND 826

Example

```
894 BASCOM-AVR
```

```
: sonysend.bas
'name
                        : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                        : code based on application note from Ger
Langezaal
'micro
                        : AT90S2313
'suited for demo
                        : yes
'commercial addon needed : no
·-----
                                      _____
------
$regfile = "2313def.dat"
                                                         ' specify
the used micro
                                                         ' used
$crystal = 4000000
crystal frequency
$baud = 19200
                                                         ' use baud
rate
\$hwstack = 32
                                                         ' default
use 32 for the hardware stack
\$swstack = 10
                                                         ' default
use 10 for the SW stack
$framesize = 40
                                                         ' default
use 40 for the frame space
1.1
  +5V <---[A Led K]---[220 Ohm]---> Pb.3 for 2313.
' RC5SEND is using TIMER1, no interrupts are used
' The resistor must be connected to the OC1(A) pin , in this case PB.3
Do
  Waitms 500
  Sonysend &HA90
Loop
End
```

6.386 SORT

Action

Sorts an array in ascending order.

Syntax

SORT array() [,elements]

Remarks

array()	The first element of the array to sort.	
	The number of elements to sort. This is an optional value. By default all elements will be sorted.	

Sorting is implemented for byte, word and integer arrays. The routines are located in mcs.lib.

See also

895

```
SORT.BAS
                  (c) 1995-2011 , MCS Electronics
' This demo demonstrates the SORT statement. It will sort an array
' sort supports, byte, integer and word arrays at the first release
!_____
                                                                     _ _ _ _ _ _
                 _ _ _ _ _ _
_____
$reqfile = "m88def.dat"
Scrystal = 8000000
$hwstack = 16
\$swstack = 8
$framesize = 30
'Dim some arrays
Dim B(10) As Byte , I(10) As Integer , W(10) As Word
Dim J As Byte
'point to data
Restore Arraydata
'read the data
For J = 1 To 10
  Read B(j)
Next
'read the words
For J = 1 To 10
  Read W(j)
Next
'read the integers
For J = 1 To 10
  Read I(j)
Next
'now sort the arrays
                                                              ' 10
Sort B(1) , 10
elements
Sort W(1)
                                                              ' all
elements
Sort I(1)
'and show the result
For J = 1 To 10
  Print J ; " " ; B(j) ; " " ; W(j) ; " " ; I(j)
Next
End
```

```
Arraydata:

Data 1 , 4 , 8 , 9 , 2 , 5 , 3 , 7 , 6 , 4

Data 1000% , 101% , 1% , 400% , 30000% , 20000% , 15000% , 0% , 999% ,

111%

Data -1000% , 101% , -1% , 400% , 30000% , 2000% , -15000% , 0% , 999% ,

111%
```

6.387 SOUND

Action

Sends pulses to a port pin.

Syntax

SOUND pin, duration, pulses

Remarks

Pin	Any I/O pin such as PORTB.0 etc.
Duration	The number of pulses to send. Byte, integer/word or constant.
Pulses	The time the pin is pulled low and high.
	This is the value for a loop counter.

When you connect a speaker or a buzzer to a port pin (see hardware) , you can use the SOUND statement to generate some tones.

The port pin is switched high and low for pulses times. This loop is executed duration times.

The SOUND statement is not intended to generate accurate frequencies. Use a TIMER to do that.

See also

NONE

'copyright 'purpose		
<pre>\$regfile = "m48def.dat" the used micro</pre>		' specify
\$crystal = 4000000		' used
crystal frequency \$baud = 19200 rate		' use baud
\$hwstack = 32		' default
<pre>use 32 for the hardware st \$swstack = 10 use 10 for the SW stack</pre>	cack	' default
\$framesize = 40 use 40 for the frame space	2	' default
Dim Pulses As Word , Peric Pulses = 65535 : Periods = variables		'set
Speaker Alias Portb.1 pin		'define port
Sound Speaker , Pulses , P noice	Periods	'make some
'note that pulses and peri 'sound is only intended to	ods must have a high value for high wake some noise!	gh XTALS

```
'pulses range from 1-65535
'periods range from 1-65535
End
```

6.388 SPACE

Action

Returns a string that consists of spaces.

Syntax

var = **SPACE**(x)

Remarks

Х	The number of spaces.
Var	The string that is assigned.

Using 0 for x will result in a string of 255 bytes because there is no check for a zero length assign.

See also

STRING 913 , SPC 898

```
_____
'copyright
                       : (c) 1995-2005, MCS Electronics
'suited for demo : ves
'commercial addon needed : no
'purpose
                       : demonstrates DEG2RAD function
·_____
                                       _____
_____
$regfile = "m48def.dat"
                                                       ' specify
the used micro
$crystal = 8000000
                                                       ' used
crystal frequency
$baud = 19200
                                                       ' use baud
rate
                                                       ' default
$hwstack = 32
use 32 for the hardware stack
                                                       ' default
\$swstack = 40
use 10 for the SW stack
$framesize = 40
                                                       ' default
use 40 for the frame space
Dim S As String * 15 , Z As String * 15
S = Space(5)
Print " { " ; S ; " } "
                                                       '{}
Dim A As Byte
A = 3
```

```
898 BASCOM-AVR
```

```
S = Space(a)
End
```

6.389 SPC

Action

Prints the number of specified spaces.

Syntax

PRINT **SPC**(x) LCD **SPC**(x)

Remarks

X The number of spaces to print.

Using 0 for x will result in a string of 255 bytes because there is no check for a zero length assign.

SPC can be used with LCD 332 too.

The difference with the SPACE function is that SPACE returns a number of spaces while SPC() can only be used with printing. Using SPACE() with printing is also possible but it will use a temporary buffer while SPC does not use a temporary buffer.

See also

SPACE 897

```
_____
_ _ _ _ _ _ _ _ _
'copyright
                        : (c) 1995-2005, MCS Electronics
                        : Mega48
'micro
'suited for demo
                        : yes
'commercial addon needed : no
                        : demonstrates DEG2RAD function
'purpose
                    _ _ _ _ _ _ _ _ _ _ _ _
_____
$regfile = "m48def.dat"
                                                          ' specify
the used micro
                                                          ' used
$crystal = 8000000
crystal frequency
$baud = 19200
                                                          ' use baud
rate
$hwstack = 32
                                                          ' default
use 32 for the hardware stack
\$swstack = 40
                                                          ' default
use 10 for the SW stack
                                                          ' default
$framesize = 40
use 40 for the frame space
Dim S As String * 15 , Z As String * 15
Print "{"; spc(5); "}"
                                                          '{}
```

_ _ _ _ _ _ _ _ _ _

Lcd "{"; Spc(5); "}"

' { }

6.390 SPIIN

Action

Reads a value from the SPI-bus.

Syntax

SPIIN var, bytes

Remarks

Var	The variable which receives the value read from the SPI-bus.
Bytes	The number of bytes to read. The maximum is 255.

In order to be able to read data from the SPI slave, the master need to send some data first. The master will send the value 0.

SPI is a 16 bit shift register. Thus writing 1 byte will cause 1 byte to be clocked out of the device which the SPIIN will read.

See also

SPIOUT Mon, SPIINIT Mon, CONFIG SPI 548, SPIMOVE Mon

Example

'name	: spi.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: demo :SPI
'micro	: Mega48
'suited for demo	: yes
'commercial addon needed	: no
·	

```
$regfile = "m48def.dat"
                                                              ' specify
the used micro
$crystal = 4000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
                                                              ' default
$hwstack = 32
use 32 for the hardware stack
$swstack = 10
                                                              ' default
use 10 for the SW stack
                                                              ' default
$framesize = 40
use 40 for the frame space
Dim B As Byte
Dim A(10) As Byte
```

Spiinit

```
900 BASCOM-AVR
```

```
B = 5
Spiout A(1) , B
Spiin A(1) , B
A(1) = Spimove(a(2))
End
```

6.391 SPIINIT

Action

Initiate the SPI pins.

Syntax

SPIINIT

Remarks

After the configuration of the SPI pins, you must initialize the SPI pins to set them for the right data direction. When the pins are not used by other hardware/software, you only need to use SPIINIT once.

When other routines change the state of the SPI pins, use SPIINIT again before using SPIIN and SPIOUT.

See also

SPIIN 8997, SPIOUT 907

ASM

Calls _init_spi

Example

See SPIIN 899

6.392 SPIMOVE

Action

Sends and receives a value or a variable to the SPI-bus.

Syntax

var = **SPIMOVE**(byte)

Remarks

Var	The variable that is assigned with the received byte(s) from the SPI-bus.
Byte	The variable or constant whose content must be send to the SPI-bus.

See also

SPIIN (899), SPIINIT (900), CONFIG SPI 548)

Example

```
Config Spi = Soft , Din = Pinb.0 , Dout = Portb.1 , Ss = Portb.2 , Clock
= Portb.3
```

Spiinit

Dim a(10) as Byte , X As Byte

```
Spiout A(1) , 5
bytes
Spiout X , 1
A(1) = Spimove(5)
SPI and store result in a(1)
End
```

'send 5

'send 1 byte ' move 5 to

6.393 SPIOUT

Action

Sends a value of a variable to the SPI-bus.

Syntax

SPIOUT var , bytes

Remarks

var	The variable whose content must be send to the SPI-bus.
bytes	The number of bytes to send. Maximum value is 255.

When SPI is used in HW(hardware) mode, there might be a small delay/pause after each byte that is sent. This is caused by the SPI hardware and the speed of the bus. After a byte is transmitted, SPSR bit 7 is checked. This bit 7 indicates that the SPI is ready for sending a new byte.

See also

SPIIN [899], SPIINIT [900], CONFIG SPI [548], SPIMOVE [900]

Example

```
Dim A(10) As Byte
Config Spi = Soft , Din =Pinb.0 , Dout =Portb.1 , Ss =Portb.2 , Clock =
Portb.3
Spiinit
Spiout A(1), 4 'write 4 bytes a(1), a(2) , a(3) and a(4)
End
```

6.394 SPLIT

Action

Split a string into a number of array elements.

Syntax

count = SPLIT (source, array, search)

Remarks

	The number of elements that SPLIT() returned. When the array is not big enough to fill the array, this will be the maximum size of the array. So make sure the array is big enough to hold the results.	
source	The source string or string constant to search for.	
array	The index of the first element of the array that will be filled	
	The character to search for. This can be a string or string constant or a byte with the ASCII value.	

When you use the serial port to receive data, in some cases you need to process the data in parts.

For example when you need to split an IP number as "123.45.24.12" you could use INSTR() or you can use SPLIT().

You must DIM the array yourself. The content of the array will be overwritten.

It is also important to know that the individual elements of the array need to be big enough to store the string part.

For example when the array has 5 elements and each element may be 10 characters long, a string that is 11 bytes long will not fit. Another element will be used in that case to store the additional info.

The SPLIT function takes care not to overwrite other memory. So when you split "1.2.2.2.2.2.3.3.3" into an array of 3 elements, you will loose the data. If empty data is encountered, an empty element will be created. Thus "1,2,3,,5" will create 5 elements. Element 4 will be empty.

See also

INSTR 747, CHARPOS 424

Example

```
mega48.bas
                      mega48 sample file
                   (c) 1995-2005, MCS Electronics
$regfile = "m48def.dat"
$crystal = 8000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As String * 80
Dim Ar(5) As String * 10
Dim Bcount As Byte
'The split function can split a string or string constant into elements
'It returns the number of elements
'You need to take care that there are enough elements and that each
element is big enough
'to hold the result
'When a result does not fit into 1 element it will be put into the next
```

```
element
'The memory is protected against overwriting.
S = "this is a test"
Bcount = Split( "this is a test" , Ar(1) , " ")
'bcount will get the number of filled elements
'ar(1) is the starting address to use
'" " means that we check for a space
'When you use " aa" , the first element will contain a space
Bcount = Split( "thiscannotfit! into the element" , Ar(1) , " ")
Dim J As Byte
For J = 1 To Bcount
 Print Ar(j)
Next
'this demonstrates that your memory is safe and will not be overwritten
when there are too many string parts
Bcount = Split( "do not overflow the array please" , Ar(1) , " ")
For J = 1 To Bcount
 Print Ar(j)
Next
End
```

6.395 SQR

Action

Returns the Square root of a variable.

Syntax

var = **SQR**(source)

Remarks

	A numeric single or double variable that is assigned with the SQR of variable source.
source	The single or double variable to get the SQR of.

When SQR is used with a single, the FP_TRIG library will be used. When SQR is used with bytes, integers, words and longs, the SQR routine from MCS. LBX will be used.

See Also

POWER 805

Example

```
$regfile = "m48def.dat"
the used micro
$crystal = 8000000
crystal frequency
```

' specify

' used

```
904
       BASCOM-AVR
    $baud = 19200
                                                                    ' use baud
    rate
    $hwstack = 32
                                                                    ' default
    use 32 for the hardware stack
    \$swstack = 40
                                                                    ' default
    use 10 for the SW stack
    $framesize = 40
                                                                    ' default
    use 40 for the frame space
    Dim A As Single
    Dim B As Double
    A = 9.0
    B = 12345678.123
    A = Sqr(A)
    Print A
                                                                    ' prints 3.0
    B = Sqr(b)
    Print B
    End
```

6.396 START

Action

Start the specified hardware source.

Syntax

START device [, cfg]

Remarks

Device	TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power), ADC(A/D converter power) or DAC(D/A converter).
 XMEGA For the Xmega you can also specify : DACA or DACB for the Dig Analog Converters A and B. ADCA and ADCB for the A/D converters. For the timers you can use TCC0, TCC1, TCD0, TCD1, TCE0, TCI TCF0 and TCF1. To start a DMA soft transfer, you can use DMACH0, DMACH1, DI and DMACH3 	
cfg	The optional cfg is only used for the TIMER when the optional CONFIGURATION is used. If CONFIG TIMERx = option , CONFIGURATION=mysetting was used, you would specify START TIMERx, mysetting.

When you configure a timer (CONFIG TIMER), the TIMER is started automatically when a pre-scaler value is provided.

When you want to halt the timer you can use the STOP TIMER statement. To start the timer after it has been stopped, you can use the START TIMER statement. The START TIMER statement will only work correctly when you have selected a clock source or pre-scaler value with the CONFIG TIMER statement.

When you stored settings using the option CONFIGURATION=setting , then you can specify which configuration the timer must use by providing the setting name as a

parameter : START TIMER1 , mysetting

When a timer is used in interrupt mode, it must be running otherwise the interrupt will never occur.

TIMER0 and COUNTER0 are the same device. And so are TIMER1 and COUNTER1.

The AC, ADC and DAC parameters will switch power to the device and thus enabling it to work.

The WATCHDOG parameter will activate the Watchdog.

See also

STOP 910

Example

```
_____
_____
'name
                       : adc.bas
'copyright
                       : (c) 1995-2005, MCS Electronics
                       : demonstration of GETADC() function for 8535
'purpose
or M163 micro
'micro
                        : Mega163
'suited for demo
                        : yes
'commercial addon needed : no
                       : possible
'use in simulator
' Getadc() will also work for other AVR chips that have an ADC converter
·_____
_____
$regfile = "m163def.dat"
                                                        ' we use the
M163
$crystal = 4000000
                                                        ' default
$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
                                                        'default use
10 for the SW stack
$framesize = 40
                                                        'default use
40 for the frame space
'configure single mode and auto prescaler setting
'The single mode must be used with the GETADC() function
'The prescaler divides the internal clock by 2,4,8,16,32,64 or 128
'Because the ADC needs a clock from 50\mathchar`-200~\mbox{KHz}
'The AUTO feature, will select the highest clockrate possible
Config Adc = Single , Prescaler = Auto
'Now give power to the chip
Start Adc
'With STOP ADC, you can remove the power from the chip
'Stop Adc
Dim W As Word , Channel As Byte
Channel = 0
'now read A/D value from channel 0
Do
 W = Getadc(channel)
 Print "Channel " ; Channel ; " value " ; W
```

```
Incr Channel
  If Channel > 7 Then Channel = 0
LOOD
End
'The new M163 has options for the reference voltage
'For this chip you can use the additional param :
'Config Adc = Single , Prescaler = Auto, Reference = Internal
'The reference param may be :
' OFF
          : AREF, internal reference turned off
: AVCC, with external capacitor at AREF pin
'INTERNAL : Internal 2.56 voltage reference with external capacitor ar
AREF pin
'Using the additional param on chip that do not have the internal
reference will have no effect.
```

6.397 STCHECK

Action

Calls a routine to check for various stack overflows. This routine is intended for debug purposes.

Syntax

STCHECK

Remarks

The different stack spaces used by BASCOM-AVR lead to lots of questions about them. The STCHECK routine can help to determine if the stack size are trashed by your program. The program STACK.BAS is used to explain the different settings.

Note that STCHECK should be removed form your final program. That is once you tested your program and found out is works fine, you can remove the call to STCHECK since it costs time and code space.

The settings used are : HW stack 8 Soft stack 2 Frame size 14

Below is a part of the memory of the 90S2313 used for the example: C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF FR SP SP SP SP SP SP SP

Since the last memory in SRAM is DF, the hardware stack is occupied by D8-DF(8 bytes) When a call is made or a push is used the data is saved at the position the hardware stack pointer is pointing to. After this the stack pointer is decreased. A call uses 2 bytes so SP will be SP-2. (DF-2) =DD When 8 bytes are stored the SP will point to D7. Another call or push will thus destroy memory position D7 which is occupied by the soft stack. The soft stack begins directly after the hardware stack and is also growing down.

The Y pointer(r28+r29) is used to point to this data.

Since the Y pointer is decreased first and then the data is saved, the pointer must point at start up to a position higher. That is D8, the end of the hardware space.

St -y,r24 will point to D8-1=D7 and will store R24 at location D7. Since 2 bytes were allocated in this example we use D7 and D6 to store the data. When the pointer is at D6 and another St -y,r24 is used, it will write to position D5 which is the end of the frame space that is used as temporarily memory.

The frame starts at C8 and ends at D5. Writing beyond will overwrite the soft stack. And when there is no soft stack needed, it will overwrite the hardware stack space. The map above shows FR(frame), YY(soft stack data) and SP(hardware stack space)

How to determine the right values?

The stack check routine can be used to determine if there is an overflow.

It will check : -if SP is below it's size. In this case below D8. -if YY is below it's size in this case when it is D5 -if the frame is above its size in this case D6

When is YY(soft stack) used? When you use a LOCAL variable inside a SUB or function. Each local variable will use 2 bytes. When you pass variables to user Subroutines or functions it uses 2 bytes for each parameter. call mysub(x,y) will use 2 * 2 = 4 bytes. local z as byte ' will use another 2 bytes

This space is freed when the routine ends. But when you call another sub inside the sub, you need more space. sub mysub(x as byte,y as byte) call testsub(r as byte) ' we must add another 2 bytes

When you use empty(no params) call like :

call mytest(), No space is used.

When do you need frame space? When ever you use a num<>string conversion routine like:

Print b (where b is a byte variable)

Bytes will use 4 bytes max (123+0) Integer will use 7 bytes max (-12345+0)c Longs will use 16 bytes max And the single will use 24 bytes max

When you add strings and use the original the value must be remembered by the

compiler.

Consider this : s\$ = "abcd" + s\$

Here you give s\$ a new value. But you append the original value so the original value must be remembered until the operation has completed. This copy is stored in the frame too.

So when string s\$ was dimmed with a length of 20, you need a frame space of 20+1 (null byte)

When you pass a variable by VALUE (BYVAL) then you actually pass a copy of the variable.

When you pass a byte, 1 byte of frame space is used, a long will take 4 bytes. When you use a LOCAL LONG, you also need 4 bytes of frame space to store the local long.

The frame space is reused and so is the soft stack space and hardware stack space. So the hard part is to determine the right sizes!

The stack check routine must be called inside the deepest nested sub or function.

Gosub test

test: gosub test1 return

test1:
' this is the deepest level so check the stack here
 stcheck
return

Stcheck will use 1 variable named ERROR. You must dimension it yourself.

Dim Error As Byte

Error will be set to :

1: if hardware stack grows down into the soft stack space

2: if the soft stack space grows down into the frame space

3: if the frame space grows up into the soft stack space.

The last 2 errors are not necessarily bad when you consider that when the soft stack is not used for passing data, it may be used by the frame space to store data. Confusing right.?

It is advised to use the simpler DBG/\$DBG method. This requires that you can simulate your program.

ASM

Routines called by STCHECK :

_StackCheck : uses R24 and R25 but these are saved and restored.

Because the call uses 2 bytes of hardware stack space and the saving of R24 and R25 also costs 2 bytes, it uses 4 more bytes of hardware stack space than your final program would do that f course does not need to use STCHECK.

Example

```
_____
'name
                        : stack.bas
                        : (c) 1995-2005, MCS Electronics
'copyright
                        : shows how to check for the stack sizes
'purpose
'micro
                        : Mega48
'suited for demo
                        : yes
'commercial addon needed : no
                                        _____
_____
$regfile = "m48def.dat"
                                                          ' specify
the used micro
$crystal = 4000000
                                                           ' used
crystal frequency
$baud = 19200
                                                           ' use baud
rate
$hwstack = 8
                                                          ' default
use 32 for the hardware stack
\$swstack = 2
                                                           ' default
use 10 for the SW stack
                                                           ' default
$framesize = 14
use 40 for the frame space
'settings must be :
'HW Stack : 8
'Soft Stack : 2
'Frame size : 14
'note that the called routine (_STACKCHECK) will use 4 bytes
'ofhardware stack space
'So when your program works, you may subtract the 4 bytes of the needed
hardware stack size
'in your final program that does not include the STCHECK
'testmode =0 will work
'testmode =1 will use too much hardware stack
'testmode =2 will use too much soft stack space
'testmode =3 will use too much frame space
Const Testmode = 0
'compile and test the program with testmode from 0-3
'you need to dim the ERROR byte !!
Dim Error As Byte
#if Testmode = 2
  Declare Sub Pass(z As Long , Byval K As Long)
#else
  Declare Sub Pass()
#endif
Dim I As Long
I = 2
```

```
Print I
'call the sub in your code at the deepest level
'normally within a function or sub
#if Testmode = 2
  Call Pass(i , 1)
#else
  Call Pass()
#endif
End
#if Testmode = 2
   Sub Pass(z As Long , Byval K As Long)
#else
 Sub Pass()
#endif
    #if Testmode = 3
       Local S As String * 13
    #else
       Local S As String * 8
    #endif
    Print I
    Gosub Test
End Sub
Test:
#if Testmode = 1
 push r0 ; eat some hardware stack space
 push r1
 push r2
#endif
  ' *** here we call the routine ***
 Stcheck
  ' *** when error <>0 then there is a problem ***
#if Testmode = 1
 pop r2
 pop rl
 pop r0
#endif
Return
```

6.398 STOP

Action

Stop the specified device. Or stop the program

Syntax

STOP device STOP

Remarks

	TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power), ADC(A/D converter power) or DAC(D/A converter)	
XMEGA	For the Xmega you can also specify : DACA or DACB for the Digital/	

Analog Converters A and B.

The single STOP statement will end your program by generating a never ending loop. When END is used it will have the same effect but in addition it will disable all interrupts.

The STOP statement with one of the above parameters will stop the specified device.

TIMER0 and COUNTER0 are the same device.

The AC and ADC parameters will switch power off the device to disable it and thus save power.

See also

START 904 , END 680

Example

See <u>START</u> 904 example

6.399 STR

Action

Returns a string representation of a number.

Syntax

var = STR(x)

Remarks

var	A string variable.
Х	A numeric variable.

The string must be big enough to store the result. So if you have a string like this : Dim S as string * 4, and you use it on a single with the value 0.00000001 then there is not enough space in the string to hold the result. Strings that are assigned with Str() should be dimmed 16 characters long.

You do not need to convert a variable into a string before you print it. When you use PRINT var, then you will get the same result as when you convert the numeric variable into a string, and print that string.

The PRINT routine will convert the numeric variable into a string before it gets printed to the serial port.

As the integer conversion routines can convert byte, integer, word and longs into a string it also means some code overhead when you do not use longs. You can include the alternative library named <u>mcsbyte</u>[sec].lbx then. This library can only print bytes. There is also a library for printing integers and words only. This library is named <u>mcsbyteint</u>[sec].

When you use these libs to print a long you will get an error message.

See also

VAL 945 , HEX 722 , HEXVAL 722 , MCSBYTE 966 , BIN 411 , STR2DIGITS 912

Difference with VB

In VB STR() returns a string with a leading space. BASCOM does not return a leading space.

Example

```
Dim A As Byte , S As String * 10
A = 123
S = Str(a)
Print S ' 123
'when you use print a, you will get the same result.
'but a string can also be manipulated with the string routines.
End
```

6.400 STR2DIGITS

Action

This statement will convert a string into an array of binary numbers.

Syntax

STR2DIGITS s, ar(1)

Remarks

s	A string variable that holds a number. For example "12345"
ar(1)	The first element of a byte array that will be assigned with the binary representation of the digits. After the conversion, the first element will be assigned with the number of processed digits. The next element will become the most right digit of the string, the last element will become the first character of the string. In this example with string "12345" ar(1) = 5 ar(2) = 5 ar(3) = 4 ar(4) = 3 ar(5) = 2 ar(6) = 1 Your array need to be big enough to hold all digits and the digit counter.

You can convert a string into a numer with VAL() and a number into a string with STR ().

In some cases, it is required to have access to all the individual digits of a variable. While you can use a loop and MOD to get all digits, the STR2DIGITS will work for bytes, word, and longs.

Non numeric digits will not be converted properly. For example, in a string "-0", the

0 which is ASCII 48, will be converted into a 0. The - is 45 and will result in 45-48=-3, and in byte form : 253. The dot (.) will be converted into 254.

See also

STR 911, VAL 945

Example

```
_____
               ARDUINO-Duemilanove.BAS
           Also tested with ARDUINO NANO V3.0
           (c) 1995-2011, MCS Electronics
  This is a sample file for the Mega328P based ARDUINO board
  Select Programmer 'ARDUINO' , 57600 baud and the proper COM port
      _____
$regfile= "m328pdef.dat" ' used micro
$crystal=16000000 ' used xtal
$baud=19200 ' baud rate we want
config clockdiv=1 ' either use this or change the divider fuse
byte
       _____
1 _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
dim w as word
dim s as string * 6, ar(6) as byte
                        ' make portb an output
config portb=output
do
 toggle portb
                         ' toggle level
 waitms 1000
                         ' wait 1 sec
 print "Duemilanove"
                         ' test serial com
 w=w+1 : s=str(w)
                         ' convert w to a string
 str2digits s,ar(1) convert string into an array with binary
numbers
```

```
loop
```

6.401 STRING

Action

Returns a string consisting of m repetitions of the character with ASCII Code n.

Syntax

var = **STRING**(m ,n)

Remarks

Var	The string that is assigned.
Ν	The ASCII-code that is assigned to the string.
М	The number of characters to assign.

Since a string is terminated by a 0 byte, you can't use 0 for n. Using 0 for m will result in a string of 255 bytes, because there is no check on a length assign of 0.

See also

SPACE 897

Example

<pre>\$regfile = "m48def.dat" the used micro</pre>	1	specify
\$crystal = 8000000	,	used
crystal frequency		una baud
\$baud = 19200 rate		use baud
<pre>\$hwstack = 32</pre>		default
use 32 for the hardware stack		1 6 1.
<pre>\$swstack = 40 use 10 for the SW stack</pre>		default
\$framesize = 40		default
use 40 for the frame space		

```
Dim S As String * 15
S = String(5, 65)
Print S
End
```

'AAAAA

6.402 SUB

Action

Defines a Sub procedure.

Syntax

SUB Name[(var1 , ...)]

Remarks

Name	Name of the sub procedure, can be any non-reserved word.
var1	The name of the parameter.

You must end each subroutine with the END SUB statement. You can copy the DECLARE SUB line and remove the DECLARE statement. This ensures that you have the right parameters.

See Also

FUNCTION 6427, CALL 4227, CONFIG SUBMODE 555

See the <u>DECLARE SUB</u> 644 topic for more details.

6.403 SYSSEC

Action

Returns a Number, which represents the System Second

Syntax

Target = **SYSSEC**() Target = **SYSSEC**(bSecMinHour) Target = **SYSSEC**(strTime, strDate) Target = **SYSSEC**(wSysDay)

Remarks

Target	A Variable (LONG), that is assigned with the System-Second		
	A Byte, which holds the Sec-value followed by Min(Byte), Hour (Byte),		
ur	Day(Byte), Month(Byte) and Year(Byte)		
StrTime	A time-string in the format "hh:mm:ss"		
StrDate	A date-string in the format specified in the Config Date statement		
wSysDay	SysDay A variable (Word) which holds the System Day (SysDay)		

The Function can be used with 4 different kind of inputs:

- 1. Without any parameter. The internal Time and Date of SOFTCLOCK (_sec, _min, _hour, _day, _month, _year) is used.
- 2. With a user defined time and Date array. It must be arranged in same way (Second, Minute, Hour, Day, Month, Year) as the internal SOFTCLOCK time/ date. The first Byte (Second) is the input by this kind of usage. So the System Second can be calculated of every time/date.
- 3. With a time-String and a date-string. The time-string must be in the Format "hh:mm:ss". The date-string must be in the format specified in the Config Date statement
- 4. With a System Day Number (Word). The result is the System Second of this day at 00:00:00.

The Return-Value is in the Range of 0 to 2147483647. 2000-01-01 at 00:00:00 starts with 0.

The Function is valid from 2000-01-01 to 2068-01-19 03:14:07. In the year 2068 a LONG – overflow will occur.

See also

Date and Time Routines [993], SYSSECELAPSED [916], SYSDAY [917]

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator =.' ANSI-Format
Dim Strdate As String * 8
Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte , Bhour As Byte
Dim Bday As Byte , Bmonth As Byte , Byear As Byte
Dim Wsysday As Word
```

916 BASCOM-AVR

```
Dim Lsyssec As Long
```

```
' Example 1 with internal RTC-Clock
' Load RTC-Clock for example - testing
_sec = 17 : _min = 35 : _hour = 8 : _day = 16 : _month = 4 : _year = 3
Lsyssec = Syssec()
Print "System Second of "; Time$; " at "; Date$; " is "; Lsyssec
' System Second of 08:35:17 at 03.04.16 is 103797317
' Example 2 with with defined Clock - Bytes (Second, Minute, Hour, Day /
Month / Year)
Bsec = 20 : Bmin = 1 : Bhour = 7 : Bday = 22 : Bmonth = 12 : Byear = 1
Lsyssec = Syssec(bsec)
Strtime = Time sb(bsec) : Strdate = Date sb(bday)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsvssec
' System Second of 07:01:20 at 01.12.22 is 62319680
' Example 3 with Time and Date - String
Strtime = "04:58:37"
strDate ="02.09.18"
Lsyssec = Syssec(strtime , Strdate)
Print "System Second of " ; Strtime ; " at " ; Strdate ; " is " ;
Lsyssec
' System Second of 04:58:37 at 02.09.18 is 85640317
' Example 4 with System Day
Wsysday = 2000
Lsyssec = Syssec(wsysday)
Print "System Second of System Day " ; Wsysday ; " (00:00:00) is " ;
Lsvssec
' System Second of System Day 2000 (00:00:00) is 172800000
```

6.404 SYSSECELAPSED

Action

Returns the elapsed Seconds to a earlier assigned system-time-stamp.

Syntax

Target = **SysSecElapsed**(SystemTimeStamp)

Remarks

Target	A variable (LONG), that is assigned with the elapsed Seconds
	A variable (LONG), which holds a Systemtimestamp like the output of an earlier called SysSec()

The Return-Value is in the Range of 0 to 2147483647. The Function is valid from 2000-01-01 to 2068-01-19 at 03:14:07. In the year 2068 a LONG – overflow will occur.

The difference to the pair DayOfSec and SecElapsed is, that SysSec and SysSecElapsed can be used for event distances larger than 24 hours.

See also

<u>Date and Time Routines</u> (993), <u>SECELAPSED</u> [849), <u>SYSSEC</u> [915]

Example

```
Enable Interrupts
Config Clock = Soft
Dim Lsystemtimestamp As Long
Dim Lsystemsecondselapsed As Long
Lsystemtimestamp = Syssec()
Print "Now it's " ; Lsystemtimestamp ; " seconds past 2000-01-01
00:00:00"
' do other stuff
' some time later
Lsystemsecondselapsed = Syssecelapsed(lsystemtimestamp)
Print "Now it's " ; Lsystemsecondselapsed ; " seconds later"
```

6.405 SYSDAY

Action

Returns a number, which represents the System Day

Syntax

Target = **SysDay**() Target = **SysDay**(bDayMonthYear) Target = **SysDay**(strDate) Target = **SysDay**(ISysSec)

Remarks

Target	A Variable (LONG), that is assigned with the System-Day
· ·	A Byte, which holds the Day-value followed by Month(Byte) and Year (Byte)
	A String, which holds a Date-String in the format specified in the CONFIG DATA statement
lSysSec	A variable, which holds a System Second (SysSec)

The Function can be used with 4 different kind of inputs:

- 1. Without any parameter. The internal Date-values of SOFTCLOCK (_day, _month, _year) are used.
- 2. With a user defined date array. It must be arranged in same way (Day, Month, Year) as the internal SOFTCLOCK date. The first Byte (Day) is the input by this kind of usage. So the Day of the Year can be calculated of every date.
- 3. With a Date-String. The date-string must be in the Format specified in the Config Date Statement.
- 4. With a System Second Number (LONG)

The Return-Value is in the Range of 0 to 36524. 2000-01-01 starts with 0. The Function is valid in the 21th century (from 2000-01-01 to 2099-12-31).

See also

Date and Time Routines [993], Config Date [473], Config Clock [461], SysSec [915]

Example

```
Enable Interrupts
Config Clock = Soft
Config Date = YMD , Separator =.' ANSI-Format
```

```
Dim Strdate As String * 8
Dim Bday Asbyte , Bmonth As Byte , Byear As Byte
Dim Wsysday As Word
Dim Lsyssec As Long
```

```
' Example 1 with internal RTC-Clock
_day = 20 : _Month = 11 : _Year = 2 ' Load RTC-Clock for example -
testing
Wsysday = Sysday()
Print "System Day of " ; Date$ ; " is " ; Wsysday
```

' System Day of 02.11.20 is 1054

```
' Example 2 with defined Clock - Bytes (Day / Month / Year)
Bday = 24 : Bmonth = 5 : Byear = 8
Wsysday = Sysday(bday)
Print "System Day of Day=" ; Bday ; " Month=" ; Bmonth ; " Year=" ;
Byear ; " is " ; Wsysday
' System Day of Day=24 Month=5 Year=8 is 3066
```

```
' Example 3 with Date - String
Strdate = "04.10.29"
Wsysday = Sysday(strdate)
Print "System Day of " ; Strdate ; " is " ; Wsysday
' System Day of 04.10.29 is 1763
```

```
' Example 4 with System Second
Lsyssec = 123456789
Wsysday = Sysday(lsyssec)
Print "System Day of System Second " ; Lsyssec ; " is " ; Wsysday
' System Day of System Second 123456789 is 1428"Now it's " ;
Lsystemsecondselapsed ; " seconds later"
```

6.406 SWAP

Action

Exchange two variables of the same type. Exchange a nibbler or 2 bytes

Syntax

SWAP var1, var2 SWAP var3

Remarks

var1 A variable of type bit, byte, integer, word, long or string.

var2	A variable of the same type as var1.
var3	A byte, integer or word.

After the swap, var1 will hold the value of var2 and var2 will hold the value of var1.

When using swap with a single variable it need to be a byte, integer or word variable. When using swap on a byte, the nibbles will be swapped. Example :

byte=&B1100_0001 : swap byte : byte will become : &B0001_1100

When using swap on a single integer or word, the 2 bytes will be swapped so the LSB becomes the MSB and the MSB becomes the LSB.

Example

'		
'copyright 'purpose		
<pre>\$regfile = "m48def.dat"</pre>		' specify
the used micro \$crystal = 4000000		' used
crystal frequency \$baud = 19200		' use baud
<pre>rate \$hwstack = 32 was 22 for the headware at</pre>		' default
use 32 for the hardware sta \$swstack = 10	ack	' default
<pre>use 10 for the SW stack \$framesize = 40 use 40 for the frame space</pre>	' default	
Dim A As Byte , B1 As Byte Dim Bbit1 As Bit , Bbit2 A Dim S1 As String * 10 , S2	s Bit	
S1 = "AAA" : S2 = "BBB" Swap S1 , S2		
A = 5 : B1 = 10		'assign some
vars Print A ; " " ; B1		'print them
Swap A , B1 Print A ; " " ; B1 again		'swap them 'print is
<pre>Set Bbit1 Swap Bbit1 , Bbit2 Print Bbit1 ; Bbit2 End</pre>		

6.407 TAN

920

Action

Returns the tangent of a float

Syntax

var = TAN(source)

Remarks

Var	A numeric variable that is assigned with tangent of variable source.
Source	The single or double variable to get the tangent of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD [649], ATN [403], COS [596], SIN [885], ATN2 [404]

Example

```
$regfile = "m48def.dat"
                                                              ' specify
the used micro
$crystal = 8000000
                                                              ' used
crystal frequency
$baud = 19200
                                                              ' use baud
rate
$hwstack = 32
                                                              ' default
use 32 for the hardware stack
\$swstack = 10
                                                              ' default
use 10 for the SW stack
$framesize = 40
                                                              ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim S As Single , X As Single
S = 0.5 : X = Tan(s) : Print X
                                                              ' prints
0.546302195
S = 0.5 : X = Sin(s) : Print X
                                                              ' prints
0.479419108
S = 0.5 : X = Cos(s) : Print X
                                                              ' prints
0.877588389
End
```

6.408 TCPCHECKSUM

Action

Return a TCP/IP checksum, also called Internet Checksum, or IP Checksum.

Syntax

res= TCPCHECKSUM(buffer , bytes [,w1] [,w2])

i cinarko			
Res	A word variable that is assigned with the TCP/IP checksum of the buffer		
Buffer	A variable or array to get the checksum of.		
Bytes	The number of bytes that must be examined.		
w1,w2	Optional words that will be included in the checksum.		

Remarks

Checksum's are used a lot in communication protocols. A checksum is a way to verify that received data is the same as it was sent. In the many Internet Protocols (TCP, UDP, IP, ICMP ...) a special Internet checksum is used. Normally the data to calculate the checksum on is stored in an array of bytes, but in some cases like TCP, and UDP, a pseudo header is added. The optional words (w1, w2) can be used for these cases. Most often w1 and w2 will be used for the Protocol number, and the UDP or TCP packet length.

This checksum is calculated by grouping the bytes in the array into 2-byte words. If the number of Bytes is an odd number, then an extra byte of zero is used to make the last 2-byte word. All of the words are added together, keeping the total in a 4-byte Long variable. If the optional words w1, w2, are included, they are also added to the total. Next, the 4-byte Long total is split into two, 2-byte words, and these words are added together to make a new 2-byte Word total. Finally the total is inverted. This is the value returned as Res.

This function using w1, w2, are very useful when working directly with Ethernet chips like the RTL8019AS or with protocols not directly supported by the WIZnet chips.

See the samples directory for more examples of use (IP_Checksum.bas).

You can use it for the PING sample below.

See also

CRC8 601, CRC16 603, CRC32 607, CHECKSUM 427

ASM

NONE

Example

'name rfcs/rfc792.html 'copyright 'purpose 'micro 'suited for demo 'commercial addon needed	: PING_TWI.bas : (c) 1995-2005, MCS : Simple PING program : Mega88 : yes : no	http://www.faqs.org/ Electronics
\$regfile = "m32def.dat" the used micro		' specify
\$crystal = 8000000		' used
crystal frequency \$baud = 19200		' use baud
rate \$hwstack = 80 use 32 for the hardware	stack	' default

BASCOM-AVR

922

\$swstack = 128		defaul	lt
use 10 for the SW stack \$framesize = 80		defaul	l+
use 40 for the frame space		uerau	it.
Const Debug = 1			
Const Sock_stream = \$01	1	Тср	
Const Sock_dgram = \$02		Udp	
Const Sock_ipl_raw = \$03 Raw Sock	1	lp Lay	er
Const Sock_macl_raw = \$04	1	Mac La	ayer
Raw Sock Const Sel_control = 0	1	Confirn	n
Socket Status Const Sel_send = 1	I.	Confirm	Тх
Free Buffer Size Const Sel_recv = 2		Confirm	Rx
Data Size		Comm	ΓX
'socket status			
Const Sock_closed = \$00	1	Status	Of
Connection Closed Const Sock_arp = \$01	1	Status	Of
Arp Const Sock_listen = \$02	I.	Status	Of
Waiting For Tcp Connection Setup Const Sock_synsent = \$03		Status	Of
Setting Up Tcp Connection	,		
Const Sock_synsent_ack = \$04 Setting Up Tcp Connection		Status	Of
Const Sock_synrecv = \$05 Setting Up Tcp Connection	'	Status	Of
Const Sock_established = \$06 Tcp Connection Established	1	Status	Of
Const Sock_close_wait = \$07	1	Status	Of
Closing Tcp Connection Const Sock_last_ack = \$08	1	Status	Of
Closing Tcp Connection Const Sock_fin_wait1 = \$09	1	Status	Of
Closing Tcp Connection Const Sock_fin_wait2 = \$0a	,	Status	Of
Closing Tcp Connection		_	
Const Sock_closing = \$0b Closing Tcp Connection		Status	Of
Const Sock_time_wait = \$0c Closing Tcp Connection	'	Status	Of
Const Sock_reset = \$0d Closing Tcp Connection	1	Status	Of
Const Sock_init = \$0e	1	Status	Of
Socket Initialization Const Sock_udp = \$0f	1	Status	Of
Udp Const Sock_raw = \$10		Status	of
IP RAW			
'we do the usual			
Print "Init TCP" message	'	display	а
Enable Interrupts use config tcpip , we need to enable the interrupts	، د	before	we
Config Tcpip = Int0, Mac = 12.128.12.34.56.78, Ip	p = 192.168.	0.8,	-
Submask = 255.255.255.0 , Gateway = 192.168.0.1 , = \$55 , Rx = \$55 , Twi = &H80 , Clock = 400000	Localport =	1000 ,	IX
Print "Init done"			

Dim Peersize As Integer, Peeraddress As Long, Peerport As Word

923

```
Dim Idx As Byte, Result As Word, J As Byte, Res As Byte
Dim lp As Long
Dim Dta(12) As Byte, Rec(12) As Byte
Dta(1) = 8
                                                                    'type
                                                                           is
echo
Dta(2) = 0
                                                                    'code
Dta(3) = 0
                                                                    1.00
                                                                      for
checksum
                initialization
Dta(4) = 0
                                                                      checksum
Dta(5) = 0
                                                                      а
signature can be any number
                                                                    ÷
Dta(6) = 1
signature
Dta(7) = 0
                                                                      sequence
number - any number
Dta(8) = 1
Dta(9) = 65
Dim W As Word At Dta + 2 Overlay
                                                                    'same as dta
(3) and dta(4)
W = Tcpchecksum(dta(1), 9)
checksum and store in dta(3) and dta(4)
                                                                    ' calculate
#i f Debug
  For J = 1 To 9
   Print Dta(j)
  Next
#endif
lp = Maketcp(192.168.0.16)
                                                                    'try
                                                                          to
check this server
Print "Socket "; Idx; "; Idx
Setipprotocol ldx , 1
                                                                    'set
protocol to 1
the protocol value must be set BEFORE the socket is openend
Idx = Getsocket(idx , 3 , 5000 , 0)
    Result = Udpwrite(ip , 7 , Idx , Dta(1) , 9)
                                                                    'write
                                                                            ping
data
   Print Result
   Waitms 100
    Result = Socketstat(idx, Sel_recv)
                                                                   'check
                                                                             for
data
   Print Result
   I f Result >= 11 Then
      Print "Ok"
Res = Tcpread(idx , Rec(1) , Result)
                                                                    'get
                                                                          data
with TCPREAD !!!
      #i f Debug
                                                                    ı.
         Print "DATA RETURNED :"; Res
         For J = 1 To Result
Print Rec(j) ; " ";
         Next
         Print
      #endif
   Else
                                                                    'there
                                                                            might
be a problem
      Print "Network not
                               available"
```

End I f

Do

Waitms 1000

6.409 TCPREAD

Action

Reads data from an open socket connection.

Syntax

Result = **TCPREAD**(socket , var, bytes)

Remarks

Result	A byte variable that will be assigned with ${f 0}$, when no errors occurred. When an error occurs, the value will be set to ${f 1}$.
	When there are not enough bytes in the reception buffer, the routine will wait until there is enough data or the socket is closed.
socket	The socket number you want to read data from (0-3).
Var	The name of the variable that will be assigned with the data from the socket.
Bytes	The number of bytes to read. Only valid for non-string variables.

When you use TCPread with a string variable, the routine will wait for CR + LF and it will return the data without the CR + LF. For strings, the function will not overwrite the string.

For example, your string is 10 bytes long and the line you receive is 80 bytes long, you will receive only the first 10 bytes after CR + LF is encountered. Also, for string variables, you do not need to specify the number of bytes to read since the routine will wait for CR + LF.

For other data types you need to specify the number of bytes. There will be no check on the length so specifying to receive 2 bytes for a byte will overwrite the memory location after the memory location of the byte.

See also

CONFIG TCPIP 559, GETSOCKET 717, SOCKETCONNECT 888, SOCKETSTAT 890, TCPWRITE 924, TCPWRITESTR 925, CLOSESOCKET 439, SOCKETLISTEN 889

Partial Example

Result = Socketstat(idx , Sel_recv)
waiting
If Result > 0 Then
 Result = Tcpread(idx , S)
End If

' get number of bytes

6.410 TCPWRITE

Action

Write data to a socket.

Syntax

Result = **TCPWRITE**(socket , var , bytes) Result = **TCPWRITE**(socket , EPROM, address , bytes)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket.
	When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned.
	When there is no space, 0 will be returned.
Socket	The socket number you want to send data to(0-3).
Var	A constant string like "test" or a variable.
	When you send a constant string, the number of bytes to send does not need to be specified.
Bytes	A word variable or numeric constant that specifies how many bytes must be send.
Address	The address of the data stored in the chips internal EEPROM. You need to specify EPROM too in that case.
EPROM	An indication for the compiler so it knows that you will send data from EPROM.

The TCPwrite function can be used to write data to a socket that is stored in EEPROM or in memory.

When you want to send data from an array, you need to specify the element : var (idx) for example.

See also

CONFIG TCPIP 559, GETSOCKET 717, SOCKETCONNECT 688, SOCKETSTAT 690, TCPWRITESTR 926, TCPREAD 924, CLOSESOCKET 439, SOCKETLISTEN 689

Example

Result = Tcpwrite(idx , "Hello from W3100A{013}{010}")

6.411 TCPWRITESTR

Action

Sends a string to an open socket connection.

Syntax

Result = **TCPWRITESTR**(socket , var , param)

Remarks

Result	A word variable that will be assigned with the number of bytes actually
	written to the socket.

	When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned. When there is no space, 0 will be returned.
Socket	The socket number you want to send data to (0-3).
Var	The name of a string variable.
Param	A parameter that might be 0 to send only the string or 255, to send the string with an additional CR + LF This option was added because many protocols expect CR + LF after the
	string.

The TCPwriteStr function is a special variant of the TCPwrite function. It will use TCPWrite to send the data.

See also

CONFIG TCPIP [559], GETSOCKET [717], SOCKETCONNECT [886], SOCKETSTAT [896], TCPWRITE [924], TCPREAD [924], CLOSESOCKET [439], SOCKETLISTEN [889]

Example

· ·		
SMTP.BAS (c) 2002 MCS Electronics sample that show how to send an email with SMTP protocol		
<pre>\$regfile = "m161def.dat" processor</pre>	,	used
<pre>\$crystal = 4000000 crystal</pre>	1	used
<pre>\$baud = 19200 \$lib "tcpip.lbx" the name of the tcp ip lib</pre>		baud rate specify
<pre>'W3100A constants Const Sock_stream = \$01 Const Sock_dgram = \$02 Const Sock_ipl_raw = \$03 Raw Sock</pre>	1	Tcp Udp Ip Layer
<pre>Const Sock_macl_raw = \$04 Raw Sock Const Sel_control = 0</pre>		Mac Layer Confirm
Socket Status Const Sel_send = 1 Free Buffer Size		Confirm Tx Confirm Rx
Const Sel_recv = 2 Data Size		CONTILUU KX
<pre>'socket status Const Sock_closed = \$00 Connection Closed Const Sock_arp = \$01</pre>		Status Of Status Of
Arp Const Sock_listen = \$02		Status Of

927

Waiting For Tcp Connection Setup **Const** Sock_synsent = \$03 ' Status Of Setting Up Tcp Connection **Const** Sock_synsent_ack = \$04 ' Status Of Setting Up Tcp Connection Const Sock_synrecv = \$05 ' Status Of Setting Up Tcp Connection **Const** Sock established = \$06 ' Status Of Tcp Connection Established **Const** Sock_close_wait = \$07 ' Status Of Closing Tcp Connection Const Sock_last_ack = \$08 ' Status Of Closing Tcp Connection **Const** Sock fin wait1 = \$09 ' Status Of Closing Tcp Connection **Const** Sock_fin_wait2 = \$0a ' Status Of Closing Tcp Connection ' Status Of **Const** Sock_closing = \$0b Closing Tcp Connection **Const** Sock_time_wait = \$0c ' Status Of Closing Tcp Connection **Const** Sock_reset = \$0d ' Status Of Closing Tcp Connection **Const** Sock_init = \$0e ' Status Of Socket Initialization **Const** Sock udp = \$0f ' Status Of Udp **Const** Sock raw = \$10 ' Status of IP RAW ' for **Const** Debug = -1 sending feeback to the terminal **#if** Debug Print "Start of SMTP demo" #endif Enable Interrupts ' enable interrupts 'specify MAC, IP, submask and gateway 'local port value will be used when you do not specify a port value while creating a connection 'TX and RX are setup to use 4 connections each with a 2KB buffer Config Tcpip = Int0 , Mac = 00.44.12.34.56.78 , Ip = 192.168.0.8 Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx = \$55 , Rx = \$55 'dim the used variables Dim S As String * 50 , I As Byte , J As Byte , Tempw As Word **#if** Debug Print "setup of W3100A complete" #endif 'First we need a socket I = Getsocket(0 , Sock_stream , 5000 , 0) ^ socket numer ^ port **#if** Debug Print "Socket : " ; I 'the socket must return the asked socket number. It returns 255 if there was an error #endif If I = 0 Then ' all ok

928

```
'connect to smtp server
  J = Socketconnect(i, 194.09.0., 25)
                                                             ' smtp
server and SMTP port 25
                     ^socket
                          ^ ip address of the smtp server
                                               ^ port 25 for smtp
   DO NOT FORGET to ENTER a valid IP number of your ISP smtp server
   #if Debug
     Print "Connection : " ; J
     Print S_status(1)
   #endif
   If J = 0 Then
                                                             ' all ok
      #if Debug
        Print "Connected"
      #endif
     Do
         Tempw = Socketstat(i, 0)
                                                             ' get status
         Select Case Tempw
          Case Sock_established
                                                             ' connection
established
               Tempw = Tcpread(i , S)
                                                             ' read line
               #if Debug
                                                             ' show info
                  Print S
from smtp server
               #endif
               If Left(s , 3) = "220" Then
                                                             ' ok
                  Tempw = Tcpwrite(i , "HELO username{013}{010}" )
' send username
                                              ^^^ fill in username there
                  #if Debug
                   Print Tempw ; " bytes written"
                                                            ' number of
bytes actual send
                  #endif
                  Tempw = Tcpread(i , S)
                                                             ' get
response
                  #if Debug
                    Print S
                                                             ' show
response
                  #endif
                  If Left(s , 3) = "250" Then
                                                             ' ok
                     Tempw = Tcpwrite(i , "MAIL FROM:<tcpip@test.com>
\{013\}\{010\}"\}
                   ' send from address
                     Tempw = Tcpread(i , S)
                                                             ' get
response
                     #if Debug
                        Print S
                     #endif
                     If Left(s , 3) = "250" Then
                                                           ' ok
                        Tempw = Tcpwrite(i , "RCPT TO:<tcpip@test.com>
\{013\}\{010\}"\}
                   ' send TO address
                        Tempw = Tcpread(i , S)
                                                             ' qet
response
                        #if Debug
                           Print S
                        #endif
                        If Left(s , 3) = "250" Then
                                                            ' ok
                           Tempw = Tcpwrite(i , "DATA{013}{010}")
' speicfy that we are going to send data
                           Tempw = Tcpread(i , S)
                                                             ' get
response
                           #if Debug
                             Print S
                           #endif
```

BASCOM Language Reference

```
929
```

```
If Left(s , 3) = "354" Then
                                                                 ' ok
                                Tempw = Tcpwrite(i , "From: tcpip@test.com
\{013\}\{010\}")
                                Tempw = Tcpwrite(i , "To: tcpip@test.com
\{013\}\{010\}")
                                Tempw = Tcpwrite(i , "Subject: BASCOM SMTP
test{013}{010}")
                                Tempw = Tcpwrite(i , "X-Mailer: BASCOM
SMTP{013}{010}")
                                Tempw = Tcpwrite(i , "{013}{010}")
Tempw = Tcpwrite(i , "This is a test email
from BASCOM SMTP{013}{010}")
                                Tempw = Tcpwrite(i , "Add more lines as
needed{013}{010}")
                                Tempw = Tcpwrite(i , ".{013}{010}")
' end with a single dot
                                Tempw = Tcpread(i , S)
                                                                 ' get
response
                                #if Debug
                                  Print S
                                #endif
                                If Left(s , 3) = "250" Then
                                                                 ' ok
                                   Tempw = Tcpwrite(i , "QUIT{013}{010}")
     ' quit connection
                                   Tempw = Tcpread(i , S)
                                   #if Debug
                                      Print S
                                   #endif
                                End If
                             End If
                         End If
                      End If
                   End If
                End If
           Case Sock_close_wait
               Print "CLOSE WAIT"
               Closesocket I
                                                                 ' close the
connection
           Case Sock_closed
               Print "Socket CLOSED"
                                                                 ' socket is
closed
               End
         End Select
      LOOD
   End If
End If
End
                                                                 'end program
```

6.412 TANH

Action

Returns the hyperbole of a single

Syntax

var = TANH(source)

Remarks

930 BASCOM-AVR

Var A numeric variable that is assigned with hyperbole of variable source. Source The single or double variable to get the hyperbole of.

All trig functions work with radians. Use deg2rad and rad2deg to convert between radians and angles.

See Also

RAD2DEG [822], DEG2RAD [649], ATN [403], COS [596], SIN [885], SINH [886], COSH [597]

Example

Show sample 983

6.413 THIRDLINE

Action

Reset LCD cursor to the third line.

Syntax

THIRDLINE

Remarks

NONE

See also

UPPERLINE [945], LOWERLINE [776], FOURTHLINE [693]

Example

Dim A As Byte A = 255 Cls Lcd A Thirdline Lcd A Upperline End

6.414 TIME\$

Action

Internal variable that holds the time.

Syntax TIME\$ = "hh:mm:ss" var = TIME\$

Remarks

The TIME\$ variable is used in combination with the CONFIG CLOCK and CONFIG DATE directive.

The CONFIG CLOCK statement will use the TIMER0 or TIMER2 in async mode to create a 1 second interrupt. In this interrupt routine the _Sec, _Min and _Hour variables are updated. The time format is 24 hours format.

When you assign TIME\$ to a string variable these variables are assigned to the TIME\$ variable.

When you assign the TIME\$ variable with a constant or other variable, the _sec, _Hour and _Min variables will be changed to the new time.

The only difference with VB is that all digits must be provided when assigning the time. This is done for minimal code. You can change this behavior of course.

The async timer is only available in the M103, 90S8535, M163 and M32(3), Mega128, Mega64, Mega8. For other chips it will not work.

As new chips are launched by Atmel, and support is added by MCS, the list above might not be complete. It is intended to serve as an example for chips with a timer that can be used in asynchrone mode. So when your micro has a timer that can be used in asynchrone mode, it should work.

Lo not confuse DATE\$ with the DATE function.

ASM

The following asm routines are called from mcs.lib. When assigning TIME\$: _set_time (calls _str2byte) When reading TIME\$: _make_dt (calls _byte2str)

See also

DATE\$ 625 , CONFIG CLOCK 461 , CONFIG DATE 473

Example

See the sample of **DATE**\$ 625

6.415 TIME

Action

Returns a time-value (String or 3 Byte for Second, Minute and Hour) depending of the Type of the Target

Syntax

bSecMinHour = **Time**(ISecOfDay) bSecMinHour = **Time**(ISysSec) bSecMinHour = **Time**(strTime)

strTime = Time(ISecOfDay)
strTime = Time(ISysSec)
strTime = Time(bSecMinHour)

Remarks

	A BYTE – variable, which holds the Second-value followed by Minute (Byte) and Hour (Byte)
strTime	A Time – String in Format "hh:mm:ss"
lSecOfDay	A LONG – variable which holds Second Of Day (SecOfDay)
lSysSec	A LONG – variable which holds System Second (SysSec)

Converting to a time-string:

The target string strTime must have a length of at least 8 Bytes, otherwise SRAM after the target-string will be overwritten.

Converting to Softclock format (3 Bytes for Second, Minute and Hour):

Three Bytes for Seconds, Minutes and Hour must follow each other in SRAM. The variable-name of the first Byte, that one for Second must be passed to the function.

See also

Date and Time Routines (993), SECOFDAY (849), SYSSEC (915)

Partial Example

Enable Interrupts Config Clock = Soft

```
Dim Strtime As String * 8
Dim Bsec As Byte , Bmin As Byte , Bhour As Byte
Dim Lsecofday As Long
Dim Lsyssec As Long
```

```
' Example 1: Converting defined Clock - Bytes (Second / Minute / Hour)
to Time - String
Bsec = 20 : Bmin = 1 : Bhour = 7
Strtime = Time(bsec)
Print "Time values: Sec=" ; Bsec ; " Min=" ; Bmin ; " Hour=" ; Bhour ; "
converted to string " ; Strtime
```

```
' Time values: Sec=20 Min=1 Hour=7 converted to string 07:01:20
```

```
' Example 2: Converting System Second to Time - String
Lsyssec = 123456789
Strtime = Time(lsyssec)
Print "Time of Systemsecond " ; Lsyssec ; " is " ; Strtime
```

```
' Time of Systemsecond 123456789 is 21:33:09
```

```
' Example 3: Converting Second of Day to Time - String
Lsecofday = 12345
Strtime = Time(lsecofday)
Print "Time of Second of Day " ; Lsecofday ; " is " ; Strtime
' Time of Second of Day 12345 is 03:25:45
```

```
' Example 4: Converting System Second to defined Clock - Bytes (Second /
Minute / Hour)
Lsyssec = 123456789
```

```
Bsec = Time(lsyssec)
Print "System Second " ; Lsyssec ; " converted to Sec=" ; Bsec ; " Min="
; Bmin ; " Hour=" ; Bhour
' System Second 123456789 converted to Sec=9 Min=33 Hour=21
' Example 4: Converting Second of Day to defined Clock - Bytes (Second /
Minute / Hour)
Lsecofday = 12345
Bsec = Time(lsecofday)
Print "Second of Day " ; Lsecofday ; " converted to Sec=" ; Bsec ; "
Min=" ; Bmin ; " Hour=" ; Bhour
' Second of Day 12345 converted to Sec=45 Min=25 Hour=3
```

6.416 TOGGLE

Action

Toggles(inverts) the state of an output pin or bit/Boolean variable. When used on a numeric variable, all bits in the variable are inverted.

Syntax

TOGGLE pin TOGGLE var

Remarks

Any port pin like PORTB.0 or boolean variable. A port pin must be configured as an output pin before TOGGLE will have effect.
A numeric variable like byte, word, integer or long. When you invert a byte, all bits of that byte will be inverted.

With TOGGLE you can simply invert the output state of a port pin. When the pin is driving a relay for example and the relay is OFF, one TOGGLE statement will turn the relays ON. Another TOGGLE will turn the relays OFF again.

When TOGGLE is used with a variable of the type Byte, Word, Integer or Long, all bits in the variable are toggled. It has the same effect as using the EXOR boolean operand with \$FF, \$FFFF or \$FFFFFFFF

Example:

Toggle Var_byte has the same effect as

Var_byte = Var_byte XOR &HFF

New AVR chips have an enhanced port architecture which allow a toggle of the PORT by setting the PIN register to 1. The DAT files have a setting under the [DEVICE] section named NEWPORT.

When the value is 1, the PIN register will be set to toggle the PORT pin. When the NEWPORT value is set to 0, an XOR will be used to toggle the port pin.

TOGGLE can also be used on numeric variables. It will invert all bits in the variable. It has the same effect as NOT. var = NOT var ' invert all bits

See also

CONFIG PORT 528

ASM

NONE

```
Example
'Bascom Help, Nov 16, 2008
'ToggleNov15 2008.bas
'Program example for use in the Help-files for
τ.
       TOGGLE
'Program has been compiled and tested using Bascom 1.11.9.2.003
'Nard Awater, November 16, 2008
$baud = 19200
$crystal = 16000000
$regfile = "m32def.dat"
\$hwstack = 40
\$swstack = 20
$framesize = 20
Dim B As Byte , W As Word , I As Integer , L As Long
Led Alias Portb.0
                                                              'the anode
of the LED connected to PortB.0, cathode with resistor (470 Ohm) to
ground
Config Pinb.0 = Output
B = 0
Reset Led
'Toggle the led
Do
  Print "Led is off "
   Waitms 500
   Toggle Led
   Print "Led is on "
   Waitms 500
   Toggle Led
   Incr B
Loop Until B = 5
'Toggle a bit in a variable
B = &B11110000
                                                              'assign a
new value: 240 in decimal
Toggle B.0
Print "B in decimal " ; B
                                                              ' print it:
result = 241 ; bit0 is set
                                                              ' print it:
Print Bin(b)
result = 11110001
Toggle B.0
Print "B in decimal " ; B
                                                              ' print it:
result = 240 ; bit0 is reset
Print Bin(b)
                                                              ' print it:
result = 11110000
W = \&HOOOF
                                                              '15 in
```

```
decimal
I = \&HOOFF
                                                                  '255 in
decimal
L = &H00CC00DD
                                                                  '13369565 in
decimal
Toggle W
Print "toggled W= " ; W
                                                                  ' print it:
result = 65520
Print Hex(w)
                                                                  ' print it:
result = &HFFF0
Toggle I
Print "toggled I= " ; I
                                                                  ' print it:
result = -256; two's complement !
Print Hex(i)
                                                                  ' print it:
result = &HFF00
Toggle L
Print "toggled L= " ; L
result = -13369566 ; two's complement !
                                                                  ' print it:
                                                                  ' print it:
Print Hex(1)
result = &HFF33FF22
```

End

6.417 TRIM

Action

Returns a copy of a string with leading and trailing blanks removed

Syntax

var = TRIM(org)

Remarks

Var	String that receives the result.
Org	The string to remove the spaces from

TRIM is the same as a LTRIM() and RTRIM() call. It will remove the spaces on the left and right side of the string.

See also

RTRIM 848 , LTRIM 763

Partial Example

```
Dim S As String * 6
S =" AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

BASCOM-AVR

6.418 UCASE

936

Action

Converts a string in to all upper case characters.

Syntax

Target = **UCASE**(source)

Remarks

Target	The string that is assigned with the upper case string of string target.
Source	The source string.

See also

LCASE 751

ASM

The following ASM routines are called from MCS.LIB : _UCASE X must point to the target string, Z must point to the source string. The generated ASM code : (can be different depending on the micro used) ;##### Z = Ucase(s) Ldi R30,\$60 Ldi R31,\$00 ; load constant in register Ldi R26,\$6D Rcall _Ucase

Example

<pre>\$regfile = "m48def.dat"</pre>	' specify
the used micro \$crystal = 4000000	' used
crystal frequency	used
\$baud = 19200	' use baud
rate \$hwstack = 32	' default
use 32 for the hardware stack	deraure
\$swstack = 10	' default
use 10 for the SW stack \$framesize = 40 use 40 for the frame space	' default

```
Dim S As String * 12 , Z As String * 12
S = "Hello World"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z
End
```

© 2011 MCS Electronics

6.419 UDPREAD

Action

Reads data via UDP protocol.

Syntax

Result = **UDPREAD**(socket , var, bytes)

Remarks

Result	A byte variable that will be assigned with 0 , when no errors occurred. When an error occurs, the value will be set to 1 .
	When there are not enough bytes in the reception buffer, the routine will wait until there is enough data or the socket is closed.
socket	The socket number you want to read data from (0-3).
Var	The name of the variable that will be assigned with the data from the socket.
Bytes	The number of bytes to read.

Reading strings is not supported for UDP.

When you need to read a string you can use the OVERLAY option of DIM.

There will be no check on the length so specifying to receive 2 bytes for a byte will overwrite the memory location after the memory location of the byte.

The socketstat function will return a length of the number of bytes + 8 for UDP. This because UDP sends also a 8 byte header. It contains the length of the data, the IP number of the peer and the port number.

The UDPread function will fill the following variables with this header data:

Peersize, PeerAddress, PeerPort

You need to DIM these variables in your program when you use UDP. Use the following line :

Dim Peersize As Integer , Peeraddress As Long , Peerport As Word

Make sure you maintain the shown order.

See also

CONFIG TCPIP [559], GETSOCKET [717], SOCKETCONNECT [886], SOCKETSTAT [890], TCPWRITE [924], TCPWRITESTR [925], CLOSESOCKET [439], SOCKETLISTEN [889], UDPWRITE [940], UDPWRITESTR [941]

Example

'name	: udptest.bas
'copyright	: (c) 1995-2005, MCS Electronics
'purpose	: start the easytcp.exe program after the chip

```
is programmed and
                           press UDP button
'micro
                         : Megal61
'suited for demo
                         : no
'commercial addon needed : yes
·-----
                                        _____
_____
$regfile = "m161def.dat"
                                                            ' specify
the used micro
                                                            ' used
Scrystal = 4000000
crystal frequency
                                                            ' use baud
$baud = 19200
rate
                                                            ' default
$hwstack = 32
use 32 for the hardware stack
\$swstack = 10
                                                            ' default
use 10 for the SW stack
$framesize = 40
                                                            ' default
use 40 for the frame space
                                                            ' Tcp
Const Sock_stream = $01
                                                            ' Udp
Const Sock_dgram = $02
Const Sock_ipl_raw = $03
                                                            ' Ip Layer
Raw Sock
Const Sock_macl_raw = $04
                                                            ' Mac Layer
Raw Sock
Const Sel_control = 0
                                                            ' Confirm
Socket Status
                                                            ' Confirm Tx
Const Sel_send = 1
Free Buffer Size
Const Sel_recv = 2
                                                            ' Confirm Rx
Data Size
'socket status
Const Sock_closed = $00
                                                            ' Status Of
Connection Closed
Const Sock_arp = $01
                                                            ' Status Of
Arp
Const Sock_listen = $02
                                                            ' Status Of
Waiting For Tcp Connection Setup
Const Sock_synsent = $03
                                                            ' Status Of
Setting Up Tcp Connection
Const Sock_synsent_ack = $04
                                                            ' Status Of
Setting Up Tcp Connection
Const Sock_synrecv = $05
                                                            ' Status Of
Setting Up Tcp Connection
Const Sock_established = $06
                                                            ' Status Of
Tcp Connection Established
                                                            ' Status Of
Const Sock_close_wait = $07
Closing Tcp Connection
Const Sock_last_ack = $08
                                                            ' Status Of
Closing Tcp Connection
Const Sock_fin_wait1 = $09
                                                            ' Status Of
Closing Tcp Connection
Const Sock_fin_wait2 = $0a
                                                            ' Status Of
Closing Tcp Connection
Const Sock_closing = $0b
                                                            ' Status Of
Closing Tcp Connection
Const Sock_time_wait = $0c
                                                            ' Status Of
Closing Tcp Connection
                                                            ' Status Of
Const Sock_reset = $0d
Closing Tcp Connection
Const Sock_init = $0e
                                                            ' Status Of
```

Socket Initialization **Const** Sock_udp = \$0f ' Status Of Udp **Const** Sock raw = \$10 ' Status of IP RAW **\$lib** "tcpip.lbx" ' specify the tcpip library **Print** "Init , set IP to 192.168.0.8" ' display a message ' before we Enable Interrupts use config tcpip , we need to enable the interrupts **Config** Tcpip = **Int0**, Mac = 12.128.12.34.56.78, Ip = 192.168.0.8, Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx = \$55 , Rx = \$55'Use the line below if you have a gate way 'Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx = \$55 , Rx = \$55 Dim Idx As Byte ' socket number Dim Result As Word ' result Dim S(80) As Byte Dim Sstr As String * 20 Dim Temp As Byte , Temp2 As Byte ' temp bytes ·-----_____ 'When you use UDP, you need to dimension the following variables in exactly the same order ! Dim Peersize As Integer , Peeraddress As Long , Peerport As Word !_____ _ _ _ _ _ _ _____ _____ Declare Function Ipnum(ip As Long) As String ' a handy function 'like with TCP, we need to get a socket first 'note that for UDP we specify sock_dgram ' get socket Idx = Getsocket(idx , Sock_dgram , 5000 , 0) for UDP mode, specify port 5000 Print "Socket " ; Idx ; " " ; Idx 'UDP is a connection less protocol which means that you can not listen, connect or can get the status 'You can just use send and receive the same way as for TCP/IP. 'But since there is no connection protocol, you need to specify the destination IP address and port 'So compare to TCP/IP you send exactly the same, but with the addition of the IP and PORT Do ' wait for Temp = Inkey() terminal input If Temp = 27 Then ' ESC pressed Sstr = "Hello" Result = Udpwritestr(192.168.0.3 , 5000 , Idx , Sstr , 255) End If Result = Socketstat(idx , Sel_recv) ' get number of bytes waiting If Result > 0 Then Print "Bytes waiting : " ; Result

```
Temp2 = Result - 8
                                                             'the first 8
bytes are always the UDP header which consist of the length, IP number
and port address
      Temp = Udpread(idx , S(1) , Result)
                                                             ' read the
result
      For Temp = 1 To Temp2
          Print S(temp) ; " " ;
                                                             ' print
result
      Next
      Print
      Print Peersize ; " " ; Peeraddress ; " " ; Peerport
                                                             ' these are
assigned when you use UDPREAD
     Print Ipnum(peeraddress)
                                                             ' print IP
in usual format
     Result = Udpwrite(192.168.0.3, Peerport, Idx, S(1), Temp2)
  ' write the received data back
  End If
LOOD
'the sample above waits for data and send the data back for that reason
temp2 is subtracted with 8, the header size
'this function can be used to display an IP number in normal format
Function Ipnum(ip As Long) As String
   Local T As Byte , J As Byte
    Ipnum = ""
   For J = 1 To 4
     T = Ip And 255
     Ipnum = Ipnum + Str(t)
      If J < 4 Then Ipnum = Ipnum + "."</pre>
      Shift Ip , Right , 8
   Next
End Function
End
```

6.420 UDPWRITE

Action

Write UDP data to a socket.

Syntax

Result = **UDPwrite**(IP, port, socket , var , bytes) Result = **UDPwrite**(IP, port, socket , EPROM, address , bytes)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket.	
	When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned.	
	When there is no space, 0 will be returned.	
IP	The IP number you want to send data to.	
	Use the format 192.168.0.5 or use a LONG variable that contains the IP number.	
Port	The port number you want to send data too.	

940

Socket	The socket number you want to send data to(0-3).	
Var	A constant string like "test" or a variable.	
	When you send a constant string, the number of bytes to send does not need to be specified.	
Bytes	A word variable or numeric constant that specifies how many bytes must be send.	
Address	The address of the data stored in the chips internal EEPROM. You need to specify EPROM too in that case.	
EPROM	An indication for the compiler so it knows that you will send data from EPROM.	

The UDPwrite function can be used to write data to a socket that is stored in EEPROM or in memory.

When you want to send data from an array, you need to specify the element : var (idx) for example.

Note that UDPwrite is almost the same as TCPwrite. Since UDP is a connection-less protocol, you need to specify the IP address and the port number.

UDP only requires an opened socket. The is no connect or close needed.

See also

CONFIG TCPIP 559, GETSOCKET 777, SOCKETCONNECT 888, SOCKETSTAT 889, TCPWRITESTR 928, TCPREAD 924, CLOSESOCKET 439, SOCKETLISTEN 889, UDPWRITESTR 941, UDPREAD 937)

Example

See UDPwriteStr 94

6.421 UDPWRITESTR

Action

Sends a string via UDP.

Syntax

Result = **UDPwriteStr**(IP, port, socket , var , param)

Remarks

Result	A word variable that will be assigned with the number of bytes actually written to the socket.	
	When the free transmission buffer is large enough to accept all the data, the result will be the same as BYTES. When there is not enough space, the number of written bytes will be returned.	
	When there is no space, 0 will be returned.	
IP	The IP number you want to send data to.	
	Use the format 192.168.0.5 or use a LONG variable that contains the IP	

	number.	
Port	The port number you want to send data too.	
Socket	The socket number you want to send data to (0-3).	
Var	The name of a string variable.	
Param	A parameter that might be 0 to send only the string or 255, to send the string with an additional CR + LF	
	This option was added because many protocols expect $CR + LF$ after the string.	

The UDPwriteStr function is a special variant of the UDPwrite function. It will use UDPWrite to send the data.

See also

CONFIG TCPIP [559], GETSOCKET [717], SOCKETCONNECT [888], SOCKETSTAT [890], TCPWRITE [924], TCPREAD [924], CLOSESOCKET [439], SOCKETLISTEN [889], UDPWRITE [940], UDPREAD [937]

Example

<pre>: udptest.bas : (c) 1995-2005, MCS Electronics : start the easytcp.exe program after the chip press UDP button : Megal61 : no : yes</pre>	
• yes	
' specify ' used ' use baud ' default ' default ' default	
' Tcp ' Udp ' Ip Layer ' Mac Layer ' Confirm ' Confirm Tx ' Confirm Rx	

'socket status **Const** Sock_closed = \$00 ' Status Of Connection Closed **Const** Sock_arp = \$01 ' Status Of Arp **Const** Sock_listen = \$02 ' Status Of Waiting For Tcp Connection Setup **Const** Sock synsent = \$03 ' Status Of Setting Up Tcp Connection ' Status Of **Const** Sock_synsent_ack = \$04 Setting Up Tcp Connection **Const** Sock_synrecv = \$05 ' Status Of Setting Up Tcp Connection **Const** Sock established = \$06 ' Status Of Tcp Connection Established ' Status Of **Const** Sock_close_wait = \$07 Closing Tcp Connection ' Status Of Const Sock_last_ack = \$08 Closing Tcp Connection Const Sock_fin_wait1 = \$09 ' Status Of Closing Tcp Connection **Const** Sock_fin_wait2 = \$0a ' Status Of Closing Tcp Connection **Const** Sock_closing = \$0b ' Status Of Closing Tcp Connection **Const** Sock time wait = \$0c ' Status Of Closing Tcp Connection **Const** Sock reset = \$0d ' Status Of Closing Tcp Connection Const Sock_init = \$0e ' Status Of Socket Initialization ' Status Of **Const** Sock_udp = \$0f Udp ' Status of **Const** Sock_raw = \$10 IP RAW \$lib "tcpip.lbx" ' specify the tcpip library **Print** "Init , set IP to 192.168.0.8" ' display a message Enable Interrupts ' before we use config tcpip , we need to enable the interrupts Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.255.255.0 , Gateway = 0.0.0.0 , Localport = 1000 , Tx = \$55 , Rx = \$55'Use the line below if you have a gate way 'Config Tcpip = Int0 , Mac = 12.128.12.34.56.78 , Ip = 192.168.0.8 , Submask = 255.255.255.0 , Gateway = 192.168.0.1 , Localport = 1000 , Tx = \$55 , Rx = \$55 Dim Idx As Byte ' socket number ' result Dim Result As Word Dim S(80) As Byte Dim Sstr As String * 20 Dim Temp As Byte , Temp2 As Byte ' temp bytes ·_____ _____ 'When you use UDP, you need to dimension the following variables in exactly the same order ! Dim Peersize As Integer , Peeraddress As Long , Peerport As Word

944

```
_____
Declare Function Ipnum(ip As Long) As String
                                                            ' a handy
function
'like with TCP, we need to get a socket first
'note that for UDP we specify sock dgram
                                                        ' get socket
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)
for UDP mode, specify port 5000
Print "Socket " ; Idx ; " " ; Idx
'UDP is a connection less protocol which means that you can not listen,
connect or can get the status
'You can just use send and receive the same way as for TCP/IP.
'But since there is no connection protocol, you need to specify the
destination IP address and port
'So compare to TCP/IP you send exactly the same, but with the addition
of the IP and PORT
Do
  Temp = Inkey()
                                                            ' wait for
terminal input
  If Temp = 27 Then
                                                            ' ESC
pressed
     Sstr = "Hello"
     Result = Udpwritestr(192.168.0.3 , 5000 , Idx , Sstr , 255)
  End If
  Result = Socketstat(idx , Sel_recv)
                                                            ' get number
of bytes waiting
   If Result > 0 Then
      Print "Bytes waiting : " ; Result
     Temp2 = Result - 8
                                                            'the first 8
bytes are always the UDP header which consist of the length, IP number
and port address
     Temp = Udpread(idx , S(1) , Result)
                                                            ' read the
result
      For Temp = 1 To Temp2
         Print S(temp) ; " " ;
                                                            ' print
result
     Next
      Print
     Print Peersize ; " " ; Peeraddress ; " " ; Peerport
                                                            ' these are
assigned when you use UDPREAD
     Print Ipnum(peeraddress)
                                                            ' print IP
in usual format
     Result = Udpwrite(192.168.0.3, Peerport, Idx, S(1), Temp2)
  ' write the received data back
  End If
Loop
'the sample above waits for data and send the data back for that reason
temp2 is subtracted with 8, the header size
'this function can be used to display an IP number in normal format
Function Ipnum(ip As Long) As String
   Local T As Byte , J As Byte
    Ipnum = ""
   For J = 1 To 4
     T = Ip And 255
      Ipnum = Ipnum + Str(t)
      If J < 4 Then Ipnum = Ipnum + "."</pre>
      Shift Ip , Right , 8
   Next
End Function
End
```

```
© 2011 MCS Electronics
```

6.422 UPPERLINE

Action

Reset LCD cursor to the upper line.

Syntax

UPPERLINE

Remarks

Optional you can also use the LOCATE statement.

See also

LOWERLINE 778, THIRDLINE 930, FOURTHLINE 693, LCD 752, CLS 433, LOCATE 771

Example

Dim A As Byte A = 255 Cls Lcd A Thirdline Lcd A Upperline End

6.423 VAL

Action

Converts a string representation of a number into a number.

Syntax

var = VAL(s)

Remarks

Var	A numeric variable that is assigned with the value of s.
S	Variable of the string type.

It depends on the variable type which conversion routine will be used. Single and Double conversion will take more code space.

When you use INPUT, internal the compiler also uses the VAL routines.

In order to safe code, there are different conversion routines. For example BINVAL and HEXVAL are separate routines.

While they could be added to the compiler, it would mean a certain overhead as they might never be needed.

With strings as input or the INPUT statement, the string is dynamic and so all conversion routines would be needed.

See also

STR 1917), HEXVAL 7227, HEX 7227, BIN 417, BINVAL 4127, STR2DIGITS 9127

Example

```
$regfile = "m48def.dat"
                                                               ' specify
the used micro
$crystal = 8000000
                                                               ' used
crystal frequency
$baud = 19200
                                                               ' use baud
rate
\$hwstack = 32
                                                               ' default
use 32 for the hardware stack
                                                               ' default
\$swstack = 10
use 10 for the SW stack
$framesize = 40
                                                               ' default
use 40 for the frame space
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
Dim A As Byte , S As String * 10
S = "123"
A = Val(s)
                                                               'convert
string
Print A
                                                               ' 123
S = "12345678"
Dim L As Long
L = Val(s)
Print L
```

6.424 VARPTR

End

Action

Retrieves the memory-address of a variable.

Syntax

var = VARPTR(var2)
var = VARPTR("var3")

Remarks

Var	The variable that receives the address of var2.	
Var2	A variable to retrieve the address from.	
var3	A constant	

Sometimes you need to know the address of a variable, for example when you like to peek at it's memory content.

The VARPTR() function assigns this address.

See also

Example Dim W As Byte Print Hex(varptr(w)) ' 0060

6.425 VER

Action

Returns the AVR-DOS version

Syntax

result = VER()

Remarks

Result	A numeric variable that is assigned with the AVR-DOS version. The
	version number is a byte and the first release is version 1.

When you have a problem, MCS can ask you for the AVR-DOS version number. The VER() function can be used to return the version number then.

See also

INITFILESYSTEM [737), OPEN [794), CLOSE [437), FLUSH [689), PRINT [810), LINE INPUT [762), LOC [766], LOF [767], EOF [687], FREEFILE [698], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [666], GET [697], PUT [819], FILEDATE [688], FILETIME [687], FILEDATETIME [686], DIR [656], WRITE [953], INPUT [744]

The <u>VERSION</u> () function is something different. It is intended to include compile time info into the program.

<u>ASM</u>

Calls	_AVRDOSVer
Input	-
Output	R16 loaded with value

Example

Print Ver()

6.426 VERSION

Action

Returns a string with the date and time of compilation.

Syntax

Var = **VERSION**(frm)

Remarks

Var is a string variable that is assigned with a constant. This version constant is set at compilation time to MM-DD-YY hh:nn:ss

Where MM is the month, DD the day of the month, YY the year. hh is the hour is 24-hour format, nn the minutes, and ss the seconds. When frm is set to 1, the format date will be shown in European DD-MM-YY hh:nn:ss format. When frm is set to 2, the version info from \$VERSION will be used.

When frm is a string constant, the string constant will be used.

While it is simple to store the version of your program in the source code, it is harder to determine which version was used for a programmed chip.

The Version() function can print this information to the serial port, or to an LCD display.

See Also

VER 947 , \$VERSION

Example

Print Version()

6.427 WAIT

Action

Suspends program execution for a given time.

Syntax

WAIT seconds

Remarks

seconds The number of seconds to wait.

No accurate timing is possible with this command. When you use interrupts, the delay may be extended.

See also

DELAY 650 , WAITMS 950

Example

WAIT 3 wait for three seconds Print "*"

6.428 WAITKEY

Action

Wait until a character is received.

Syntax

var = WAITKEY()
var = WAITKEY(#channel)

Remarks

var Variable that receives the ASCII value of the serial buffer	
	Can be a numeric variable or a string variable.
#channel	The channel used for the software UART.

While Inkey() returns a character from the serial buffer too, INKEY() continues when there is no character. Waitkey() waits until there is a character received. This blocks your program.

See also

INKEY 7387, ISCHARWAITING 7507

Example

```
_____
_____
                      : inkey.bas
'name
'copyright
                      : (c) 1995-2005, MCS Electronics
'purpose
                     : demo: INKEY , WAITKEY
                     : Mega48
'micro
'suited for demo : yes
'commercial addon needed : no
                                        _____
!_____
_____
$regfile = "m48def.dat"
                                                   ' specify
the used micro
$crystal = 4000000
                                                   ' used
crystal frequency
$baud = 19200
                                                   ' use baud
rate
\$hwstack = 32
                                                   ' default
use 32 for the hardware stack
$swstack = 10
                                                   ' default
use 10 for the SW stack
$framesize = 40
                                                   ' default
use 40 for the frame space
Dim A As Byte , S As String * 2
Do
                                                   'get ascii
  A = Inkey()
value from serial port
  's = Inkey()
  If A > 0 Then
                                                   'we got
something
     Print "ASCII code " ; A ; " from serial"
```

```
End If
Loop Until A = 27
                                                               'until ESC
is pressed
A = Waitkey()
                                                               'wait for a
key
's = waitkey()
Print Chr(a)
'wait until ESC is pressed
Do
Loop Until Inkey() = 27
'When you need to receive binary data and the bibary value {\tt 0} ,
'you can use the IScharwaiting() function.
'This will return 1 when there is a char waiting and 0 if there is no
char waiting.
'You can get the char with inkey or waitkey then.
End
```

6.429 WAITMS

Action

Suspends program execution for a given time in mS.

Syntax

WAITMS mS

Remarks

Ms

The number of milliseconds to wait. (1-65535)

No accurate timing is possible with this command. In addition, the use of interrupts can slow this routine.

See also

DELAY (650), WAIT (948), WAITUS (951)

ASM

WaitMS will call the routine _WAITMS. R24 and R25 are loaded with the number of milliseconds to wait. Uses and saves R30 and R31. Depending on the used XTAL the asm code can look like : _WaitMS: WaitMS1F: Push R30 ; save Z Push R31 _WaitMS_1: Ldi R30,\$E8 ;delay for 1 mS Ldi R31,\$03 WaitMS 2: Sbiw R30,1 ; -1 Brne _WaitMS_2 ; until 1 mS is ticked away Sbiw R24,1 Brne _WaitMS_1 ; for number of mS Pop R31 Pop R30

Ret

Example

WAITMS 10 'wait for 10 mS Print "*"

6.430 WAITUS

Action

Suspends program execution for a given time in uS.

Syntax

WAITUS uS

Remarks

US	The number of microseconds to wait. (1-65535)
	This must be a constant. Not a variable! In version 1.12.x.x and higher you can use a variable as well.

No accurate timing is possible with this command. In addition, the use of interrupts can slow down this routine.

The minimum delay possible is determined by the used frequency. The number of cycles that are needed to set and save registers is 17.

When the loop is set to 1, the minimum delay is 21 uS. In this case you can better use a NOP that generates 1 clock cycle delay.

At 4 MHz the minimum delay is 5 uS. So a waitus 3 will also generate 5 uS delay. Above these values the delay will become accurate.

When you really need an accurate delay you should use a timer. Set the timer to a value and poll until the overflow flag is set. The disadvantage is that you can not use the timer for other tasks during this hardware delay.

The philosophy behind BASCOM is that it should not use hardware resources unless there is no other way to accomplish a task. The WAITUS is used internal by some statements. It was added to the BASCOM

statements but it does NOT accept a variable. Only a constant is accepted.

When you use a variable for the delay, the accuracy will depend on the value of the oscillator. A higher oscillator value will result in a better accuracy. When you clock the micro with 1 MHz, it means that you have a million cycles per second. A NOP instructions takes 1 cycle and will thus delay for 1 us. If you must load registers, and call a subroutine, it means you need more cycles.

At 8 Mhz the following results were measured:

With a constant lus : 2.6 us l0us : 11.5 us 25us : 26.3 us With a variable lus : 1.4 us l0us : 10.2 us 25us : 25.0 us

See also

DELAY 650 , WAIT 948 , WAITMS 950

Example

WAITUS 10 'wait for 10 uS Print "*"

6.431 WHILE-WEND

Action

Executes a series of statements in a loop, as long as a given condition is true.

Syntax

WHILE condition statements WEND

Remarks

If the condition is true then any intervening statements are executed until the WEND statement is encountered.

BASCOM then returns to the WHILE statement and checks the condition. If it is still true, the process is repeated.

If it is not true, execution resumes with the statement following the WEND statement.

So in contrast with the DO-LOOP structure, a WHILE-WEND condition is tested first so that if the condition fails, the statements in the WHILE-WEND structure are never executed.

See also

DO-LOOP 664

Example

_____ _____ 'name : while_w.bas : (c) 1995-2005, MCS Electronics 'copyright : demo: WHILE, WEND 'purpose : Mega48 'micro 'suited for demo : yes 'commercial addon needed : no ·_____ _____ _ _ _ _ _ _ _____ \$regfile = "m48def.dat" ' specify the used micro \$crystal = 4000000' used

```
crystal frequency
$baud = 19200
rate
$hwstack = 32
use 32 for the hardware stack
$swstack = 10
use 10 for the SW stack
$framesize = 40
use 40 for the frame space
Dim A As Byte
A = 1
While A < 10
expression
Print A
Incr A
```

'assign var 'test 'print var 'increase by

' use baud

' default

' default

' default

'continue

6.432 WRITE

one

Wend loop End

Action

Writes data to a sequential file

Syntax

WRITE #ch , data [,data1]

Remarks

	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Data , data1	A variable who's content are written to the file.

When you write a variables value, you do not write the binary representation but the ASCII representation. When you look in a file it contains readable text.

When you use PUT, to write binary info, the files are not readable or contain unreadable characters.

Strings written are surrounded by string delimiters "". Multiple variables written are separated by a comma. Consider this example :

Dim S as String * 10 , W as Word S="hello" : W = 100 OPEN "test.txt" For OUTPUT as #1 WRITE #1, S , W CLOSE #1

The file content will look like this : "hello",100 Use INPUT to read the values from value.

See also

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [695], FILEATTR [684], SEEK [850], BSAVE [420], BLOAD [416], KILL [757], DISKFREE [660], GET [697], PUT [819], FILEDATE [685], FILETIME [687], FILEDATETIME [686], DIR [656], WRITE [953], INPUT [744]

ASM

Calls	_FileWriteQuotationMark	_FileWriteDecInt
	_FileWriteDecByte	_FileWriteDecWord
	_FileWriteDecLong	_FileWriteDecSingle
Input	Z points to variable	
Output		

Partial Example

Dim S As String * 10 , W As Word , L As Long

```
S = "write"
Open "write.dmo"for Output As #2
Write #2 , S , W , L
also supported
Close #2
```

Open "write.dmo"for Input As #2
Input #2 , S , W , L
also supported
Close #2
Print S ; " " ; W ; " " ; L

' write is

' write is

6.433 WRITEEEPROM

Action

Write a variables content to the DATA EEPROM.

Syntax

WRITEEEPROM var , address

Remarks

var	The name of the variable that must be stored
address	The address in the EEPROM where the variable must be stored. A new option is that you can provide a label name for the address. See example 2.

This statement is provided for compatibility with BASCOM-8051.

You can also use : Dim V as Eram Byte 'store in EEPROM Dim B As Byte 'normal variable B = 10V = B 'store variable in EEPROM

When you use the assignment version, the data types must be the same!

According to a data sheet from ATMEL, the first location in the EEPROM with address 0, can be overwritten during a reset. It is advised not to use this location.

For security, register R23 is set to a magic value before the data is written to the EEPROM.

All interrupts are disabled while the EEPROM data is written. Interrupts are enabled automatic when the data is written.

It is advised to use the Brownout circuit that is available on most AVR processors. This will prevent that data is written to the EEPROM when the voltage drops under the specified level.

When data is written to the EEPROM, all interrupts are disabled, and after the EEPROM has been written, the interrupts are re-enabled.

In the XMEGA, you need to set the mode to mapped : <u>CONFIG EEPROM</u> [49th] = MAPPED.

See also

READEEPROM 830

ASM

NONE

Example

```
_____
                       : eeprom2.bas
'name
'copyright
                        : (c) 1995-2005, MCS Electronics
'purpose
                       : shows how to use labels with READEEPROM
                       : Mega48
'micro
'suited for demo
                       : yes
'commercial addon needed : no
!_____
                            _____
_____
$regfile = "m48def.dat"
                                                       ' specify
the used micro
$crystal = 4000000
                                                       ' used
crystal frequency
$baud = 19200
                                                       ' use baud
rate
Shwstack = 32
                                                       ' default
use 32 for the hardware stack
Sswstack = 10
                                                       ' default
use 10 for the SW stack
$framesize = 40
                                                       ' default
use 40 for the frame space
'first dimension a variable
Dim B As Byte
Dim Yes As String * 1
'Usage for readeeprom and writeeprom :
'readeeprom var, address
'A new option is to use a label for the address of the data
'Since this data is in an external file and not in the code the eeprom
data
```

956

```
'should be specified first. This in contrast with the normal DATA lines
which must
'be placed at the end of your program!!
'first tell the compiler that we are using EEPROM to store the DATA
$eeprom
'the generated EEP file is a binary file.
'Use $EEPROMHEX to create an Intel Hex file usable with AVR Studio.
'$eepromhex
'specify a label
Label1:
Data 1 , 2 , 3 , 4 , 5
Label2:
Data 10 , 20 , 30 , 40 , 50
'Switch back to normal data lines in case they are used
$data
'All the code above does not generate real object code
'It only creates a file with the EEP extension
'Use the new label option
Readeeprom B , Label1
Print B
                                                              'prints 1
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B
                                                              'prints 2
Readeeprom B , Label2
Print B
                                                              'prints 10
Readeeprom B
Print B
                                                              'prints 20
'And it works for writing too :
'but since the programming can interfere we add a stop here
Input "Ready?" , Yes
B = 100
Writeeeprom B , Labell
B = 101
Writeeeprom B
'read it back
Readeeprom B , Labell
Print B
                                                              'prints 100
'Succesive reads will read the next value
'But the first time the label must be specified so the start is known
Readeeprom B
Print B
                                                              'prints 101
End
```

6.434 X10DETECT

Action

Returns a byte that indicates if a X10 Power line interface is found.

Syntax

Result = **X10DETECT(**)

Rema	rks
Result	A variable that will be assigned with 0 if there is no Power Line Interface found.
	1 will be returned if the interface is found, and the detected mains frequency is 50 Hz. 2 will be returned if the interface is found and the detected mains frequency is 60 Hz.

When no TW-523 or other suitable interface is found, the other X10 routines will not work.

See also

CONFIG X10 587, X10SEND 958

Example

```
_____
                       : x10.bas
'name
                       : (c) 1995-2005, MCS Electronics
'copyright
'purpose
                       : example needs a TW-523 X10 interface
                       : Mega48
'micro
'suited for demo
                       : yes
'commercial addon needed : no
! _____
                                    _____
_____
$regfile = "m48def.dat"
                                                       ' specify
the used micro
                                                       ' used
$crystal = 8000000
crystal frequency
$baud = 19200
                                                       ' use baud
rate
\$hwstack = 32
                                                       ' default
use 32 for the hardware stack
$swstack = 10
                                                       ' default
use 10 for the SW stack
$framesize = 40
                                                       ' default
use 40 for the frame space
'define the house code
Const House = "M"
                                                       ' use code
A-P
Waitms 500
                                                       ' optional
delay not really needed
'dim the used variables
Dim X As Byte
'configure the zero cross pin and TX pin
Config X10 = Pind.4 , Tx = Portb.0
            ^--zero cross
                         ^--- transmission pin
'detect the TW-523
```

```
X = X10detect()
                                                              ' 0 means
Print X
error, 1 means 50 Hz, 2 means 60 Hz
Do
   Input "Send (1-32) " , X
   'enter a key code from 1-31
   '1-16 to address a unit
   '17 all units off
   '18 all lights on
   '19 ON
   '20 OFF
   '21 DIM
   '22 BRIGHT
   '23 All lights off
   '24 extended code
   '25 hail request
   '26 hail acknowledge
   '27 preset dim
   '28 preset dim
   '29 extended data analog
   '30 status on
   '31 status off
   '32 status request
  X10send House , X
                                                               ' send the
code
Loop
Dim Ar(4) As Byte
X10send House , X , Ar(1) , 4
                                                              ' send 4
additional bytes
End
```

6.435 X10SEND

Action

Sends a house and key code with the X10 protocol.

Syntax

X10SEND house , code

Remarks

House	The house code in the form of a letter A-P.
	You can use a constant, or you can use a variable
Code	The code or function to send. This is a number between 1-32.

The X10SEND command needs a TW-523 interface. Only ground, TX and Zero Cross, needs to be connected for transmission. Use CONFIG X10 to specify the pins.

X10 is a popular protocol used to control equipment via the mains. A 110 KHz signal is added to the normal 50/60 Hz , 220/110 V power.

Notice that experimenting with 110V-240V can be very dangerous when you do not know exactly what you are doing !!!

958

In the US, X10 is very popular and wide spread. In Europe it is hard to get a TW-523 for 220/230/240 V.

I modified an 110V version so it worked for 220V. On the Internet you can find modification information. But as noticed before, MODIFY ONLY WHEN YOU UNDERSTAND WHAT YOU ARE DOING.

A bad modified device could result in a fire, and your insurance will most likely not pay. A modified device will not pass any CE, or other test.

When the TW-523 is connected to the mains and you use the X10SEND command, you will notice that the LED on the TW-523 will blink.

Code value	Description
1-16	Used to address a unit. X10 can use a maximum of 16 units per house code.
17	All units off
18	All lights on
19	ON
20	OFF
21	DIM
22	BRIGHT
23	All lights off
24	Extended ode
25	Hail request
26	Hail acknowledge
27	Preset dim
28	Preset dim
29	Extended data analog
30	Status on
31	Status off
32	Status request

The following table lists all X10 codes.

At www.x10.com you can find all X10 information. The intension of BASCOM is not to learn you everything about X10, but to show you how you can use it with BASCOM.

See also

CONFIG X10 587, X10DETECT 956, X10SEND 958

Example

See X10DETECT 956



7 International Resellers

7.1 International Resellers

Since the resellers list changes so now and then, it is not printed in this help. You can best look at the list at the MCS website. See <u>MCS Electronics web</u>.

There is always a reseller near you. A reseller can help you in your own language and you are in the same time zone.

Sometimes there are multiple resellers in your country. All resellers have their own unique expertise. For example : industrial, robotics, educational, etc.



8 ASM Libraries and Add-Ons

ASM Libs are libraries that are used by the compiler. They contain machine language statements for various statements and functions.

A library can also be used to modify an existing function. For example when you use a conversion routine num<>string with a byte variable only, the routine from the MCS.LIB has some overhead as it can also convert integers, word and longs.

You can specify the MCSBYTE.LIB or MCSBYTE.LBX library then to override the function from MCS.LIB.

8.1 EXTENDED I2C

Action

Instruct the compiler to use parts of the extended i2c library

Syntax

\$LIB = "i2c_extended.lib"

Remarks

The I2C library was written when the AVR architecture did not have extended registers. The designers of the AVR chips did not preserve enough space for registers. So when they made bigger chips with more ports they ran out of registers. They solved it to use space from the RAM memory and move the RAM memory from &H60 to &H100.

In the free space from &60 to &H100 the new extended register were located.

While this is a practical solution, some ASM instructions could not be used anymore. This made it a problem to use the I2C statements on PORTF and PORTG of the Mega128.

The extended i2c library is intended to use I2C on portF and portG on the M64 and M128.

It uses a bit more space then the normal I2C lib.

Best would be that you use the TWI interface and the i2c_twi library as this uses less code. The disadvantage is that you need fixed pins as TWI used a fix pin for SCL and SDA.

See also

ASM NONE

Example

© 2011 MCS Electronics

```
964
      BASCOM-AVR
    π.
                                 (c) 2005 MCS Electronics
                      This demo shows an example of I2C on the M128 portF
    ' PORTF is an extened port and requires a special I2C driver
    ·_____
    $regfile = "m128def.dat"
                                                                ' the used
    chip
    $crystal = 8000000
    $baud = 19200
                                                                ' baud rate
    $lib "i2c extended.lib"
    Config Scl = Portf.0
                                                                ' we need to
    provide the SCL pin name
    Config Sda = Portf.1
                                                                ' we need to
    provide the SDA pin name
    Dim B1 As Byte , B2 As Byte
    Dim W As Word At B1 Overlay
    I2cinit
                                                                ' we need to
    set the pins in the proper state
    Dim B As Byte , X As Byte
    Print "Mega128 master demo"
    Print "Scan start"
    For B = 1 To 254 Step 2
      I2cstart
      I2cwbyte B
      If Err = 0 Then
         Print "Slave at : " ; B
      End If
      I2cstop
    Next
    Print "End Scan"
    Do
      I2cstart
      I2cwbyte &H70
                                                                ' slave
    address write
                                                                ' write
      I2cwbyte & B10101010
    command
      I2cwbyte 2
      I2cstop
      Print Err
      I2cstart
      I2cwbyte &H71
      I2crbyte B1 , Ack
      I2crbyte B2 , Nack
      I2cstop
      Print "Error : " ; Err
                                                                ' show error
      Waitms 500
                                                                'wait a bit
```

Loop End

8.2 FM24C16

The FM24C16 library is a library that uses a RAMTRON I2C serial EEPROM. Ramtron memory chips are as quick as RAM and can be overwritten almost unlimited times.

An external EEPROM is a safe alternative to the internal EEPROM.

By using : \$lib "fm24c16.lib" The EEPROM read and write routines from the library will be used instead of the internal EEPROM. Thus you can still use : Dim BE as ERAM Byte And you can use READEEPROM and WRITEEEPROM, but instead of using the internal EEPROM, the external I2C EEPROM is used. The lib is for the FM24C16. It uses I2C/TWI.

This library is only included in the full version. It is not included with the DEMO.

8.3 FM24C64_256

The FM24C64_256 library is a library that uses a RAMTRON I2C serial EEPROM. Ramtron memory chips are as quick as RAM and can be overwritten almost unlimited times.

An external EEPROM is a safe alternative to the internal EEPROM.

By using : \$lib "fm24c64_256.lib" The EEPROM read and write routines from the library will be used instead of the internal EEPROM. Thus you can still use : Dim BE as ERAM Byte And you can use READEEPROM and WRITEEEPROM, but instead of using the internal EEPROM, the external I2C EEPROM is used. The lib is for the FM24C64 to FM24C256. It uses I2C/TWI.

This library is only included in the full version. It is not included with the DEMO.

8.4 FM25C256

The FM24C256 library is a library that uses a RAMTRON SPI serial EEPROM. Ramtron memory chips are as quick as RAM and can be overwritten almost unlimited times.

An external EEPROM is a safe alternative to the internal EEPROM. You can also increase the size of the EEPROM this way.

By using : \$lib "fm25c256.lib" The EEPROM read and write routines from the library will be used instead of the internal EEPROM. Thus you can still use : Dim BE as ERAM Byte And you can use READEEPROM and WRITEEEPROM, but instead of using the internal

EEPROM, the external I2C EEPROM is used. The lib is for the FM25C256. It uses SPI

For the SPI you have to define the pins. The pin named fram_so is connected to SO of the FRAM. SI is connected to SI. A sample is shown below. The clock and, cs and SI pins need to be configured as output pins.

Fram_cs Alias Portl.7 : Const Fram_csp = 7 : Const Fram_csport = Portl Fram_so Alias Pind.1 : Const Fram_sop = 1 : Const Fram_soport = Pind Fram_si Alias Portd.0 : Const Fram_sip = 0 : Const Fram_siport = Portd Fram_sck Alias Portl.6 : Const Fram_sckp = 6 : Const Fram_sckport = Portl

Of course you can also use a cheaper normal SPI EEPROM. These are slower and have less write cycles.

This library is only included in the full version. It is not included with the DEMO.

8.5 I2C_TWI

By default BASCOM will use software routines when you use I2C statements. This because when the first AVR chips were introduced, there was no TWI yet. Atmel named it TWI because Philips is the inventor of I2C. But TWI is the same as I2C.

So BASCOM allows you to use I2C on every AVR chip. Most newer AVR chips have build in hardware support for I2C. With the I2C_TWI lib you can use the TWI which has advantages as it require less code.

Read more about I2C in the hardware 173 section.

To force BASCOM to use the TWI, you need to insert the following statement into your code:

\$LIB "I2C_TWI.LBX"

You also need to choose the correct SCL and SDA pins with the CONFIG SCL and CONFIG SDA statements.

The TWI will save code but the disadvantage is that you can only use the fixed SCL and SDA pins.

8.6 MCSBYTE

The numeric<>string conversion routines are optimized when used for byte, integer, word and longs.

When do you use a conversion routine ?

- When you use STR(), VAL() or HEX().
- When you print a numeric variable
- When you use INPUT on numeric variables.

To support all data types the built in routines are efficient in terms of code size. But when you use only conversion routines on bytes there is a overhead.

The mcsbyte.lib library is an optimized version that only support bytes. Use it by including : \$LIB "mcsbyte.lbx" in your code. Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager are.

See also

mcsbyteint.lib

8.7 MCSBYTEINT

The numeric<>string conversion routines are optimized when used for byte, integer, word and longs.

When do you use a conversion routine ?

-When you use STR() , VAL() or HEX().-When you print a numeric variable-When you use INPUT on numeric variables.

To support all data types the built in routines are efficient in terms of code size. But when you use only conversion routines on bytes there is a overhead.

The mcsbyteint.lib library is an optimized version that only support bytes, integers and words.

Use it by including : \$LIB "mcsbyteint.lbx" in your code.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

See also

mcsbyte.lib 966

8.8 TCPIP

The TCPIP library allows you to use the W3100A internet chip from www.iinchip.com

MCS has developed a special development board that can get you started quickly with TCP/IP communication. Look at <u>http://www.mcselec.com</u> for more info.

The tcpip.lbx is shipped with BASCOM-AVR

The following functions are provided:

<u>CONFIG</u> <u>TCPIP</u> ज्ञिजे	Configures the W3100 chip.
GETSOCKET	Creates a socket for TCP/IP communication.
SOCKETCONN	Establishes a connection to a TCP/IP server.
SOCKETSTAT	Returns information of a socket.
TCPWRITE 924	Write data to a socket.

8	
TCPWRITEST R 925	Sends a string to an open socket connection.
TCPREAD 924	Reads data from an open socket connection.
CLOSESOCKE T 439	Closes a socket connection.
SOCKETLISTE	Opens a socket in server(listen) mode.
GETDSTIP 708	Returns the IP address of the peer.
GETDSTPORT	Returns the port number of the peer.
BASE64DEC	Converts Base-64 data into the original data.
BASE64ENC	Convert a string into a BASE64 encoded string.
MAKETCP 782	Encodes a constant or 4 byte constant/variables into an IP number
UDPWRITE 940	Write UDP data to a socket.
<u>UDPWRITEST</u> <u>R</u> 94ि	Sends a string via UDP.
UDPREAD 937	Reads data via UDP protocol.

8.9 LCD

8.9.1 LCD4BUSY

BASCOM supports LCD displays in a way that you can choose all pins random. This is great for making a simple PCB but has the disadvantage of more code usage. BASCOM also does not use the WR-pin so that you can use this pin for other purposes.

The LCD4BUSY.LIB can be used when timing is critical.

The default LCD library uses delays to wait until the LCD is ready. The lcd4busy.lib is using an additional pin (WR) to read the status flag of the LCD.

The db4-db7 pins of the LCD must be connected to the higher nibble of the port.

The other pins can be defined.

' (c) 2004 MCS Electronics

' lcd4busy.bas shows how to use LCD with busy check

۱_____

'code tested on a 8515 \$regfile="8515def.dat"

'stk200 has 4 MHz \$crystal= 4000000

'define the custom library 'uses 184 hex bytes total

\$lib"lcd4busy.lib"

'define the used constants 'I used portA for testing Const _lcdport =Porta Const _lcdddr =Ddra Const _lcdin =Pina Const _lcd_e = 1 Const _lcd_rw = 2 Const _lcd_rs = 3

'this is like always, define the kind of LCD ConfigLcd= 16 * 2

'and here some simple lcd code Cls Lcd"test" Lowerline Lcd"this" End

8.9.2 LCD4.LIB

The built in LCD driver for the PIN mode is written to support a worst case scenario where you use random pins of the microprocessor to drive the LCD pins.

This makes it easy to design your PCB but it needs more code. When you want to have less code you need fixed pins for the LCD display.

With the statement \$LIB "LCD4.LBX" you specify that the LCD4.LIB will be used.

The following connections are used in the asm code:

Rs = PortB.0 RW = PortB.1 we dont use the R/W option of the LCD in this version so connect to ground E = PortB.2 E2 = PortB.3 optional for lcd with 2 chips Db4 = PortB.4 the data bits must be in a nibble to save code Db5 = PortB.5 Db6 = PortB.6 Db7 = PortB.7

You can change the lines from the lcd4.lib file to use another port. Just change the address used : .EQU LCDDDR=\$17 ; change to another address for DDRD (\$11) .EQU LCDPORT=\$18 ; change to another address for PORTD (\$12)

See the demo lcdcustom4bit.bas in the SAMPLES dir.

Note that you still must select the display that you use with the <u>CONFIG LCD</u> statement.

See also the <u>lcd42.lib</u> for driving displays with 2 E lines.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

8.9.3 LCD4E2

The built in LCD driver for the PIN mode is written to support a worst case scenario where you use random pins of the microprocessor to drive the LCD pins.

This makes it easy to design your PCB but it needs more code.

When you want to have less code you need fixed pins for the LCD display. With the statement \$LIB "LCD4E2.LBX" you specify that the LCD4.LIB will be used.

The following connections are used in the asm code: Rs = PortB.0 RW = PortB.1 we don't use the R/W option of the LCD in this version so connect to ground E = PortB.2 E2 = PortB.3 the second E pin of the LCD Db4 = PortB.4 the data bits must be in a nibble to save code Db5 = PortB.5 Db6 = PortB.6 Db7 = PortB.7

You can change the lines from the lcd4e2.lib file to use another port. Just change the address used : .EQU LCDDDR=\$17 ; change to another address for DDRD (\$11) .EQU LCDPORT=\$18 ; change to another address for PORTD (\$12)

See the demo lcdcustom4bit2e.bas in the SAMPLES dir.

Note that you still must select the display that you use with the <u>CONFIG LCD</u> [516] statement.

See also the <u>lcd4.lib</u> for driving a display with 1 E line.

A display with 2 E lines actually is a display with 2 control chips. They must both be controlled. This library allows you to select the active E line from your code.

In your basic code you must first select the E line before you use a LCD statement.

The initialization of the display will handle both chips.

Note that LBX is a compiled LIB file. In order to change the routines you need the commercial edition with the source code(lib files). After a change you should compile the library with the library manager.

8.9.4 GLCD

GLCD.LIB (LBX) is a library for Graphic LCD's based on the T6963C chip.

The library contains code for LOCATE 77th, CLS 433h, PSET 813h, LINE 759h, CIRCLE 429h, SHOWPIC 883h and SHOWPICE 883h.

8.9.5 **GLCDSED**

GLCDSED.LIB (LBX) is a library for Graphic LCD's based on the SEDXXXX chip.

The library contains modified code for this type of display. New special statements for this display are :

LCDAT 755 SETFONT 855 GLCDCMD 718 GLCDDATA 718

See the SED.BAS sample from the sample directory

8.9.6 **PCF8533**

COLOR LCD

Color displays were always relatively expensive. The mobile phone market changed that. And Display3000.com , sorted out how to connect these small nice colorfully displays.

You can buy brand new Color displays from Display3000. MCS Electronics offers the same displays.

There are two different chip sets used. One chip set is from EPSON and the other from Philips. For this reason there are two different libraries. When you select the wrong one it will not work, but you will not damage anything.

LCD-EPSON.LBX need to be used with the EPSON chip set.

LCD-PCF8833.LBX need to be used with the Philips chip set.

Config Graphlcd = Color, Controlport = Portc, Cs = 1, Rs = 0, Scl = 3, Sda = 2		
Controlport	The port that is used to control the pins. PORTA, PORTB, etc.	
	The chip select pin of the display screen. Specify the pin number. 1 will mean PORTC.1	
	The RESET pin of the display	
	The clock pin of the display	
SDA	The data pin of the display	

As the color display does not have a built in font, you need to generate the fonts vourself.

You can use the Fonteditor 148 for this task.

A number of statements accept a color parameter. See the samples below in **bold**.

LINE	Line(0 , 0) -(130 , 130) , Blue
LCDAT	Lcdat 100 , 0 , "12345678" , Blue , Yellow
CIRCLE	Circle(30,30),10, Blue
PSET	32 , 110 , Black
BOX	Box(10, 30) -(60, 100), Red

See Also

LCD Graphic converter 1020

Example

© 2011 MCS Electronics

' The support for this display has been made possible by Peter Küsters from (c) Display3000 'You can buy the displays from Display3000 or MCS Electronics ----' \$lib "lcd-pcf8833.lbx" 'special color display support \$regfile = "m88def.dat"
change if using different
\$crystal = 8000000 'ATMega 8, processors '8 MHz **Config** GraphIcd = Color, Controlport = Portc, Cs = 1, Rs = 0, Scl = 3, Sda = 2 'here we define the colors **Const** Blue = **&**B00000011 'predefined contants are making programming easier **Const** Yellow = **&**B11111100 **Const** Red = **&**B11100000 **Const** Green = &B00011100 **Const** Black = &B00000000 **Const** White = **&**B11111111 ConstBrightgreen = &B001111ConstDarkgreen = &B00010100ConstDarkred = &B1010000 Brightgreen = &B00111110 Const Darkblue = &B00000010 Brightblue = &B00011111Const **Const** Orange = **&**B11111000 'clear the display CIs 'create a cross Line(0,0) - (130,130), Blue Line(130,0) - (0,130), Red **Waitms** 1000 'show an RLE encoded picture Showpic 0 , 0 , Plaatje Showpic 40 , 40 , Plaatje **Waitms** 1000 'select a font Setfont Color16x16 and show some text Lcdat 100, 0, "12345678", Blue, Yellow **Waitms** 1000 Circle(30, 30), 10, Blue **Waitms** 1000 'make a box Box(10, 30) - (60, 100), Red 'set some pixels Pset 32, 110, Black Pset 38, 110, Black

```
Pset 35, 112, Black
End
```

Plaatje: **\$bgf** "a.bgc"

\$include "color.font"
\$include "color16x16.font"

8.9.7 LCD-EPSON

This chip is compatible with PCF8533 97h.

8.9.8 glcdR7565R

The glcdR7565R.lib is intended to be used with 128x64 displays using the ST7565R chip.

```
_____
                (c) 1995-2011, MCS
                 xm128A1-ST7565R.bas
 This sample demonstrates the ST7565R chip with an Xmega128A1
 Display used : 64128N SERIES from DisplayTech
 this is a parallel display with read/write options
·_____.
                                       _____
$regfile = "xm128a1def.dat"
Scrystal = 3200000
Shwstack = 64
\$swstack = 40
$framesize = 40
'include the following lib and code, the routines will be
replaced since they are a workaround
$lib "xmega.lib"
$external _xmegafix_clear
$external _xmegafix_rol_r1014
'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled
'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1
Config Com1 = 38400 , Mode = Asynchroneous , Parity = None ,
Stopbits = 1 , Databits = 8
$lib "glcdST7565r.lbx"
                                                   ' specify
the used lib
$lib "glcd.lbx"
                                                  ' and this
one of you use circle/line etc
'the display was connected with these pins
Config Graphlcd = 128 * 64eadogm ,dataport=portj, Cs1 = Porth.0
, A0 = Porth.2 , rst= Porth.1 , wr = Porth.3 , Rd = Porth.4, c86=
```

```
porth.6
cls
Setfont Font8x8tt ' set font
dim y as byte
'You can use locate but the columns have a range from 1-128
'When you want to show somthing on the LCD, use the LDAT command
'LCDAT Y , COL, value
Lcdat 1 , 1 , "111111111"
Lcdat 2 , 1 , "ABCDEFGHIJKL1234"
Lcdat 3 , 1 , "MCS Electronics" , 1 ' inverse
Lcdat 4 , 1 , "MCS Electronics"
Waitms 3000
Setfont My12_16 ' use a bigger font
Cls
Lcdat 1 , 1 , "112345678"
                                                             'a
bigger font
Waitms 3000
wait
Line(0 , 0) -(127 , 64) , 1
                                                             'make
line
Waitms 2000 'wait 2 secs
Line(0, 0) - (127, 64), 0
'remove line by inverting the color
For Y = 1 To 20
   Circle(30, 30), Y, 1
growing circle
  Waitms 100
Next
End
```

\$include "font8x8TT.font"
\$include "my12_16.font"

8.9.9 glcdSSD1325_96x64

This lib is for SSD1325 based displays. This lib supports screen 96x64. The lib is based on bascom code from Robert Wolgajew.

SSD1325 is used for OLED displays. Each pixel can have 16 tints. The usual graphic statements are supported. Images such as bitmaps can be converted into 16 grey tone images. The ssd1325 conversion tool you can <u>download</u> from the MCS web server.

The sample below is using porta pins to control BS1 and BS2. Of course you would connect them to VDD directly.

The pins used, and bascom pins names are :

SSD pin	BASCOM pin
WR	WR
RD	RD
D0-D7	PORTx
D/C	A0
RES	RST
RES CS	CS1
VCC	VCC

The VCC pin controls a 12V generator.

Since the display is using a pallet of 16 grey tones, you must specify the foreground and background colors with LCDAT.

-----. (c) 1995-2011 MCS Electronics oled_ssd1325.bas demonstrates OLED display 96x64 with SSD1325 chip Based on bascom SSD1325 code from Robert Wolgajew - - - - - -\$regfile = "m8535.dat" \$crystal = 3686000 \$hwstack = 48 \$swstack = 48 \$framesize = 48 'normally the BS1 and BS2 pins would be connected to VCC on the PCB 'but the test PCB used 2 port pins Config Porta. 0 = Output **Config** Porta. 1 = **Output** Porta. 0 = 1 Porta. 1 = 1\$lib "glcdSSD1325_96x64.lbx" 1 include the lib 'vcc is 12V and must be enabled later. This means vcc needs a control pin. Config GraphIcd = 96x64ssd1325, Dataport = Portc, Wr = Portd. 6, Rd = Portd. 7, Cs1 = Portd. 3, A0 = Portd. 5, Rst = Portd. 4, Vcc = Portd. 2 Cls 'as clear display usual Dim J As Byte, K As Byte, W As Word . diagonal line . with other line diagonal color Pset 1 , 0 , 15 'set a pixel Setfont Color8x8 Lcdat 20 , 0 , "123" , 15 , 0 font to use 'and show some text **Waitms** 3000 Showpic 0, 0, Plaatje End 'include font \$include "color8x8.font" "color16x16.font" '\$include Plaatje:

\$bgf "ssd1325.bgc"

8.10 AVR-DOS

976

8.10.1 AVR-DOS File System

The AVR-DOS file system is written by Josef Franz Vögel. He can be contacted via the BASCOM forum. Note that it is not permitted to use the AVR-DOS file system for commercial applications without the purchase of a license. A license comes with the ASM source.

You can buy a user license that is suited for most private users.

When you develop a commercial product with AVR-DOS you need the company license.

The ASM source is shipped with both licenses.

Josef has put a lot of effort in writing and especially testing the routines. Josef nor MCS Electronics can be held responsible for any damage or data loss of your CF-cards.

The File-System works with Compact – Flash Cards (see AN 123 Accessing a Compact Flash Card from BASCOM and <u>Compact Flash</u>) and is written for the needs for embedded systems for logging data. There are further functions for binary read and write.

You do not need AN123. AN123 was used to develop AVR-DOS. So you should use AVR-DOS.

The intention in developing the DOS – file system was to keep close to the equivalent VB functions.

The Filesystem works with:

- FAT16, this means you need to use >= 32MB CF cards
- FAT32
- Short file name (8.3)
- (Files with a long file name can be accessed by their short file name alias)
- Files in Root Directory. The root dir can store 512 files. Take in mind that when you use long file names, less filenames can be stored.
- Files in SUBDIRS

Requirements:

- Hardware: see AN 123 on http://www.mcselec.com/an_123.htm
- Software: appr. 2K-Word Code-Space (4000 Bytes)
- SRAM: 561 Bytes for File system Info and DIR-Handle buffer
- 517 Bytes if FAT is handled in own buffer (for higher speed), otherwise it is handled with the DIR Buffer
- 534 Bytes for each File handle
- This means that a Mega103 or Mega128 is the perfect chip. Other chips have too little internal memory. You could use XRAM memory too with a Mega8515 for example.

File System Configuration in CONFIG_AVR-DOS.BAS

	Count of File handles: for each file opened at same time, a file handle buffer of 534 Bytes is needed
cSepFATHandle:	For higher speed in handling file operations the FAT info can be stored in a own buffer, which needs additional 517 Bytes.

Assign Constant cSepFATHandle with 1, if wanted, otherwise with
0.

Memory Usage of DOS – File System:

1. General File System information

Variable Name	Туре	Usage	
gbDOSError	Byte	holds DOS Error of last file handling routine	
gbFileSystem	Byte	File System Code from Master Boot Record	
glFATFirstSector	Long	Number of first Sector of FAT Area on the Card	
gbNumberOfFATs	Byte	Count of FAT copies	
gwSectorsPerFat	Word	Count of Sectors per FAT	
glRootFirstSector	Long	Number of first Sector of Root Area on the Card	
gwRootEntries	Word	Count of Root Entries	
glDataFirstSector	Long	Number of first Sector of Data Area on the Card	
gbSectorsPerCluster	Byte	Count of Sectors per Cluster	
gwMaxClusterNumber	Word	Highest usable Cluster number	
gwLastSearchedClust er	Word	Last cluster number found as free	
gwFreeDirEntry	Word	Last directory entry number found as free	
glFS_Temp1	Long	temporary Long variable for file system	
gsTempFileName	String * 11	temporary String for converting file names	

2. Directory

Variable Name	Type Usage		
gwDirRootEntry	Word number of last handled root entry		
glDirSectorNumber	Long Number of current loaded Sector		
gbDirBufferStatus	Byte	Buffer Status	
gbDirBuffer	Byte (512)	Buffer for directory Sector	

3. FAT

Variable Name Type		Usage
glFATSectorNumber	Long	Number of current loaded FAT sector
gbFATBufferStatus	Byte	Buffer status
gbFATBuffer	Byte(512)	buffer for FAT sector

4. File handling

Each file handle has a block of 534 Bytes in the variable abFileHandle which is a byte-array of size (534 * cFileHandles)

978 BASCOM-AVR

Variable Name	Туре	Usage
FileNumber	Byte	File number for identification of the file in I/O operations to the opened file
FileMode	Byte	File open mode
FileRootEntry	Word	Number of root entry
FileFirstCluster	Word	First cluster
FATCluster	Word	cluster of current loaded sector
FileSize	Long	file size in bytes
FilePosition	Long	file pointer (next read/write) 0-based
FileSectorNumber	Long	number of current loaded sector
FileBufferStatus	Byte	buffer Status
FileBuffer	Byte(512)	buffer for the file sector
SectorTerminator	Byte	additional 00 Byte (string terminator) for direct reading ASCII files from the buffer

Code	Compiler – Alias	Remark	
0	cpNoError	No Error	
1	cpEndOfFile	Attempt behind End of File	
17	cpNoMBR	Sector 0 on Card is not a Master Boot Record	
18	cpNoPBR	No Partition Sector	
19	cpFileSystemNotSupported	Only FAT16 File system is supported	
20	cpSectorSizeNotSupported	Only sector size of 512 Bytes is supported	
21	cpSectorsPerClusterNotSupported	Only 1, 2, 4, 8, 16, 32, 64 Sectors per Cluster is supported. This are values of normal formatted partitions. Exotic sizes, which are not power of 2 are not supported	
33	cpNoNextCluster	Error in file cluster chain	
34	cpNoFreeCluster	No free cluster to allocate (Disk full)	
35	cpClusterError	Error in file cluster chain	
49	cpNoFreeDirEntry	Directory full	
50	cpFileExist		
65	cpNoFreeFileNumber	No free file number available, only theoretical error, if 255 file handles in use	
66	cpFileNotFound	File not found	
67	cpFileNumberNotFound	No file handle with such file number	
68	cpFileOpenNoHandle	All file handles occupied	
69	cpFileOpenHandleInUse File handle number in use, can create a new file handle with sa number		
70	cpFileOpenShareConflict Tried to open a file in read and w modus in two file handles		
71	cpFileInUse	Can't delete file, which is in use	
72	cpFileReadOnly	Can't open a read only file for writing	
73	cpFileNoWildCardAllowed	No wildcard allowed in this function	

© 2011 MCS Electronics

ASM Libraries and Add-Ons 979

97	cpFilePositionError	
98	cpFileAccessError	function not allowed in this file open mode
99	cpInvalidFilePosition	new file position pointer is invalid (minus or 0)
100	cpFileSizeToGreat	File size to great for function BLoad

Buffer Status: Bit definitions of Buffer Status Byte (Directory, FAT and File)

Bit	DIR	FAT	File	Compiler Alias	Remark
0 (LSB)			•	dBOF	Bottom of File (not yet supported)
1			•	dEOF	End of File
2			•	dEOFinSector	End of File in this sector (last sector)
3	•	•	•	dWritePendin g	Something was written to sector, it must be saved to Card, before loading next sector
4		•			This is an FAT Sector, at writing to Card, Number of FAT copies must be checked and copy updated if necessary
5			•	dFileEmpty	File is empty, no sector (Cluster) is allocated in FAT to this file

Validity of the file I/O operations regarding the opening modes

	Open mode				
Action	Input	Output	Append	Binary	
Attr	•	•	•	•	
Close	•	•	•	•	
Put				•	
Get				•	
LOF	•	•	•	•	
LOC	•	•	•	•	
EOF	•	1)	1)	•	
SEEK	•	•	•	•	
SEEK-Set				•	
Line Input	•			•	
Print		•	•	•	
Input	•			•	
Write		•	•	•	

1) Position pointer is always at End of File

Supported statements and functions:

INITFILESYSTEM [737], OPEN [794], CLOSE [437], FLUSH [689], PRINT [810], LINE INPUT [762], LOC [766], LOF [767], EOF [687], FREEFILE [686], FILEATTR [684], SEEK [680], BSAVE [420], BLOAD [416], KILL [751], DISKFREE [660], DISKSIZE [661], GET [697], PUT [819], FILEDATE [688], FILETIME [687], FILEDATETIME [688], DIR [656], WRITE [953], INPUT [744], FILELEN [687]

8.11 CF Card

980

8.11.1 Compact FlashCard Driver

The compact flash card driver library is written by Josef Franz Vögel. He can be contacted via the BASCOM user list.

Josef has put a lot of effort in writing and especially testing the routines. Josef nor MCS Electronics can be held responsible for any damage or data loss of your CF-cards.

Compact flash cards are very small cards that are compatible with IDE drives. They work at 3.3V or 5V and have a huge storage capacity.

The Flash Card Driver provides the functions to access a Compact Flash Card.

At the moment there are six functions: <u>DriveCheck</u> (acch, <u>DriveReset</u> (acch), <u>DriveInit</u> (acch), <u>DriveGetIdentity</u> (acch), <u>DriveWriteSector</u> (acch), <u>DriveReadSector</u> (acch)

The Driver can be used to access the Card directly and to read and write each sector of the card or the driver can be used in combination with a file-system with basic drive access functions.

Because the file system is separated from the driver you can write your own driver.

This way you could use the file system with a serial EEPROM for example.

For a file system at least the functions for reading (DriveReadSector / _DriveReadSector) and writing (DriveWriteSector / _DriveWriteSector) must be provided. The preceding under slash _ is the label of the according asm-routine. The other functions can, if possible implemented as a NOP – Function, which only returns a No-Error (0) or a Not Supported (224) Code, depending, what makes more sense.

For writing your own Driver to the AVR-DOS File system, check the ASM-part of the functions-description.

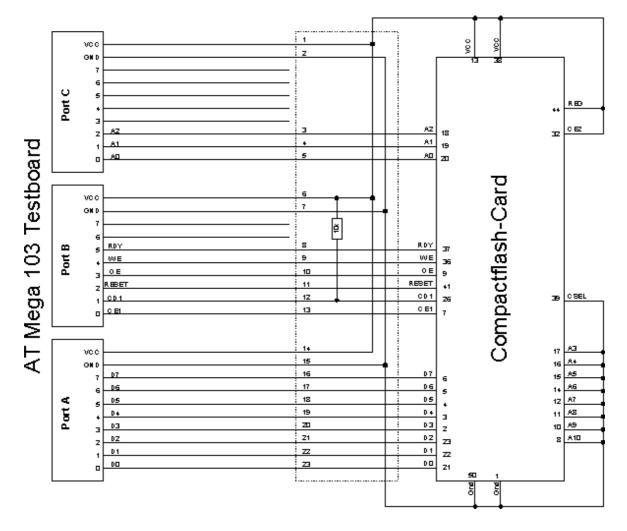
Code	Compiler – Alias	Remark
0	CpErrDriveNoError	No Error
224	cpErrDriveFunctionNo tSupported	This driver does not supports this function
225	cpErrDriveNotPresent	No Drive is attached
226	cpErrDriveTimeOut	During Reading or writing a time out occurred
227	cpErrDriveWriteError	Error during writing
228	cpErrDriveReadError	Error during reading

Error Codes:

At the MCS Web AN section you can find the application note 123.

More info about Compact Flash you can find at :

http://www.sandisk.com/download/Product%20Manuals/cf_r7.pdf



A typical connection to the micro is shown below.

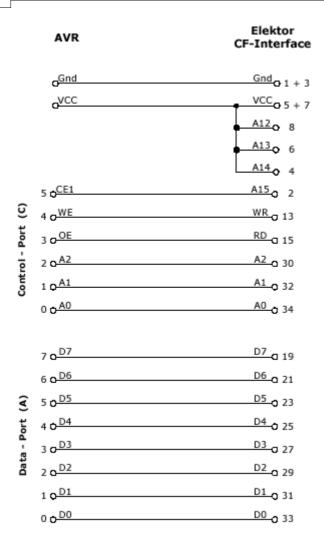
8.11.2 Elektor CF-Interface

The popular Electronics magazine Elektor, published an article about a CF-card interface. This interface was connected to an 89S8252. This interface can be used and will use little pins of the micro.

Note that because of the FAT buffer requirement, it is not possible to use a 8051 micro.,

At this moment, only the Mega128 and the Mega103 AVR micro's are good chips to use with AVR-DOS.

You can use external memory with other chips like the Mega162.



Changes of the hardware pins is possible in the file Config_FlashCardDrive_EL_PIN. bas.

The default library is FlashCardDrive.lib but this interface uses the library FlashCardDrive_EL_PIN.lib.

8.11.3 XRAM CF-Interface for simulation

The XRAM CF-Card interface is created for the purpose of testing the File System routines without hardware.

You can use an external RAM chip (XRAM) for the CF-interface but of course it is not practical in a real world application unless you backup the power with a battery.

For tests with the simulator it is ideal.

Just specify the Config_XRAMDrive.bas file and select a micro that can address external memory such as the M128. Then specify that the system is equipped with 64KB of external RAM.

You can now simulate the flashdisk.bas sample program !

In order to simulate Flashdisk.bas, set the constant XRAMDRIVE to 1. Then select 64KB of external RAM and compile.

8.11.4 New CF-Card Drivers

New CF-Card drivers can be made relatively simple.

Have a look at the supplied drivers.

There are always a few files needed :

- A config file in the format : CONFIG_XXX.bas
- FlashCardDrive_XXX.LIB
- FlashCardDrive_XXX.lbx is derived from the LIB file

XXX stands for the name of your driver.

At the AVR-DOS web you can find more drivers.

8.12 Floating Point

8.12.1 FP_TRIG

The FP_TRIG library is written by Josef Franz Vögel.

All trig functions are stored in fp_trig.lib library. The fp_trig.lbx contains the compiled object code and is used by BASCOM.

This sample demonstrates all the functions from the library:

```
: test_fptrig2.bas
: (c) 1995-2005, MCS Electronics
: demonstates FP trig library from Josef Franz Vögel
'name
'copyright
'purpose
                                       : Mega8515
micro
'suited for demo
                                        : no
'commercial addon needed : no
$regfile = "m8515.dat"
$crystal = 4000000
$baud = 19200
                                                                                             specify the used micro
used crystal frequency
                                                                                          1.1
                                                                                          .
                                                                                             use baud rate
$hwstack = 32
hardware stack
                                                                                          .
                                                                                             default use 32 for the
$swstack = 10
                                                                                          ' default use 10 for the SW
stack
$framesize = 40
                                                                                          ' default use 40 for the frame
space
Dim S1 As Single, S2 As Single, S3 As Single, S4 As Single, S5 As Single, S6 As
Single
Dim Vcos As Single
Dim Vcos As Single, vsu according to bin Wi As Single, B1 As Byte Dim Ms1 As Single
                               Vsin As Single, Vtan As Single, Vatan As Single, S7 As Single
Const Pi = 3.14159265358979
\begin{array}{rrr} \text{'calculate} & \text{PI} \\ \text{Ms1} = \text{Atn(1)} & 4 \end{array}
Testing_power:
Print "Testing Power X ^ Y"
Print "Testing Force. Y

Print "X Y

For S1 = 0.25 To 14 Step 0.25
                                                 x^Y"
For S1 = 0.25 To 14 Step 0.25

S2 = S1 \ 2

S3 = Power(s1 , S2)

Print S1 ; " ^ "; S2 ; " = "; S3
Next
Print: Print: Print
```

984 BASCOM-AVR

Testing_exp_log:

```
Print"TestingEXP and LOG"Print"xexp(x)log([exp(x)])Error-absError-restrictionPrint"Errorisforcalculatingexpandbackwithlogtogether"
                                                                                                 Error-rel"
For S1 = -88 To 88

S2 = Exp(s1)

S3 = Log(s2)

S4 = S3 - S1

S5 = S4 \setminus S1

Print S1 - "
     Print S1;
                                "; S2; " "; S3; " "; S4; " "; S5; " ";
     Print
Next
Print: Print: Print
Testing_trig:

Print "Testing COS,

Print "Angle Degree

For Wi = -48 To 48

S1 = Wi * 15

S2 = Deg2rad(s1)

Vigeo = Cor(s2)
                                                                    Cos
                                                                                           Sin
                                                                                                      Tan"
     V\cos = Cos(s2)
V\sin = Sin(s2)
      Vtan = Tan(s2)
     Print S1 ;
                              "; S2; " "; Vcos; " "; Vsin; " "; Vtan
Next
Print : Print : Print
Testing_atan:

Print "Testing Arctan"

Print "X atan in Radiant,

S1 = 1 / 1024
                                                     Dearee"
Do
    S2 = Atn(s1)
S3 = Rad2deg(s2)
Totat S1 : " ; S2 ; " " ; S3
     Print S1 ; " ;
S1 = S1 * 2
I f S1 > 1000000 Then
         Exit Do
     End I f
Loop
Print: Print: Print
Testing_int_fract:

Print "Testing Int und Fract of Single"

Print "Value Int Frac"

S2 = Pi \setminus 10

For S1 = 1 To 8
     S3 = Int(s2)
     S4 = Frac(s2)
     Print S2 ; "
S2 = S2 * 10
                              "; S3; " "; S4
Next
Print: Print: Print
Print "Testing degree - radiant - degree converting"
Print "Degree Radiant Degree Diff-abs rel"
For S1 = 0 To 90
   S2 = Deg2rad(s1)
S3 = Rad2deg(s2)
S4 = S3 - S1
S5 = S4 \ S1
Print S1 ;
                          "; S2; " "; S3; " "; S4; " "; S5
Next
Testing_hyperbolicus:

Print : Print : Print

Print "Testing SINH, COSH and TANH"

Print "X sinh(x) co

For S1 = -20 To 20

S3 = Sinh(s1)

S2 = Corp(c1)
                                                 cosh(x)
                                                                               tanh(x)"
   S3 = Sinn(S);
S2 = Cosh(S1)
S4 = Tanh(S1)
Print S1 ; " "; S3 ; " "; S2 ; " "; S4
Next
Print : Print : Print
```

Testing_log10: Print "Testing LOG10" Print "X log1 log10(x)" Print "X log10(x S1 = 0.01 S2 = Log10(s1) Print S1 ; " " ; S2 S1 = 0.1 S2 = Log10(s1) Print S1 ; " " ; S2 For S1 = 1 To 100 S2 = Log10(s1) Print S1 ; " " ; S2 Next Next Print: Print: Print 'test MOD on FP S1 = 10000 S2 = 3 S3 = S1 Mod S2 Print S3 Print S1; Next Print For S1 = 1000000 To 1000100 S2 = Sqr(s1) Print S1 ; " "; S2 Print S1; Next Testing_atn2: Print "Testing Sin / Cos / ATN2 / Deg2Rad / Rad2Deg / Round" Print "X[deg] X[Rad] Sin(x) Cos(x) Atn2 For S1 = -180 To 180 Step 5 S2 = Deg2rad(s1) S3 = Sin(s2) S4 = Cos(s2) Deg of Atn2 Rounded" S4 = Cos(s2) S5 = Atn2(s3 , S4) S6 = Rad2deg(s5) S7 = Round(s6) Print S1 ; " "; S2 ; " "; S3 ; " "; S4 ; " "; S5 ; " "; S6 ; " "; S7 Next Print "note: -180° is equivalent to +180°" Print Testing_asin_acos: Print "Testing ASIN, ACOS" Print "X asin(x) acos(x)" For S1 = -1.125 To 1.125 Step 0.0625 S2 = Asin(s1) S3 = Acos(s1) Print S1 ; " "; S2 ; " "; S3 Next Print "Note: > 1.0 and < -1.0 are invalid and shown here for error handling" Testing_shift: S1 = 12 For B1 = 1 To 20 S2 = S1 : S3 = S1 Shift S2 , Left , B1 Shift S3 , Right , B1 Print S1 ; " " ; S2 ; " " ; S3 Next Print "End of testing" End Back

BASCOM-AVR

8.12.2 DOUBLE

986

The double.lbx (lib) is written by Josef Franz Vögel. The library supports the basic operations :

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Val(), INPUT
- Str(), PRINT
- Int()
- Frac()
- Fix()
- Round()
- Conversion from double to single and long
- Conversion from single and long to double

The double library uses special Mega instructions not available in all AVR chips. But as the old chips are not manufactured anymore, this should not be a problem. In version 1.11.9.8 a software multiplication is performed so the trig functions can be used on any chip that has enough internal memory.

In the report file you can find out if your micro supports the HWMUL. the _HWMUL conststant is set to 1 in that case.

When software multiplication is used, the multiply routine needs more processor cycles. A number of trig functions depend on the multiplication code and as a result, they become more slow too.

All Trig() functions are supported by the double too!

8.13 I2C SLAVE

8.13.1 I2CSLAVE

The I2C-Slave library is intended to create I2C slave chips. This is an add-on library that is not included by default. It is a commercial add on library. It is available from <u>MCS Electronics</u>

All BASCOM I2C routines are master I2C routines. The AVR is a fast chip and allows to implement the I2C slave protocol.

You can control the chips with the BASCOM I2C statements like I2CINIT, I2CSEND, I2CRECEIVE, I2CWBYTE, etc. Please consult the BASCOM Help file for using I2C in master mode.

Before you begin

Copy the i2cslave.lib and i2cslave.lbx files into the BASCOM-AVR\LIB directory. The i2cslave.lib file contains the ASM source. The i2cslave.lbx file contains the compiled ASM source.

Slave address

Every I2C device must have an address so it can be addressed by the master I2C routines.

When you write to an I2C-slave chip the least significant bit (bit0) is used to specify if we want to read from the chip or that we want to write to the chip. When you specify the slave address, do not use bit 0 in the address!

For example a PCF8574 has address &H40. To write to the chip use &H40, to read from the chip, use &H41. When emulating a PCF8574 we would specify address &H40.

Use the <u>CONFIG</u> statement to specify the slave address:

Config I2cslave = &B01000000 ' same as &H40 Optional use : **CONFIG I2CSLAVE = address, INT= int , TIMER = tmr**

Where INT is INTO, INT1 etc. and TIMER is TIMER0, TIMER1 etc. When using other interrupts or timers, you need to change the library source. The library was written for TIMER0 and INT0.

The I2C slave routines use the TIMER0 and INT0. You can not use these interrupts yourself. It also means that the SCL and SDA pins are fixed.

Chip	SCL	SDA
AT90S1200	PORTD.4	PORTD.2
AT90S2313	PORTD.4	PORTD.2
AT90S2323	PORTB.2	PORTB.1
AT90S2333	PORTD.4	PORTD.2
AT90S2343	PORTB.2	PORTB.1
AT90S4433	PORTD.4	PORTD.2
ATTINY22	PORTB.2	PORTB.1
ATTINY13	PORTB.2	PORTB.1
ATTINY2313	PORTD.4	PORTD.2
ATMEGA1280	PORTD.7	PORTD.0
ATMEGA128CAN	PORTD.7	PORTD.0
ATMEGA168	PORTD.4	PORTD.2
ATMEGA2560	PORTD.7	PORTD.0
ATMEGA2561	PORTD.7	PORTD.0
ATMEGA48	PORTD.4	PORTD.2
ATMEGA88	PORTD.4	PORTD.2
ATMEGA8	PORTD.4	PORTD.2

The following table lists the pins for the various chips

Note that new AVR chips have a TWI or hardware I2C implementation. It is better to use hardware I2C, then the software I2C. The slave library is intended for AVR chips that do not have hardware I2C.

CONFIG I2CSLAVE will enable the global interrupts.

After you have configured the slave address, you can insert your code.

A do-loop would be best:

Do ' your code here Loop This is a simple never-ending loop. You can use a GOTO with a label or a While Wend loop too but ensure that the program will never end.

After your main program you need to insert two labels with a return:

When the master needs to read a byte, the following label is always called. You must put the data you want to send to the master in variable _a1 which is register R16

I2c_master_needs_data:

'when your code is short, you need to put in a waitms statement 'Take in mind that during this routine, a wait state is active and the master will wait 'After the return, the waitstate is ended Config Portb = Input ' make it an input

_a1 = Pinb ' Get input from portB and assign it Return

When the master writes a byte, the following label is always called. It is your task to retrieve variable _A1 and do something with it _A1 is register R16 that could be destroyed/altered by BASIC statements For that reason it is important that you first save this variable.

I2c_master_has_data:

'when your code is short, you need to put in a waitms statement 'Take in mind that during this routine, a wait state is active and the master will wait 'After the return, the waitstate is ended

Bfake = _a1 ' this is not needed but it shows how you can store _A1 in a byte 'after you have stored the received data into bFake, you can alter R16 Config Portb = Output ' make it an output since it could be an input Portb = a1 'assign A1 (R16)

```
Portb = _a1
Return
```

8.13.2 I2C TWI Slave

The I2C Slave add on can turn some chips into a I2C slave device. You can start your own chip plant this way.

Most new AVR chips have a so called TWI interface. As a customer of the I2C slave lib, you can get both libs.

The TWI slave lib works in interrupt mode and is the best way as it adds less overhead and also less system resources.

In the following example the code for older compilers

Example

```
'name : twi-slave.bas
'copyright : (c) 1995-2005, MCS Electronics
'purpose : shows an example of the TWI in SLAVE mode
'micro : Mega128
'suited for demo : yes
'commercial addon needed : yes
```

```
_____
$regfile = "m128def.dat"
                                                             ' specify
the used micro
$crystal = 8000000
                                                             ' used
crystal frequency
$baud = 19200
                                                             ' use baud
rate
$hwstack = 32
                                                             ' default
use 32 for the hardware stack
\$swstack = 10
                                                             ' default
use 10 for the SW stack
Sframesize = 40
                                                             ' default
use 40 for the frame space
' Not all AVR chips have TWI (hardware I2C)
' IMPORTANT : this example ONLY works when you have the TWI slave
library
' which is a commercial add on library, not part of BASCOM
Print "MCS Electronics TWI-slave demo"
Config Twislave = &H70 , Btr = 1 , Bitrate = 100000
'as you might need other interrupts as well, you need to enable them
manual
Enable Interrupts
'this is just an empty loop but you could perform other tasks there
Do
 nop
Loop
End
'A master can send or receive bytes.
'A master protocol can also send some bytes, then receive some bytes
'The master and slave must match.
'the following labels are called from the library
Twi_stop_rstart_received:
 Print "Master sent stop or repeated start"
Return
Twi_addressed_goread:
 Print "We were addressed and master will send data"
Return
Twi_addressed_gowrite:
 Print "We were addressed and master will read data"
Return
'this label is called when the master sends data and the slave has
received the byte
'the variable TWI holds the received value
Twi_gotdata:
  Print "received : " ; Twi
Return
```

BASCOM-AVR

```
'when the mast has all bytes received this label will be called
Twi_master_need_nomore_byte:
    Print "Master does not need anymore bytes"
Return
```

8.14 SPI

8.14.1 SPISLAVE

SPISLAVE.LIB (LBX) is a library that can be used to create a SPI slave chip when the chip does not have a hardware SPI interface.

Although most AVR chips have an ISP interface to program the chip, the 2313 for example does not have a SPI interface.

When you want to control various micro's with the SPI protocol you can use the SPISLAVE library.

The SPI-softslave.bas sample from the samples directory shows how you can use the SPISLAVE library.

Also look at the spi-slave.bas sample that is intended to be used with hardware SPI.

The sendspi.bas sample from the samples directory shows how you can use the SPI hardware interface for the master controller chip.

```
_____
_____
'name
                       : spi-softslave.bas
'copyright
                      : (c) 1995-2005, MCS Electronics
                       : shows how to implement a SPI SLAVE with
'purpose
software
'micro
                       : AT90S2313
'suited for demo
                       : yes
'commercial addon needed : no
!_____
_____
$regfile = "2313def.dat"
                                                      ' specify
the used micro
$crystal = 4000000
                                                      ' used
crystal frequency
$baud = 19200
                                                      ' use baud
rate
$hwstack = 32
                                                      ' default
use 32 for the hardware stack
$swstack = 10
                                                      ' default
use 10 for the SW stack
$framesize = 40
                                                      ' default
use 40 for the frame space
'Some atmel chips like the 2313 do not have a SPI port.
```

990

'The BASCOM SPI routines are all master mode routines 'This example show how to create a slave using the 2313 'ISP slave code 'define the constants used by the SPI slave **Const** _softslavespi_port = Portd ' we used portD **Const** _softslavespi_pin = Pind 'we use the PIND register for reading Const _softslavespi_ddr = Ddrd ' data direction of port D **Const** softslavespi clock = 5 'pD.5 is used for the CLOCK **Const** _softslavespi_miso = 3 'pD.3 is MISO Const _softslavespi_mosi = 4 'pd.4 is MOSI Const _softslavespi_ss = 2 ' pd.2 is SS 'while you may choose all pins you must use the INTO pin for the SS 'for the 2313 this is pin 2 'PD.3(7), MISO must be output 'PD.4(8), MOSI 'Pd.5(9) , Clock 'PD.2(6), SS /INT0 'define the spi slave lib \$lib "spislave.lbx" 'sepcify wich routine to use \$external _spisoftslave 'we use the int0 interrupt to detect that our slave is addressed On IntO Isr_sspi Nosave 'we enable the int0 interrupt Enable Int0 'we configure the INTO interrupt to trigger when a falling edge is detected Config Int0 = Falling 'finally we enabled interrupts Enable Interrupts Dim _ssspdr As Byte ' this is out SPI SLAVE SPDR register Dim _ssspif As Bit ' SPI interrupt revceive bit Dim Bsend As Byte , I As Byte , B As Byte ' some other demo variables ' we send a $_ssspdr = 0$ 0 the first time the master sends data Do If _ssspif = 1 Then Print "received: " ; _ssspdr **Reset** _ssspif _ssspdr = _ssspdr + 1 ' we send this the next time End If Loop

When the chip has a SPI interface, you can also use the following

example:

```
_____
_____
                      : spi-slave.bas
'name
                      : (c) 1995-2005, MCS Electronics
'copyright
                      : shows how to create a SPI SLAVE
'purpose
'micro
                      : AT90S8515
                     : yes
'suited for demo
'commercial addon needed : no
·_____
_____
$regfile = "8515def.dat"
                                                    ' specify
the used micro
$crystal = 3680000
                                                    ' used
crystal frequency
$baud = 19200
                                                    ' use baud
rate
$hwstack = 32
                                                    ' default
use 32 for the hardware stack
$swstack = 10
                                                    ' default
use 10 for the SW stack
$framesize = 40
                                                    ' default
use 40 for the frame space
' use together with sendspi.bas
 _____
' Tested on the STK500. The STK200 will NOT work.
' Use the STK500 or another circuit
Dim B As Byte , Rbit As Bit , Bsend As Byte
'First configure the MISO pin
Config Pinb.6 = Output
                                                    ' MISO
'Then configure the SPI hardware SPCR register
Config Spi = Hard , Interrupt = On , Data Order = Msb , Master = No ,
Polarity = Low , Phase = 0 , Clockrate = 128
'Then init the SPI pins directly after the CONFIG SPI statement.
Spiinit
'specify the SPI interrupt
On Spi Spi_isr Nosave
'enable global interrupts
Enable Interrupts
'show that we started
Print "start"
Spdr = 0
                                                    ' start with
sending 0 the first time
Do
 If Rbit = 1 Then
   Print "received : " ; B
   Reset Rbit
   Bsend = Bsend + 1 : Spdr = Bsend
                                                    'increase
SPDR
 End If
  ' your code goes here
```

Loop

```
'Interrupt routine
'since we used NOSAVE, we must save and restore the registers ourself
'when this ISR is called it will send the content from SPDR to the
master
'the first time this is 0
Spi_isr:
  push r24 ; save used register
  in r24, sreg ; save sreg
  push r24
  B = Spdr
  Set Rbit
                                                             ' we
received something
  pop r24
  !out sreg,r24 ; restore sreg
  pop r24
                ; and the used register
                                                             ' this will
Return
generate a reti
```

8.15 DATE TIME

8.15.1 EUROTIMEDATE

The CONFIG CLOCK statement for using the asynchrony timer of the 8535, M163, M103 or M128 (and others) allows you to use a software based clock. See $\underline{\text{TIME}}$ and $\underline{\text{DATE}}$ [33]

By default the date format is in MM/DD/YY.

By specifying: <u>\$LIB</u> and "EURODATETIME.LBX"

The DATE\$ will work in European format : DD-MM-YY

Note that the eurotimedate library should not be used anymore. It is replaced by the <u>DATETIME</u> library which offers many more features.

8.15.2 DATETIME

The DateTime library is written by Josef Franz Vögel. It extends the clock routines with date and time calculation.

The following functions are available:

<u>DayOfWee</u> <u>k</u> ढि15ी	Returns the day of the week
DayOfYear	Returns the day of the year
SecOfDay 849	Returns the second of the day
SecElapse	Returns the elapsed Seconds to a former assigned time-stamp
<u>SysDay</u> ७१७	Returns a number, which represents the System Day
SysSec 915	Returns a Number, which represents the System Second
<u>SysSecEla</u>	Returns the elapsed Seconds to a earlier assigned system-time-stamp

psed 916	
<u>Time</u> छि	Returns a time-value (String or 3 Byte for Second, Minute and Hour) depending of the Type of the Target
Date 627	Returns a date-value (String or 3 Bytes for Day, Month and Year) depending of the Type of the Target



Date and time not to be confused with Date\$ and Time\$!

PS2-AT Mouse and Keyboard Emulation 8.16

8.16.1 AT_EMULATOR

The PS2 AT Keyboard emulator library is an optional add on library you can purchase.

The library allows you to emulate an AT PS/2 keyboard or mouse. The following statements become available:

CONFIG ATEMU 455 SENDSCANKBD 864

8.16.2 PS2MOUSE EMULATOR

The PS2 Mouse emulator library is an optional addon library you can purchase. The library allows you to emulate an AT PS/2 mouse. The following statements become available:

CONFIG PS2EMU 535 PS2MOUSEXY 816 SENDSCAN 862

8.17 BCCARD

8.17.1 BCCARD

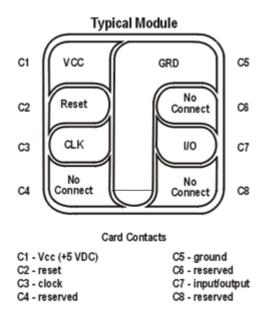
BCCARD.LIB is a commercial addon library that is available separately from MCS Electronics.

With the BCCARD library you can interface with the BasicCards from www.basiccard. com

BasicCards are also available from MCS Electronics

A BasicCard is a smart card that can be programmed in BASIC.

The chip on the card looks like this :



To interface it you need a smart card connector.

Smart Card PIN	Connect to
C1	+5 Volt
C2	PORTD.4, RESET
C3	PIN 4 of 2313 , CLOCK
C5	GND
C7	PORTD.5 , I/O

In the provided example the connections are made as following:

The microprocessor must be clocked with a 3579545 crystal since that is the frequency the Smart Card is working on. The output clock of the microprocessor is connected to the clock pin of the Smart card.

Some global variables are needed by the library. They are dimensioned automatic by the compiler when you use the CONFIG BCCARD statement.

These variables are:

 $_Bc_pcb$: a byte needed by the communication protocol. Sw1 and SW2 : both bytes that correspondent to the BasicCard variables SW1 and SW2

The following statements are especially for the BasicCard:

CONFIG BCCARD458to init the libraryBCRESET1002to reset the cardBCDEF1996to define your function in the cardBCCALL1996to call the function in the card

Encryption is not supported by the library yet.

8.17.2 BCDEF

996

Action

Defines a subroutine name and it's parameters in BASCOM so it can be called in the BasicCard.

Syntax

BCDEF name([param1 , paramn])

Remarks

name	The name of the procedure. It may be different than the name of the procedure in the BasicCard but it is advised to use the same names.
Param1	Optional you might want to pass parameters. For each parameter you pass, you must specify the data type. Supported data types are byte, Integer, Word, Long, Single and String

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

BCDEF Calc(string)

Would define a name 'Calc' with one string parameter. When you use strings, it must be the last parameter passed.

BCDEF name(byte, string)

BCDEF does not generate any code. It only informs the compiler about the data types of the passed parameters.

See Also

CONFIG BCCARD 458 , BCCALL 996 , BCRESET 1002

Partial Example

Bcdef Calc(string)

8.17.3 BCCALL

Action

Calls a subroutine or procedure in the BasicCard.

Syntax

BCCALL name(nad , cla, ins, p1, p2 [param1 , paramn])

Remarks

name	The name of the procedure to all in the BasicCard. It must be defined
	first with BCDEF. The name used with BCDEF and BCCALL do not

	need to be the same as the procedure in the BasicCard but it is advised to use the same names.
NAD	Node address byte. The BasicCard responds to all node address values. Use 0 for default.
CLA	Class byte. First byte of two byte CLA-INS command. Must match the value in the BasicCard procedure.
INS	Instruction byte. Second byte of two byte CLA-INS command. Must match the value in the BasicCard procedure.
P1	Parameter 1 of CLA-INS header.
P2	Parameter 2 of CLA-INS header

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

When in your BasicCard basic program you use: 'test of passing parameters Command &hf6 &h01 ParamTest(b as byte, w as integer, I as long) b=b+1 w=w+1 I=I+1 end command

You need to use &HF6 for CLA and 1 for INS when you call the program:

Bccall Paramtest(0 , &HF6 , 1 , 0 , 0 , B , W , L) ^ NAD

^CLA

^INS

^P1

^P2

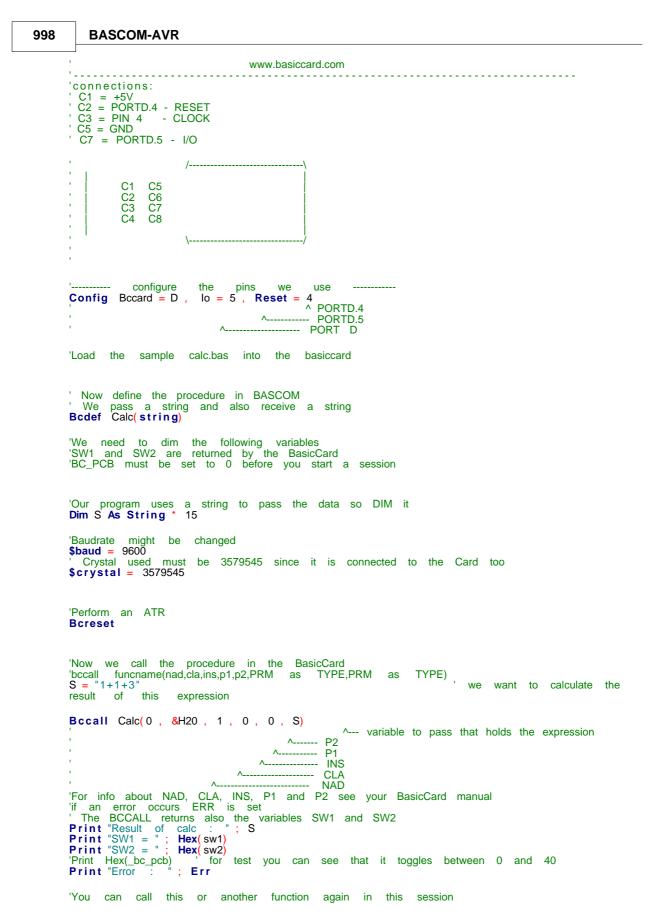
When you use BCCALL, the NAD, CLA, INS, P1 and P2 are sent to the BasicCard. The parameter values are also sent to the BasicCard. The BasicCard will execute the command defined with CLA and INS and will return the result in SW1 and SW2.

The parameter values altered by the BasicCard are also sent by the BasicCard.

You can not sent constant values. Only variables may be sent. This because a constant can not be changed.

See Also CONFIG BCCARD 458 , BCDEF 996 , BCRESET 1002

Example BCCARD.BAS This AN shows how to use the BasicCard from Zeitcontrol



 $\begin{array}{l} S = "2+2"\\ \textbf{Bccall Calc}(0, \& H20, 1, 0, 0, S)\\ \textbf{Print "Result of calc : "; S}\\ \textbf{Print "SW1 = "; Hex(sw1)}\\ \textbf{Print "SW2 = "; Hex(sw2)}\\ \textbf{'Print Hex(_bc_pcb)} & \text{for test you can see that it toggles between 0 and 40} \end{array}$

Print "Error : "; Err 'perform another ATR Bcreset
 Bccall
 Calc(0, &H20, 1, 0, 0, S)

 Print
 Answer
 :
 ;
 S
 '----and now perform an ATR as Dim Buf(25) As Byte, I As Byte Buf(1) = Bcreset() For I = 1 To 25 Print I; " "; Hex(buf(i)) Next as a function Next 'typical 'TS = 3B returns : 'T0 = EF 'TB1 = 00 'TC1 = FF 'and another test define the procedure in the BasicCard program Bcdef Paramtest(byte, Word, Long) 'dim some variables Dim B As Byte , W As Word , L As Long the variables 'assign B = 1 : W = &H1234 : L = &H12345678 'try the echotest command Bcdef Echotest(byte) Bccall Echotest(0, &HC0, &H14, 1, 0, B) Print B End 'end program

Rem BasicCard Sample Source Code Rem -------Rem Copyright (C) 1997-2001 ZeitControl GmbH Rem You have a royalty-free right to use, modify, reproduce and Rem distribute the Sample Application Files (and/or any modified Rem version) in any way you find useful, provided that you agree Rem that ZeitControl GmbH has no warranty, obligations or liability Rem for any Sample Application Files. Rem ------

#Include CALCKEYS.BAS

Declare ApplicationID = "BasicCard Mini-Calculator"

Rem This BasicCard program contains recursive procedure calls, so the Rem compiler will allocate all available RAM to the P-Code stack unless Rem otherwise advised. This slows execution, because all strings have to Rem be allocated from EEPROM. So we specify a stack size here:

1000 BASCOM-AVR

#Stack 120 ' Calculator Command (CLA = &H20, INS = &H01) 'Input: an ASCII expression involving integers, and these operators: * / % + - & ^ | (Parentheses are also allowed.) Output: the value of the expression, in ASCII. ' P1 = 0: all numbers are decimal ' P1 <> 0: all numbers are hex ' Constants Const SyntaxError = &H81 Const ParenthesisMismatch = &H82 Const InvalidNumber = &H83 Const BadOperator = &H84 ' Forward references Declare Function EvaluateExpression (S\$, Precedence) As Long Declare Function EvaluateTerm (S\$) As Long Declare Sub Error (Code@) 'test for passing a string Command &H20 &H01 Calculator (S\$) Private X As Long S = Trim\$ (S\$) X = EvaluateExpression (S\$, 0)If Len (Trim\$ (S\$)) <> 0 Then Call Error (SyntaxError) If P1 = 0 Then S = Str (X) : Else S = Hex (X) End Command 'test of passing parameters Command &hf6 &h01 ParamTest(b as byte, w as integer, l as long) b=b+1w = w + 1|=|+1end command Function EvaluateExpression (S\$, Precedence) As Long EvaluateExpression = EvaluateTerm (S\$) Do S = LTrim\$ (S\$) If Len (S\$) = 0 Then Exit Function

Select Case S\$(1)

```
Case "*"
  If Precedence > 5 Then Exit Function
  S = Mid$ (S$, 2)
  EvaluateExpression = EvaluateExpression * _
     EvaluateExpression (S$, 6)
Case "/"
  If Precedence > 5 Then Exit Function
  S_{s} = Mid_{s}(S_{s}, 2)
  EvaluateExpression = EvaluateExpression /
     EvaluateExpression (S$, 6)
Case "%"
  If Precedence > 5 Then Exit Function
  S$ = Mid$ (S$, 2)
  EvaluateExpression = EvaluateExpression Mod _
     EvaluateExpression (S$, 6)
Case "+"
  If Precedence > 4 Then Exit Function
  S$ = Mid$ (S$, 2)
  EvaluateExpression = EvaluateExpression + _
     EvaluateExpression (S$, 5)
Case "-"
  If Precedence > 4 Then Exit Function
  S = Mid$ (S$, 2)
  EvaluateExpression = EvaluateExpression -
     EvaluateExpression (S$, 5)
Case "&"
  If Precedence > 3 Then Exit Function
  S$ = Mid$ (S$, 2)
  EvaluateExpression = EvaluateExpression And _
     EvaluateExpression (S$, 4)
Case "^"
  If Precedence > 2 Then Exit Function
  S = Mid$ (S$, 2)
  EvaluateExpression = EvaluateExpression Xor _
     EvaluateExpression (S$, 3)
Case "|"
  If Precedence > 1 Then Exit Function
  S$ = Mid$ (S$, 2)
  EvaluateExpression = EvaluateExpression Or
     EvaluateExpression (S$, 2)
Case Else
  Exit Function
End Select
```

Loop

End Function

Function EvaluateTerm (S\$) As Long

```
Do 'Ignore unary plus
S$ = LTrim$ (S$)
If Len (S$) = 0 Then Call Error (SyntaxError)
If S$(1) <> "+" Then Exit Do
S$ = Mid$ (S$, 2)
Loop
If S$(1) = "(" Then 'Expression in parentheses
```

```
S$ = Mid$ (S$, 2)
     EvaluateTerm = EvaluateExpression (S$, 0)
     S = LTrim$ (S$)
     If S$(1) <> ")" Then Call Error (ParenthesisMismatch)
     S$ = Mid$ (S$, 2)
     Exit Function
  ElseIf S$(1) = "-" Then 'Unary minus
     S$ = Mid$ (S$, 2)
     EvaluateTerm = -EvaluateTerm (S$)
     Exit Function
  Else
                          ' Must be a number
     If P1 = 0 Then
                             ' If decimal
       EvaluateTerm = Val& (S$, L@)
     Else
       EvaluateTerm = ValH (S$, L@)
     End If
     If L@ = 0 Then Call Error (InvalidNumber)
     S = Mid$ (S$, L@ + 1)
  End If
End Function
Sub Error (Code@)
  SW1 = &H64
  SW2 = Code@
  Exit
End Sub
```

8.17.4 BCRESET

Action

Resets the BasicCard by performing an ATR.

Syntax

BCRESET Array(1) = **BCRESET**()

Remarks

Array(1)	When BCRESET is used as a function it returns the result of the	
	ATR to the array named array(1). The array must be big enough	
	to hold the result. Dim it as a byte array of 25.	

This statements uses BCCARD.LIB, a library that is available separately from MCS Electronics.

An example of the returned output when used as a function:

'TS = 3B 'T0 = EF 'TB1 = 00 'TC1 = FF 'TD1 = 81 T=1 indication
'TD2 = 31 TA3,TB3 follow T=1 indicator
'TA3 = 50 or 20 IFSC ,50 =Compact Card, 20 = Enhanced Card
'TB3 = 45 BWT block waiting time
'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00

'BasicCardZC123

See the BasicCard manual for more information When you do not need the result you can also use the BCRESET statement.

See Also

CONFIG BCCARD 458 , BCDEF 996 , BCCALL 996

Partial Example (no init code shown)

'----and now perform an ATR as a function Dim Buf(25)AsByte, I AsByte Buf(1)=Bcreset() For I = 1 To 25 Print I ;" ";Hex(buf(i)) Next 'typical returns : 'TS = 3B'T0 = EF'TB1 = 00'TC1 = FF'TD1 = 81 T=1 indication 'TD2 = 31 TA3, TB3 follow T=1 indicator 'TA3 = 50 or 20 IFSC ,50 =Compact Card, 20 = Enhanced Card 'TB3 = 45 BWT blocl waiting time 'T1 -Tk = 42 61 73 69 63 43 61 72 64 20 5A 43 31 32 33 00 00 'BasicCardZC123

8.18 USB

8.18.1 USB Add On

The USB Add On is a commercial add on which is available from the MCS Electronics Web Shop.

The CONFIG USB statement needs this add on. The add on is written in BASCOM BASIC mixed with assembler. Since the examples from Atmel were not really consistent, it took some effort to create reusable code. At a later stage, a number of routines will be moved to an assembler library.

The advantage of the BASCOM code is that it is similar to the C-code examples.

Please read this entire topic first before you start with experiments.

The Add On only supports the device mode. There is no support for host mode yet. In fact the add on is just the first step into USB support.

To use the USB Add on, unzip all the files to the SAMPLES\USB directory. You will find three samples :

hid_generic-162.bas

- virtcom-162.bas
- hid_keyboard-162.bas

The same samples are also provided for the USB1287.

And you will find the include file : **usbinc.bas**. It is not allowed to distribute any of the files.

Further, you will find a subdirectory named VB which contains a simple VB generic HID sample that uses the HIDX.OCX from the OCX subdirectory.

The PDF directory contains a PDF with a translation between PS2 scan codes and USB key codes.

The TOOLS directory contains the USBDEVIEW.EXE which can be used to display all USB devices,

The CDC-Driver directory contains the INF file you need for the CDC/Virtual COM port example.

The USB162 has a boot loader which can be programmed by USB using FLIP. BASCOM will also support this USB boot loader in version 1.11.9.2. It is great for development but of course the boot loader uses some space which you probably need. The chip is also programmable via the normal way with the ISP protocol. when you do not use FLIP, and you erase the chip, the boot loader from Atmel is erased too! You can always reprogram the Atmel boot loader. But not using FLIP which depends on the boot loader.

For USB to work properly the chip needs a good oscillator. The internal oscillator is not good enough. For that reason, the USB162 module from MCS has a 8 MHz crystal. Your hardware should use a crystal or crystal oscillator too.

It is not the intention of MCS or the documentation to learn you everything about USB. There is a lot of information available from various sources. It is the goal of MCS to make it easy to use USB with your AVR micro. When there is enough demand for it, a special Wizard will be created to be able to generate HID applications.

HID Keyboard

Let's begin with a simple program. Load the hid_keyboard-162.bas sample and compile it. Use either FLIP or a different programmer to program the chip. Each program has some important settings.

Const Mdbg = 1	' add print to see what is happening
Const Chiddevice = 1	' this is a HID device

MDBG is a constant that can be set to 0 since all the print statements will use flash code. When you are new to USB and want to look at the events, it is good to have it turned on. You can view all events from the program.

cHIDdevice need to be set to 1 when the application is a HID device. Most of your own devices will be HID devices. But the virtual COM example uses a different USB class and in that program, the constant is set to 0.

These constants are used in the add on to keep all code generic for all applications. Since not all USB chips have the same options, the code also checks which microprocessor is used.

The USB1287 is a kind of M128 with USB support. It supports host and device mode.

The USB162 is a cheap host chip. It does not support the HOST mode and it does not have all registers found in the USB1287. It also can not detect when a device is plugged/unplugged.

Atmel solved this in the STK526 in a simple way that we recommend too : A voltage divider is connected to PORTC.4 which serves as a simple way to detect plug/unplug. In the USB_TASK() routine you will find this code :

If Usb_connected = 0 And Pinc.4 = 1 Then ' portc.4 is used as vbus detection on stk526

This is used with the STK526. If you want to use a different pin, you have to change PINC.4.

When you use the USB1287 this is not needed since the 1287 has a Usbsta register which can determine if a device is plugged or removed.

The USB program structure is always the same :

- 1. constants are defined that describe the end points, interfaces, vendor ID, product ID
- 2. you call a subroutine that initializes your variables
- 3. In a loop you call :
- 4. the generic USB_TASK routine so that the USB communication with the PC is executed
- 5. the specific task is called
- 6. your other code is called

This is clear in the keyboard sample :

Print "init usb task" Usb_task_init Do Usb_task Kbd_task 'call your other code here Loop

While the word Task might give you the idea that multi task switching is used, this is not the case! The USB_Task must be called by your code in order to process pending USB events. It will also find out if a device is plugged or unplugged. Events are handled in the background by the **Usb_gen_int** interrupt.

In the example the KBD_TASK is a user routine which is called in regular intervals. There is always the normal USB_TASK and there is an additional task specific to the program. In the generic-hid example this is the hid_task routine.

HID classes are simple to use since they do not require additional drivers. FTDI chips need additional drivers. But the Atmel USB chips do not need additional drivers since they use standard implemented HID classes.

When you compile the program and program it into a chip you are ready to test it. When you use FLIP you need to switch to application mode so your device can be recognized by windows. Windows will show some info that your device is found. And after installing the driver, it will report that your device is ready to be used. On the terminal emulator, press a space, and set the focus to notepad or the bascom editor. The text data from the **keys**: label is send as if it was typed on a keyboard! You in fact created a HID-keyboard, or USB keyboard. The document translatePS2-HID.pdf contains HID key codes which are different then PS2 key scan codes.

When you do not have a terminal emulator connected you can also modify the program and connect a push button. Which makes more sense for a keyboard :-)

1006 BASCOM-AVR

So modify the code into : If Inkey() = 32 Or Pinb.0 = 0 Then 'if you press SPACE BAR or make PINB.0 low

Now you can test the code without the terminal emulator.

All USB programs are similar. You specify the number of end points , the interfaces and the class. There is a lot of information available at http://www.usb.org/home

Atmel has a number of samples and you will find tools and info at various places. MCS will publish some convenient tools too.

FLIP

The USB chips are programmed with a boot loader. This is very convenient since you do not need any hardware to program the chip. FLIP can be downloaded from the Atmel site.

URL : <u>http://www.atmel.com/dyn/resources/prod_documents/Flip%20Installer%20-%203.3.1.exe</u> The FLIP website you can find at :

http://www.atmel.com/dyn/products/tools_card.asp?family_id=604&family_name=8051+ Architecture&tool_id=3886

FLIP is a Java application. The BASCOM-IDE can use the FLIP software to program the chip too. But in order to use the FLIP programmer, you need to install FLIP first.

When FLIP is working, you can select FLIP from Options, Programmer, in order to program quickly without the FLIP executable.

On Vista there is a problem with loading some of the FLIP DLL's. In case you get an error, copy the FLIP DLL's to the BASCOM application directory.

You need to copy the following files :

- atjniisp.dll
- AtLibUsbDfu.dll
- msvcp60.dll
- msvcrt.dll

You can run the **flipDLLcopy.cmd** file from the BASCOM application directory to copy these files. The content of the command file :

copy "c:\program files\atmel\flip 3.3.1\bin\atjniisp.dll" . copy "c:\program files\atmel\flip 3.3.1\bin\AtLibUsbDfu.dll" . copy "c:\program files\atmel\flip 3.3.1\bin\msvcp60.dll" . copy "c:\program files\atmel\flip 3.3.1\bin\msvcrt.dll" . pause

The last line pauses so you can view the result. Notice the . (dot) that will copy the file to the current directory, which is the reason that you need to run this file from the BASCOM application directory.

As with other programmers, you press F4 to program the HEX file into the chip. A small window will become visible.

A number of dialogs are possible:

BASCOM-AVR	X
Device not supported : ATmega88	
	ОК

In this case, you try to program a chip which is not supported by FLIP. The Mega88 is not an USB chip so the error makes sense.

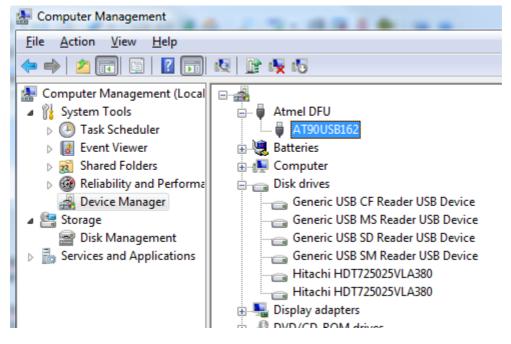
The next dialog informs you about a missing DFU device.

BASCOM FLIP Programmer
 Push the HWB (Hardware Bootloader) button Push the RST (Reset) button Release the RST button Release the HWB button Close this message box
OK Cancel

In this case, the boot loader is not found. You can run the boot loader by following the sequence from the dialog box.

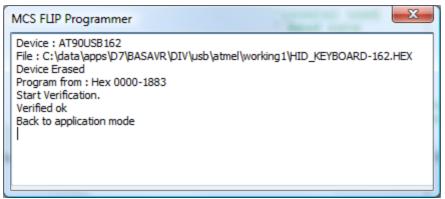
In order to make this work, the HWB and RST input both need a small switch to ground. When HWB is pressed(low) during a reset, the boot loader will be executed.

In the device manager you will find the USB device :

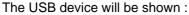


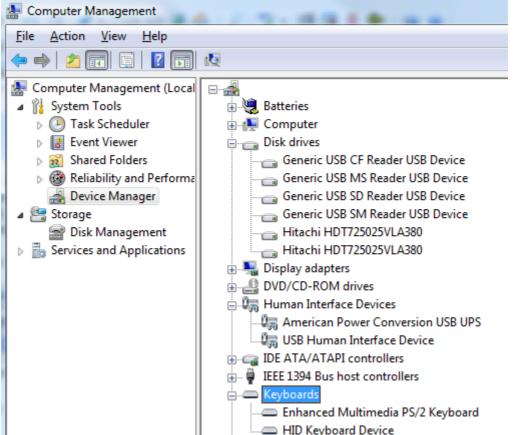
When you have a different chip, a different device will be shown !

When the programming succeeds, and there is no verify error, the application mode will be selected. This will disconnect the DFU and will connect your USB device !



The FLIP programmer window will be closed automatic when the programming succeeds.





Since you created a keyboard device, the device will be shown under the KEYBOARDS node.

When you load a generic HID device it will be shown under HUMAN INTERFACE DEVICES



HID Generic

The generic HID class is the class that is well suited for transferring bytes between the PC and the micro processor.

As with any USB application, you specify the number of end points, The example just transfers 8 bytes in and 8 bytes out.

You need to change the Ep_in_length_1, Ep_out_length, Length_of_report_in and Length_of_report_out constants when you want to transfer a different amount of bytes. You also need to take into account the maximum data size which will depend on the used chip.

The **Usb_user_endpoint_init** sub routine also need to be adjusted. The size_8 constant specifies how many bytes are used by the endpoint.

'init the user endpoints Sub Usb_user_endpoint_init(byval Nm As Byte) Call Usb_configure_endpoint(ep_hid_in, Type_interrupt, Direction_in, Size_8, One_bank, Nvet enabled) Call Usb_configure_endpoint(ep_hid_out, Type_interrupt, Direction_out, Size_8, One_bank, Nvet enabled) End Sub As with all USB program, we first initialize the USB task and the HID task. Then we call the tasks in a loop; Usb_task_init ' init the usb task Hid_task_init ' init the USB task Do 'call this subroutine once in a while Usb_task Hid task 'call this subroutine once in a while 'you can call your sub program here Loop The Hid task itself is very simple : Sub Hid_task() If Usb_connected = 1 Then ' Check USB HID is enumerated Usb_select_endpoint Ep_hid_out ' Get Data Repport From Host If Ueintx.rxouti = 1 Then 'Is usb receive out()) Dummy1 = Uedatx : Print "Got : " ; Dummy1 ' it is important that you read the same amount of bytes here as were sent by the host ! Dummy2 = Uedatx : Print "Got : " ; Dummy2 Dummy = Uedatx : Print "Got : " ; Dummy Dummy = Uedatx : Print "Got : " ; Dummy Dummy = Uedatx : Print "Got : " ; Dummy Dummy = Uedatx : Print "Got : " ; Dummy Dummy = Uedatx : Print "Got : "; Dummy Dummy = Uedatx : Print "Got : " ; Dummy Usb_ack_receive_out End If If Dummy1 = &H55 And Dummy2 = &HAA Then ' Check if we received DFU mode command from host Usb detach ' Detach Actual Generic Hid Application Waitms 500 Goto &H1800 'goto bootloader 'here you could call the bootloader then End If Usb_select_endpoint Ep_hid_in ' Ready to send these information to the host application If Ueintx.txini = 1 Then ' ls_usb_in_ready()) Uedatx = 1Uedatx = 2Uedatx = 3Uedatx = 4Uedatx = 5Uedatx = 6Uedatx = 7 Uedatx = 8' Send data over the USB Usb_ack_fifocon End If End If End Sub

We first check if the device is connected to the USB bus. Then we use Usb_select_endpoint with

the number of the end point, to select the end point.

When we want to communicate with an end point, we always have to select this end point using the **Usb_select_endpoint** procedure.

In the sample, we first select the EP_HID_OUT end point. We check the UEINTX.RXOUTI flag to determine if we received an interrupt with data. If that is the case, we read the UEDATX register to read the data byte.

The UEDATX register is the USB data register. When you read it, you read data from the USB bus. When you write it, you write data to the USB bus.

After reading the bytes you MUST acknowledge with the **Usb_ack_receive_out** macro.

The sample also shows how to run the boot loader from your code. In order to run the boot loader you must detach the current device from the USB bus. Then there is some delay to have windows process it. Finally the GOTO jumps to the boot loader address of the USB162.

If you want to write some data back, you need to select the end point, and check if you may send data. If that is the case, you assign the data to the UEDATX register and finally, you MUST acknowledge with the USB_ACK_FIFOCON macro.

Finally, you will find in the report data the length of the end points specified : Data &H75 , &H08 You need to adjust these values when you want to send/receive more data.

HIDX.OCX

There are plenty of examples on the internet that show how to communicate with HID devices using the windows API.

The HIDX.OCX is an OCX control that can be used for simple communication. Like all OCX controls, you must register it first with REGSVR32 : regsvr32 hidx.ocx After it has been registered you can run the VB test application named HIDdemo.exe.

The application will list all HID devices :

S. Form1	
Device VID=051D PID=0002 Device VID=16D0 PID=201D	
Device VID=16D0 PID=201D	CheckOut
	Checkin
	Write
I	
Device changed	,

Our device is the device with VID 16D0 and PID 201D.

There can only be one application/process at the time that communicates with an USB device. You must click the checkout-button the device to start communication. This will call the

SelectDevice method of the OCX.

As soon as you do this, you will notice that the **OnDataRead** event will receive data.

5. Form1	
Device VID=051D PID=0002	
Device VID=16D0 PID=201D	CheckOut
	Checkin
	Write
Device changed 1 selected Device : 1, report: 0, size: 8, data: 1 U -•0 Device : 1, report: 0, size: 8, data: 1 U -•0	
Device: 1, report: 0, size: 8, data: 1, $\Box = 1$ Device: 1, report: 0, size: 8, data: 1, $\Box = 1$ Device: 1, report: 0, size: 8, data: 1, $\Box = 1$ Device: 1, report: 0, size: 8, data: 1, $\Box = 1$ Device: 1, report: 0, size: 8, data: 1, $\Box = 1$	
Device : 1, report: 0, size: 8, data: ¬□□-□ Device : 1, report: 0, size: 8, data: ¬□□-□ Device : 1, report: 0, size: 8, data: ¬□□-□ Device : 1, report: 0, size: 8, data: ¬□□-□	

The event has the following parameters : (ByVal Device As Long, ByVal ReportID As Long, ByVal Data As String, ByVal Size As Long)

The device is a number with the index of all HID devices. The first device will have number 0. The report number is passed in ReportID. The data is passed as a string. You can use MID to access this data : firstByte= Asc(Mid(data,1,1))

To write to the device, you can use the **WriteDevice** method. The same parameters are used as with the OnDataRead event.

Example : WriteDevice curdev, 0, s, 8

Curdev is the index of the device. 0 is the report ID and s contains the data. You must specify the length of the data to send.

To stop communication you can click the Checkin-button. This will call the ReleaseDevice method.

When the device changes, or will be removed or inserted, you will receive a notification. In the sample program, all these events will result in a release of the device. This is done since the curdev variable can change when a new device is added. The index will not correspond to the existing index then anymore. The sample is very simple. In an application you could add a function or procedure that will examine the new list of devices and return the index of our device. When our device is found we could open it automatic again.

Notice that you can not add too much lines to a listbox in VB. Since data arrives at a very high rate, it will not take long before VB/Windows will give some error.

Property	Description
NumCheckedInDevic	Number of available devices
es	
NumCheckedOutDevi	Number of devices that are checked out and communicating.
ces	

	l
NumUnpluggedDevic es	
	The time in mS that the HID thread will sleep. You can see this as a timer interval. The lower the interval the more process time it will take. 100 mS is a good value for most applications.
Version	The version of the control
DeviceCount	The number of devices.
Methods	
SelectDevice	Parameters
	 Device : LONG that specifies the index of the device to select. The index starts at 0.
ReleaseDevice	 Parameters Device : LONG that specifies the index of the device to release. The index starts at 0.
WriteDevice	 Parameters Device : LONG that specifies the index of the device to write to. The index starts at 0. Report : LONG that specifies the report number. This would be 0 in most cases. Data : string that contains the data to send. Size : the length of the data to send.
Events	
OnDeviceChange	Parameters • none. This event fires when a device changes. This can be because a
	new device is added, or a device is removed.
OnDeviceArrival	ParametersDevice : LONG that specifies the index of the device that arrived. The index starts at 0.
	This event fires when a device is inserted. When a device is added or removed, the index that was used previously, does not need to match the new index anymore. For this reason you have to checkout the device again.
OnDeviceRemoval	 Parameters Device : LONG that specifies the index of the device that has been removed. The index starts at 0.
	This event fires when a device is removed. When a device is added or removed, the index that was used previously, does not need to match the new index anymore. For this reason you have to checkout the device again.
OnDataRead	 Parameters Device : LONG that specifies the index of the device that sent data. The index starts at 0. ReportID : LONG with the report ID of the device that sent the data. Data : string that contains the data. This string might contain 0-bytes. Size : LONG that contains the length of the received data. When data is received you can read it in this event. For

example : dim ar(8) as Byte For J=1 to Size ar(j) = ASC(Mid(data,J,1)) ' fill the array Next

The OCX can be used with all programming languages that can host OCX controls. The OCX was tested with Delphi and VB.

Your windows must support USB in order to use the OCX. So it will not work on Windows 95.

Virtual COM sample

The virtual COM demo shows how to implement an USB device with a virtual COM port. The Demo will echo data sent to the UART to the USB and vise versa.

When you compile and program the sample, you will notice that you find a new COM port in the device manager.

When you press CTRL+D, BASCOM will launch the device manager.

Device Manager	
jile <u>A</u> ction <u>V</u> iew <u>H</u> elp	
= 🔿 📰 📴 🔽 📷 😣 😫 🍢 🐻	
a- 🏯	
🗄 💘 Batteries	ſ
🕀 🚛 Computer	
🗈 👝 Disk drives	
👜 📲 Display adapters	
🗈 🔮 DVD/CD-ROM drives	
👜 🦣 Human Interface Devices	
🗈 😋 IDE ATA/ATAPI controllers	
🗄 🟺 IEEE 1394 Bus host controllers	
🖶 🛲 Keyboards	
🕀 🖞 Mice and other pointing devices	
🖶 🛄 Monitors	
🖶 🛄 Multifunction adapters	
🕢 🔮 Network adapters	
🕢 🖬 Portable Devices	
Ports (COM & LPT)	
Prolific USB-to-Serial Comm Port (COM3)	
🖶 🛄 Processors	
🖅 📲 Sound, video and game controllers	
🗈 💠 Storage controllers	
🕢 🚛 System devices	
😥 🖶 Universal Serial Bus controllers	

As you can see, the CDC class is used for the virtual COM port. As with most virtual COM devices, you can change the settings :

1014 **BASCOM-AVR**

AT90USBxxx CDC USB to UART MGM (COM4) Properties		
General Port Settings Driver Details		
<u>B</u> its per second: 19200 ▼		
Data bits: 8		
Parity: None		
Stop bits: 1		
Flow control: None		
<u>A</u> dvanced <u>R</u> estore Defaults		
OK Cancel		

In the BASCOM application the procedure Cdc_get_line_coding is called when the PC need to know the settings.

The Cdc_set_line_coding is called when the settings are changed by the user. You need to change the settings according to the received parameters.

Notice that these settings are virtual too : for the USB it does not matter how the baud rate is set ! Only for a real UART this is important. For an USB-RS232 converter for example it is very convenient to be able to change the baud rate and other settings. But when you just use the USB port for communication, and choose to use the COM port in your program as a way for communication, then you do not really need the settings.

When you want to send date to the USB/COM you can use the Uart_usb_putchar procedure. Like any USB routine, it will select the proper end point. After the end point for sending data is selected it will wait if it may send data, and finally it will send this data. The Uart usb getchar() function can be used to receive data from the USB/COM.

When you create your own device, the virtual COM port has the advantage that the PC application is simple. In most cases you already have the experience to read/write data to the PC COM port. The disadvantage is that it requires mode code. It also need an INF file. This INF file you can change to suite your own needs.

When you create your own device, the HID device is the simplest way to go.

CDC INF file

The CDC INF file looks like this. The bold parts need to be changed if you want to customize with your own text and VID/PID.

; Windows 2000, XP & Vista setup File for AT90USBxx2 demo

[Version] Signature="\$Windows NT\$" Class=Ports ClassGuid={4D36E978-E325-11CE-BFC1-08002BE10318}

Provider=%ATMEL% LayoutFile=layout.inf DriverVer=10/15/1999,5.0.2153.1

[Manufacturer] %ATMEL%=ATMEL

[ATMEL] %ATMEL_CDC%=Reader, USB\VID_03EB&PID_2018

[Reader_Install.NTx86] ;Windows2000

[DestinationDirs] DefaultDestDir=12 Reader.NT.Copy=12

[Reader.NT] include=mdmcpq.inf CopyFiles=Reader.NT.Copy AddReg=Reader.NT.AddReg

[Reader.NT.Copy] usbser.sys

[Reader.NT.AddReg] HKR,,DevLoader,,*ntkern HKR,,NTMPDriver,,usbser.sys HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider"

[Reader.NT.Services] AddService = usbser, 0x00000002, Service_Inst

[Service_Inst] DisplayName = %Serial.SvcDesc% ServiceType = 1 ; SERVICE_KERNEL_DRIVER StartType = 3 ; SERVICE_DEMAND_START ErrorControl = 1 ; SERVICE_ERROR_NORMAL ServiceBinary = %12%\usbser.sys LoadOrderGroup = Base

[Strings] ATMEL = "ATMEL, Inc." ATMEL_CDC = "AT90USBxxx CDC USB to UART MGM" Serial.SvcDesc = "USB Serial emulation driver"

;---- END OF INF FILE

You can also change the key names.

8.19 MODBUS Slave/Server

The MODBUS protocol is used a lot in the industry. With the MODBUS add-on, you can create a slave or server.

This add-on is a MODBUS server-RTU that implements function 03,06 and 16. (decimal)

We use the term master and slave to indicate that there is at least one master, and that there is at least one slave device that will respond.

A slave could be a master too. Another term is client/server. The server is the MODBUS device that will respond to the client. It is the same as master/slave and thus slave=server and master=client.

Like a web server, the server does not initiate the communication. It simply waits for data and when it is addressed, it will respond.

When it is not addressed, it should not respond. When it is addressed, it should process the data and send a response.

A client sends the following data : server address, function, data, checksum The server address is a byte , the function code is a byte too. The data depends on the function and the checksum is a 16 bit CRC checksum.

MODBUS uses the term registers for the data. A register is 16 bit width. You can pass words or integers with a single register.

In order to send a long, single, double or string, you need to send multiple registers.

There are a lot of functions defined in the MODBUS protocol. The add-on implements the functions that are most suited for an own MODBUS server device. These functions are :

- 03 : read (multiple) register(s)
- 06 : write a single register
- 16 : write multiple registers

If needed you can add other functions yourself. The implemented functions should be sufficient however.

Constants

There are a few constants that you might need to change.

Registersize : this constant defines how many registers can be processed. For example if a client asks to return 10 registers with function 03, you should set this constant to 10.

The reason for the constant is that RAM space is limited. And each register need storage space (2 bytes for each register) thus we do not want to take more bytes then needed.

Mdbg : this can be used for debugging. The add-on uses a Mega162 since it has 2 UARTS. One UART can be used for debugging. You need to set mdbg to a non-zero value to enable debugging to the serial port.

RS232-RS485

The protocol can be used with RS-232 and RS-485 and TCP/IP, etc. The add-on can be used with RS-232 and RS-485.

RS-485 half duplex needs a data direction pin. It is defined in the source like this :

Rs485dir Alias Portb.1 Config Rs485dir = Output Rs485dir = 0 'Config Print1 = Portb.1 , Mode = Set

You can remark or remove the mark depending on the mode you need. For testing, RS-232 is most simple.

TIMER

A timer is used to detect the start of a frame. With RTU (binary data) a silence of 3.5 characters is needed between frames. A frame is a complete MODBUS message. A timer is used to detect such a silence. The statement : GENRELOAD , is used to generate the proper timer divisor and timer reload values. GENRELOAD will only work on TIMER0 and TIMER1. You pass the names of the constants which are free to chose, and in the sample are named _RL and _TS, and these constant values will be calculated and assigned to constants by the compiler.

The TM_FRAME constant is the time of 4 characters. When the timer reaches this value it will overflow and execute the ISR_TMR0 interrupt. The interrupt routine will set the start state since now the server can expect an address.

In the TM_FRAME calculation the baud rate value is used. In the add on this is 9600. When you use a different value, you need to change the constant here as well.

Server Address

The server address need to be set. The MBSLAVE variable need to be set by you. Optional, you could change the variable into a constant.

But when you use a DIP switch for example to set the address, it is better to use a variable.

Event mode

The MODBUS handeling is coded into a state machine and executed as a task. You can call the Modbustask() in your code yourself in the main program loop, or you can have it called in the interrupt of the buffered serial input routine. The sample uses the last option :

Config Serialin1 = Buffered , Size = 50 , Bytematch = All

Notice that BYTEMATCH = ALL is used so the Serial1bytereceived routine is called for every received byte. If the state is right, the modbustask code is executed and otherwise, the data is read to remove it from the buffer. Since there can be multiple slaves, the data will keep coming and we may only handle the data when we are addressed.

Functions

Each function that is requested will call a sub routine.

Function 03 (read registers) : Sub Modbus03(addr3 As Word , Idx3 As Byte , Wval3 As Word)

addr3 contains the address that was passed by the client.

Idx3 contains an index in case multiple registers are read. It is 1 for the first register, 2 for the second, etc.

With these 2 values you can fill the wval3 value.

In the sample, a select case is used to send different values.

You should NOT change the addr3 and idx3 values ! There variables are passed by reference and changes will corrupt the data.

1018 BASCOM-AVR

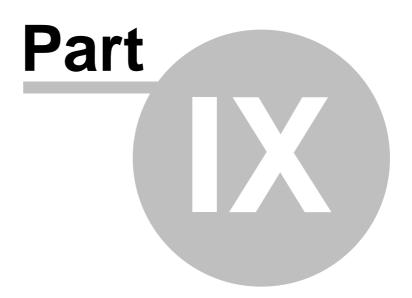
Notice that the function is called for each register. When the client want to read 2 word registers, the sub routine is called twice.

Function 06(write register) Sub Modbus06(addr3 As Word , Wval3 As Word) Addr3 contains the address that was passed by the client. wval3 contains a word value passed by the client. You can use the address to change some variable in your code.

Function 16 (write multiple registers) Sub Modbus16w(addr3 As Word , Idx As Byte , Bw As Word)Addr3 contains the address send by the client.Idx contain the index to a word register.Bw contains the value that was send.

Notice that the sub routine is called for each register. You can use the address and index to alter the proper variable in your code.

For functions that are not implemented, an error response will be sent.



9 Tools

9.1 LCD RGB-8 Converter

Action

This tool is intended to convert normal bitmaps into BGC files. The BGC format is the **B**ascom **G**raphic **C**olor Format. This is a special RLE compressed format to save space.

The SHOWPIC statement can display graphic bitmaps. The color display uses a special RGB8 format. The LCD converter has the task to convert a normal windows bitmap into a 256-color RGB8 coded format.

When you run the tool you will see the following window :

🍄 MCS Electronics LCD RGB-8 Converter
File Edit Help 🗸
[Image] Colors Output

You can use File , Open, to load an image from disk. Or you can use Edit, Paste, to paste an image from the clipboard.

Option	Description
File, Open	Open a graphical file from disk.
File, Save, Image	Save the file as a windows graphical file
File, Save, Binary	Save the BGC file, the file you need with SHOWPIC
File, Save , Data Lines	Save the file as data lines into a text file
File, Convert	Converts the bitmap into a RGB8 bitmap
Edit, Bitmap height	height of the image. Change it to make the image smaller or larger
Edit, Bitmap width	width of the image. Change it to make the image wider.
Edit, Select All	Select entire image
Edit, Copy	Copy selection to the clipboard
Edit, Paste	Paste clipboard to the selection. You must have an area

	selected !
Edit, Delete	Delete the selected area

The Output TAB, has an option : Save as RLE. This must be checked. By default it is checked.

When you do not want the image to be RLE encoded, you can uncheck this option.

The bottom area is used to store the DATA lines.

The Color TAB shows the effect on the table inside the color display. When a picture uses a lot of different red colors, you can put the most used into the table.

It is well explained in the manuals from display3000.

By clicking on the color , you can view which colors are used by the picture. You can match them with the color table.

You can download the LCD Converter tool from : <u>http://www.mcselec.com/index.php?</u> <u>option=com_docman&task=doc_download&gid=168&Itemid=54</u>

Index

- # -

1022

#AUTOCODE 307 #IF ELSE ENDIF 307

- \$ -

\$AESKEY 309 \$ASM 309 \$BAUD 310 \$BAUD1 311 \$BGF 312 \$BIGSTRINGS 314 \$BOOT 314 \$CRYPT 315 \$CRYSTAL 316 \$DATA 316 \$DBG 318 \$DEFAULT 320 \$EEPLEAVE 321 \$EEPROM 321 \$EEPROMHEX 322 \$EEPROMSIZE 323 \$END ASM 309 \$EXTERNAL 324 \$FILE 325 \$FRAMECHECK 327 \$FRAMESIZE 325 \$HWCHECK 327 \$HWSTACK 326 \$INC 330 \$INCLUDE 330 \$INITMICRO 331 \$LCD 332 \$LCDPUTCTRL 335 \$LCDPUTDATA 336 \$LCDRS 337 \$LCDVFO 340 \$LIB 340 \$LOADER 343 \$LOADERSIZE 348 \$MAP 349 \$NOCOMPILE 350 \$NOINIT 351

\$NORAMCLEAR 351 \$NORAMPZ 352 \$NOTRANSFORM 352 \$PROG 354 \$PROGRAMMER 355 \$PROJECTTIME 353 \$REGFILE 356 \$RESOURCE 357 \$ROMSTART 360 \$SERIALINPUT 360 \$SERIALINPUT1 362 \$SERIALINPUT2LCD 363 \$SERIALOUTPUT 363 \$SERIALOUTPUT1 364 \$SIM 365 \$SOFTCHECK 327 \$SWSTACK 365 \$TIMEOUT 366 \$TINY 368 \$VERSION 369 \$WAITSTATE 370 \$XA 370 \$XRAMSIZE 371 \$XRAMSTART 371

- 1 -

1WIRECOUNT 372 1WIRESEARCHNEXT 382 1WREAD 377 1WRESET 375 1WSEARCHFIRST 379 1WVERIFY 384 1WWRITE 386

- A -

ABS 388 ACOS 389 Adding XRAM 161 Additional Hardware 152 ADR 390 ADR2 390 AESDECRYPT 394 AESENCRYPT 396 ALIAS 398 ARDUINO 131 ARRAY 274 ASC 399

© 2011 MCS Electronics

ASCII chart 303	ATMEGA48P 219
ASIN 402	ATMEGA603 236
Assembler mnemonics 291	ATMEGA64 221
AT_EMULATOR 994	ATMEGA640 238
	ATMEGA644P 239
AT90CAN128 202	ATMEGA645 240
AT90PWM216 201	ATMEGA649 241
AT90PWM2-3 200	ATMEGA64M1 216
AT90S1200 192	ATMEGA8 212
AT90S2313 192	ATMEGA8515 244
AT90S2323 193	ATMEGA8535 244
AT90S2333 194	ATMEGA88 220
AT90S2343 194	ATMEGA88P 220
AT90S4414 196	ATMEGA8U2 216
AT90S4433 196	ATN 403
AT90S4434 198	ATN2 404
AT90S8515 199	Attaching an LCD Display 162
AT90S8535 199	ATtiny12 203
AT90USB162 203	ATtiny13 204
ATMEGA103 221	ATtiny15 204
ATMEGA128 223	ATTINY167 209
ATMEGA1284P 223	ATTINY20 204
ATMEGA16 213	ATtiny22 205
ATMEGA161 225	ATtiny2313 211
ATMEGA162 225	ATtiny24 205
ATMEGA163 226	ATtiny25 205
ATMEGA164P 227	ATtiny26 206
ATMEGA165 228	ATtiny261 210
ATMEGA168 229	ATtiny4313 212
ATMEGA168P 229	ATtiny44 206
ATMEGA169 230	ATtiny45 207
ATMEGA16M1 216	ATtiny461 210
ATMEGA16U2 216	ATtiny48 207
ATMEGA16U4 214	ATtiny84 208
ATMEGA2560 242	ATtiny85 208
ATMEGA2561 243	ATtiny861 211
ATMEGA32 215	ATtiny88 208
ATMEGA323 230	ATXMEGA 245
ATMEGA324P 231	ATXMEGA128A1 252
ATMEGA325 232	ATXMEGA128A3 254
ATMEGA3250P 233	ATXMEGA128D3 254
ATMEGA328 234	ATXMEGA128D4 255
ATMEGA328P 234	ATXMEGA16A4 245
ATMEGA329 235	ATXMEGA16D4 246
ATMEGA32M1 216	ATXMEGA192A3 256
ATMEGA32U 218	ATXMEGA192D3 257
ATMEGA32U2 216	ATXMEGA256A3 258
ATMEGA406 235	ATXMEGA256A3B 259
ATMEGA48 219	ATXMEGA256D3 260

1023

ATXMEGA32A4 247 ATXMEGA32D4 247 ATXMEGA64A1 248 ATXMEGA64A3 250 ATXMEGA64D3 250 ATXMEGA64D4 251 AVR Internal Hardware 152 AVR Internal Hardware Port B 158 AVR Internal Hardware Port D 160 AVR Internal Hardware Watchdog timer 158 AVR Internal Registers 153 AVR ISP Programmer 113 AVR-DOS File System 976

147

- B -

BASCOM Editor Keys BASE64DEC 405 BASE64ENC 406 BAUD 407 BAUD1 408 BCCALL 996 BCD 409 BCDEF 996 BCINIT 458 BCRESET 1002 BIN 411 BIN2GRAY 413 BINVAL 412 BIT 274 BITS 415 BITWAIT 414 BLOAD 416 BOX 417 BOXFILL 419 BSAVE 420 BUFSPACE 421 BYTE 274 BYVAL 421

- C -

CALL 422, 555 CASE 852 Changes compared to BASCOM-8051 273 CHARPOS 424 CHECKFLOAT 425 CHECKSUM 427 CHECKSUMXOR 427 CHR 428 CIRCLE 429 CLEAR 432 CLOCKDIVISION 436 CLOSE 437 CLOSESOCKET 439 CLS 433 Compact FlashCard Driver 980 CONFIG 442 **CONFIG 1WIRE** 444 CONFIG ACI 448 CONFIG ADC 451 CONFIG ADCx 453 CONFIG ATEMU 455 CONFIG BASE 457 CONFIG CLOCK 461 CONFIG CLOCKDIV 465 CONFIG COM1 466 CONFIG COM2 467 CONFIG COMx 469 CONFIG DAC 471 CONFIG DACA 471 CONFIG DACB 471 CONFIG DATE 473 CONFIG DCF77 475 CONFIG DEBOUNCE 482 CONFIG DMA 483, 484 CONFIG DMACHx 484 CONFIG DMXSLAVE 488 490 CONFIG DP CONFIG EEPROM 491 CONFIG EVENT SYSTEM 492 CONFIG EXTENDED PORT 493 CONFIG GRAPHLCD 506 CONFIG HITAG 494 CONFIG I2CDELAY 498 CONFIG I2CSLAVE 500 CONFIG INPUT 503 CONFIG INTx 504 CONFIG KBD 511 CONFIG KEYBOARD 513 CONFIG LCD 516 CONFIG LCDBUS 520 CONFIG LCDMODE 523 CONFIG LCDPIN 523 CONFIG OSC 527 CONFIG PORT 528 CONFIG POWER REDUCTION 530 CONFIG PRINT 534

Index

CONFIG PRINT0 534 CONFIG PRINT1 534 534 CONFIG PRINT2 CONFIG PRINT3 534 CONFIG PRINTBIN 535 CONFIG PRIORITY 533 CONFIG PS2EMU 535 CONFIG RC5 538 CONFIG SCL 539 CONFIG SDA 538 CONFIG SERIALIN 540 CONFIG SERIALOUT 545 CONFIG SERVOS 552 CONFIG SHIFTIN 548 CONFIG SINGLE 547 CONFIG SPI 548 CONFIG SPIC, SPID, SPIE, SPIF 550 CONFIG SUBMODE 555 CONFIG SYSCLOCK 556 CONFIG TC 557 CONFIG TCPIP 559 CONFIG TIMER0 562 CONFIG TIMER1 564 CONFIG TIMER2 567 CONFIG TWI 569 CONFIG TWISLAVE 571 CONFIG USB 575 CONFIG VPORT 582 CONFIG WAITSUART 584 CONFIG WATCHDOG 584 CONFIG X10 587 CONFIG XRAM 589 CONST 594 Constants 163 COS 596 COSH 597 COUNTER0 and COUNTER1 597 CPEEK 598 CPEEKH 600 CRC16 603 CRC16UNI 605 CRC32 607 CRC8 601 CRYSTAL 608 CURSOR 609 custom design 28

- D -

DATA 612 DATE 627 DATE\$ 625 DATETIME 993 DAYOFWEEK 615 DAYOFYEAR 624 DBG 637 DCF77TIMEZONE 637 DEBOUNCE 639 DEBUG 638 DECLARE 555 DECLARE FUNCTION 642 DECLARE SUB 644 **DECR 640** DEFBIT 648 648 DEFINT DEFLCDCHAR 648 DEFLNG 648 DEFSNG 648 DEFWORD 648 DEFxxx 648 DEG2RAD 649 DELAY 650 DELCHAR 651 DELCHARS 652 DIM 653 DIR 656 DISABLE 657 DISKFREE 660 DISKSIZE 661 DISPLAY 661 DMACH0 484 DMACH1 484 DMACH2 484 DMACH3 484 DO 664 DOUBLE 986 DOWNTO 691 DriveCheck 666 DriveGetIdentity 666 DriveInit 667 DriveReadSector 669 DriveReset 668 DriveWriteSector 670 DTMFOUT 671

- E -

ECHO 673 Edit Copy 51 Edit Cut 51 Edit Find 52 Edit Find Next 52 Edit Goto 52 Edit Goto Bookmark 52 Edit Indent Block 53 Edit Paste 52 Edit Redo 51 Edit Remark Block 53 Edit Replace 52 Edit Toggle Bookmark 52 Edit Undo 51 Edit Unindent Block 53 Elektor CF-Interface 981 ELSE 675, 735 ENABLE 676 ENCODER 678 ENCRYPT 53 END 680 END IF 735 END SELECT 852 EOF 681 ERAM 163 Error Codes 297 EUROTIMEDATE 993 EXIT 682 EXP 683 EXTENDED I2C 963

- F -

File Close50File Exit51File New49File Open50File Print51File Print Preview51File Save50File Save50FILEATTR684FILEDATE685FILEDATETIME686FILELEN687FILETIME687

FIX 688 FLIP 125 FLUSH 689 FM24C16 965 FM24C256 965 FM24C64 965 FM25C256 965 Font Editor 148 FOR 691 FORMAT 690 FOR-NEXT 691 FOURTHLINE 693 FP_TRIG 983 FRAC 694 FREEFILE 695 FUNCTION 642 FUSING 696

- G -

GET 697 GETADC 700 GETATKBD 703 GETATKBDRAW 707 GETDSTIP 708 GETDSTPORT 708 GETKBD 709 GETRC 711 GETRC5 712 GETREG 716 GETSOCKET 717 GETTCPREGS 716 GLCD 970 GLCDCMD 718 GLCDDATA 718 glcdR7565R 973 GLCDSED 971 GOSUB 719 GOTO 720 GRAY2BIN 720

- H -

Help About 142 Help Credits 146 Help Index 143 Help Knowledge Base 145 Help MCS Forum 143 Help MCS Shop 144

1026

Help Support 145 HEX 722 HEXVAL 722 HIGH 723 HIGHW 724 HOME 724

- | -

I2C TWI Slave 988 I2C TWI 966 I2CINIT 725 I2CRBYTE 728 I2CRECEIVE 726 I2CREPSTART 728 12CSEND 727 I2CSLAVE 986 I2CSTART 728 I2CSTOP 728 I2CSTOP: I2CRBYTE: I2CWBYTE 728 **I2CWBYTE** 728 I2START 728 IDLE 734 IF 735 **IF-THEN-ELSE-END IF** 735 INCR 736 Index 22 INITFILESYSTEM 737 INITLCD 738 INKEY 738 INP 740 INPUT 744 INPUTBIN 741 INPUTHEX 742 **INSERTCHAR** 746 Install on multiple computers 45 Installation of BASCOM 30 INSTR 747 INT 748 INTEGER 274 IP2STR 749 ISCHARWAITING 750 ISP programmer 103



Keyword Reference 23 KILL 751 KITSRUS Programmer 106

- L -

Language Fundamentals 274 Lawicel BootLoader 112 LCASE 751 LCD 752 LCD RGB-8 Converter 1020 LCD4.LIB 969 LCD4BUSY 968 LCD4E2 970 LCDAT 755 LCDAUTODIM 757 LCDCONTRAST 758 LCD-EPSON 973 LEFT 758 LEN 759 LIBUSB 131 759 LINE LINE INPUT 762 LINEINPUT 762 LOAD 764 LOADADR 764 LOADLABEL 765 LOADWORDADR 765 LOC 766 LOCAL 768 LOCATE 771 LOF 767 LOG 772 LOG10 772 LONG 274 LOOKDOWN 772 LOOKUP 774 LOOKUPSTR 775 LOOP 664 LOW 776 LOWERLINE 776 LTRIM 763 - M -

MACRO 777 MAKEBCD 778 MAKEDEC 779 MAKEINT 778 MAKEMODBUS 780 MAKETCP 782 MAX 782

BASCOM-AVR

MCS Bootloader 122 MCS Universal Interface Programmer 107 MCSBYTE 966 MCSBYTEINT 967 MEMCOPY 784 Memory usage 163 MID 787 MIN 785 Mixing ASM and BASIC 286 MOD 787

- N -

1028

NBITS 788 New CF-Card Drivers 983 Newbie problems 301 NEXT 691

- 0 -

ON INTERRUPT 789 ON VALUE 792 794 OPEN **Options Communication** 95 Options Compiler 93 **Options Compiler 1WIRE** 93 Options Compiler Chip 90 **Options Compiler Communication** 92 **Options Compiler I2C** 93 Options Compiler LCD 94 Options Compiler Output 91 **Options Compiler SPI** 93 Options Environment 96 **Options Monitor** 140 **Options Printer** 140 Options Programmer 101 Options Simulator 100 OUT 802

- P -

PCF8533 971 PEEK 803 PG302 programmer 104 POKE 804 POPALL 805 POWER 805 POWER MODE 808 Power Up 191 POWERDOWN 809 POWERSAVE 809 PRINT 810 PRINTBIN 812 PROGGY 124 Program Compile 61 Program Development Order 148 Program Send to Chip 75 **Program Show Result** 63 Program Simulate 64 Program Syntax Check 61 PS2MOUSE EMULATOR 994 PS2MOUSEXY 816 PSET 813 PULSEIN 816 PULSEOUT 817 PUSHALL 818 PUT 819

- Q -

QUOTE 821

- R -

RAD2DEG 822 RC5SEND 822 RC5SENDEXT 824 RC6SEND 826 **READ 828** READEEPROM 830 READHITAG 832 READMAGCARD 835 READSIG 837 READSIGNATURE 837 REM 839 REPLACECHARS 840 Resellers 961 Reserved Words 296 RESET 840 RESTORE 841 RETURN 843 RIGHT 844 RND 844 ROTATE 846 ROUND 847 RTRIM 848 Running BASCOM-AVR 47

© 2011 MCS Electronics

Index

- S -

Sample Electronics cable programmer SECELAPSED 849 SECOFDAY 849 SEEK 850 SELECT 852 SELECT-CASE-END SELECT 852 SENDSCAN 862 SENDSCANKBD 864 SERIN 868 SEROUT 870 SET 853 SETFONT 855 SETIPPROTOCOL 872 SETREG 858 SETTCP 858 SETTCPREGS 860 SGN 874 SHIFT 875 SHIFTCURSOR 877 SHIFTIN 877 SHIFTLCD 882 SHIFTOUT 881 SHOWPIC 883 SHOWPICE 883 SIGNATURE 837 SIN 885 SINGLE 274 SINH 886 SOCKETCONNECT 886 SOCKETLISTEN 889 SOCKETSTAT 890 SONYSEND 891 SORT 894 SOUND 895 SPACE 897 SPC 898 SPIIN 899 SPIINIT 900 SPIMOVE 900 SPIOUT 901 SPISLAVE 990 SPLIT 901 SQR 903 START 904 STCHECK 906 STEP 691

STK500 Programmer 109 STK600 128 STOP 910 STR 911 STR2DIGITS 912 STRING 913 SUB 555, 914 Supported Programmers 102 SWAP 918 SYSDAY 917 SYSSEC 915 SYSSECELAPSED 916

- T -

105

TAN 920 TANH 929 TCC0 557 TCC1 557 TCD0 557 TCD1 557 TCPCHECKSUM 920 TCPIP 967 TCPREAD 924 TCPWRITE 924 TCPWRITESTR 925 THEN 735 THIRDLINE 930 TIME 931 TIME\$ 930 TIMER 557 TIMER0 155 TIMER1 156 Tips and tricks 302 TOGGLE 933 **Tools Batch Compile** 84 **Tools Graphic Converter** 82 Tools LCD Designer 79 Tools LIB Manager 81 Tools PDF Update 87 Tools Plugin Manager 83 Tools Resource Editor 88 Tools Stack Analyzer 83 Tools Terminal Emulator 78 TRIM 935 TWI START 569

- U -

UCASE 936 UDPREAD 937 UDPWRITE 940 UDPWRITESTR 941 UPPERLINE 945 USB-ISP Programmer 113 Using RS485 171 Using the 1 WIRE protocol 180 Using the I2C protocol 173 Using the SPI protocol 183 USING the UART 165

- V -

VAL 945 VARPTR 946 VER 947 VERSION 947 View Error Panel 60 View PDF viewer 58 View PinOut 54 View Tip 60

- W -

WAIT 948 WAITKEY 949 WAITMS 950 WAITUS 951 WEND 952 WHILE 952 WHILE-WEND 952 Window Arrange Icons 141 141 Window Maximize All Window Minimize All 141 Window Tile 141 Windows Cascade 141 WORD 274 WRITE 953 WRITEEEPROM 954



X10DETECT 956 X10SEND 958 XRAM 163 XRAM CF-Interface for simulation 982

© MCS Electronics 1995-2011

www.mcselec.com

Making BASIC and Micro's Easy