BASCOM AVR

Help Reference

MCS ELECTRONICS

EMBEDDED SYSTEMS
BASIC COMPILERS
DEVELOPMENT

Making things easy

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

Printed: 31-10-2008

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

Part I

1 Index

Version 1.11.9.3 document build 22

1.1 Keyword Reference

1WIRE

1Wire routines allow you to communicate with Dallas 1wire chips.
1WRESET 314 , 1WREAD 317 , 1WWRITE 326 , 1WSEARCHFIRST 319 , 1WSEARCHNEXT 321 ,1WVERIFY 324 , 1WIRECOUNT 312

Conditions

Conditions execute a part of the program depending on a condition being True or False
IF-THEN-ELSE-END IF 613 , WHILE-WEND 813 , ELSE 560 , DO-LOOP 550 , SELECT CASE - END SELECT 719 , FOR-NEXT 576

Configuration

Configuration commands initialize the hardware to the desired state.
CONFIG 375 , CONFIG ACI 379 , CONFIG ADC 380 , CONFIG BCCARD 384 , CONFIG CLOCK 387 , CONFIG COM1 390 , CONFIG COM2 392 , CONFIG DATE 395 , CONFIG PS2EMU 440 , CONFIG ATEMU 382 , CONFIG I2CSLAVE 411 , CONFIG INPUT 413, CONFIG GRAPHLCD 416 , CONFIG KEYBOARD 423 , CONFIG TIMER0 459 , CONFIG TIMER1 461 , CONFIG LCDBUS 430 , CONFIG LCDMODE 433 , CONFIG 1WIRE 377 , CONFIG LCD 426 , CONFIG SERIALOUT 449 , CONFIG SERIALIN 444 , CONFIG SPI 453 , CONFIG LCDPIN 433 , CONFIG SDA 443 , CONFIG SCL 444 , CONFIG DEBOUNCE 403 , CONFIG WATCHDOG 478 , CONFIG PORT , 436 COUNTER0 AND COUNTER1 486 , CONFIG TCPIP 456 , CONFIG TWISLAVE 467 , CONFIG SINGLE 451 , CONFIG X10 480 , CONFIG XRAM 482 , CONFIG USB 471

Conversion

A conversion routine is a function that converts a number or string from one form to another.
BCD 345 , GRAY2BIN 603 , BIN2GRAY 349 , BIN 347 , MAKEBCD 654 , MAKEDEC 655 , MAKEINT 654 , FORMAT 575 , FUSING 581 , BINVAL 348 , CRC8 490 , CRC16 491 , CRC16UNI 494 , CRC32 496 , HIGH 606 , HIGHW 607 , LOW 651

DateTime

Date Time routines can be used to calculate with date and/or times.
DATE 516 , TIME 794 , DATE$ 514 , TIME$ 793 , DAYOFWEEK 504 , DAYOFYEAR 513 , SECOFDAY 717 , SECELAPSED 716 , SYSDAY 780 , SYSSEC 777 , SYSSECELAPSED 779

Delay

Delay routines delay the program for the specified time.
WAIT 809 , WAITMS 811 , WAITUS 812 , DELAY 538

Directives

Directives are special instructions for the compiler. They can override a setting from the IDE.
$ASM 257 , $BAUD 257 , $BAUD1 258 , $BGF 259 , $BOOT 261 , $CRYSTAL 262 , $DATA 262 , $DBG 264 , $DEFAULT 266 , $EEPLEAVE 267 , $EEPROM 267 , $EEPROMHEX 268 , $EXTERNAL 269 , $HWSTACK 271 , $INC 272 , $INCLUDE 273 , $INITMICRO 274 , $LCD 275 , $LCDRS 280 , $LCDPUTCTRL 277 , $LCDPUTDATA 279 , $LCDVFO 282 , $LIB 283 , $LOADER 285 , $LOADERSIZE 291 , $MAP 292 , $NOCOMPILE 292 , $NOINIT 293 , $NORAMCLEAR 294

, $PROG 294 , $PROGRAMMER 295 , $REGFILE 296 , $RESOURCE 297 , $ROMSTART 300 , $SERIALINPUT 300, $SERIALINPUT1 302 , $SERIALINPUT2LCD 303 , $SERIALOUTPUT 304 , $SERIALOUTPUT1 304 , $SIM 305 , $SWSTACK 306 , $TIMEOUT 307 , $TINY 308 , $WAITSTATE 309 , $XRAMSIZE 310 , $XRAMSTART 311 , $XA 310

File

File commands can be used with AVR-DOS, the Disk Operating System for AVR.
BSAVE 356 , BLOAD 352 , GET 582 , VER 808 , DISKFREE 545 , DIR 542 , DriveReset 553 , DriveInit 553 , LINE INPUT 638 , INITFILESYSTEM 615 , EOF 566 , WRITE 814 , FLUSH 574 , FREEFILE 580 , FILEATTR 569 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , FILELEN 571 , SEEK 718 , KILL 627 , DriveGetIdentity 552 , DriveWriteSector 555 , DriveReadSector 554 , LOC 642 , LOF 643 , PUT 688 , OPEN 669 , CLOSE 370

Graphical LCD

Graphical LCD commands extend the normal text LCD commands.
GLCDCMD 601 , GLCDDATA 601 , SETFONT 723 , LINE 635 , PSET 682 , SHOWPIC 749 , SHOWPICE 750 , CIRCLE 362 , BOX 353

I2C

I2C commands allow you to communicate with I2C chips with the TWI hardware or with emulated I2C hardware.
I2CINIT 608 , I2CRECEIVE 608 , I2CSEND 609 , I2CSTART,I2CSTOP,I2CRBYTE,I2CWBYTE 610

IO

I/O commands are related to the I/O pins and ports of the processor chip.
ALIAS 334 , BITWAIT 350 , TOGGLE 796 , RESET 707 , SET 721 , SHIFTIN 743 , SHIFTOUT 747 , DEBOUNCE 527 , PULSEIN 685 , PULSEOUT 686

Micro

Micro statements are specific to the micro processor chip.
IDLE 613 , POWERDOWN 678 , POWERSAVE 679 , ON INTERRUPT 664 , ENABLE 562 , DISABLE 543 , START 769 , END 565 , VERSION 809 , CLOCKDIVISION 369 , CRYSTAL 497 , STOP 775

Memory

Memory functions set or read RAM , EEPROM or flash memory.
ADR 330 , ADR2 330 , WRITEEEPROM 815 , CPEEK 487 , CPEEKH 488 , PEEK 674 , POKE 675 , OUT 673 , READEEPROM 699 , DATA 501 , INP 618 , READ 697 , RESTORE 709 , LOOKDOWN 648 , LOOKUP 650 , LOOKUPSTR 651 , CPEEKH 488 , LOAD 640 , LOADADR 640 , LOADLABEL 641 , LOADWORDADR 641 , MEMCOPY 660

Remote Control

Remote control statements send or receive IR commands for remote control.
RC5SEND 691 , RC6SEND 695 , GETRC5 596 , SONYSEND 757

RS-232

RS-232 are serial routines that use the UART or emulate a UART.

BAUD 343 , BAUD1 344, BUFSPACE 357 , CLEAR 365, ECHO 558 , WAITKEY 810 , ISCHARWAITING 626 , INKEY 617 , INPUTBIN 620 , INPUTHEX 620 , INPUT 622 , PRINT 679 , PRINTBIN 681 , SERIN 734 , SEROUT 736 , SPC 763 , MAKEMODBUS 655

SPI

SPI routines communicate according to the SPI protocol with either hardware SPI or software emulated SPI.
SPIIN 764 , SPIINIT 765 , SPIMOVE 765 , SPIOUT 766

String

String routines are used to manipulate strings.
ASC 335 , UCASE 797 , LCASE 628 , TRIM 796 , SPLIT 766 , LTRIM 639 , INSTR 624 , SPACE 762 , STRING 776 , RTRIM 715 , LEFT 634 , LEN 635 , MID 662 , RIGHT 711 , VAL 806 , STR 775 , CHR 361 , CHECKSUM 360 , HEX 604, HEXVAL 605 , QUOTE 690

TCP/IP

TCP/IP routines can be used with the W3100/IIM7000/IIM7010 modules.
BASE64DEC 341 , BASE64ENC 342 , IP2STR 626 , UDPREAD 798 , UDPWRITE 801 , UDPWRITESTR 802 , TCPWRITE 787 , TCPWRITESTR 788 , TCPREAD 786 , GETDSTIP 591 , GETDSTPORT 592 , SOCKETSTAT 756 , SOCKETCONNECT 752 , SOCKETLISTEN 755 , GETSOCKET 600 , CLOSESOCKET 372 , SETTCP 725 , GETTCPREGS 599 , SETTCPREGS 726 , SETIPPROTOCOL 738 , TCPCHECKSUM 783

Text LCD

Text LCD routines work with normal text based LCD displays.
HOME 607 , CURSOR 498 , UPPERLINE 806 , THIRDLINE 793 , INITLCD 616 , LOWERLINE 652 , LCD 629 , LCDAT 632 , FOURTHLINE 578 , DISPLAY 547 , LCDCONTRAST 634 , LOCATE 647 , SHIFTCURSOR 743 , DEFLCDCHAR 536 , SHIFTLCD 748 , CLS 366

Trig & Math

Trig and Math routines work with numeric variables.
ACOS 329 , ASIN 338 , ATN 339 , ATN2 340 , EXP 568 , RAD2DEG 690 , FRAC 579 , TAN 782 , TANH 792 , COS 485 , COSH 486 , LOG 647 , LOG10 648 , ROUND 714 , ABS 328 , INT 625 , MAX 658 , MIN 661 , SQR 768 , SGN 740, POWER 676 , SIN 751 , SINH 752 , FIX 573 , INCR 615 , DECR 529 , DEG2RAD 537

Various

This section contains all statements that were hard to put into another group
CONST 483 , DBG 525 , DECLARE FUNCTION 530 , DEBUG 526, DECLARE SUB 532 , DEFXXX 535 , DIM 539 , DTMFOUT 556 , EXIT 567 , ENCODER 563 , GETADC 585 , GETKBD 593 , GETATKBD 587 , GETRC 595 , GOSUB 602 , GOTO 603 , LOCAL 644 ,ON VALUE 667 , POPALL 675 , PS2MOUSEXY 685 , PUSHALL 687 , RETURN 710 , RND 712 , ROTATE 713 , SENDSCAN 728 , SENDSCANKBD 730 , SHIFT 741 , SOUND 760 , STCHECK 770 , SUB 777 , SWAP 781 , VARPTR 807 , X10DETECT 817 , X10SEND 819 , READMAGCARD 704 , REM 706 , BITS 351 , BYVAL 357 , CALL 358 , #IF 820 , #ELSE 820 , #ENDIF 820 , READHITAG 701

1.2 About MCS Electronics

About the founder

Since i was young i was intrigued by remote control, robots, transmitters and in short all electronics. I created countless electronic devices. I designed a lot of PCB's by hand(using ink) and when ATARI came with the ST1040 and an affordable PCB design tool, I bought my first real computer.
It turned out that the printers at that time(matrix printers) were not able to produce a good print. And the design of the PCB was still time consuming. But 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 which was easy to master. And very intuitive.

When I found out that Atmel made the 89c2051 which was a 20 pin chip with flash memory, i was excited that there was a small micro processor that could be erased/ reprogrammed without the need of UV-erasing of the EPROM. Before the Atmel chip i used the 8052AH, a BASIC interpreter. It worked nice but code ran too slow. And the EPROM's had to be erased by UV light which took a long time.
At those days, electronic circuits consisted of numerous CMOS and TTL chips. And i saw the 89C2051 as an ideal replacement for a lot of CMOS/TTL chips. It would make PCB design much simpler.
And 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 but in 1993 it was not so common. At least not to my knowledge.

I wrote a complete DOS tool and when i was satisfied Windows became reasonable stable and a standard(windows 3.1).
And thus i rewrote the tool. The tool was for my own usage. When i learned it would be usable to others as well 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 why 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. And when you needed a different LCD display, you could simply change the CONFIG line.
And when a different processor was needed, you only had to change the definition file !
Another reason for the success is that we hidden all complexity for the user. No ASM to deal with, simple statements, and of course free updates and support.
Small companies that used the BASIC Stamp also recognized another advantage : 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. So BASCOM-LT was rewritten for a big part and support for arrays and floating point (single) was added.
The many different 8051 variants make it impossible to support all chips but the DAT files were easy to add by the user.

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, and it has simple linear memory. The best part is that you can program the chip inside the circuit. No wonder this chip family became so popular.
And because the 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 that we even support hardware!

Since a long time we are working on a more professional version of the software.
Some times we put a feature of it to the current BASCOM version.
An ARM version is under development too.
Note that we do not give details or time frames for these versions, nor do we do for other features.
In order to migrate to a future version it is however important that you keep your software up to date. This will make a migration more simple.

Things we find important :

- the environment. we reuse all usable packing material like foam, plastic bubbles we receive when we ship your order.
- that everybody can use microprocessors. They are not scary, but are just chips 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 and they are from us only.
- free updates. (they are free since 1995 but it is not a guarantee it 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 use new 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 life, no matter which religion you have, no matter how old you are, if you are male, female, purple or white.
- that you can use the free demo for free. But that you pay for a full version if you use it commercial. Do not use cracked soft. Using cracks 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 EM4095 Reference Design 212 or the TCP TWI motherboard and adapter 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 'About MCS 24'

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.

Part II

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.

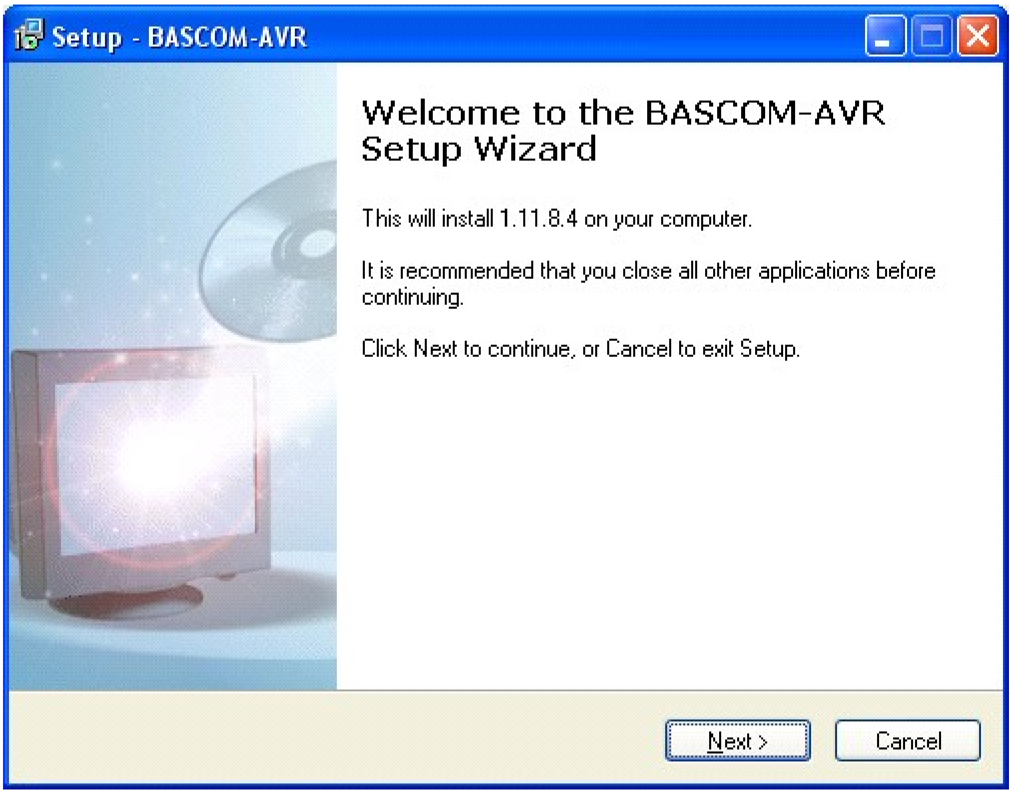
When you are installing on a NT machine like NT4 , W2000, XP or Vista, you need to have Administrator rights.
After installing BASCOM you must reboot the computer before you run BASCOM.

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.

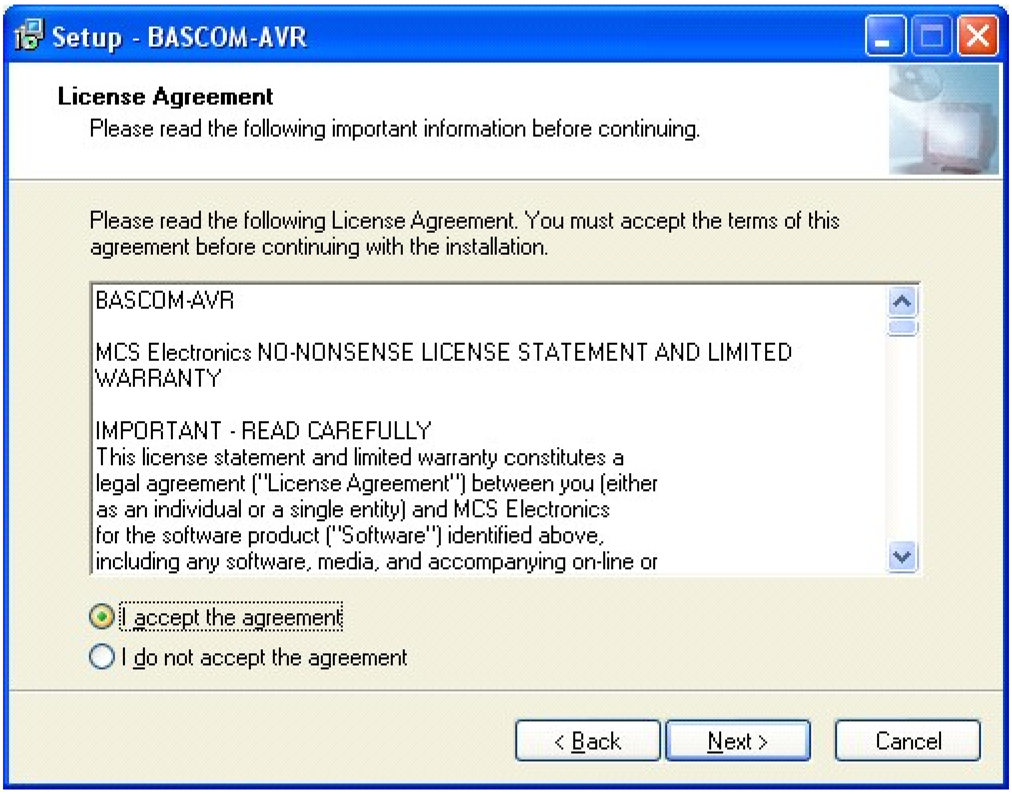
The following window will appear:

(screen shots may differ a bit)



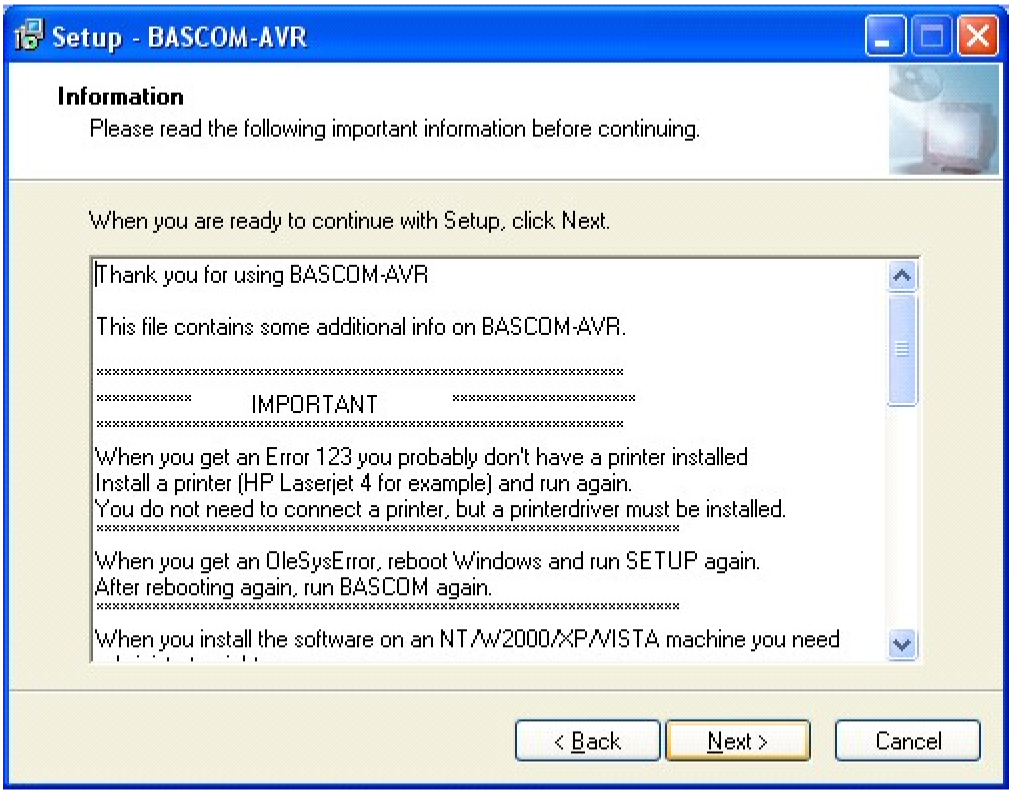
Click on the **Next button** to continue installation.

The following license info window will appear:



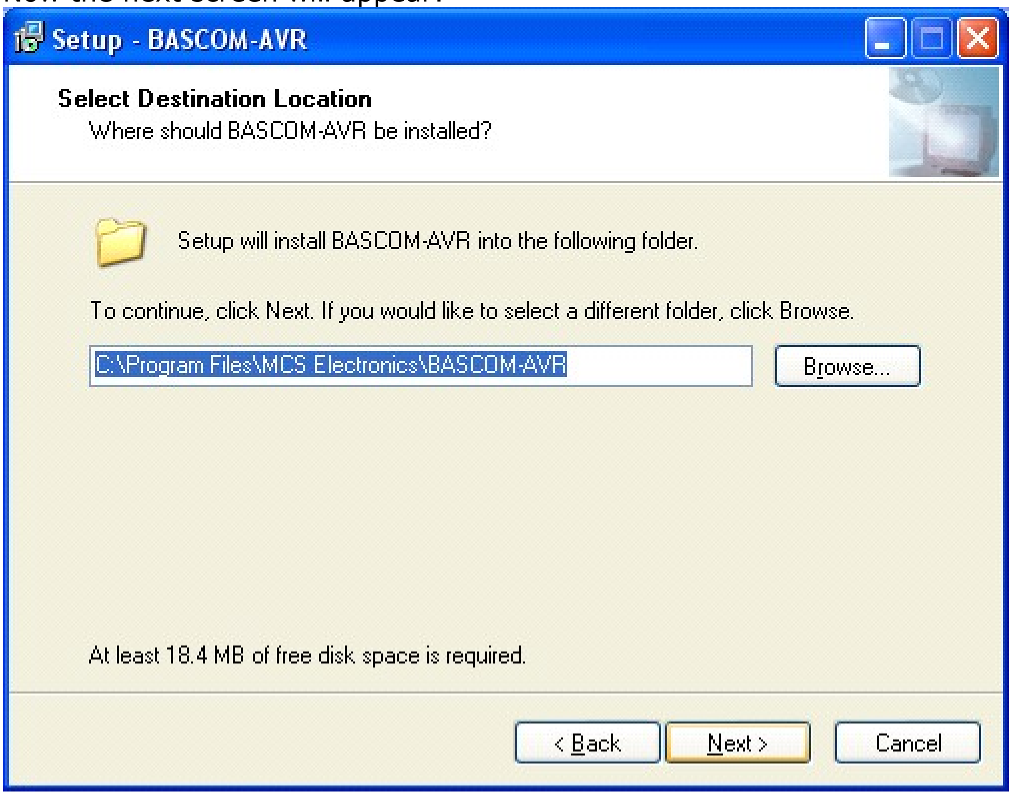
Read the instructions , select **'I accept the agreement'** and press the **Next button.**

The following window will be shown :



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

Now the next screen will appear:



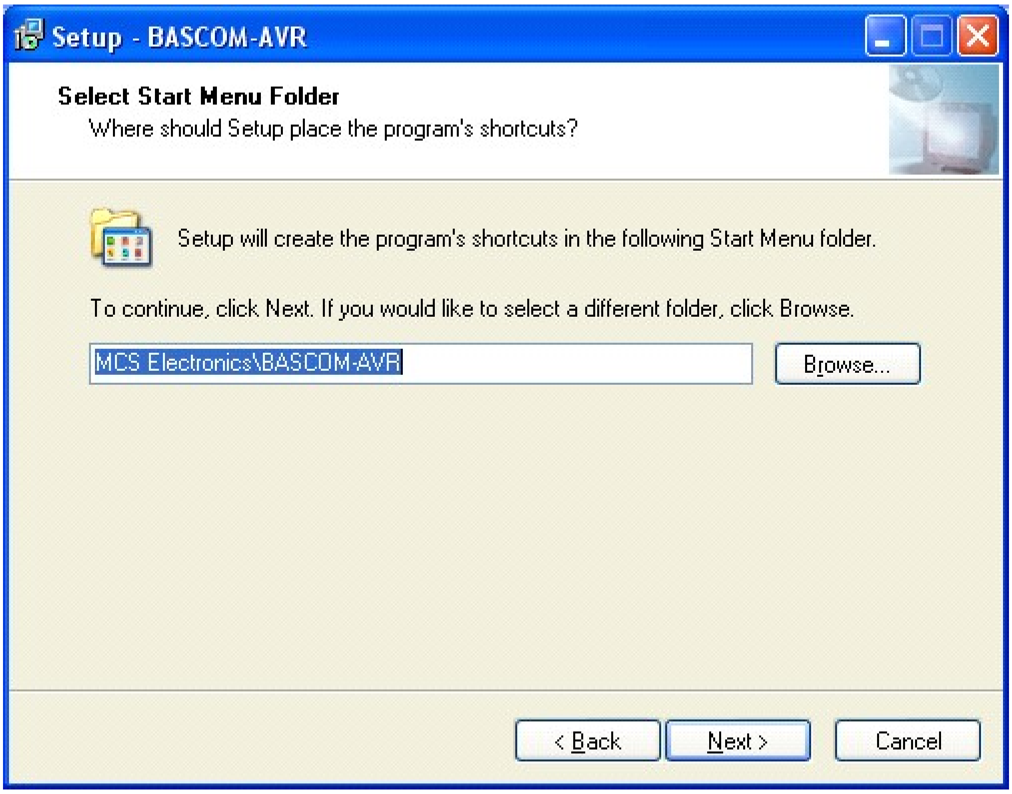
You can select the drive and path where you like BASCOM to be installed. You can also accept the default value which is :

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 :

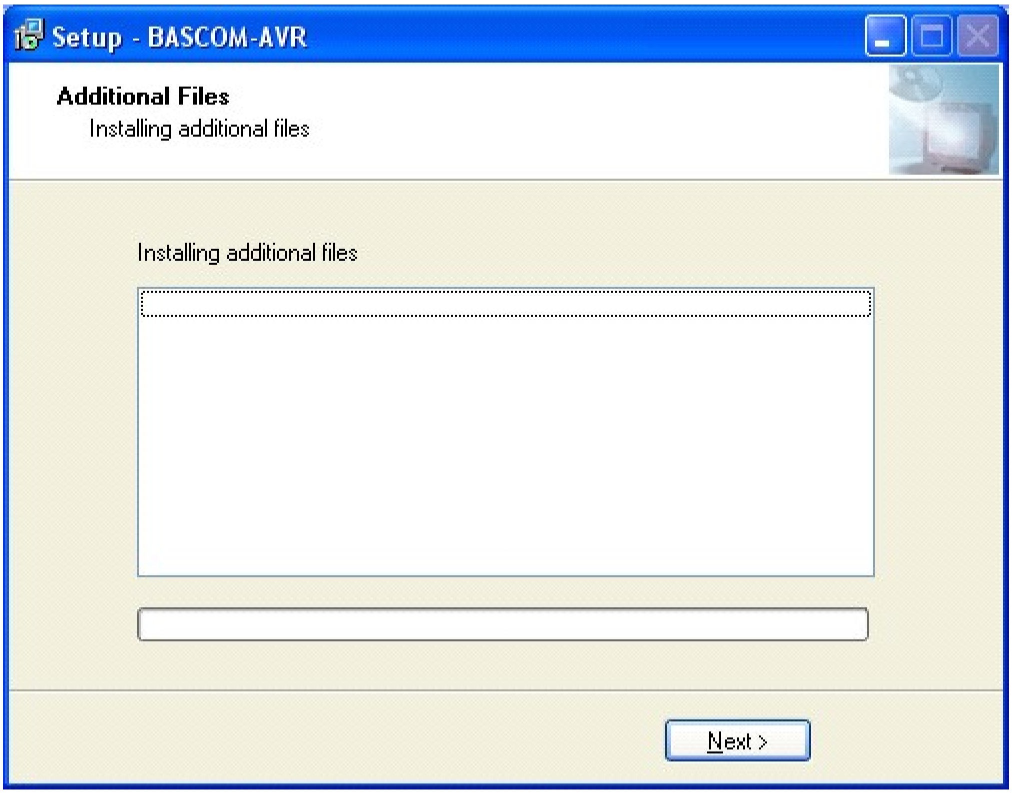
In case of this warning, select Yes.

You will now see the following window:



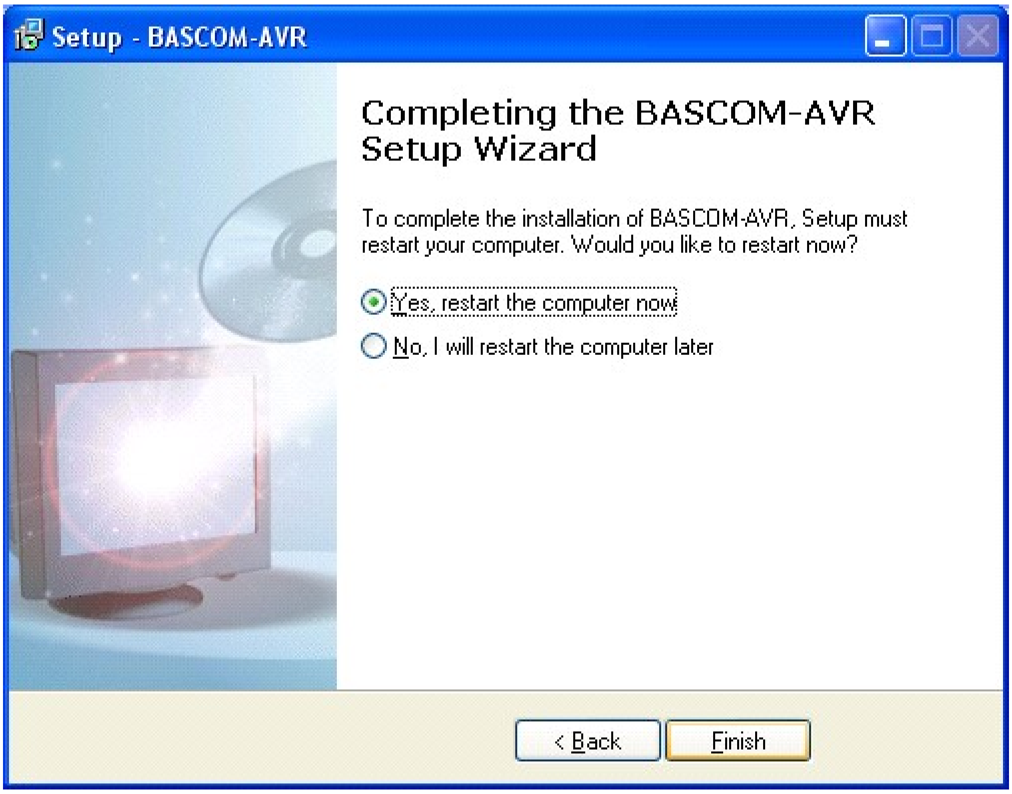
You can choose to create into a new Program Group named 'BASCOM-AVR' , or you can modify the name, or install into an existing Program Group. Press the **Next-button** after you have made your choice.

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



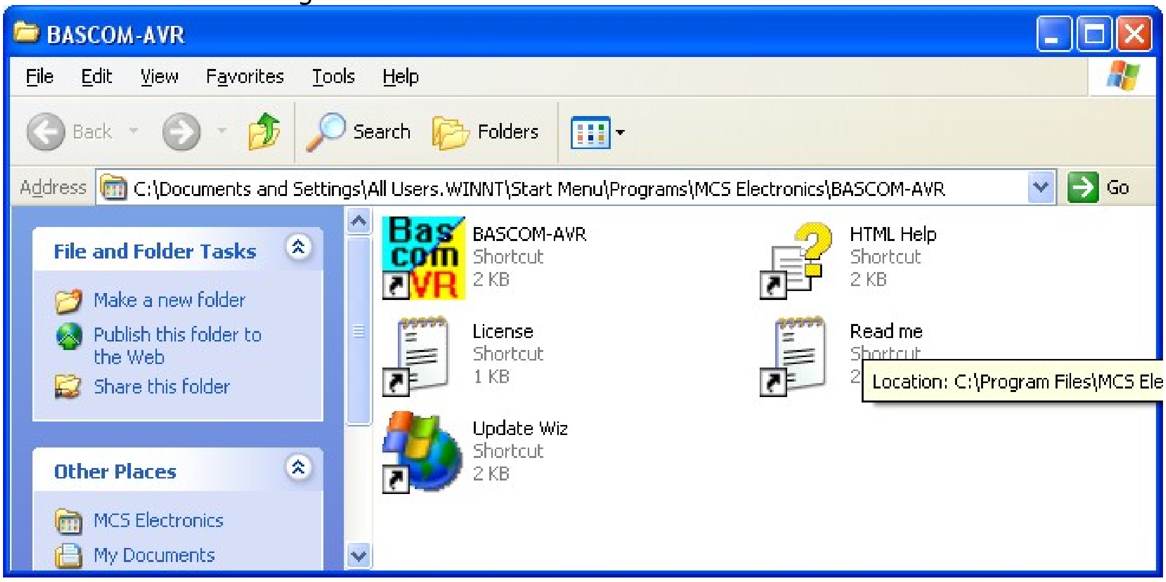
These additional files can be PDF files when the program is distributed on a CD-ROM.

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:



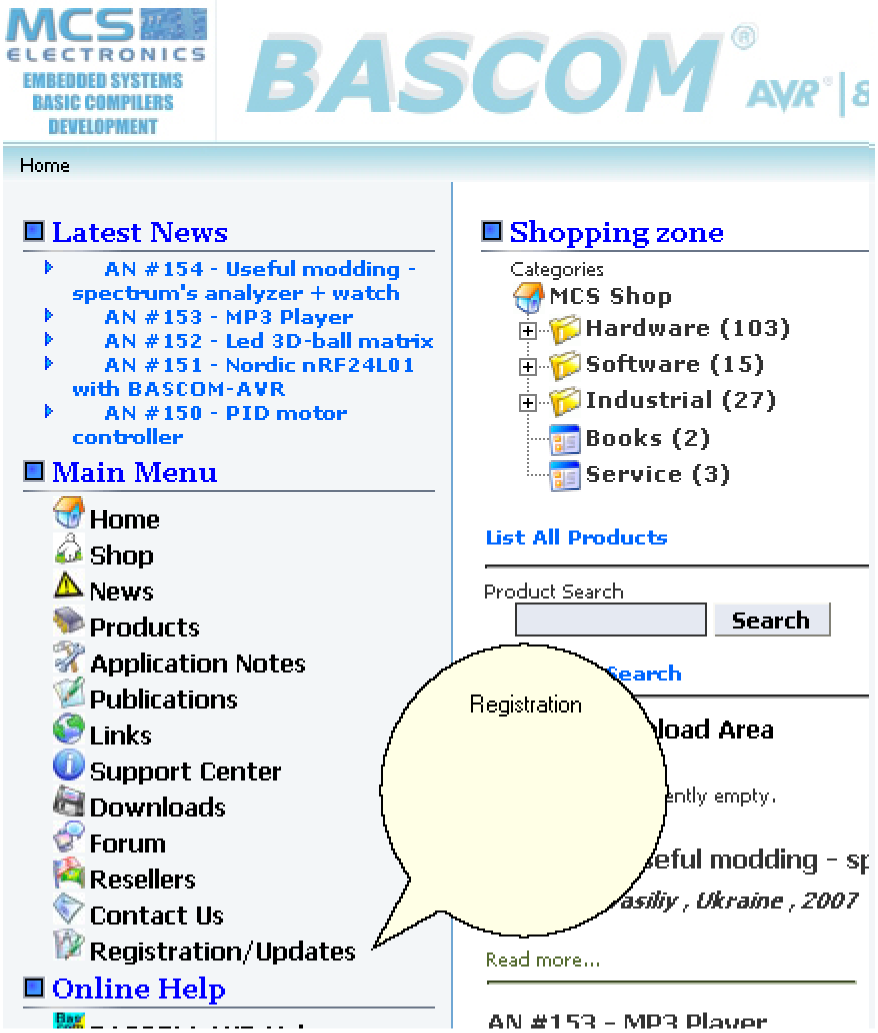
You can view the "Read me" and "License" files content and you can start BASCOM-AVR.
BASCOM supports both HTML Help and old Win help(HLP). The HLP file is not distributed in the setup. You need to use the Update Wiz to download it. But it is advised to use the HTML-Help file.
When you used to use the HLP file, and find it missing now, turn on 'Use HTML Help' in Options, Environment, IDE. 93

When the UpdateWiz is not installed, you can download it from the register 34.

2.2 Updates

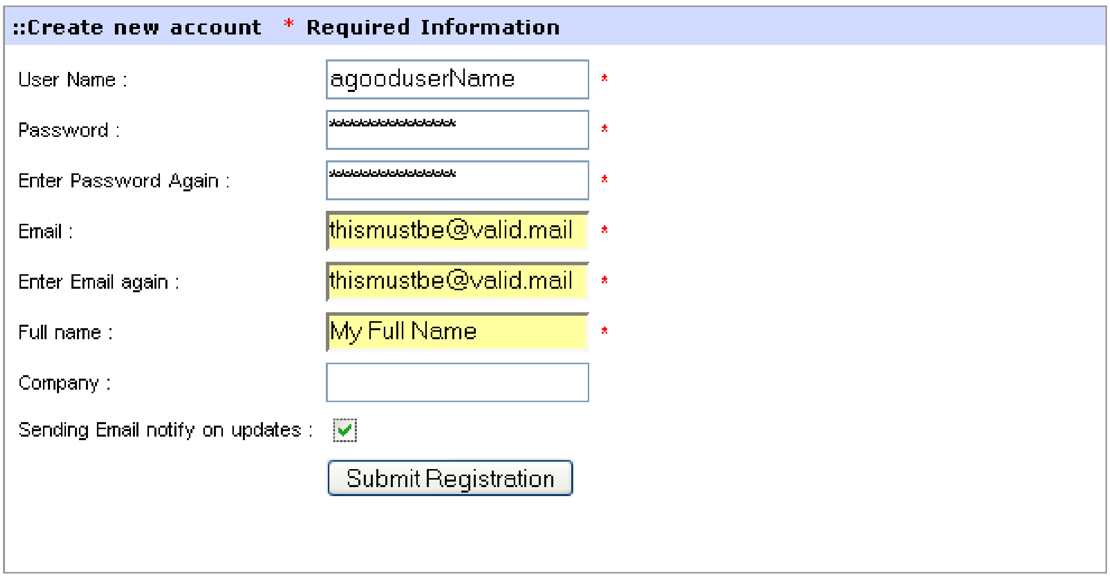
The update process is simple.

- Go to the main MCS website at http://www.mcselec.com
- 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:

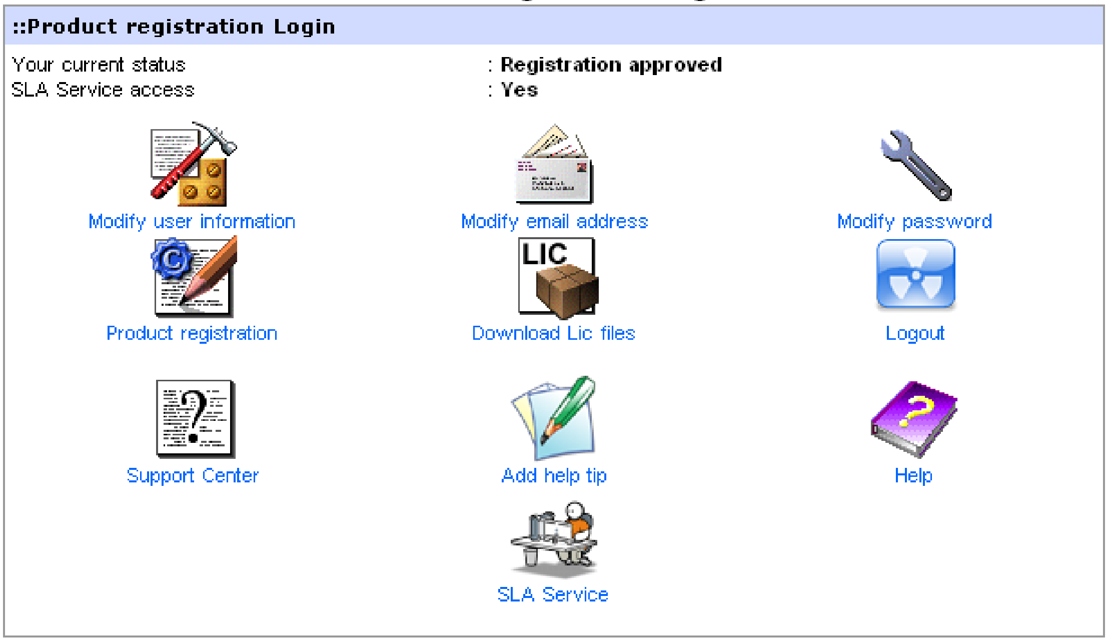
- Click the link and select 'Create new account'



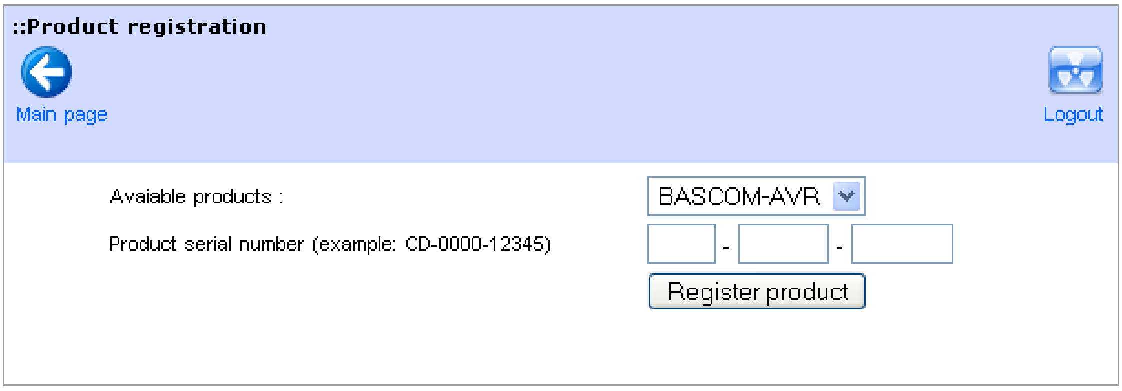
You need to provide a username, password, email and full name. Company name is optional. When you want to receive notifications when updates are available, select this option.
When you filled in the information, click 'Submit Registration'.

- 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 :

Product registration Login

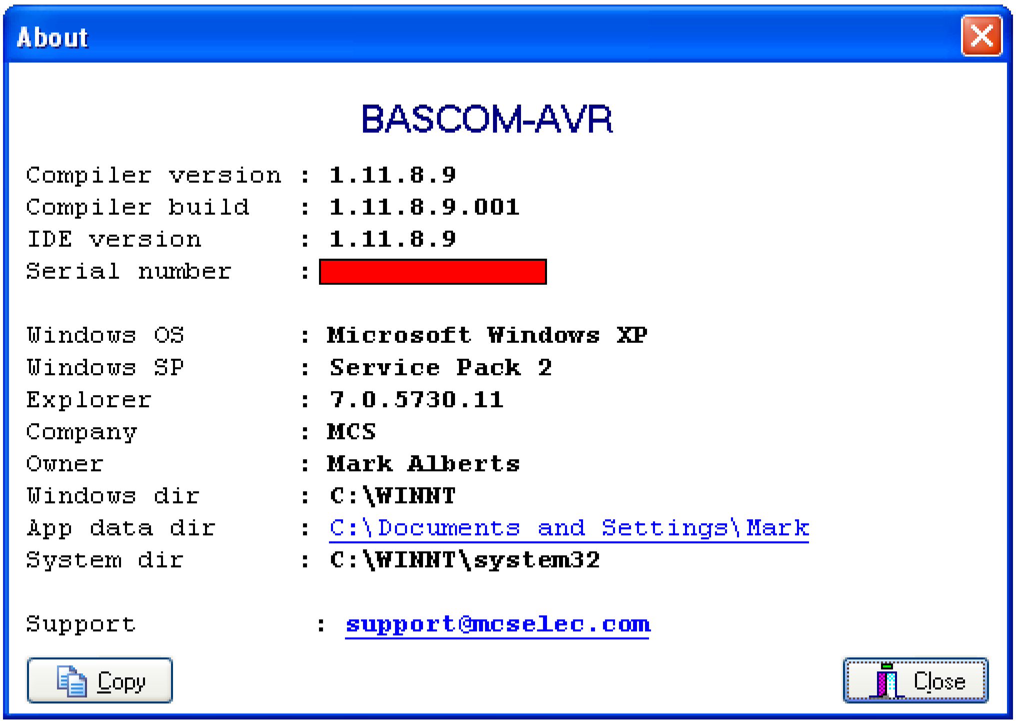


- You need to chose 'Product registration'.
- The following screen will be shown:

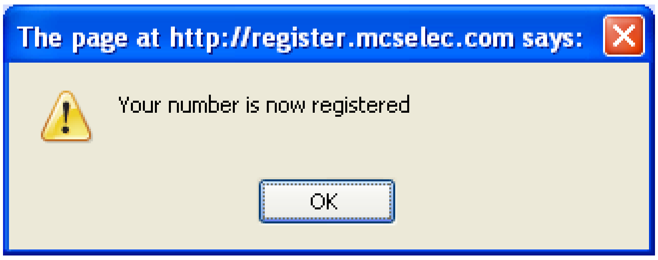


- Select a product from the list.
- Enter the serial number

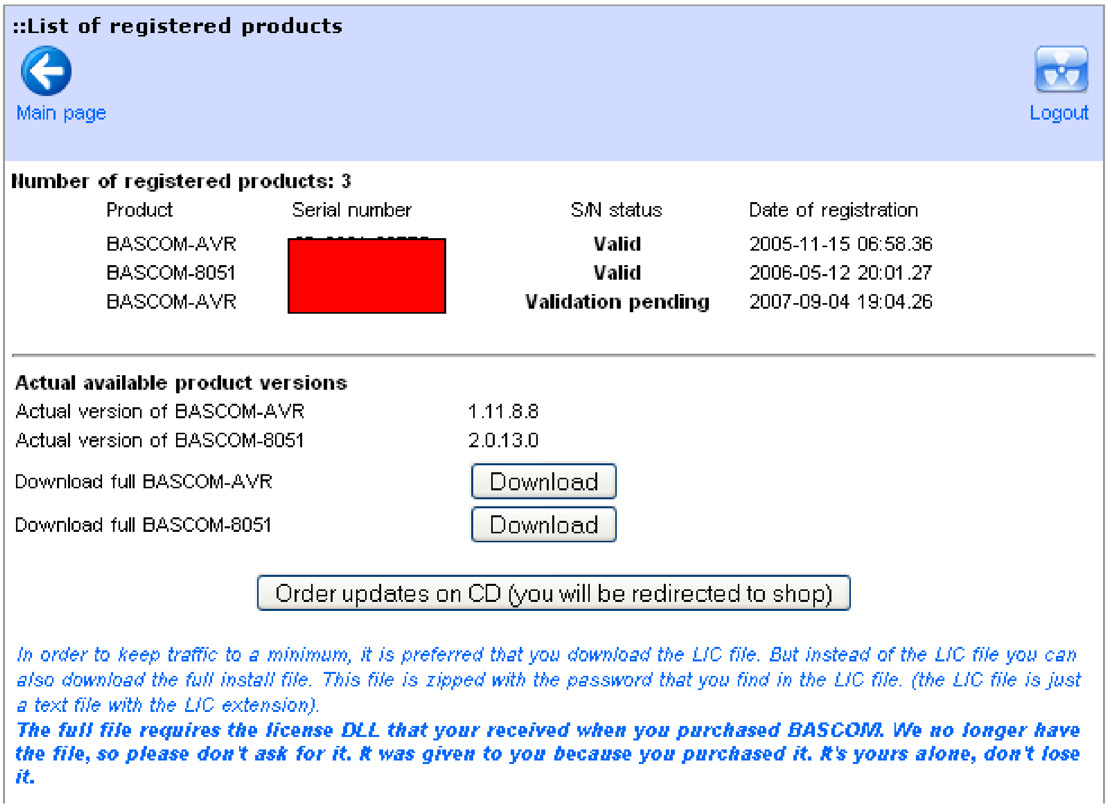
It is important that you enter a **valid** serial number. Do not try to enter serial numbers from cracked versions. When you enter invalid serial numbers, you will loose support and the ability to update.
The valid serial number is shown in the Help, About box.



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



- 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:



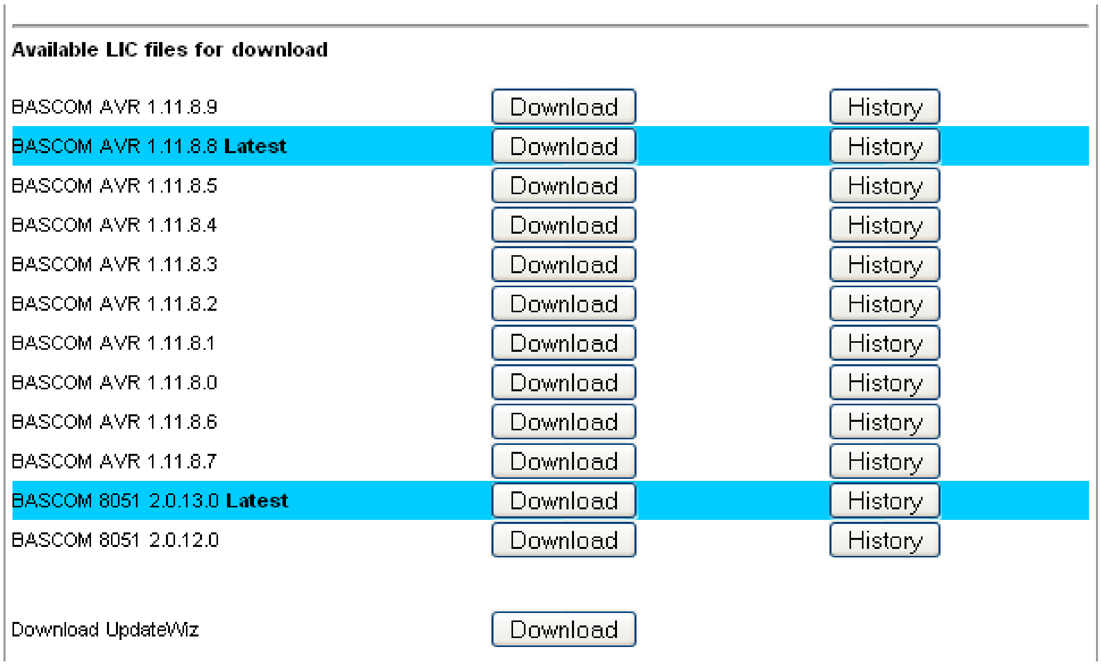
At the top you can see which products are registered, and which status they have.
When you want to do a FULL SETUP, you need to download the full version.
You **do not** need to **uninstall a previous version**. You can install an update into the same directory or a new directory.
You can also order 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 28 on how to do a full install.

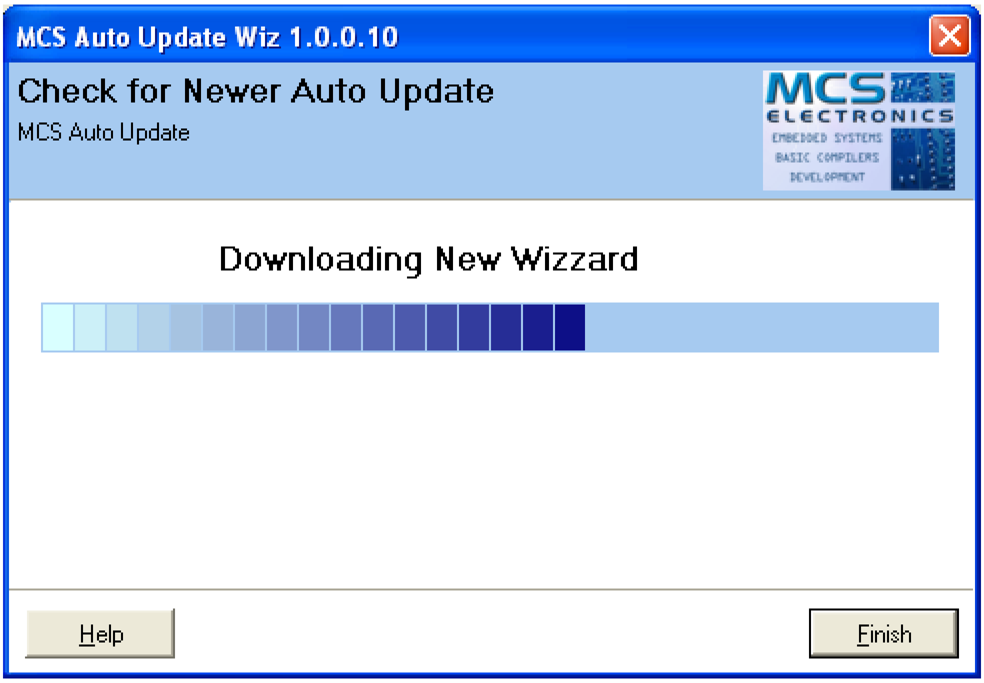
It is also possible to do a partial update. For example to update some DAT files, or to update to a beta which is only available as an update.
For partial updates, you need the Update Wiz.



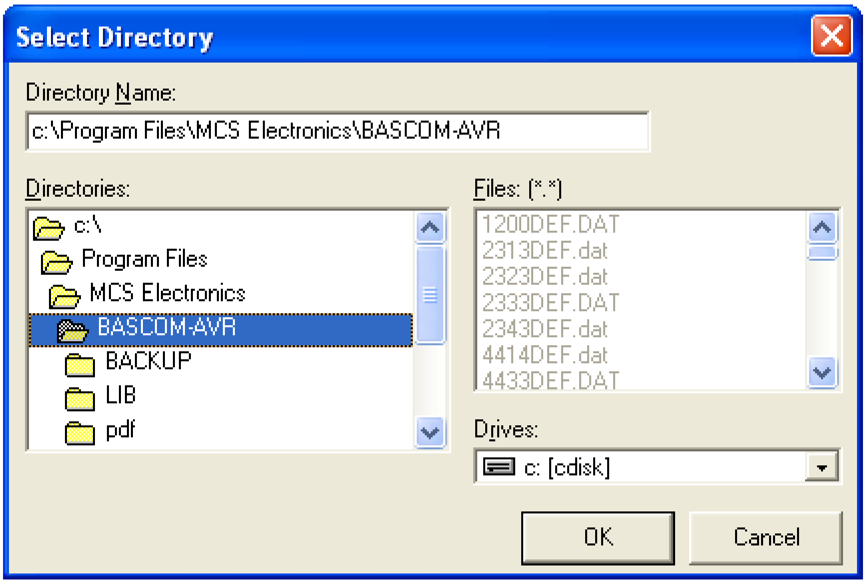
When you do not have the Update Wiz, you can download it.
Unzip it to the same directory as BASCOM.

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.

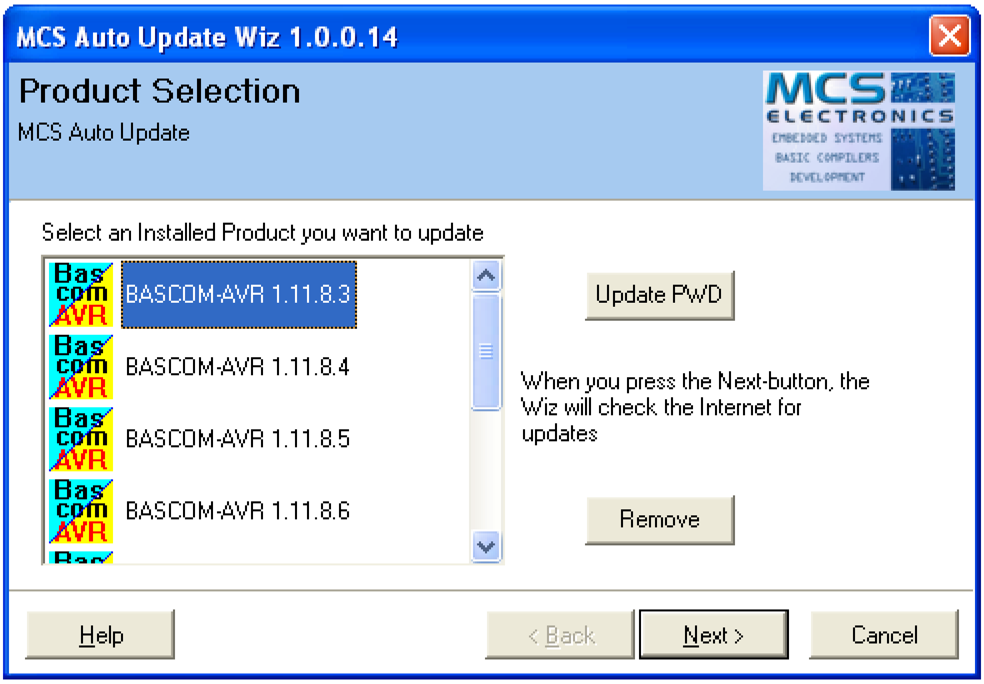


When the Update Wiz finds a LIC file, it will check if the update/install location is specified. For new downloaded LIC files, the update wiz does not know the update directory, and will ask for the directory you want to update. This can be any (new) directory, but usually is the BASCOM application directory.



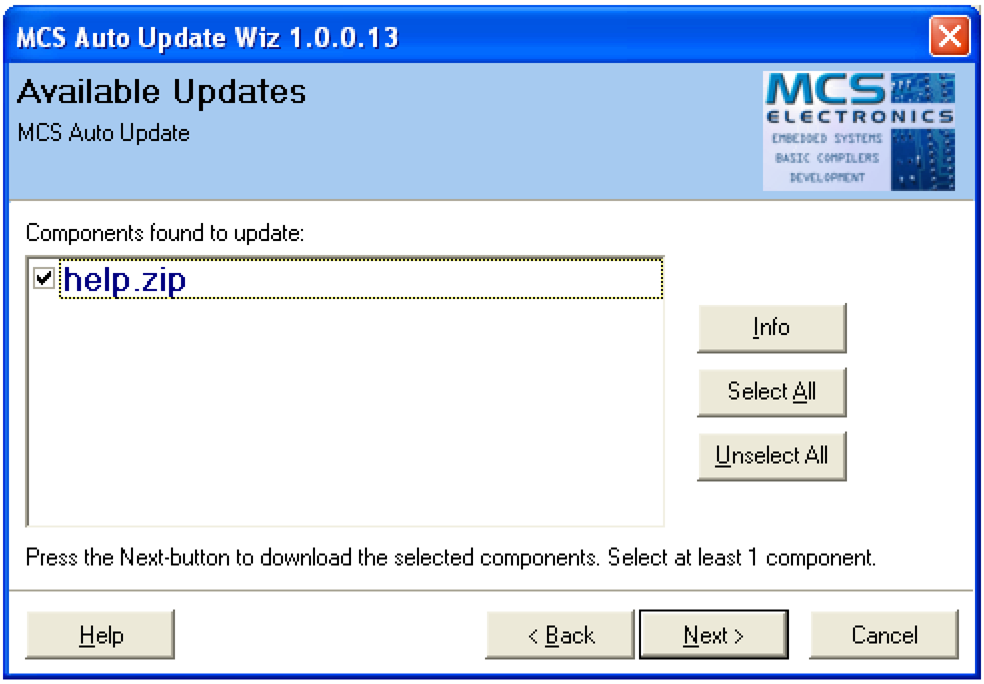
After you click Ok, the directory to update is stored in the LIC file.
It will not be asked again.

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 :

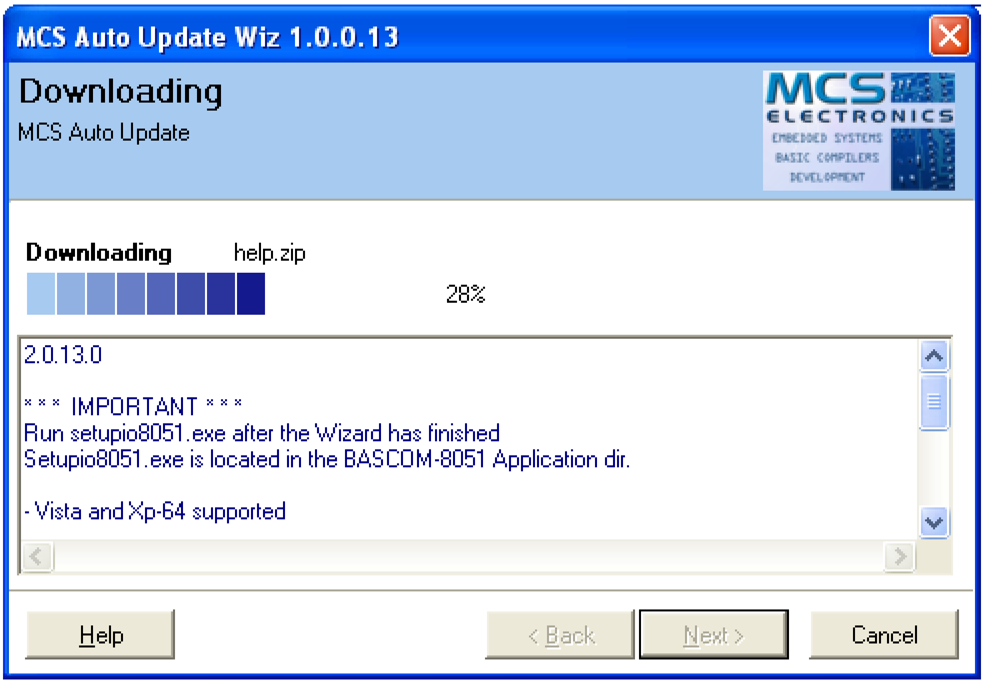


You need to select the product that you want to update. In the sample there are multiple choices.
Press the Next-button to continue.

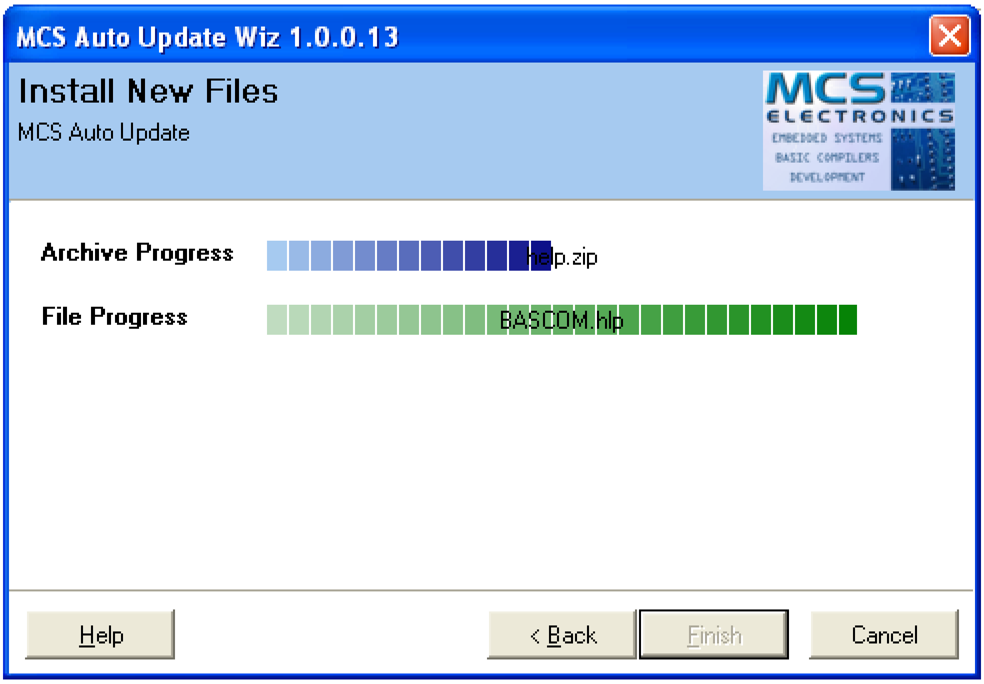
The Wiz will compare files on the web with your local files in the specified directory.



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.



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.



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 http://register.mcselec.com
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.

Part III

3 BASCOM IDE

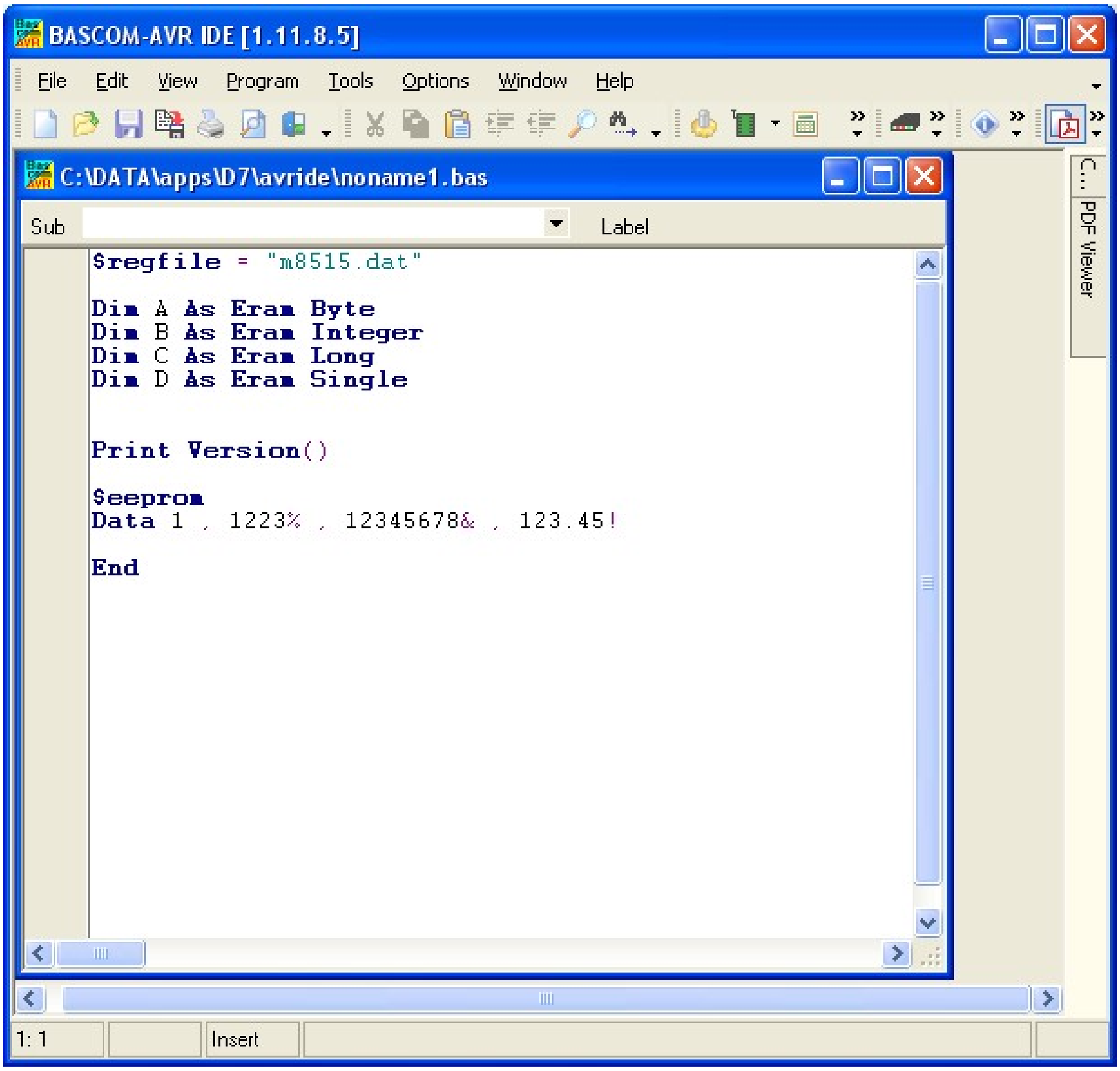
3.1 Running BASCOM-AVR

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

Double-click the BASCOM-AVR icon 

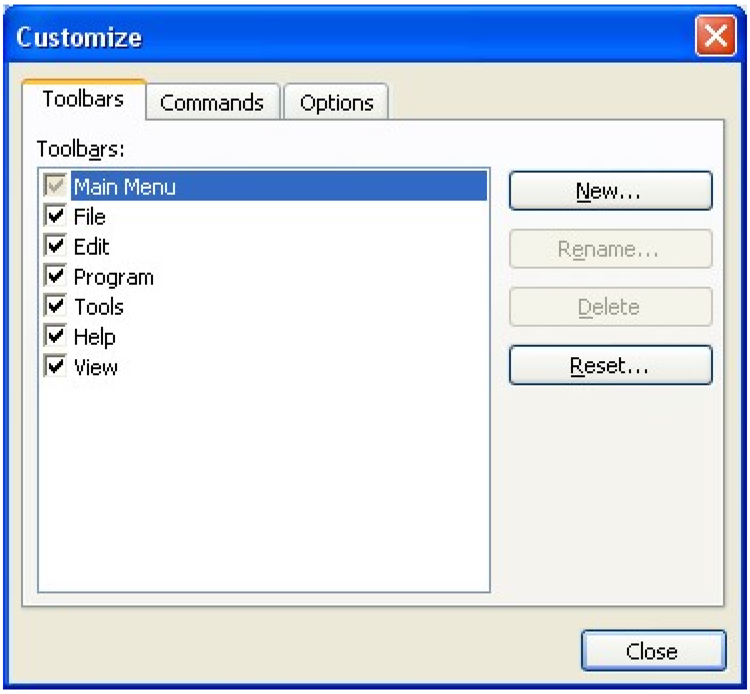
to run BASCOM.

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'.

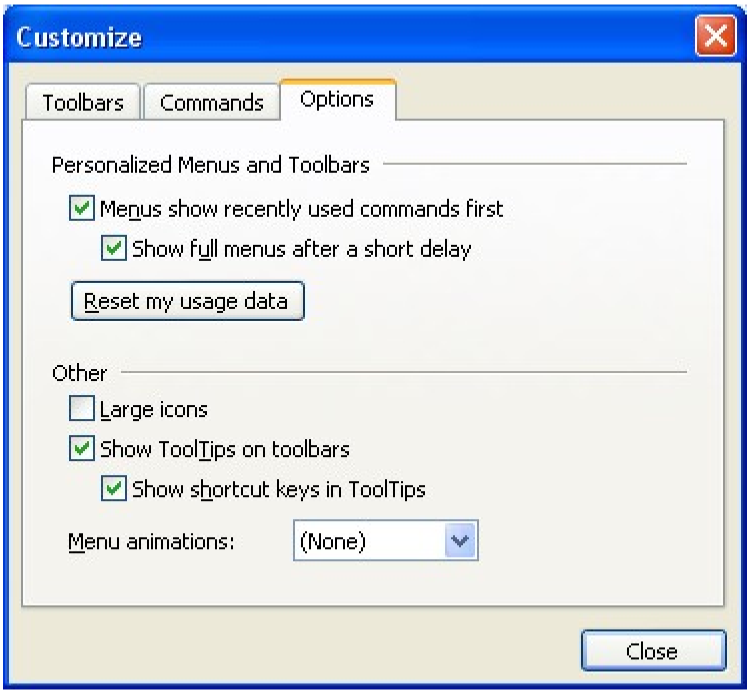
This will show the following window:



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.

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:



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 Simulate 61 or Program Chip 72, 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 + N

3.3 File Open

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 93 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 File New 47 option, you will be asked to name the file first. Use the File Save As 48 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 : 

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 :

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

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.

Edit Find shortcut : , CTRL+F

3.16 Edit Find Next

With this option, you can search again for the last specified search item.
Edit Find Next shortcut : , 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 : ,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.

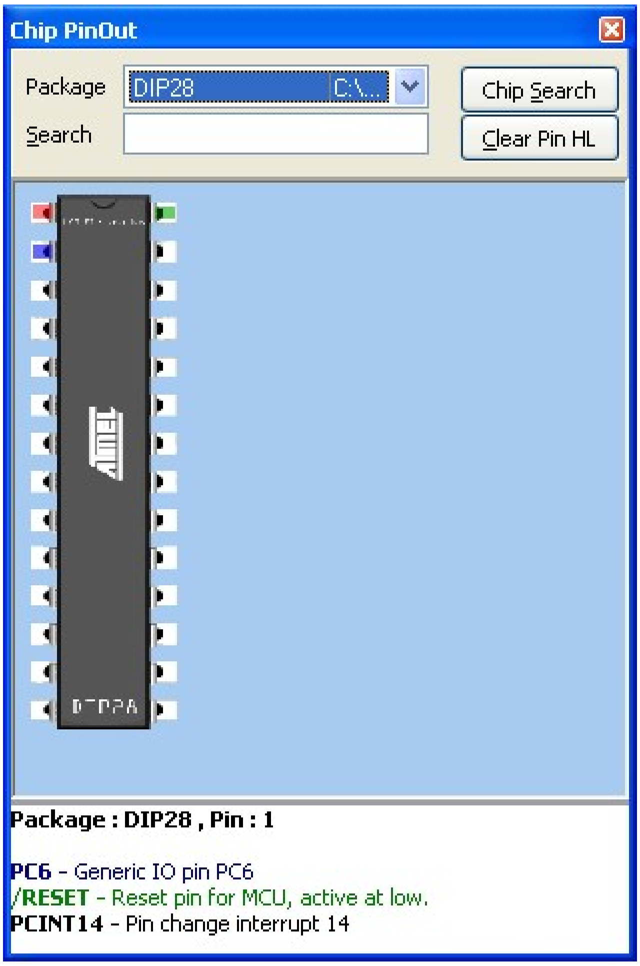
Edit Unindent Block shortcut : , CTRL+SHIFT+U

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 View PinOut

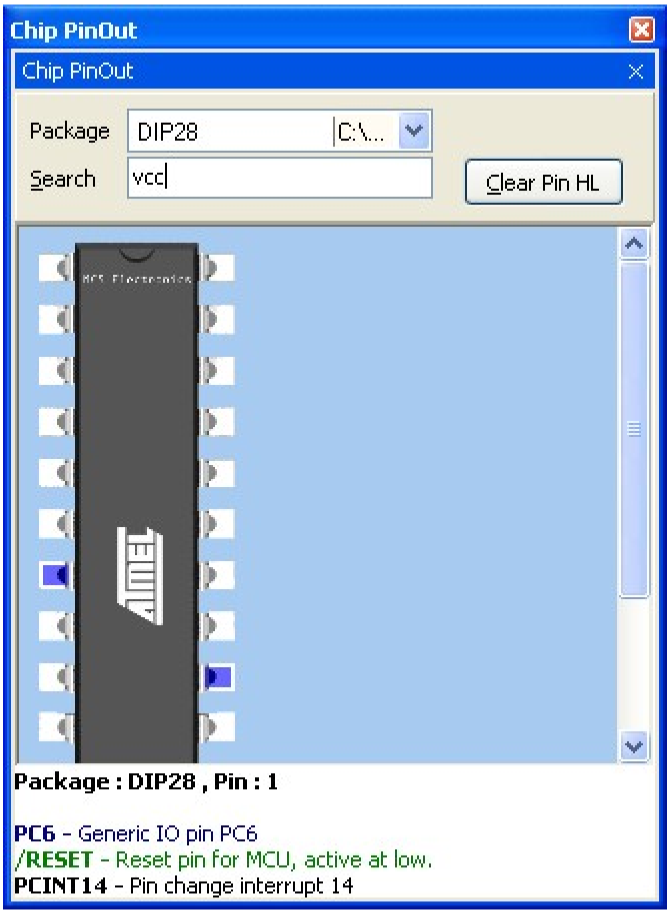
The Pin Out viewer is a dock able window that shows the case of the active chip.
The active chip is determined by the value of $REGFILE 296.



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.

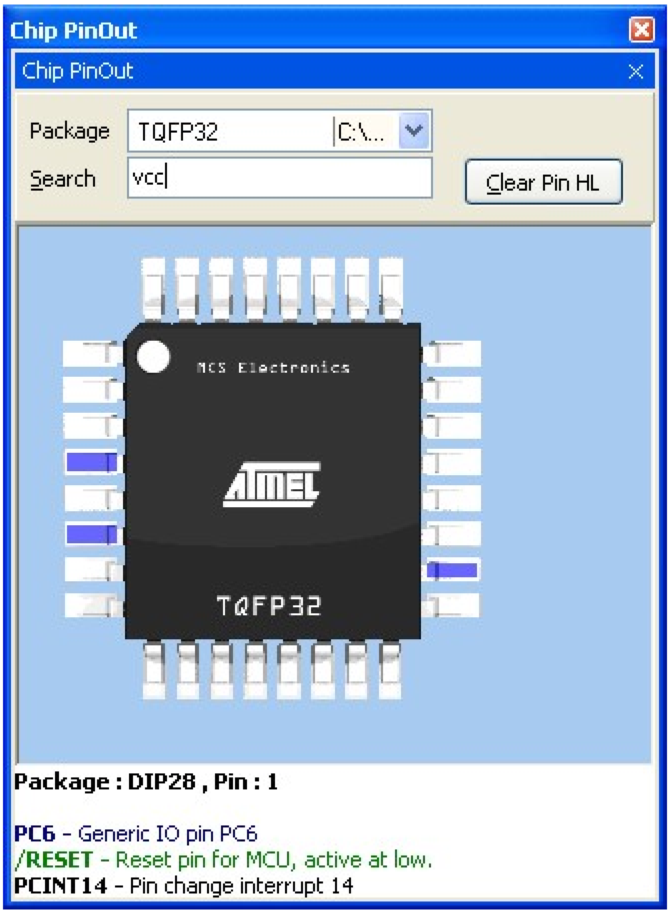
A pin can have one or more functions. Some functions can be used together.
When you move the mouse cursor away, the pin will be colored blue to indicate that you viewed this pin. For example, when you need to look at it again.

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.

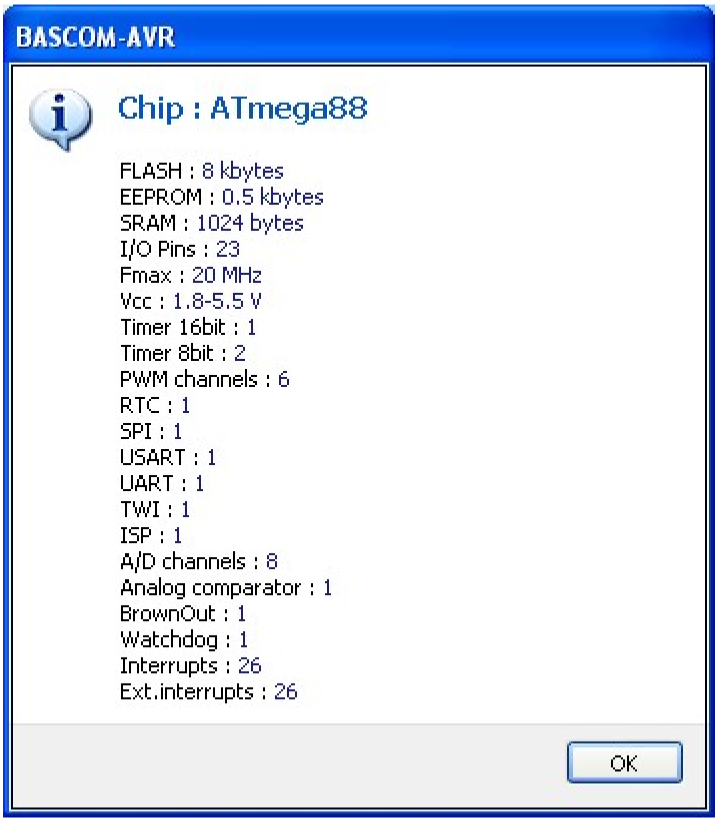
Some chips might have multiple cases. You can select the case from the package list.



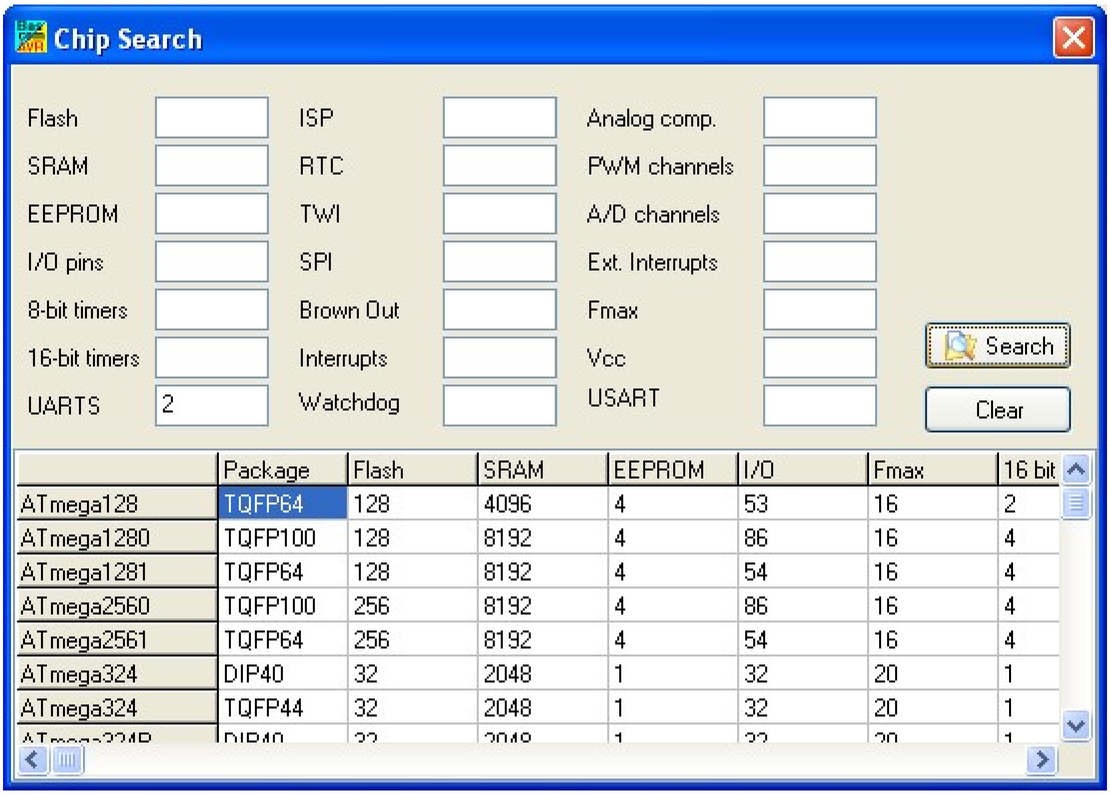
When you change from package, all pin colors will be cleared.
When you double click a pin, the pin will be colored green. Another double click will color it red/blue.
When a pin is green, it will not be colored red/blue. The green color serves as a kind of bookmark.
The only exception is the search function. 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.



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



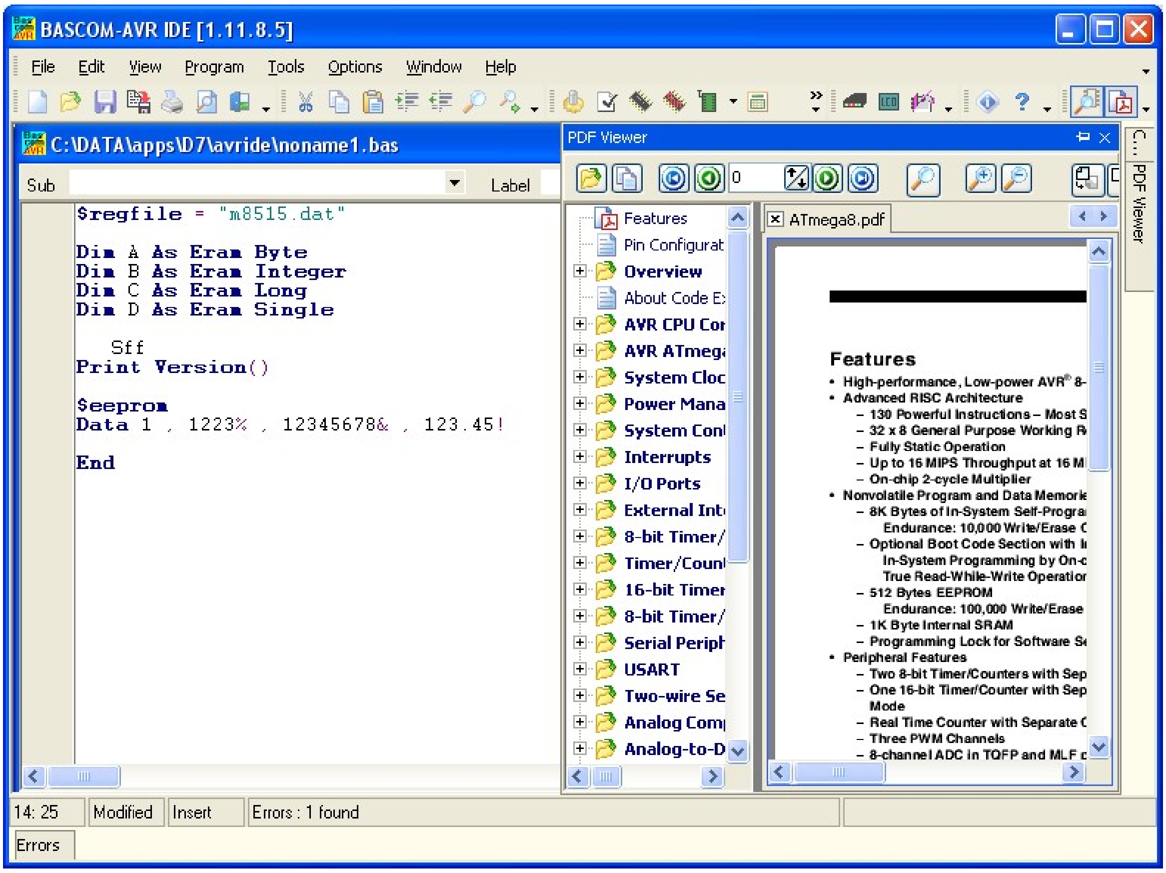
You can provide criteria such as 2 UARTS. All criteria are OR-ed together. This means that when one of the criteria is met, the chip will be included in the list.

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.

3.25 View PDF viewer

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



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**.

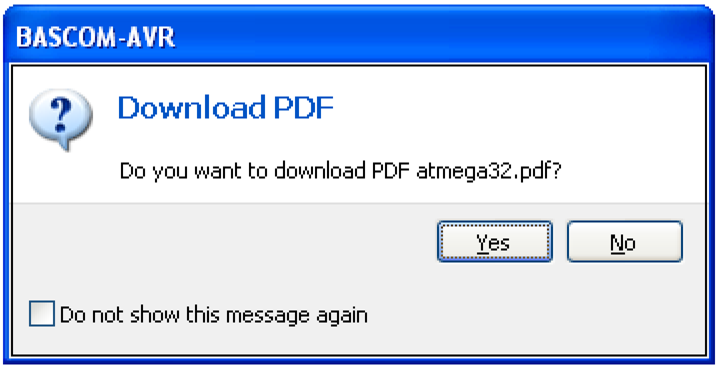
When you have enabled 'Auto open Processor PDF' in Options, Environment, PDF, the data sheet will be automatically loaded when you change the $REGFILE value.
It can be shown in a new sheet or it can replace the current PDF.

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

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

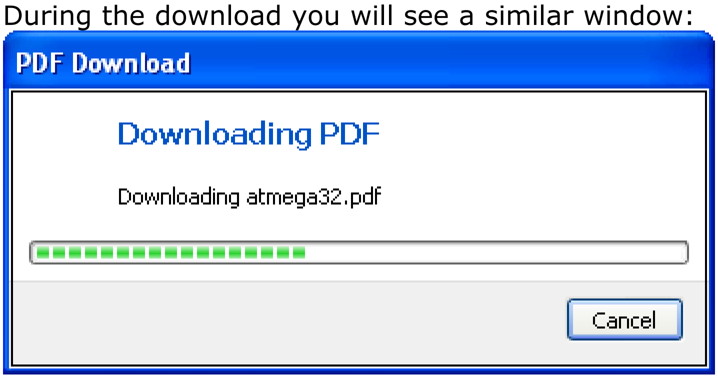
When you right click in the PDF, a pop up menu with the most common options will appear.
In Options, Environment, PDF 93 you can specify how data sheets must be downloaded.

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.



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:

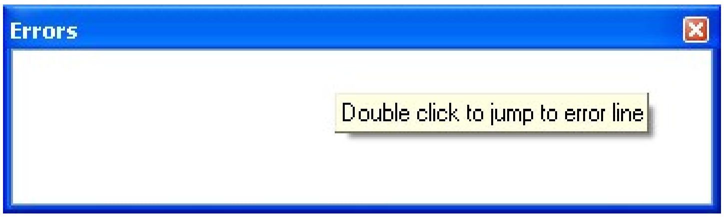


You can also download all newer PDF's from the Atmel website with the option :
Tools, PDF Update 84

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

3.26 View Error Panel

This option will show the Error panel.



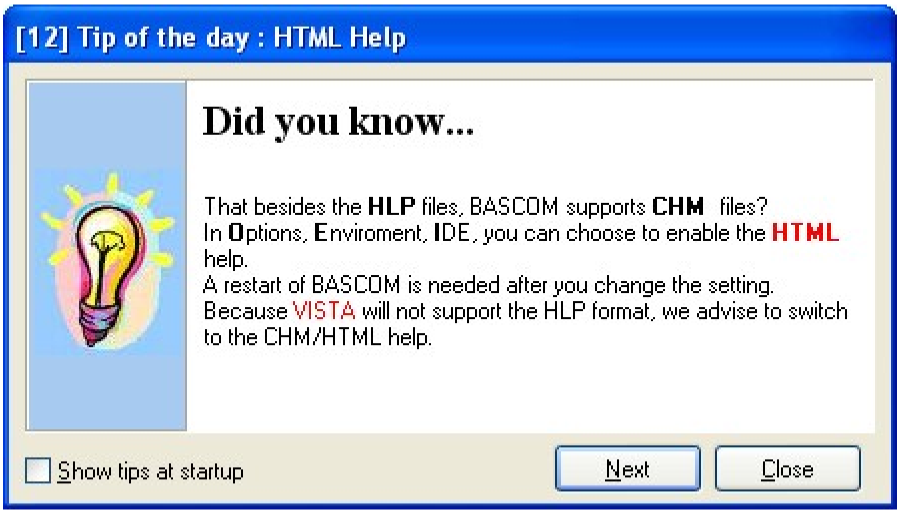
When there are no errors, the list will be empty. You will also be able to close the window.
When there are errors :

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.

3.27 View Tip

Action

Shows the Tip of the day Window



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 : http://register.mcselec.com

3.28 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. 86

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 sign.

At the next compilation, the error window will disappear or reappear if there are still errors.
See also 'Syntax Check' 58 for further explanation of the Error window.

Program compile shortcut: 

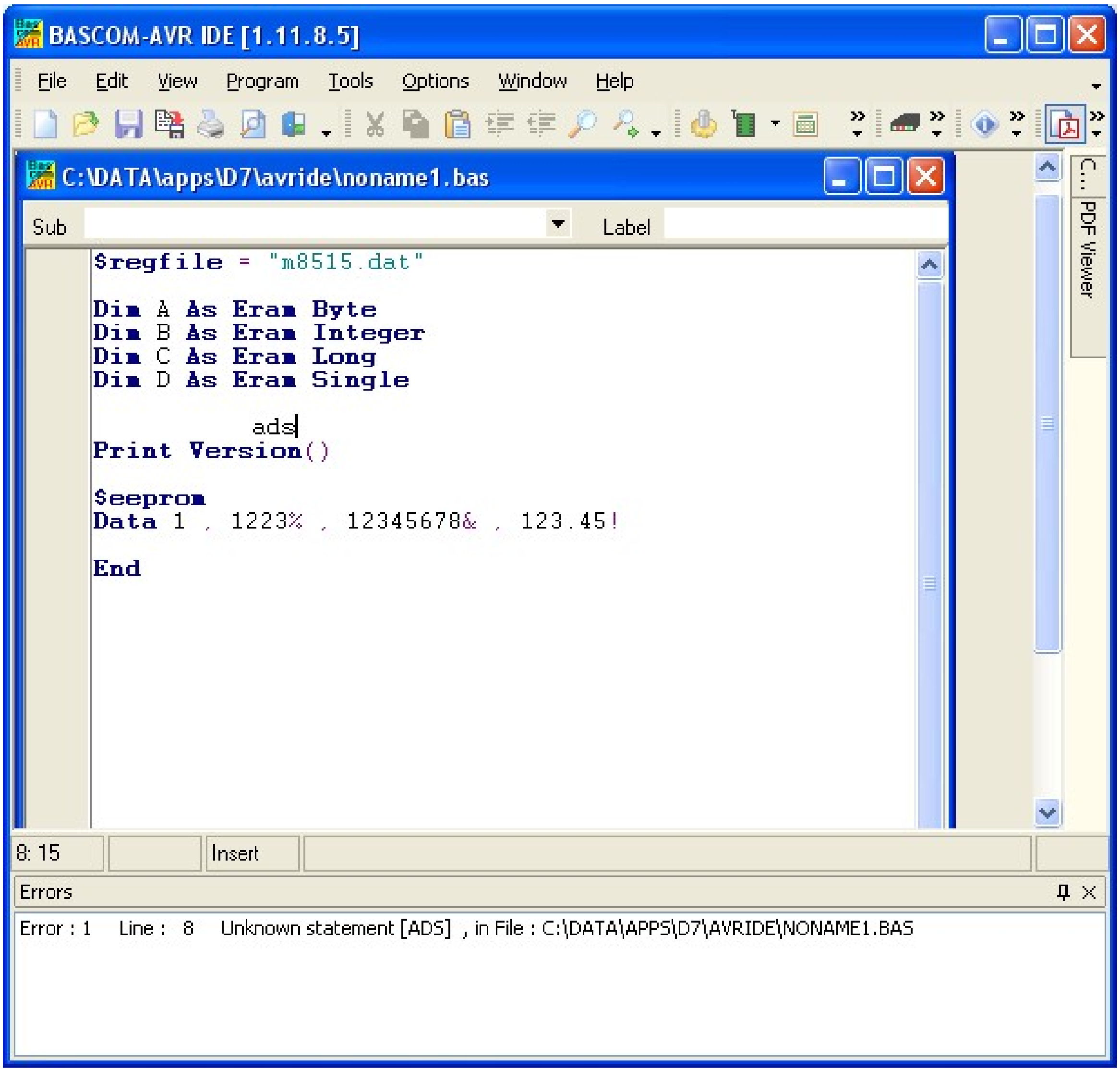
, F7

3.29 Program Syntax Check

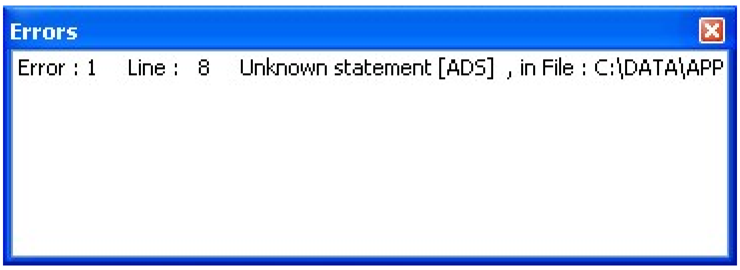
With this option, your program is checked for syntax errors. No file will be created except for an error file, if an error is found.

Program syntax check shortcut , CTRL + F7

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



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 :



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.

3.30 Program Show Result

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

See the Options Compiler Output 88 for specifying which files will be created.

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

File show result shortcut : ,CTRL+W

Information provided in the report:

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

	_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 501 lines are used to create the EEP file. (EEPROM binary image).

3.31 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.

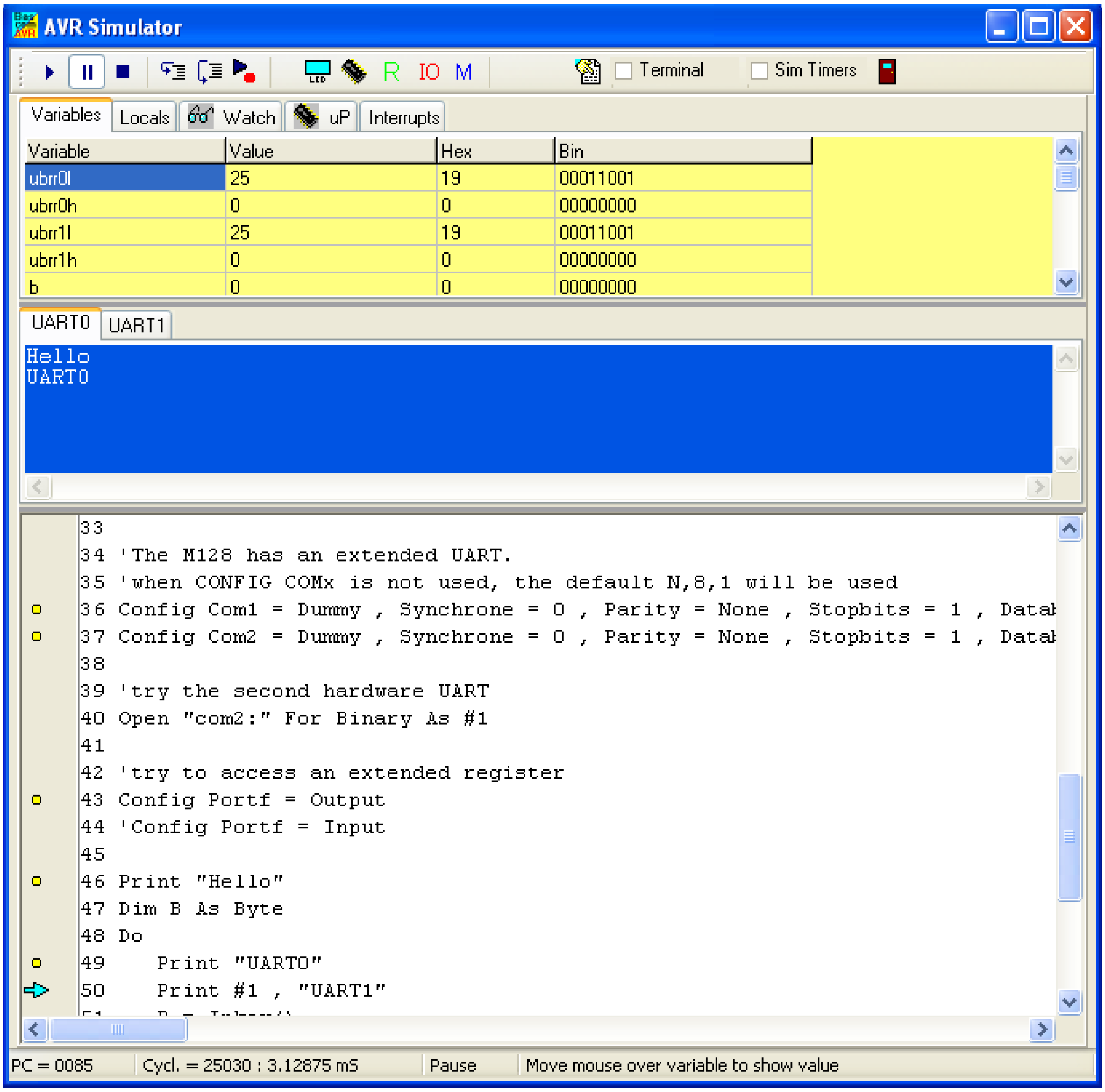
The simulator that will be used when you press F2, depends on the selection you made in the Options Simulator TAB. The default is the built in Simulator.

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.

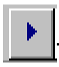
The DBG file contains info about variables and many other info needed to simulate a program.

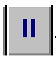


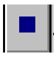
The Simulator window is divided into a few sections:

The Toolbar

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

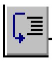
This is the RUN button, it starts a simulation. You can also press F5. The simulation will pause when you press the pause button. It is advised, that you step through your code at the first debug session. When you press F8, you step through the code line by line which is a clearer way to see what is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

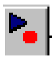
This is the PAUSE button. Pressing this button will pause the simulation.

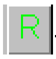
This is the STOP button. Pressing this button will stop the simulation. You can't continue from this point, because all of the variables are reset. You need to press the RUN button when you want to simulate your program again.

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

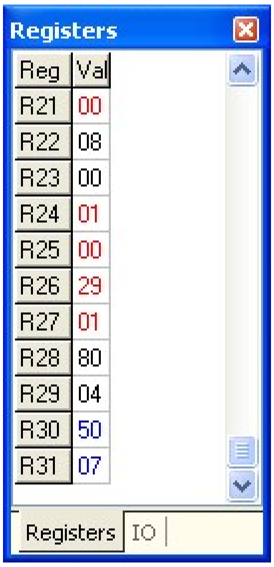
executed the simulator will be in the PAUSE state. If you press F8 again, and it takes a long time too simulate the code, press F8 again, and the simulator will go to the pause state.

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.

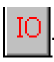
This is the RUN TO button. The simulator will RUN until it gets to the current line. The line must contain executable code. Move the cursor to the desired line before pressing the button.



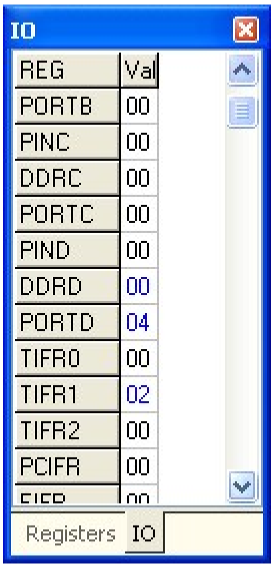
This button will show the processor registers window.



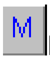
The values are shown in hexadecimal format. To change a value, click the cell in the VAL column, and type the new value. When you right click the mouse, you can choose between the Decimal, Hexadecimal and Binary formats.
The register window will show the values by default in **black**. When a register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



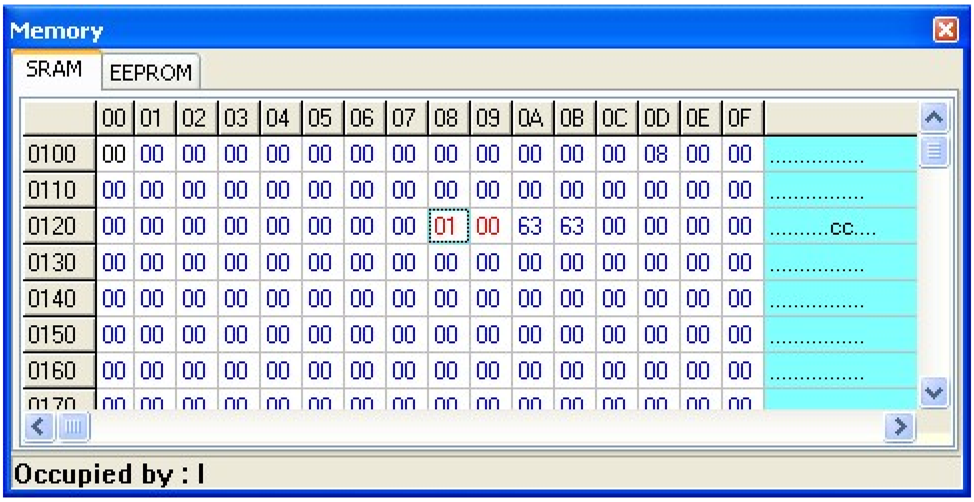
This is the IO button and will show processor Input and Output registers.



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.



Pressing this button shows the Memory window.



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

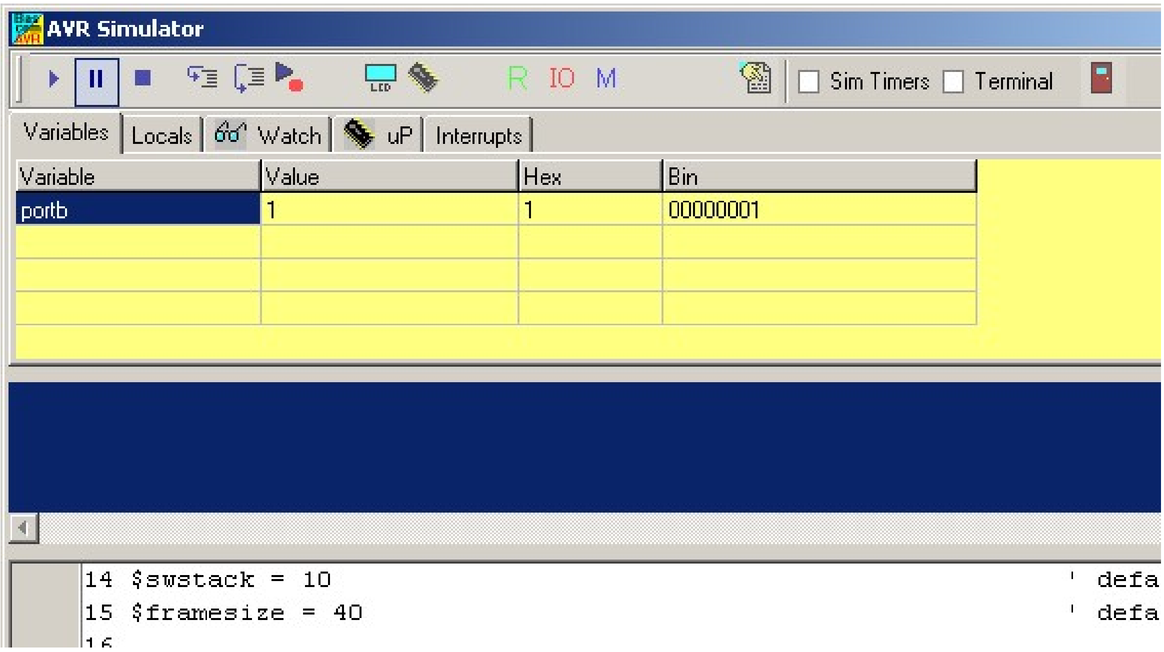
enabled this option. Note that using this option will slow down simulation. That is why it is an option. When you use F8 to step through your code you do not need to turn this option on as the variables are refreshed after each step.

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.

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:

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.

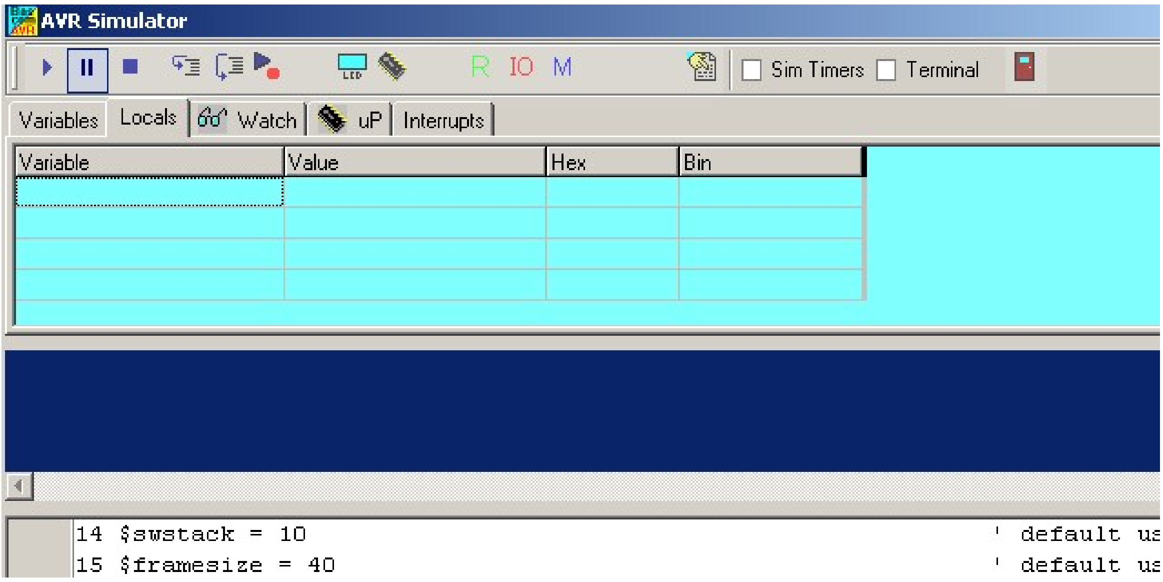
During simulation you can change the values of the variables in the Value-column, Hex-column or Bin-column. You must press ENTER to store the changes.

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

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

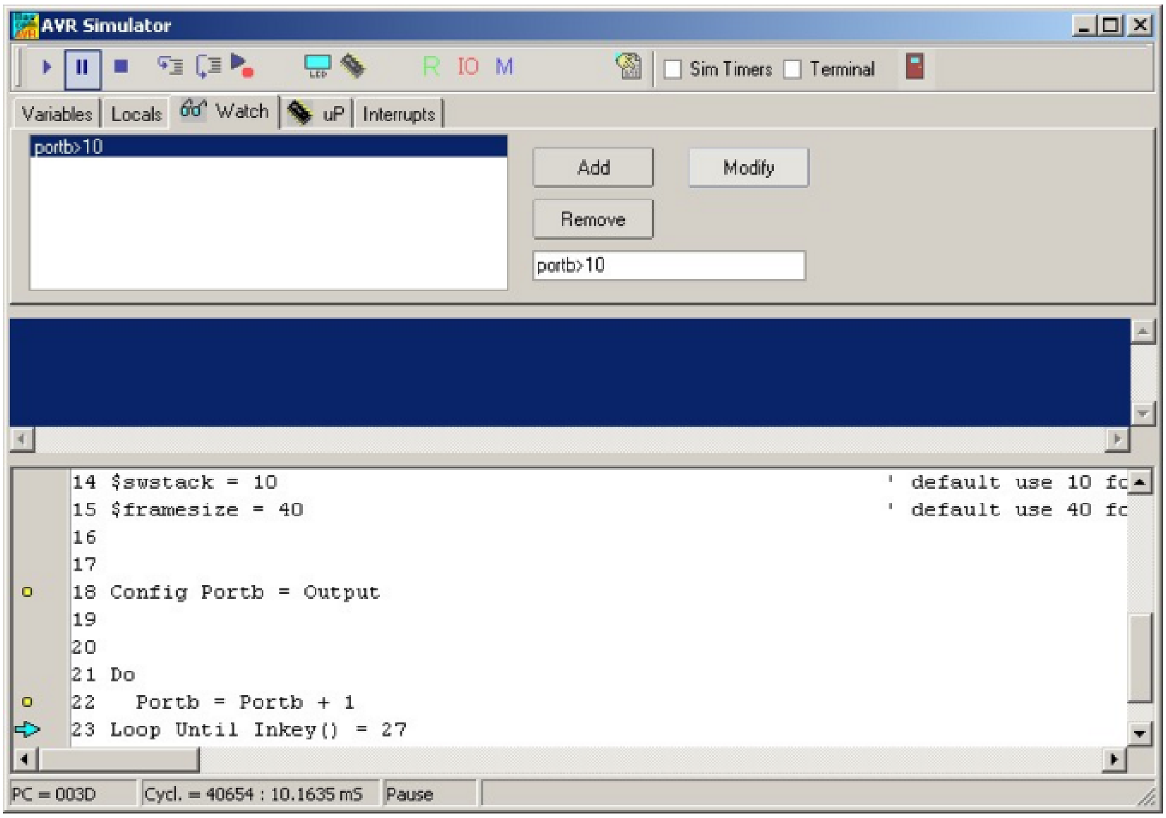
It is also possible to watch a variable by selecting it in the code window, and then pressing enter. It will be added to the variable list automatically.
Notice that it takes time to refresh the variables. So remove variables that do not need to be watched anymore for faster simulation speed.

LOCALS



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

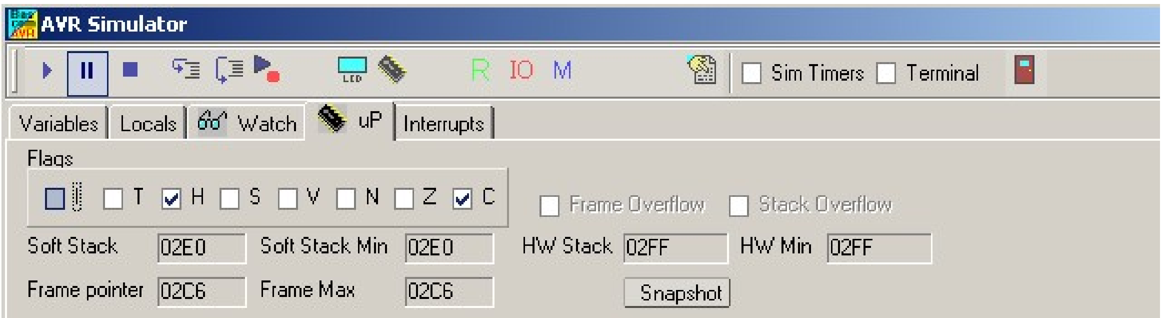


The Watch-TAB can be used to enter an expression that will be evaluated during simulation. When the expression is true the simulation is paused.

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



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

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

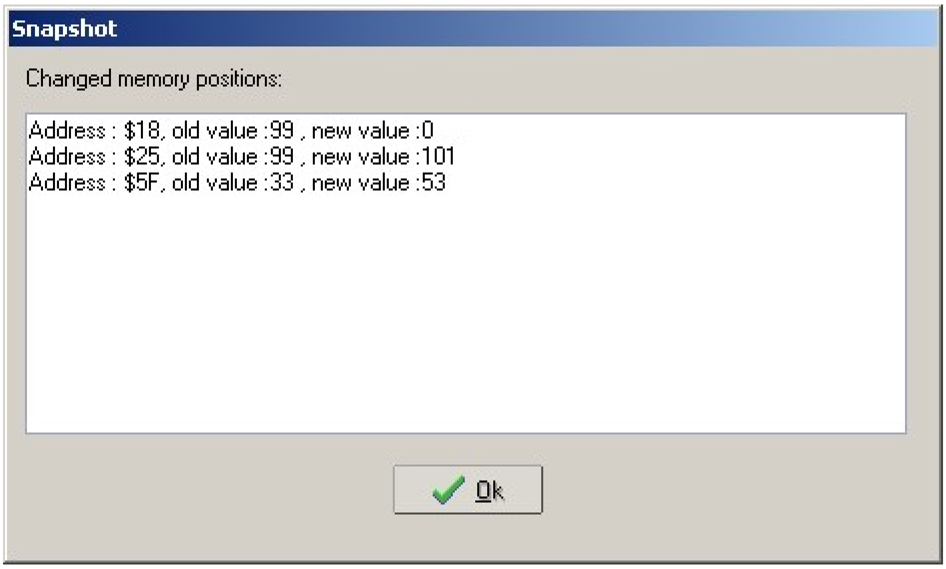
The software stack, hardware stack, and frame pointer values are shown. The minimum or maximum value that occurred during simulation is also shown. When

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’

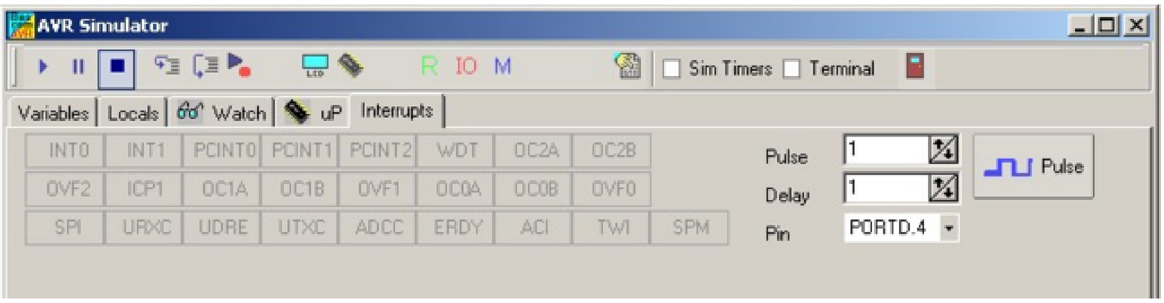
Now execute some code by pressing F8 and press the Snapshot-button again.

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



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



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

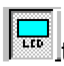
Under the Terminal section you find the Source Window.
It contains the source code of the program you are simulating. All lines that contain executable code have a yellow point in the left margin.
You can set a breakpoint on these lines by selecting the line and pressing F9.

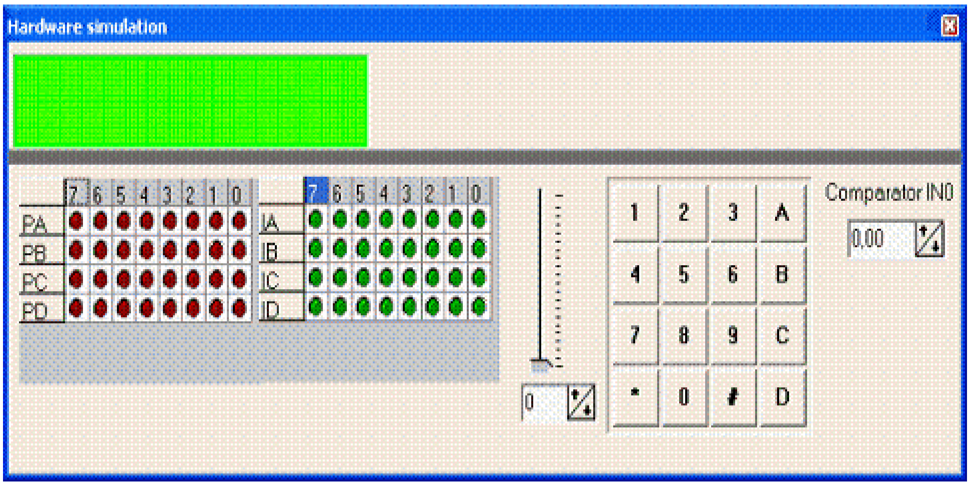
By holding the mouse cursor over a variable name, the value of the variable is shown in the status bar.
If you select a variable, and press ENTER, it will be added to the Variable window.

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.

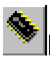
Right beside the PIN led's, there is a track bar. This bar can be used to simulate the input voltage applied the ADC converter. Note that not all chips have an AD converter. You can set a value for each channel by selecting the desired channel below the track bar.

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 IN0.

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.

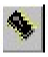
Power up or reset the DT006. It probably already is powered since you just previously compiled the basmon.bas program and stored it in the 2313.
When you press the real hardware simulation button now the simulator will send and receive data when a port, pin or DDR register is changed.

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 button again.

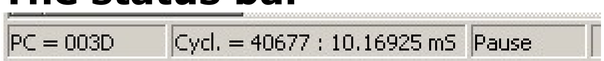
The Real Hardware Simulation is a cost effective way to test attached hardware.

 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



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.

3.32 Program Send to Chip

Program send to chip shortcut , F4

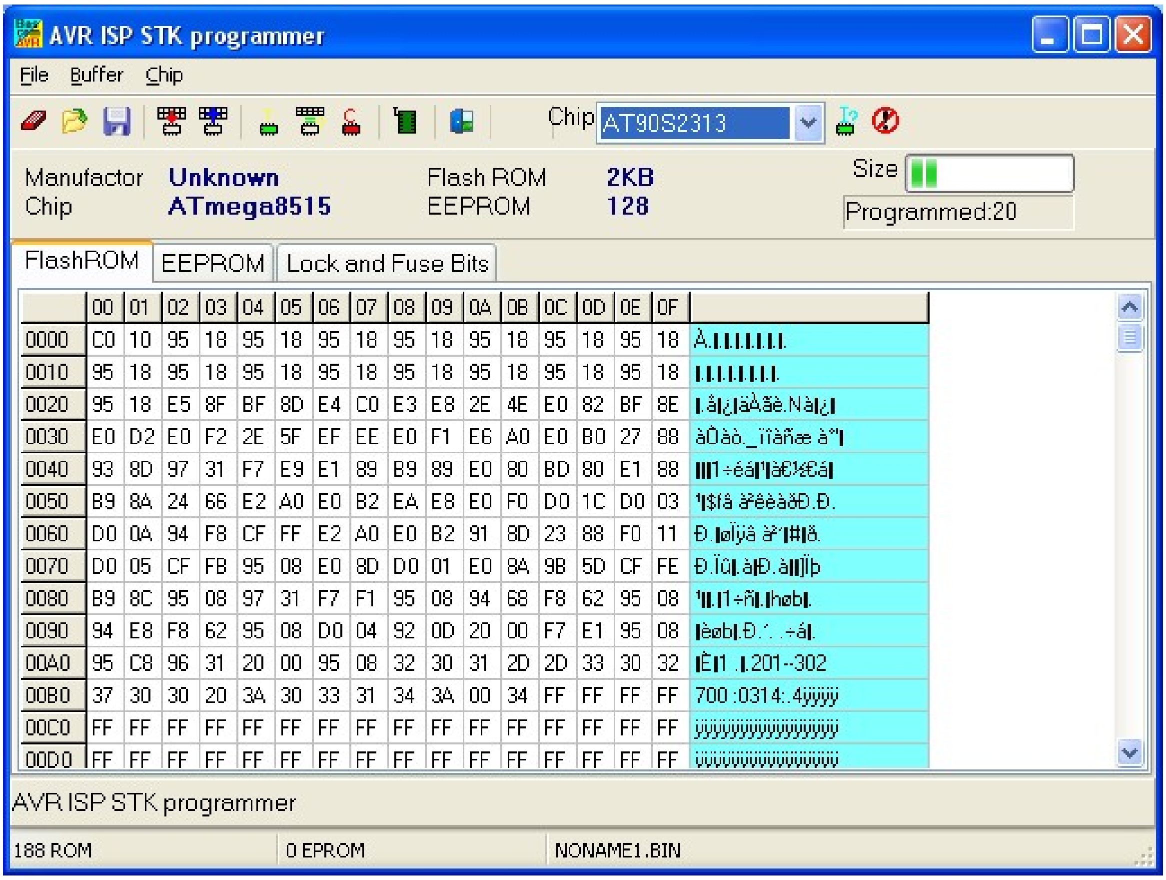
This option will bring up the selected programmer window, or will program the chip directly if the 'Auto Flash' option is selected in the Programmer options 98 section.

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.

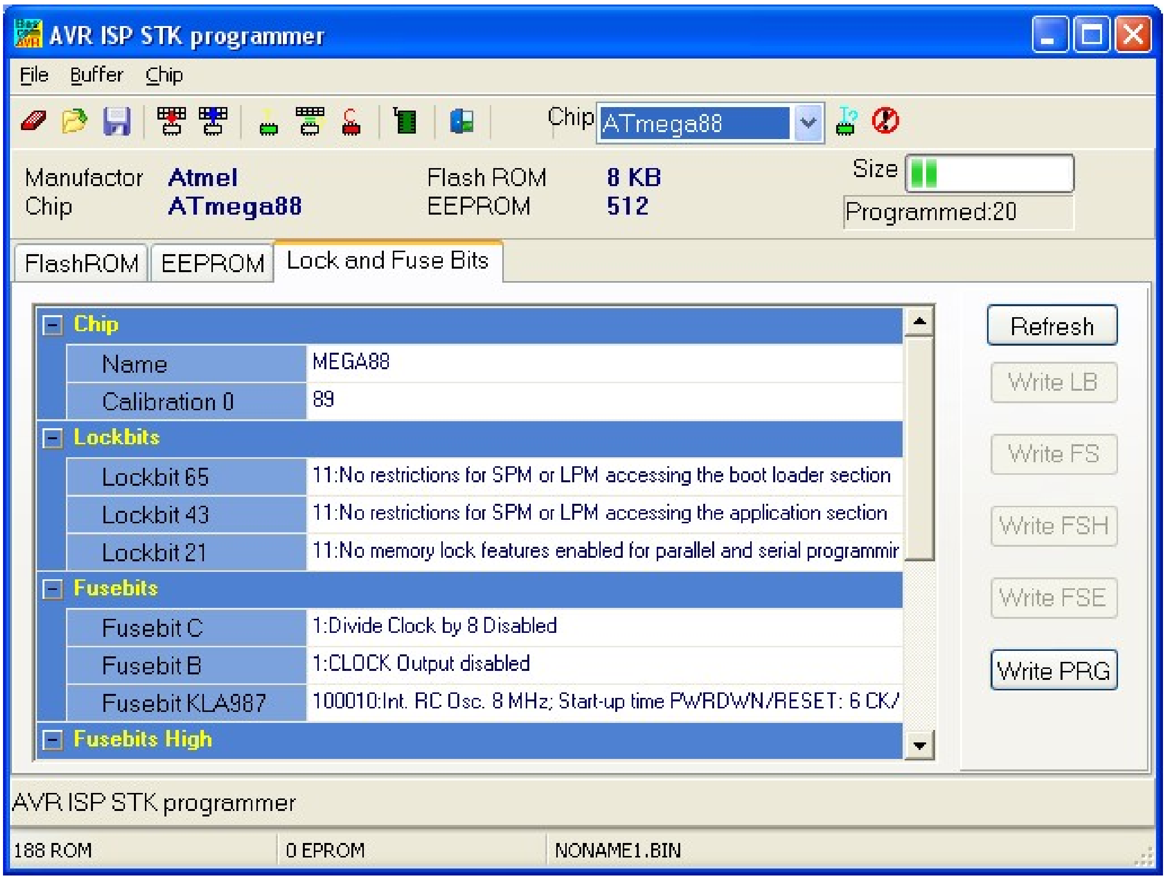
The following window will be shown for most programmers:



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.



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 $PROG 294 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 $crystal 262 = 8000000 then.

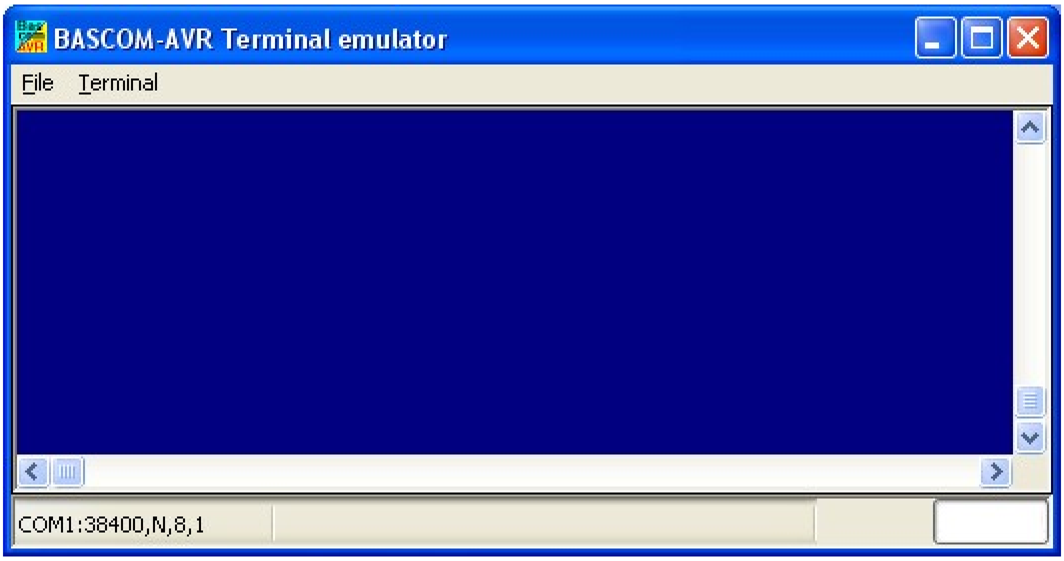
$crystal will only inform the compiler which oscillator speed you have selected.

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.

3.33 Tools Terminal Emulator

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



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.

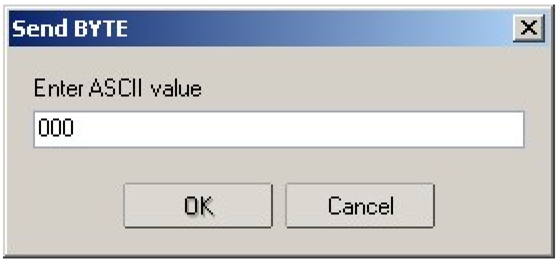
File Escape
Aborts the upload to the monitor program.

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.



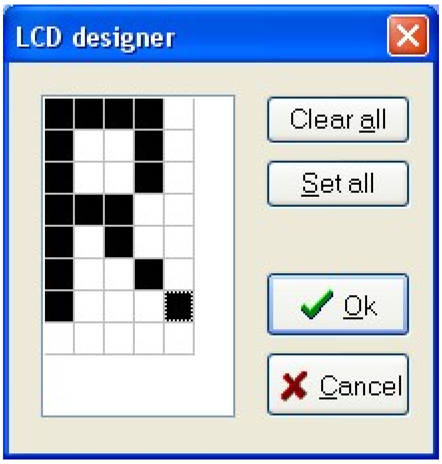
Terminal Send Magic number
This option will send 4 bytes to the terminal emulator. The intention is to use it together with the boot loader examples. Some of the boot loader samples check for a number of characters when the chip resets. When they receive 4 'magic' characters after each other, they will 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 Options, Communication 92.

3.34 Tools LCD Designer

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

The following window will appear:



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

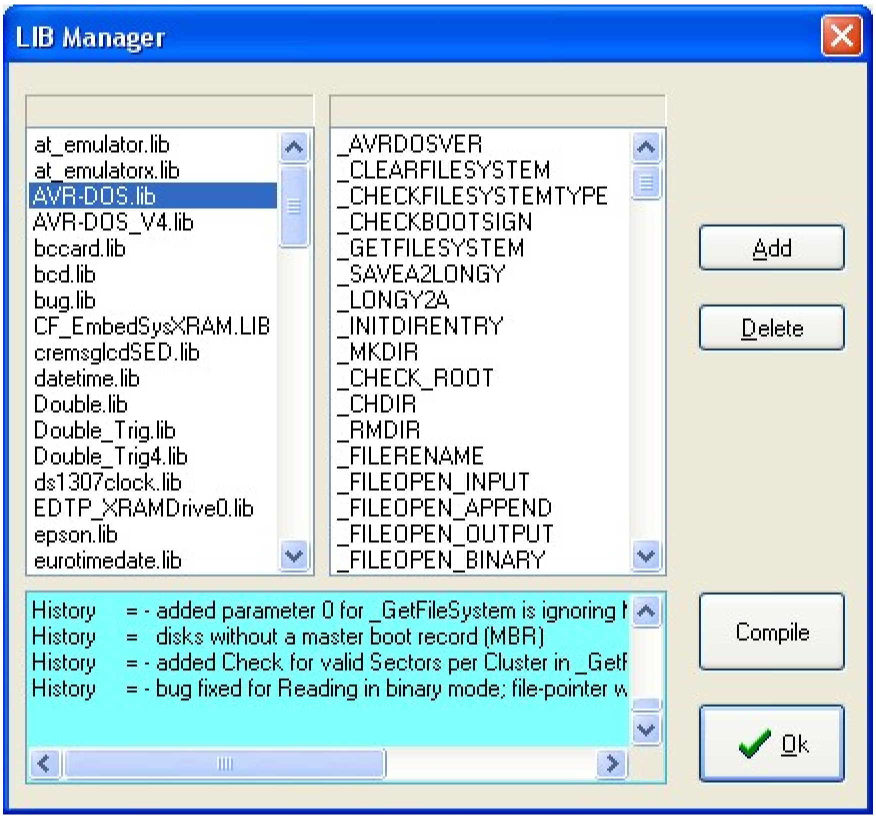
You must replace the ?-sign with a character number ranging from 0-7.
The eight bytes define how the character will appear. So they will be different depending on the character you have drawn.

See Also

Font Editor 125

3.35 Tools LIB Manager

With this option the following window will appear:



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

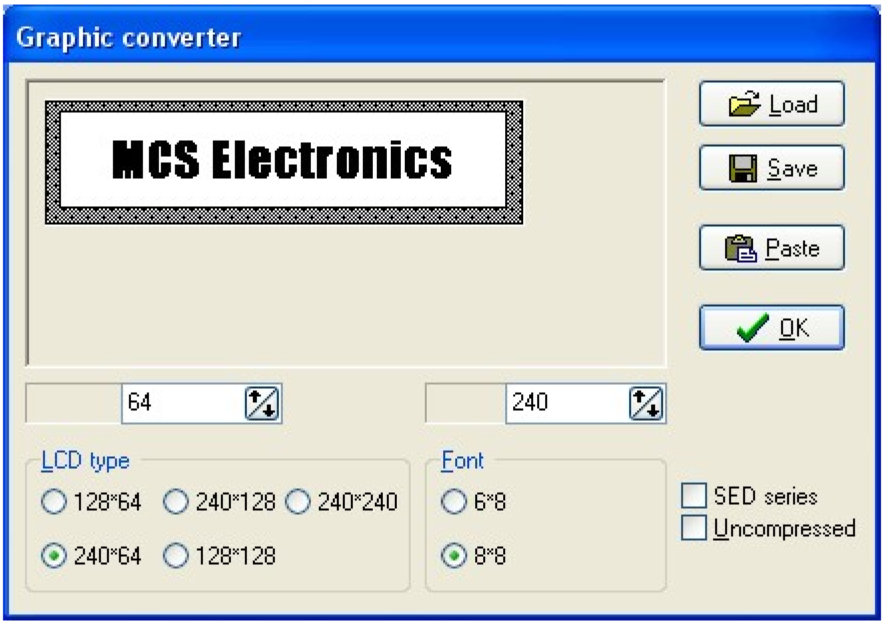
See Also

$LIB 283 for writing your own libraries

3.36 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.

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 ShowPic 749 statement or the ShowpicE statement.

The BGF files are RLE encoded to save space.

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 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.
Font	The T6963 supports 6x8 and 8x8 fonts. This is the font select that must match the CONFIG statement. For other displays, use 8*8.
Type	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.
Uncompressed	Images are RLE encoded. Select this option when you do not want to compress the image.

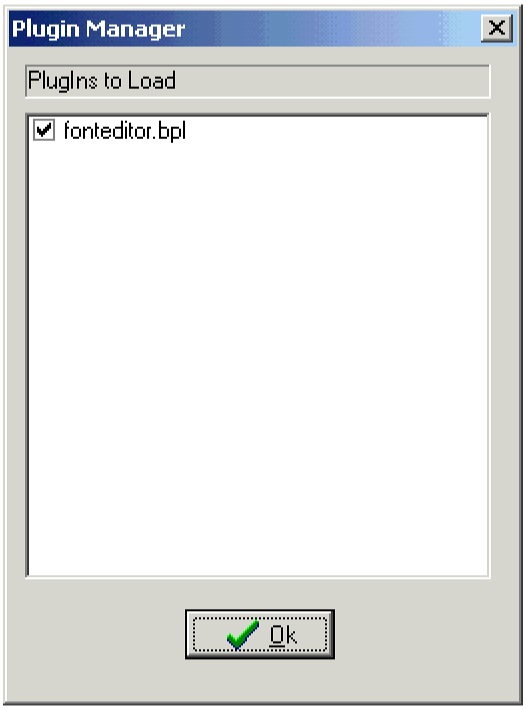
3.37 Tools Stack Analyzer

The Stack analyzer helps to determine the proper stack size.

See $DBG 264 for the proper usage of this option.

3.38 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.



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.

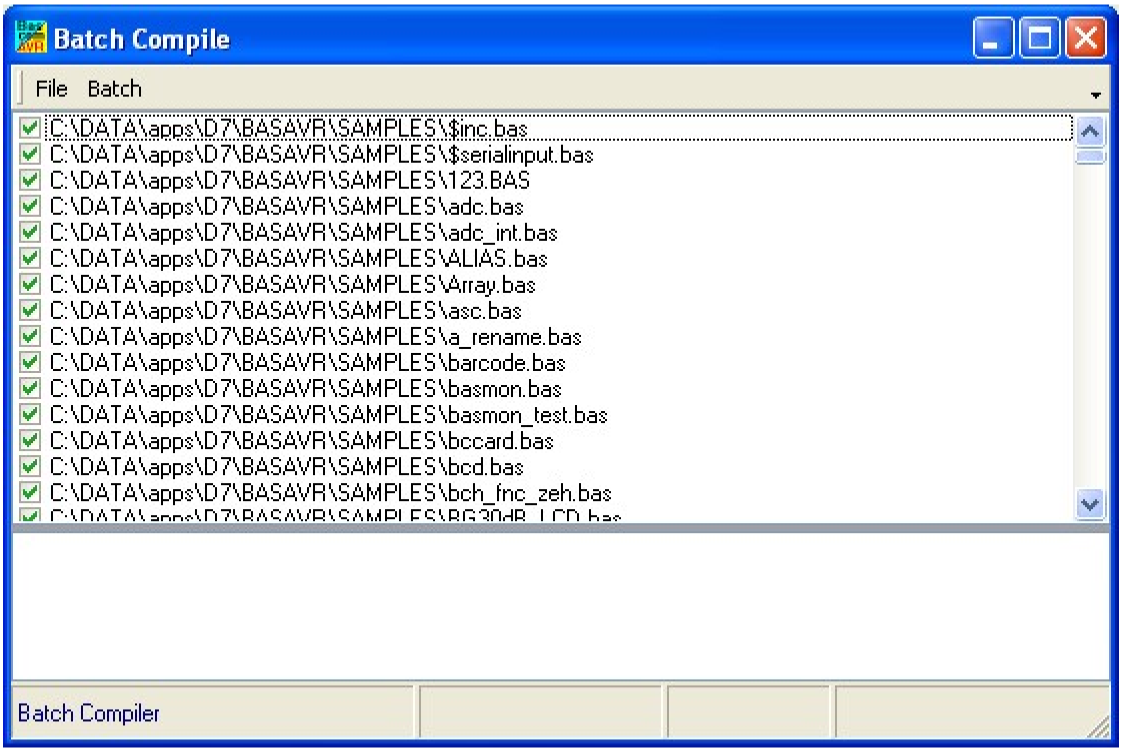
To add a button to the toolbar, right click the mouse on the menu bar, and choose customize.

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

3.39 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 :



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

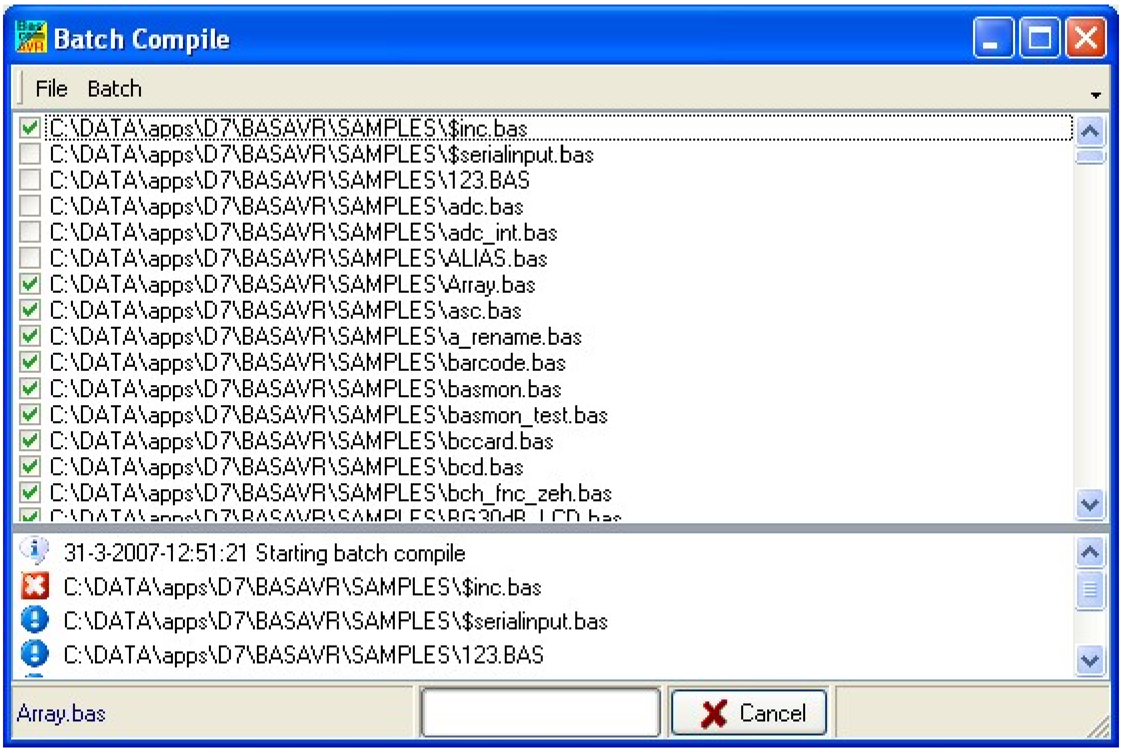
Save the batch compile log file to disk. A file named batchresult.txt will be saved in the BASCOM application directory.

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.



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.

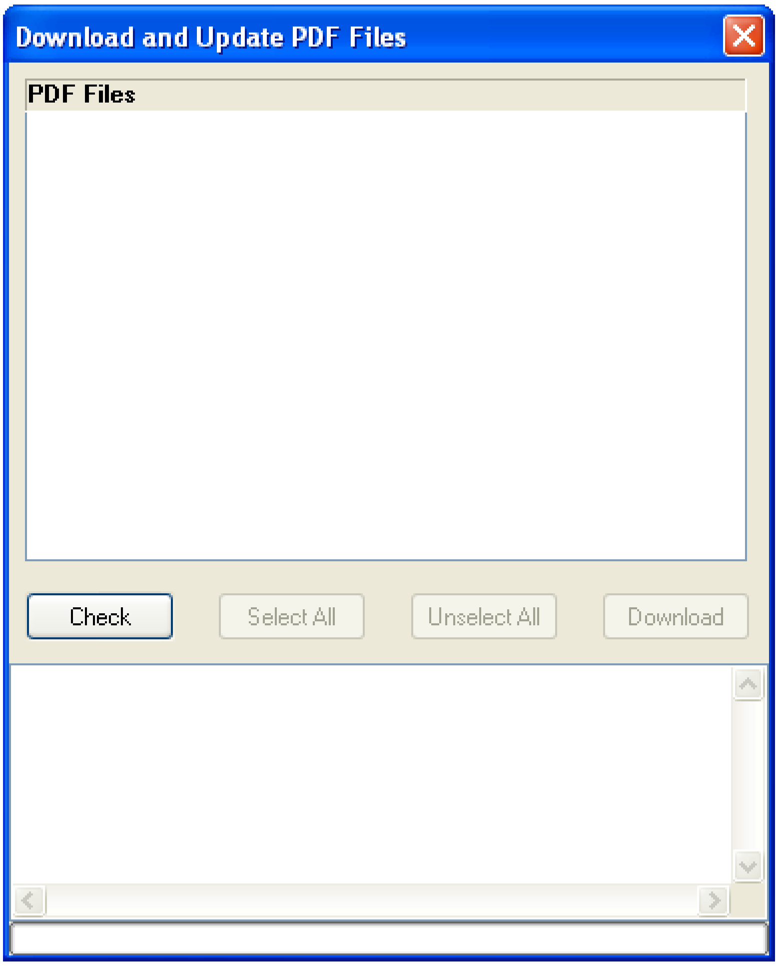
All results are shown in an error list at the bottom of the screen.
When you double click an item, the file will be opened by the editor.

See Also

$NOCOMP 292

3.40 Tools PDF Update

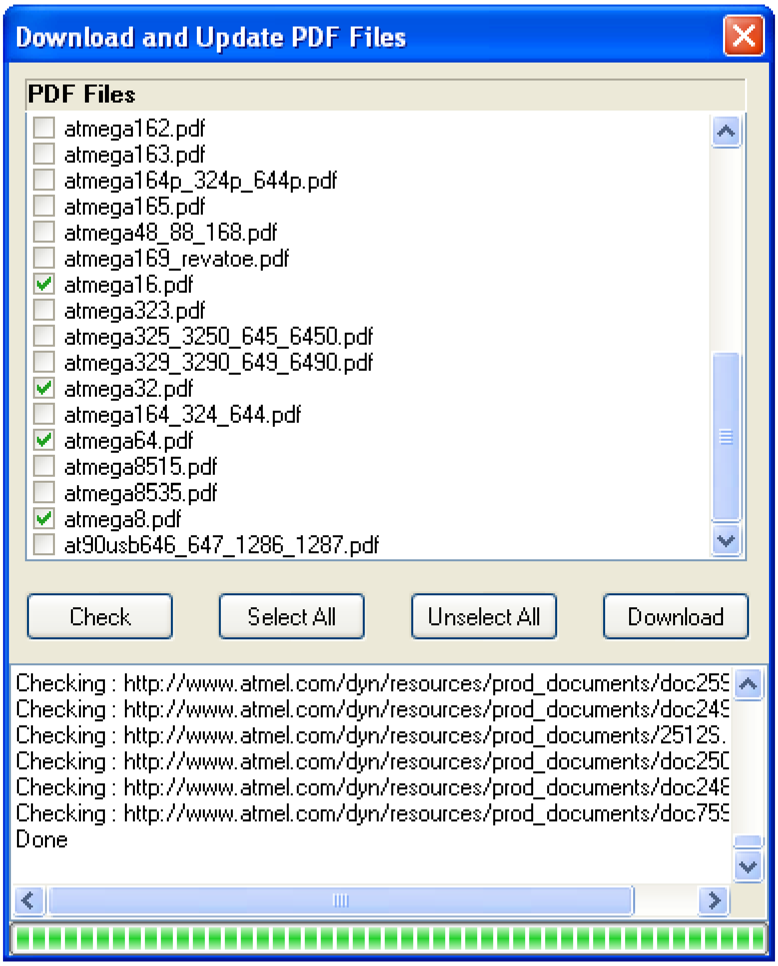
Use this option to update all Atmel PDF files.
The Atmel data sheets are stored in the **\PDF** subdirectory.
The following window will be shown :



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 :



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.

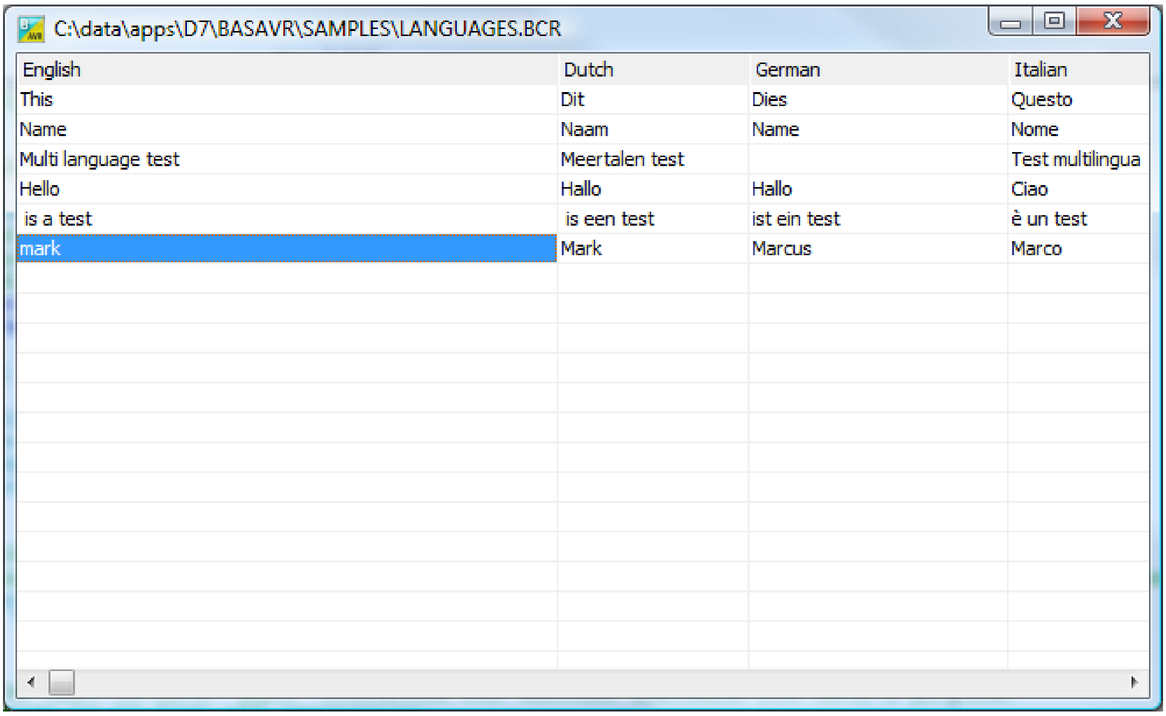
When you want to download the selected files, press the Download-button.
This will close all PDF documents in the PDF viewer. A backup of each PDF file downloaded will be made before it is downloaded. You need to restore it when something goes wrong during the download(server drops the connection for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

3.41 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.



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).

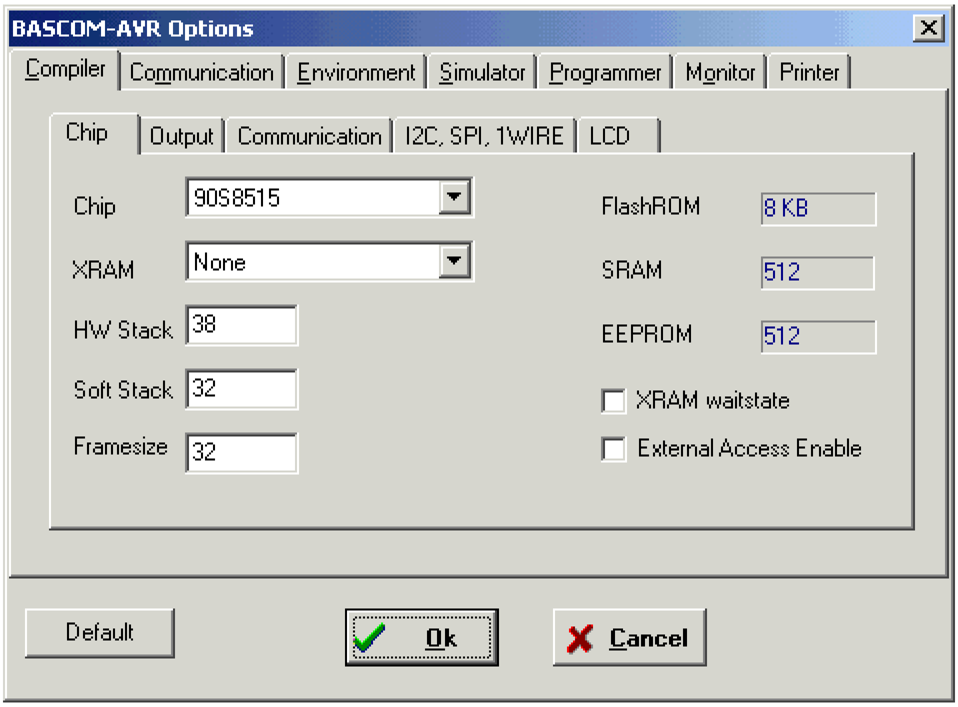
The resource editor is pretty simple. The only task is allow you to edit the various strings. You can also use notepad or Excel to create the BCR file which is explained in the $RESOURCE 297 topic.

3.42 Options Compiler

With this option, you can modify the compiler options.

The following TAB pages are available:

3.42.1 Options Compiler Chip



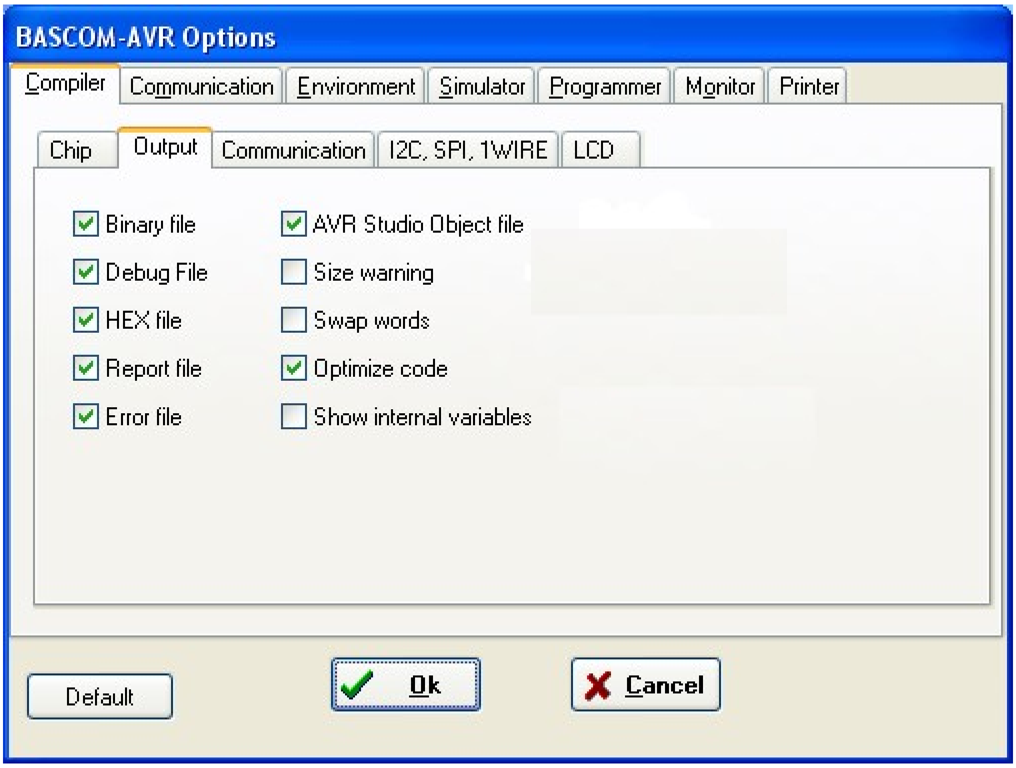
The following options are available:

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. For 32 KB you need a 62256 STATIC RAM chip.
HW Stack	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.
External Access 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
Default	Press or click this button to use the current Compiler Chip settings as default for all new projects.

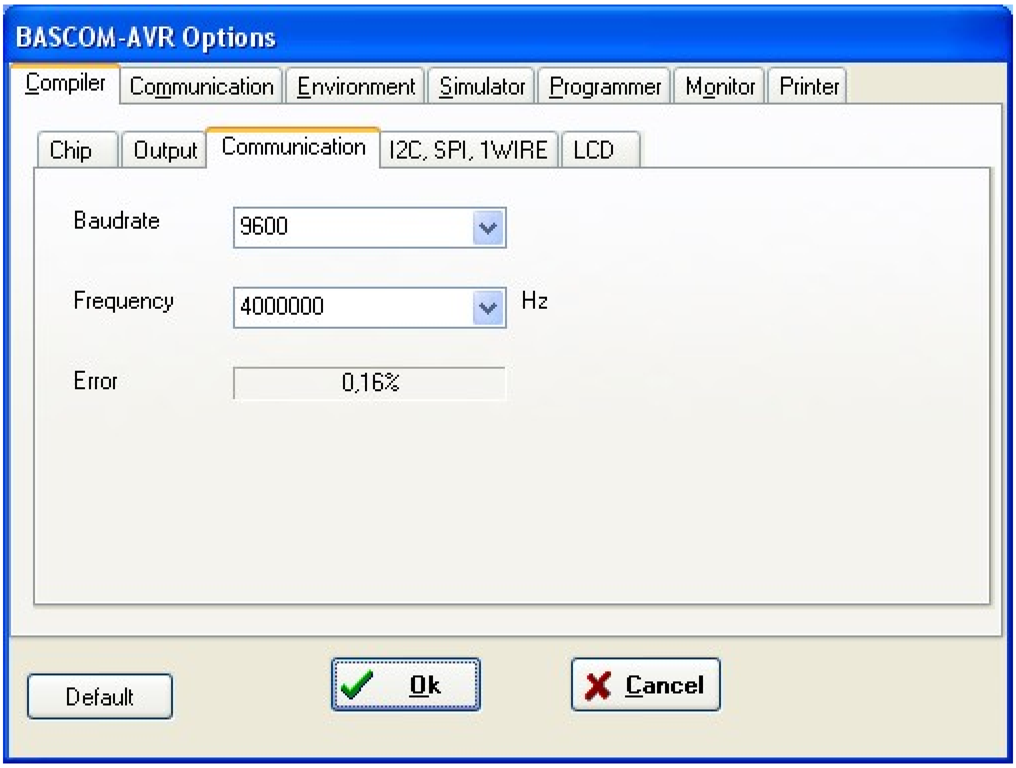
3.42.2 Options Compiler Output



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. Like _TEMP1 = R24. This option shows these variables in the report.

3.42.3 Options Compiler Communication



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 $BAUD 257 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 $CRYSTAL 262 in the source code which overrides this setting. Settings in source code are preferred since it is more clear.

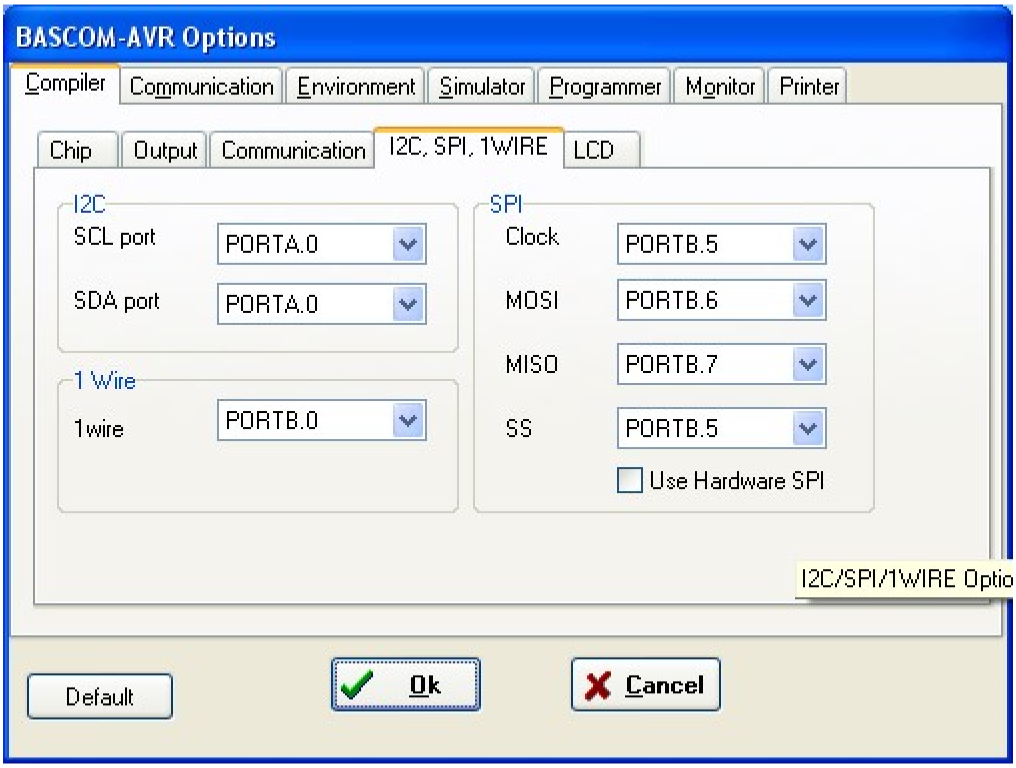
The settings for the internal hardware UART are:

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.

3.42.4 Options Compiler I2C, SPI, 1WIRE

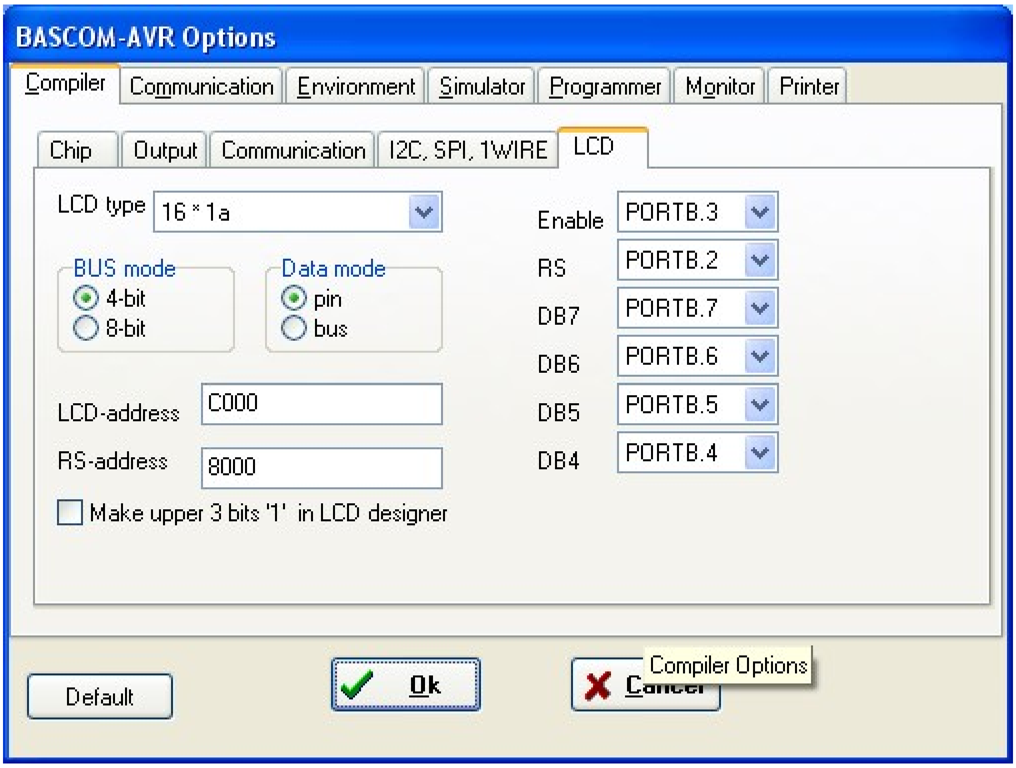


Options Compiler I2C, SPI, 1WIRE

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 CONFIG 375 commands in your source code. It make more clear in the source code which pins are used.

3.42.5 Options Compiler LCD



Options Compiler LCD

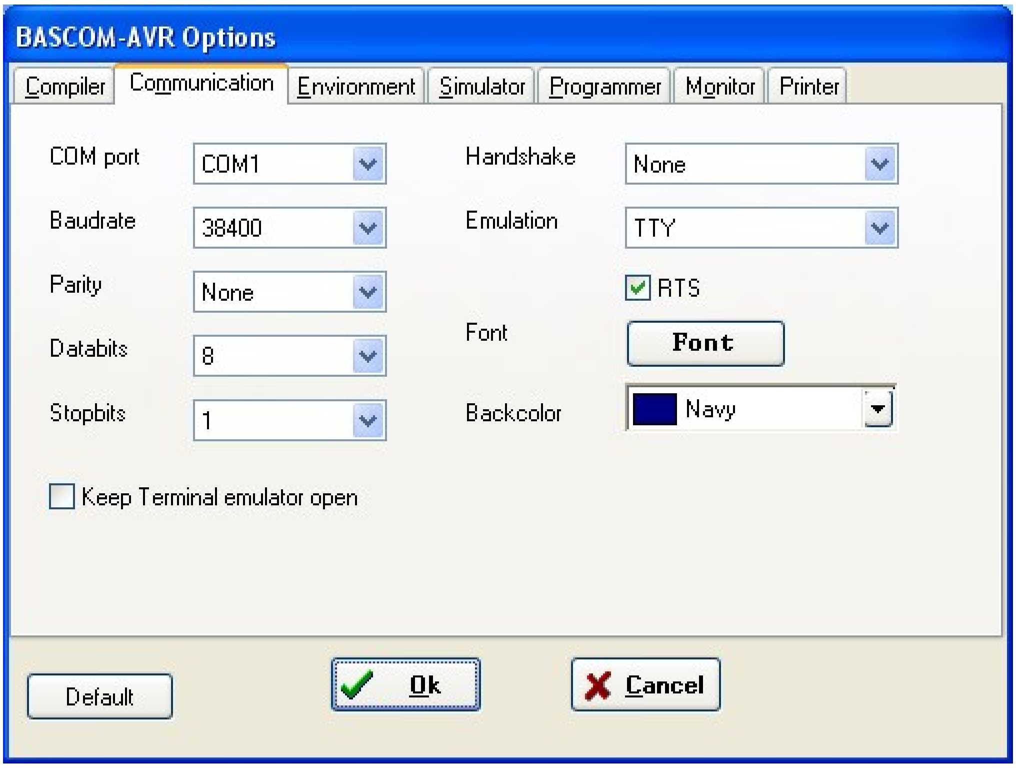
Item	Description
LCD type	The LCD display used.
Bus mode	The LCD can be operated in BUS mode or in PIN mode. In PIN mode, the data lines of the LCD are connected to the processor port pins. In BUS mode the data lines of the LCD are connected to the data lines of the BUS. Select 4 when you have only connect 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.

3.43 Options Communication

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

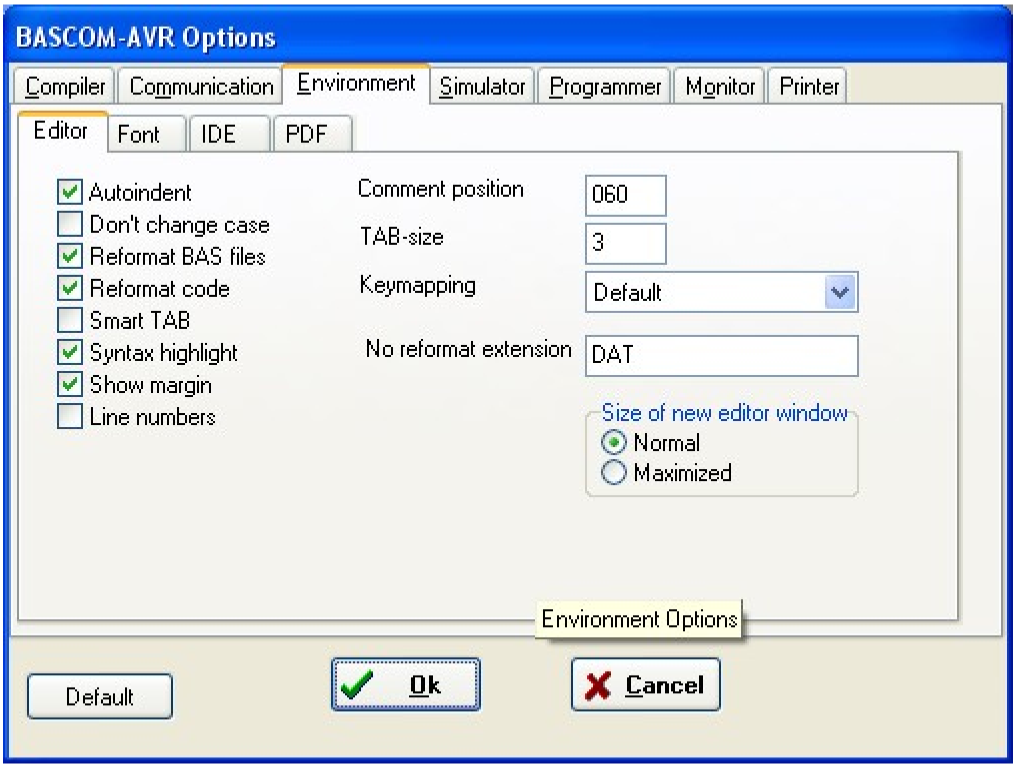


Item	Description
Comport	The communication port of your PC that you use for the terminal emulator.
Baud rate	The baud rate to use.
Parity	Parity, default None.
Data bits	Number of data bits, default 8.
Stop bits	Number of stop bits, default 1.
Handshake	The handshake used, default is none.
Emulation	Emulation used, default BBS ANSI.
Font	Font type and color used by the emulator.
Back color	Background color of the terminal emulator.

Note that the baud rate of the terminal emulator and the baud rate setting of the compiler options 89, must be the same in order to work correctly.

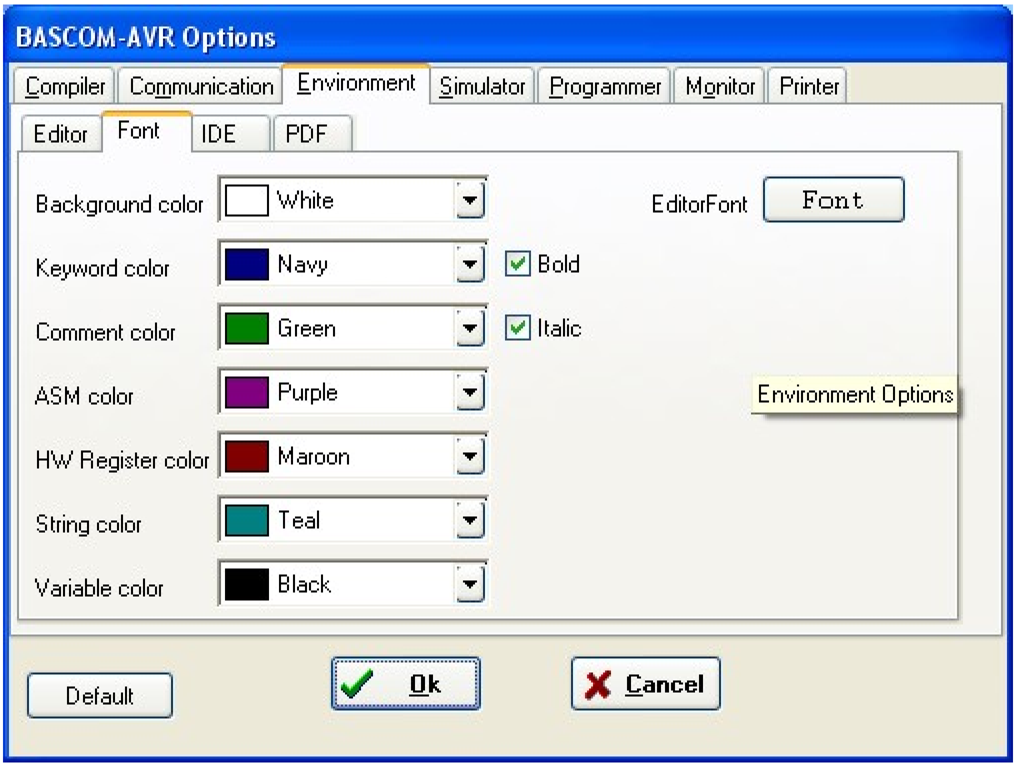
The reason why you can specify them both to be different is that you can use the terminal emulator for other purposes too.

3.44 Options Environment

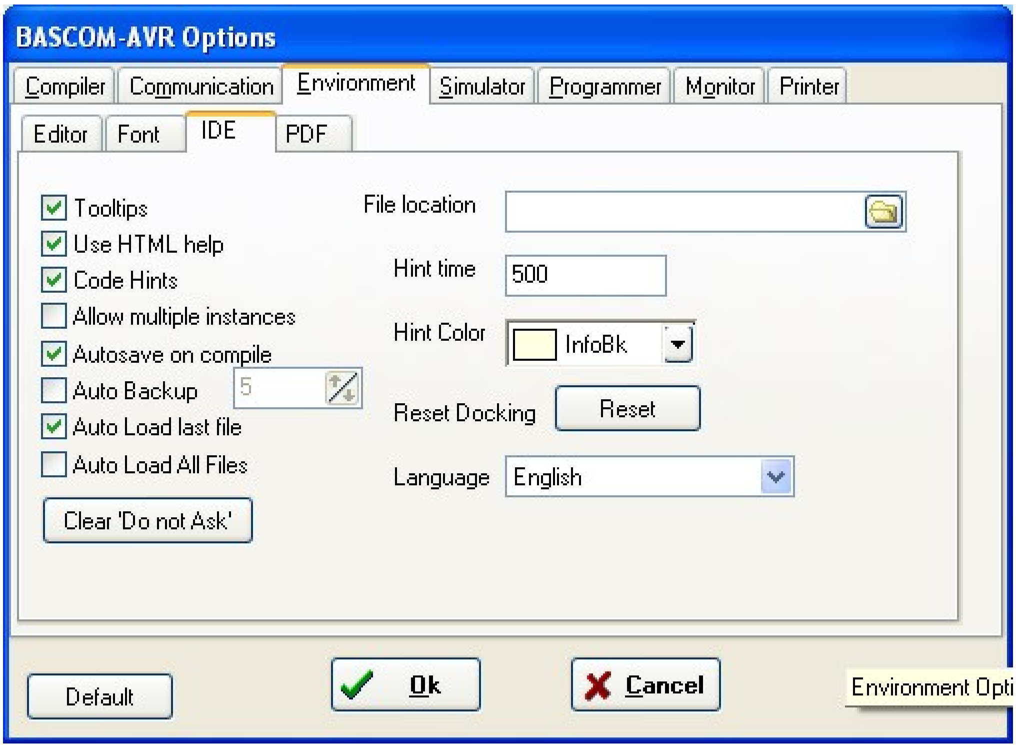


OPTION	DESCRIPTION
Auto Indent	When you press return, the cursor is set to the next line at the current column position.
Don't change case	When set, the reformat won't change the case of the line after you have edited it. Default is that the text is reformatted so every word begins with upper case.
Reformat 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	File extensions separated by a space that will not be reformatted

extension	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.

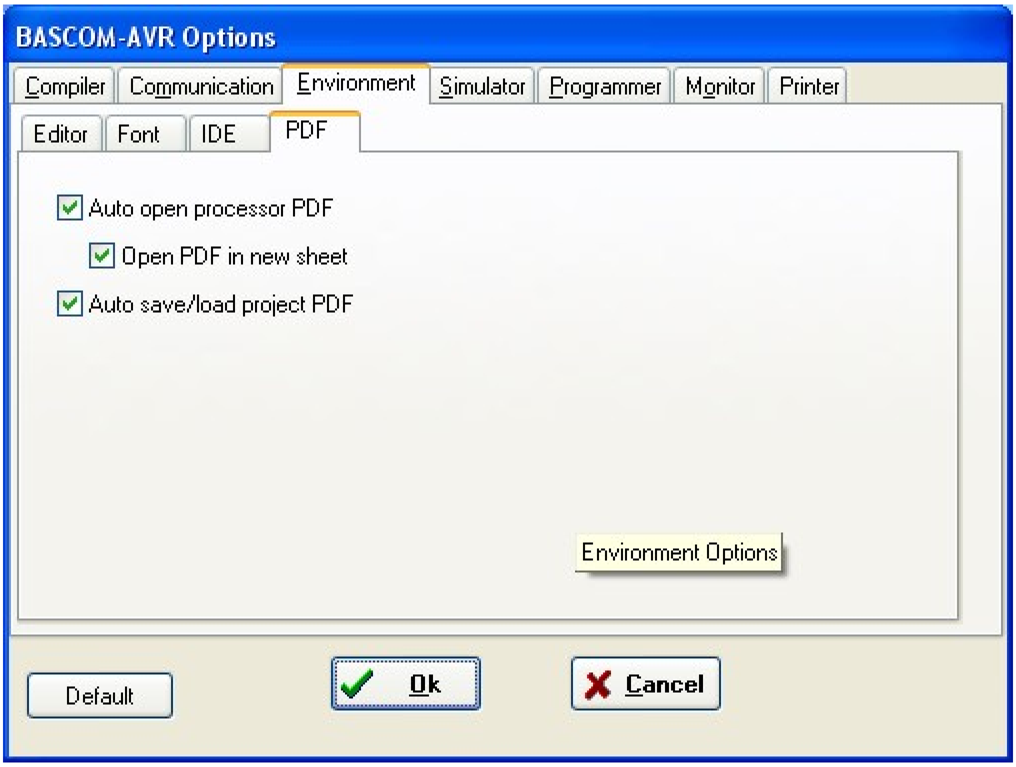


OPTION	DESCRIPTION
Background color	The background color of the editor window.
Keyword color	The color of the reserved words. Default Navy. The keywords can be displayed in **bold** too.
Comment color	The color of comment. Default green. Comment can be shown in *Italic* too.
ASM color	Color to use for ASM statements. Default purple.
HW registers color	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.



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.
Auto save on compile	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

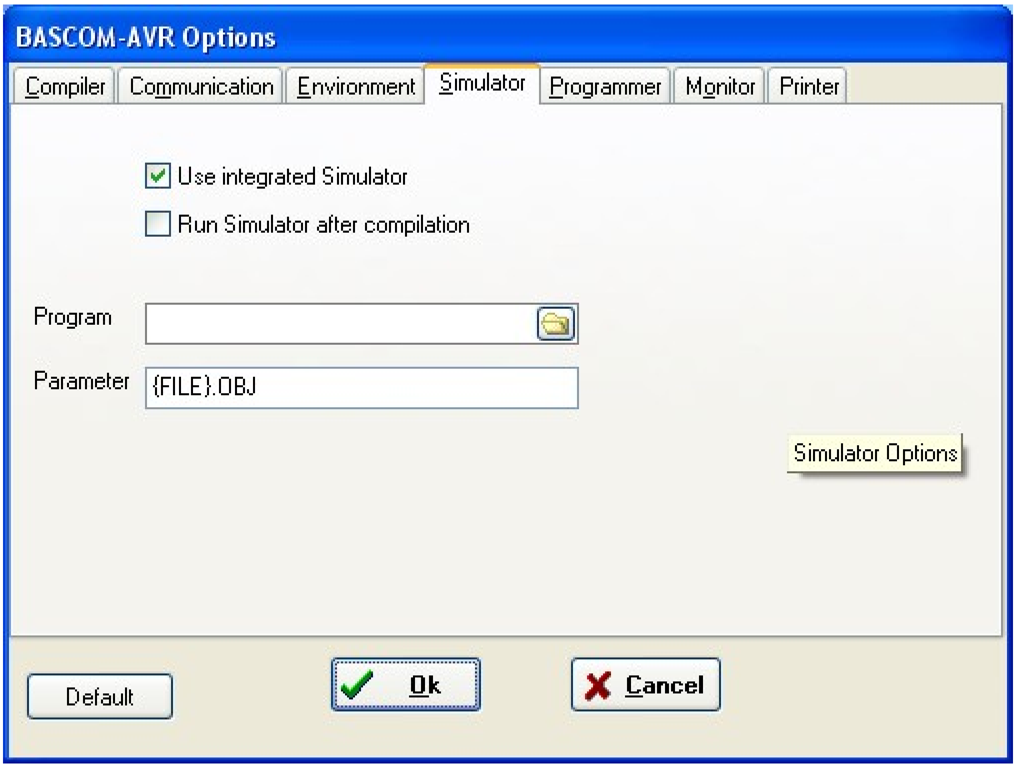
	yet.



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.
Open PDF in new sheet	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.

3.45 Options Simulator

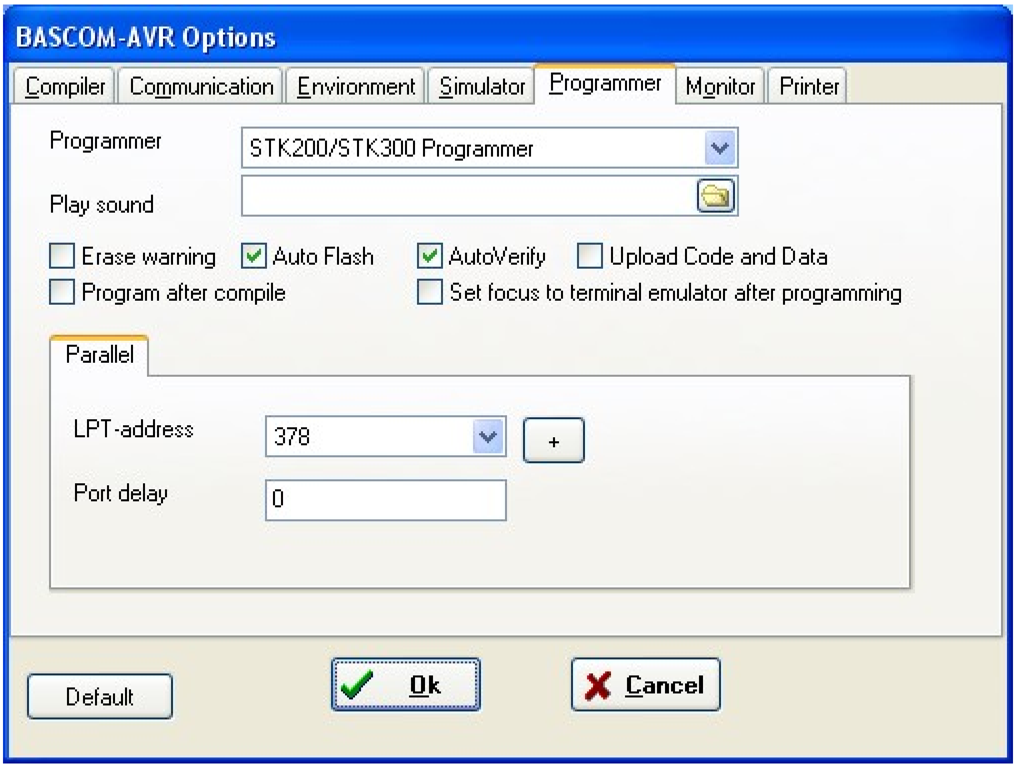
With this option you can modify the simulator settings.



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.

3.46 Options Programmer

With this option you can modify the programmer settings.



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 systems a 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 99

3.46.1 Supported Programmers

BASCOM supports the following programmers

AVR ICP910 based on the AVR910.ASM application note 109

STK200 ISP programmer from Atmel 109

The PG302 programmer from Iguana Labs 101

The simple cable programmer from Sample Electronics. 101

KITSRUS KIT122 Programmer 102

MCS Universal Interface Programmer 103

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. 105

Lawicel BootLoader 108

USB-ISP Programmer 109

MCS Bootloader 113

PROGGY 115

FLIP 115

Elektor / AVR ISP mkII 117

3.46.1.1 ISP programmer

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

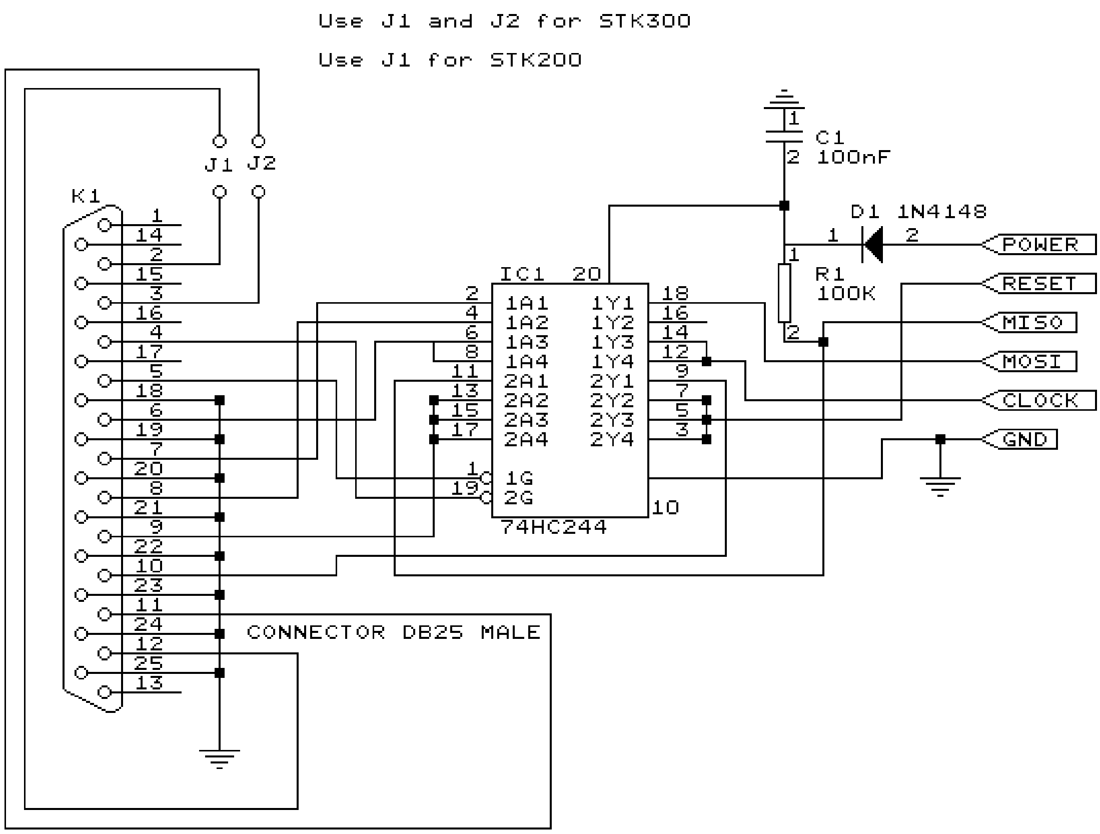
This is a very reliable parallel printer port programmer.
The STK200 ISP programmer is included in the STK200 starter kit.
Most programs were tested with the STK200.

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.

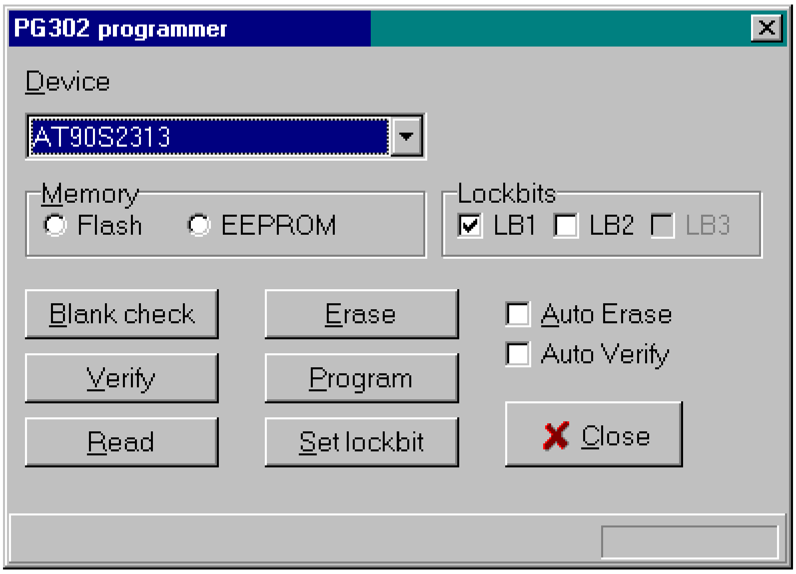
Here is a tip received from a user :

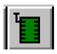
If the parallel port is disconnected from the interface and left floating, the '244 latch outputs will waver, causing your micro controller to randomly reset during operation. The simple addition of a 100K pull-up resistor between pin 1 and 20 of the latch, and another between pin 19 and 20, will eliminate 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.



3.46.1.2 PG302 programmer

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



Select the programmer from The Option Programmer menu or right click on the  button to show the Option Programmer 98 menu

3.46.1.3 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.

The connections to make are as following:

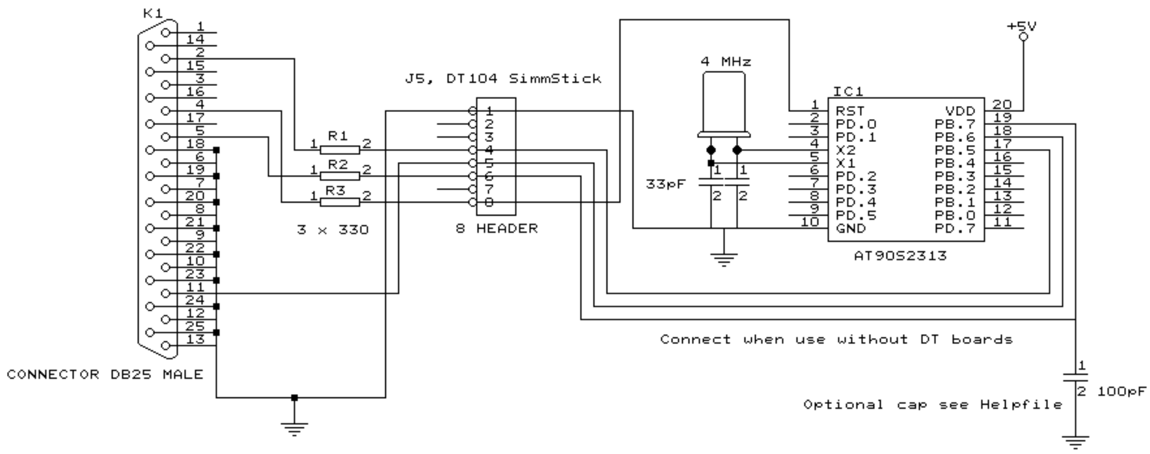
DB25 pin	Target MCU pin (AT90S8535)	Target MCU M103/M128	Target MCU pin 8515	DT104
2, D0	MOSI, pin 6	PE.0, 2	MOSI, 6	J5, pin 4
4, D2	RESET, pin 9	RESET, 20	RESET, 9	J5, pin 8
5, D3	CLOCK, pin 8	PB.1,11	CLOCK, 8	J5, pin 6
11, BUSY	MISO, pin 7	PE.1, 3	MISO, 7	J5, pin 5
18-25,GND	GROUND	GROUND	GND,20	J5, pin 1

The 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:

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.

3.46.1.4 KITSRUS Programmer

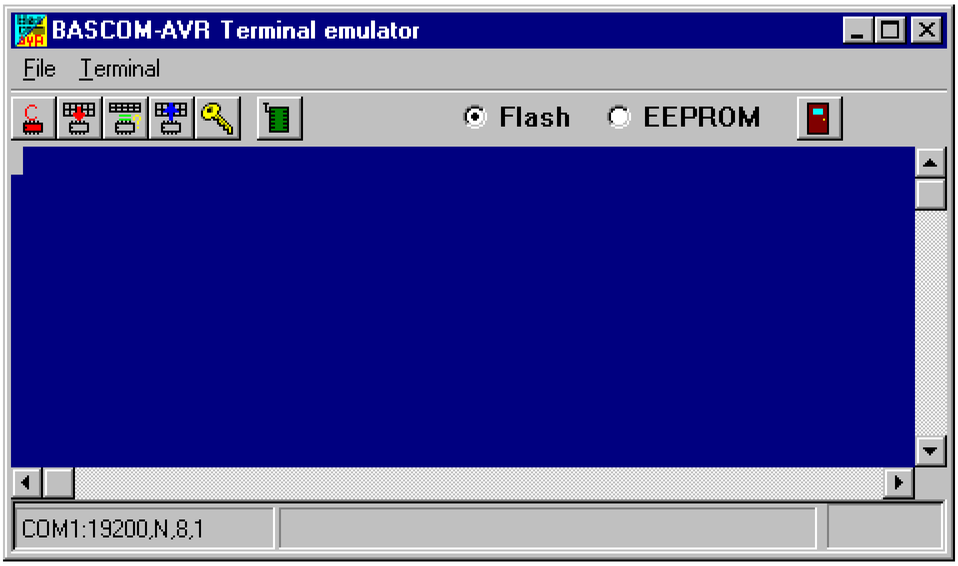
The K122 is a KIT from KITSRUS. (www.kitsrus.com)

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:



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.

3.46.1.5 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.

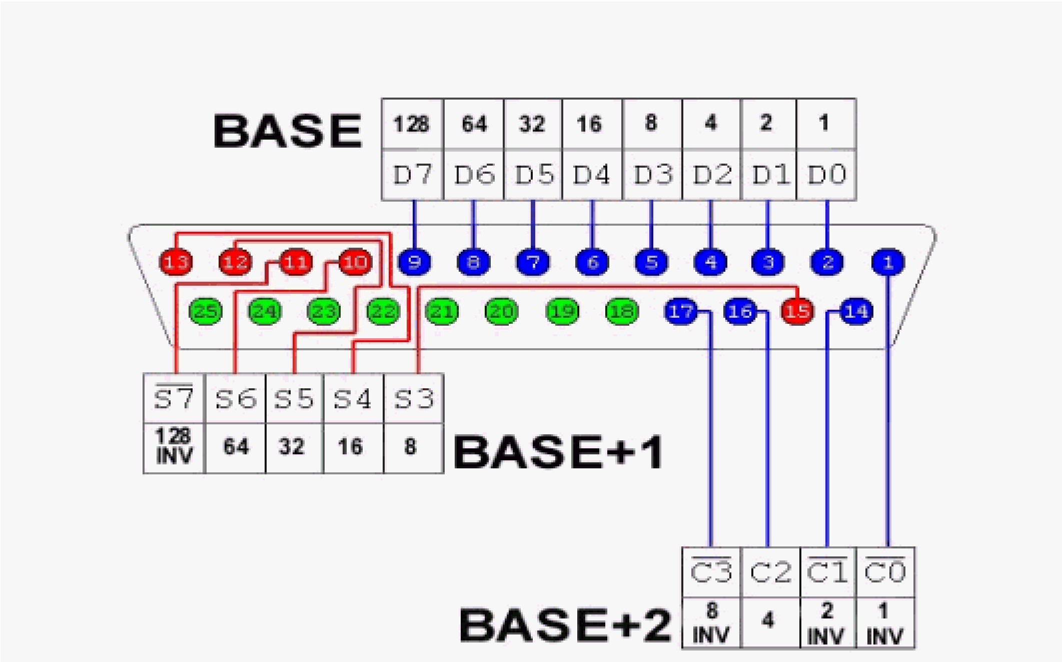
The pins that are needed are MOSI, CLOCK, RESET and MISO.
Add the pin name MOSI =

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.



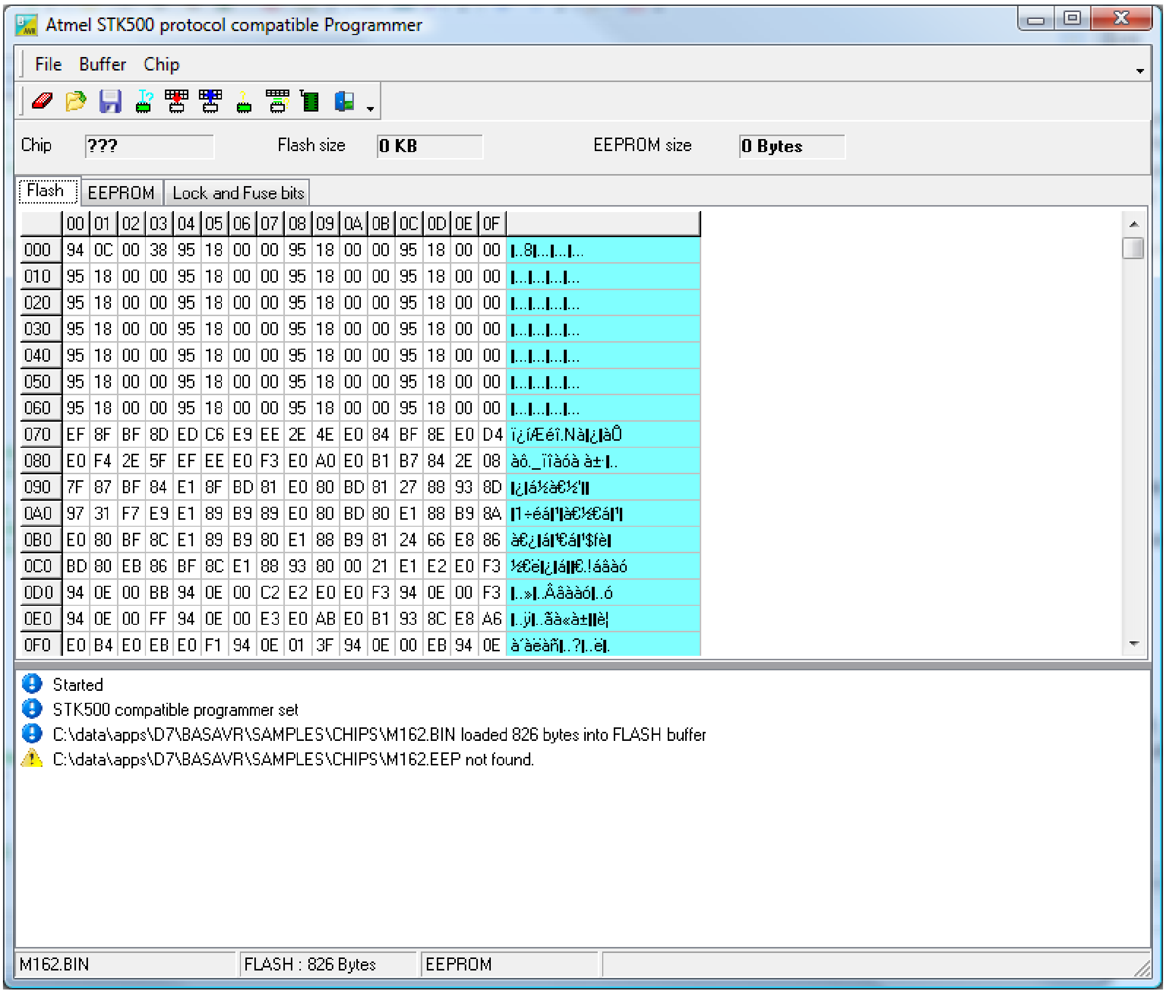
Always add your entry to the bottom of the file and email the settings to support@mcselec.com so it can be added to BASCOM.

3.46.1.6 STK500 Programmer

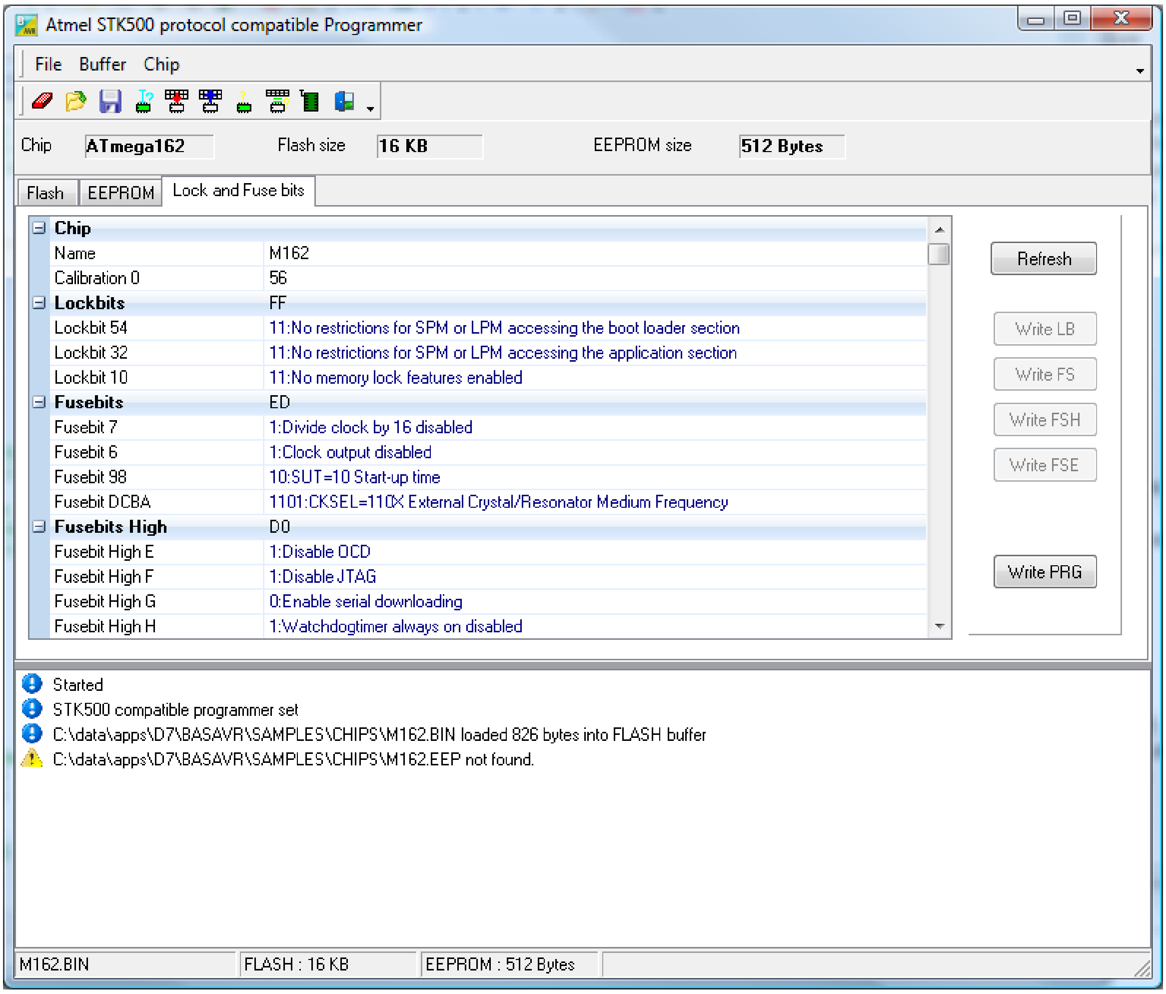
When you select the STK500 programmer, BASCOM will run the files named stk500. exe that is installed with AVR Studio.

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 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)



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.



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 fusebit, 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 fusebit '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 fusebit (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 a $PROG directive.
For example : $prog &HFF , &HED , &HD0 , &HFF
When you compile your program with the $PROG 294 directive 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:

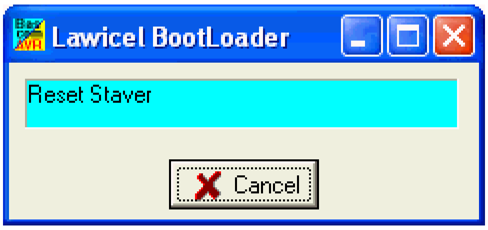
Option	Description
File	
Exit	Close programmer.
Buffer	
Clear	Clear buffer. Will put a value of 255 (FF hex) into each memory location. When the FLASH-TAB has the focus, the FLASH buffer will be cleared. When the EEPROM-TAB has the focus, the EEPROM buffer will be cleared. 255 is the value of an empty 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.

3.46.1.7 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.



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.

3.46.1.8 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 STK500 programmer 105 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.

3.46.1.9 USB-ISP Programmer

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.

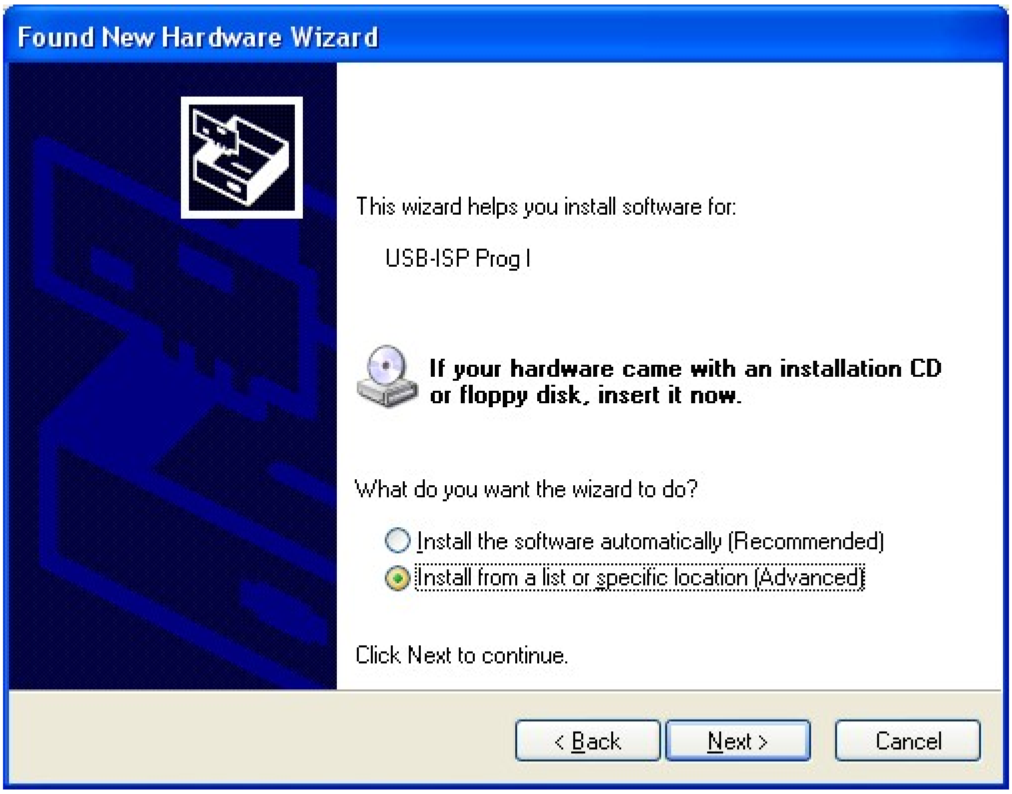
The drivers are located in the USB sub directory that is located in the BASCOM-AVR application folder.

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

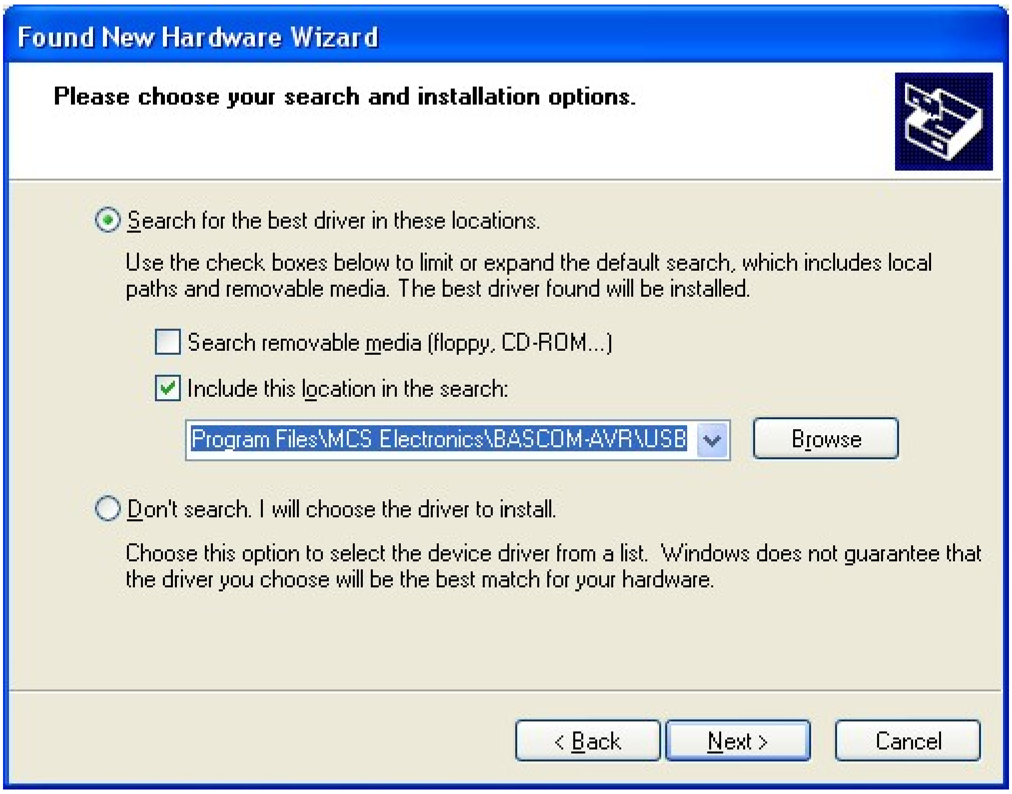
Then the Hardware wizard will be started :

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

The Wiz will show :



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



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

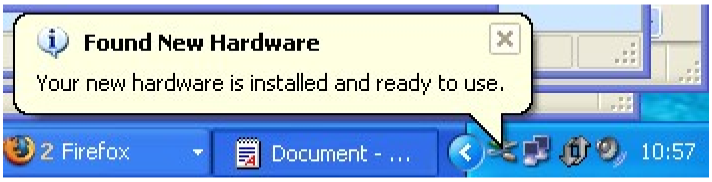
C:\Program Files\MCS Electronics\BASCOM-AVR\USB

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

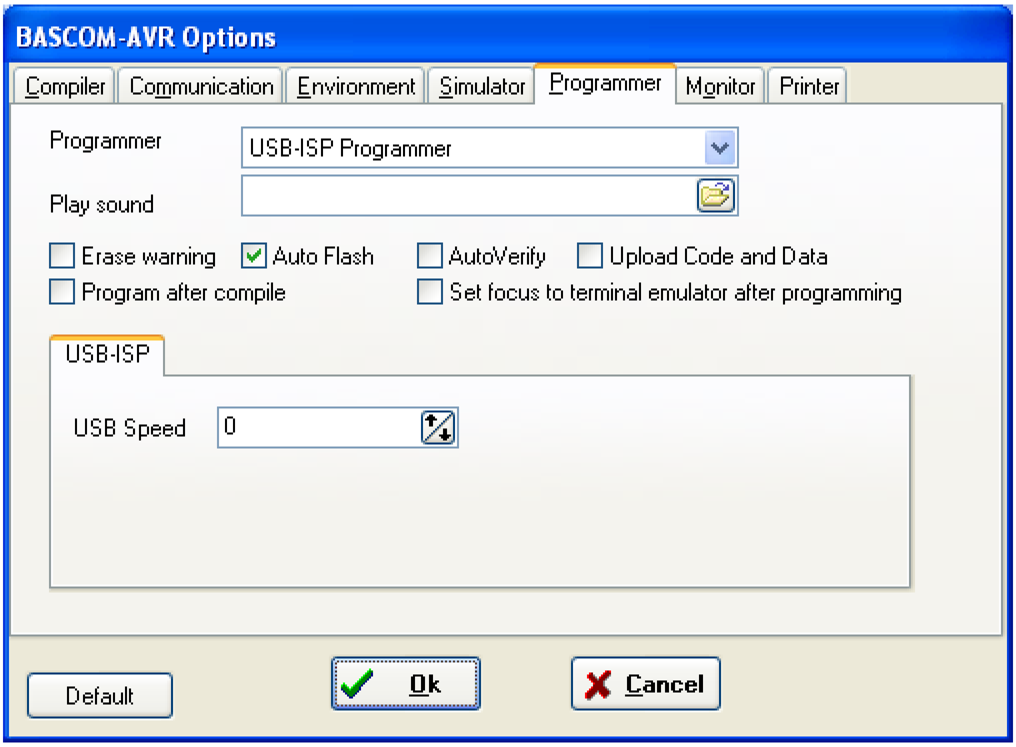
As the driver is not certified by Micros ft, you will see the following window:

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 :

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



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



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.

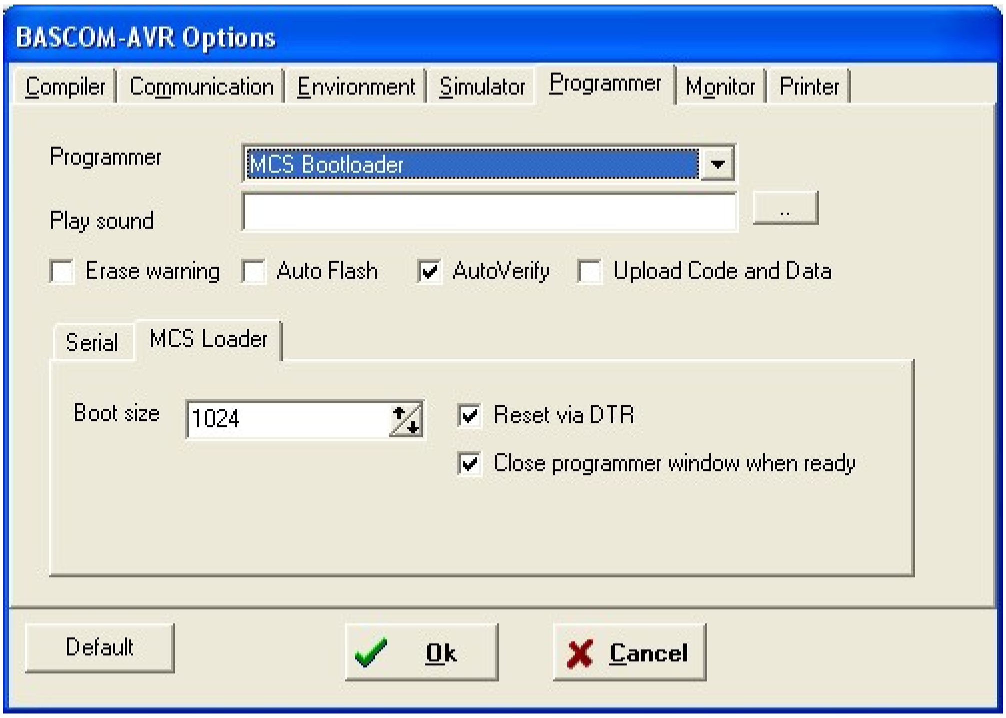
When you use other USB devices that use the FTDI drivers, there might occur a problem. Manual install the drivers of these other devices, then install the USB-ISP driver.

3.46.1.10 MCS Bootloader

The MCS Boot loader is intended to be used with the $LOADER 285 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.

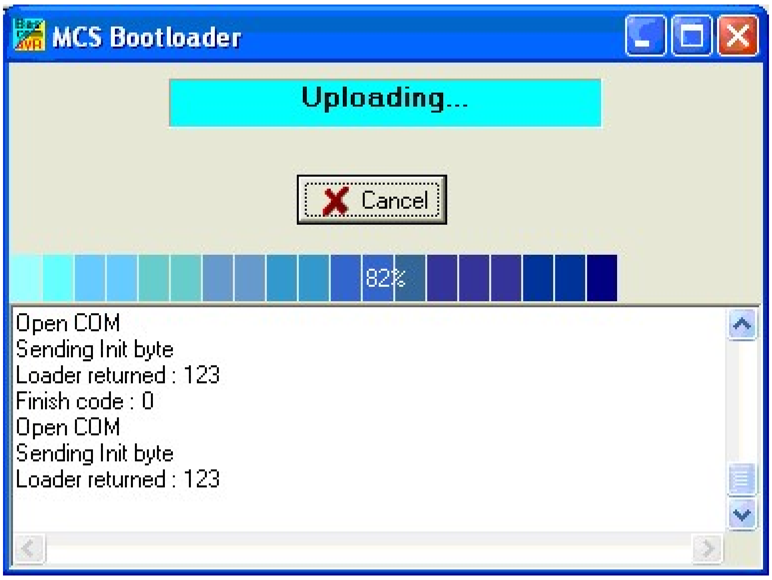


You can choose the boot size which is 1024 for the BASCOM $LOADER example. Since this space is used from the normal flash memory, it means your application has 1024 less words for the main application. (A word is 2 byte, so 2KB less)

The 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.

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.

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 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.

3.46.1.11 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.

3.46.1.12 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 : http://www.atmel.com/dyn/resources/prod_documents/Flip%20Installer%20-%203.3.1.exe
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.

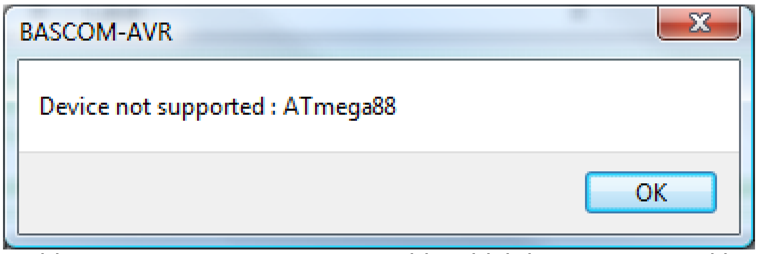
But in order to use the FLIP programmer, you need to install FLIP first.
When FLIP is working, you can select FLIP from Options, Programmer, in order to program quickly without the FLIP executable.
On Vista there is a problem with loading some of the FLIP DLL's. In case you get an error, copy the FLIP 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

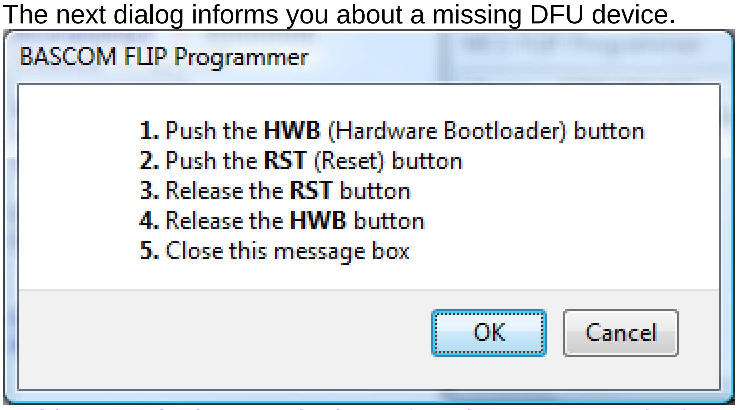
The last line pauses so you can view the result. Notice the . (dot) that will copy the file to the current directory, which is the reason that you need to run this file from the BASCOM application directory.

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:



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. (GRRRR)

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.

3.46.1.13 Elektor / AVR ISP mkII

The Elektor programmer is a neat small USB programmer which is fully compatible to the AVR ISP mKII programmer.
When you select this programmer, you will get the same interface as the STK500 native 105 programmer.
F4 will launch the programmer. For more details read the help section for the STK500 programmer.

Just like Atmel AVR Studio, the programmer uses the LIBUSB USB drivers. When you do a full setup, you can select to install this driver. Like other USB drivers/products, it will NOT work on Windows 95.
When you performed an update, you must manual install the USB drivers. Download the file name USBLIBSETUP.EXE from the MCS web and run the setup as an administrator. Installing LIBUSB is at your own risk. Make a backup before you do so!

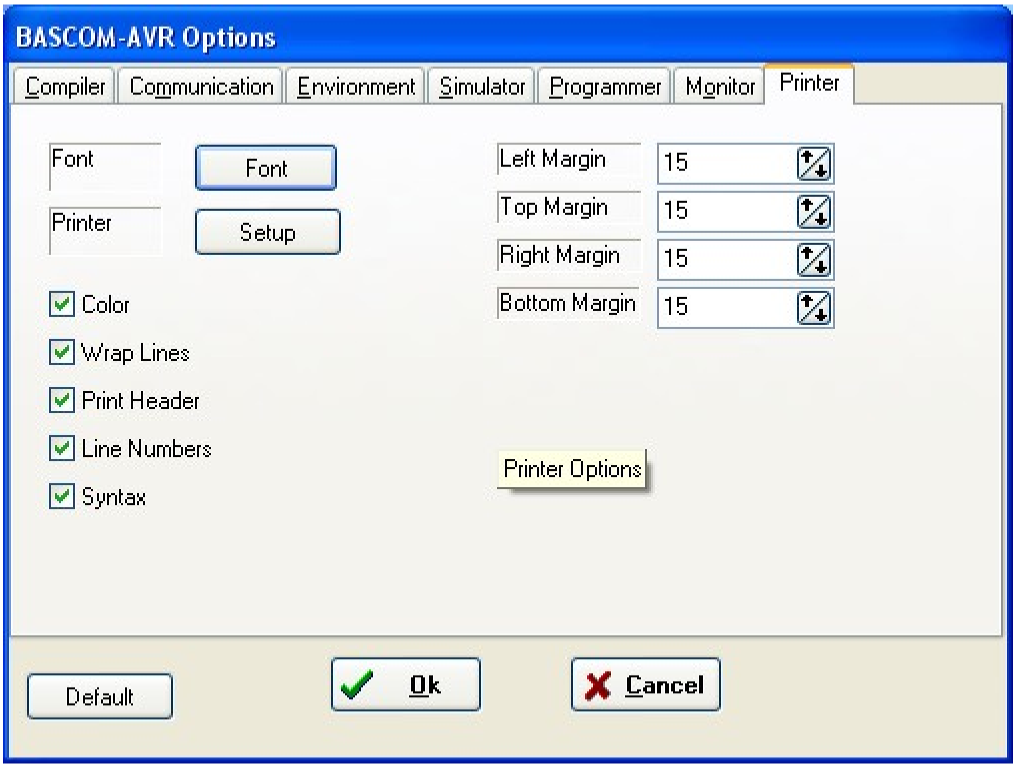
3.47 Options Monitor

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.

3.48 Options Printer

With this option you can modify the printer settings.



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 header	Print a header with the filename.
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.

3.49 Window Cascade

Cascade all open editor windows.

3.50 Window Tile

Tile all open editor windows.

3.51 Window Arrange Icons

Arrange the icons of the minimized editor windows.

3.52 Windows Maximize All

Maximize all open editor windows.

3.53 Window Minimize All

Minimize all open editor windows.

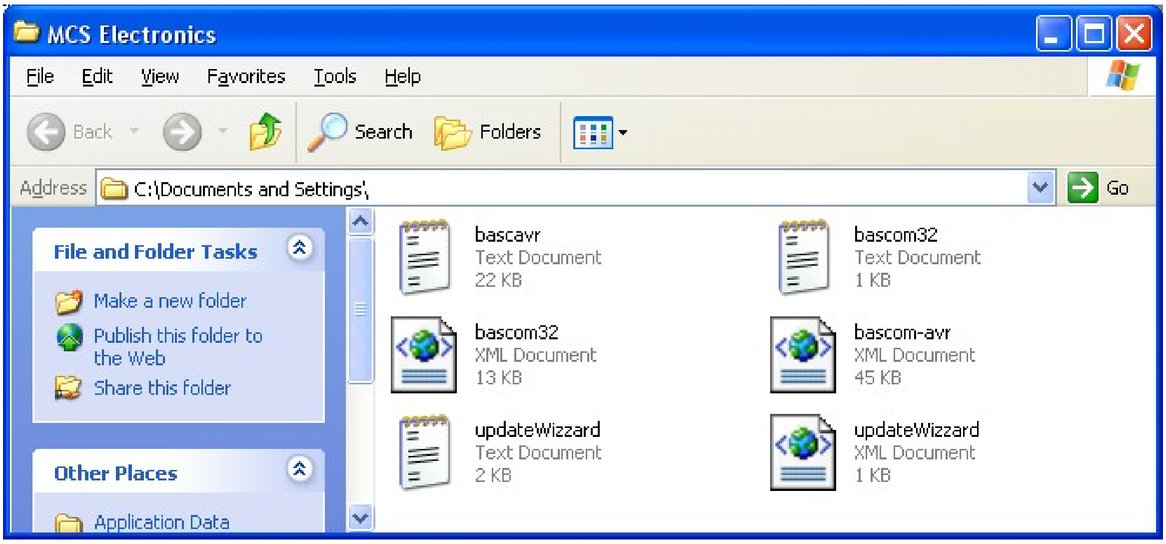
3.54 Help About

This option shows an about box as shown below.

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
Explorer :7.0.5730.11
Company :MCS
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.

3.55 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.

3.56 Help MCS Forum

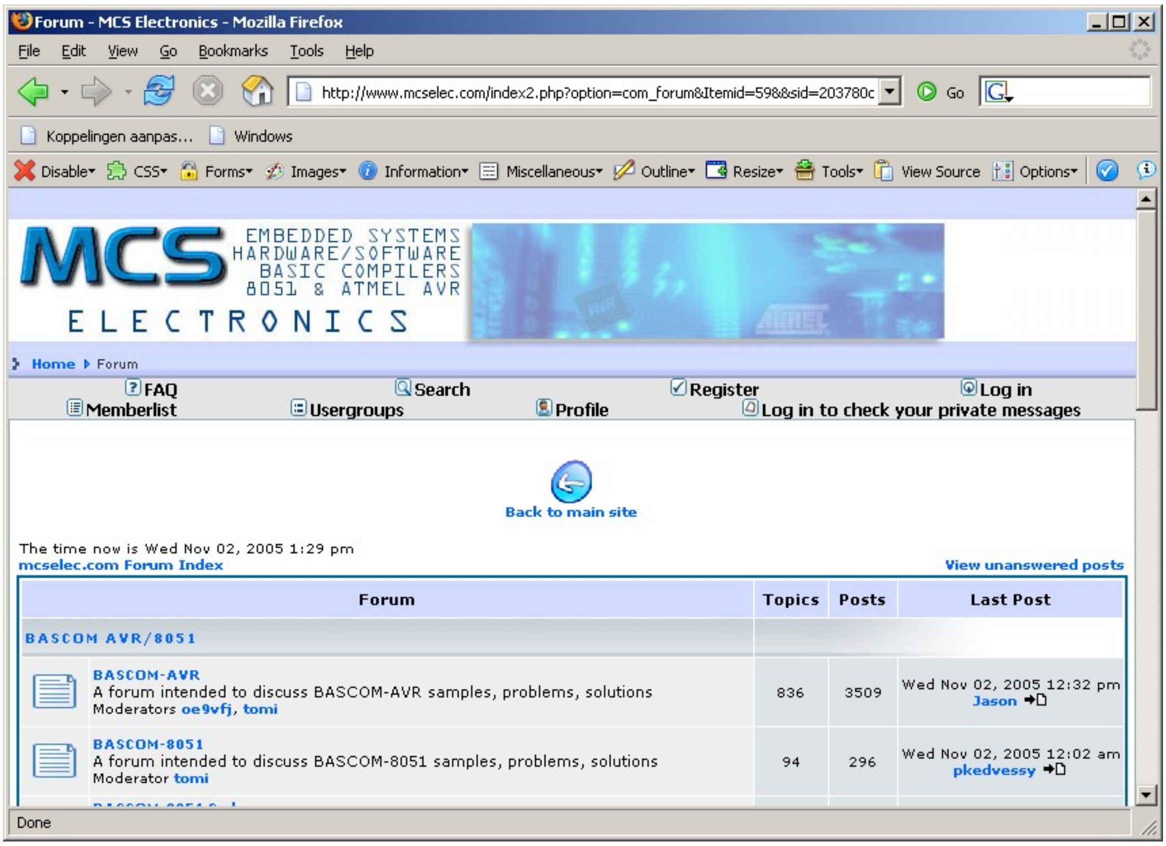
This option will start your default Web browser and direct it to http://www.mcselec.com/index2.php?option=com_forum&Itemid=59

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.

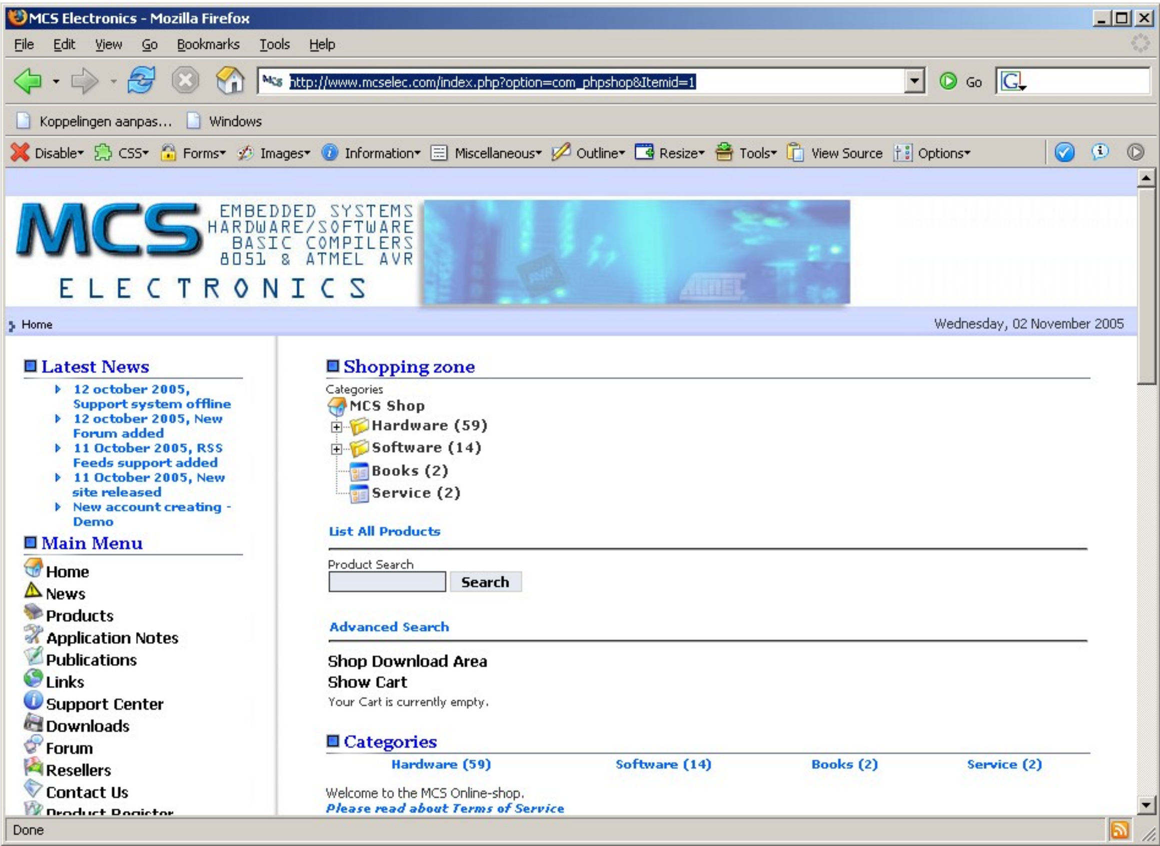


3.57 Help MCS Shop

This option will start your default web browser and direct it to :http://www.mcselec.com/index.php?option=com_phpshop&Itemid=1

You can order items and pay with PayPal. PayPal will accept most credit cards.

Before you order, it is best to check the resellers 824 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.



Before you can order items, you need to create an account.
Read the following about the new website : http://www.mcselec.com/index.php?option=com_content&task=view&id=133&Itemid=1

3.58 Help Support

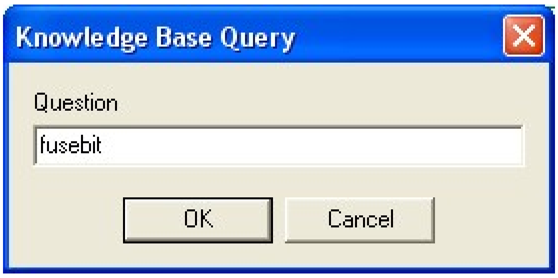
This option will start your default browser with the following URL :

http://www.mcselec.com/support-center/

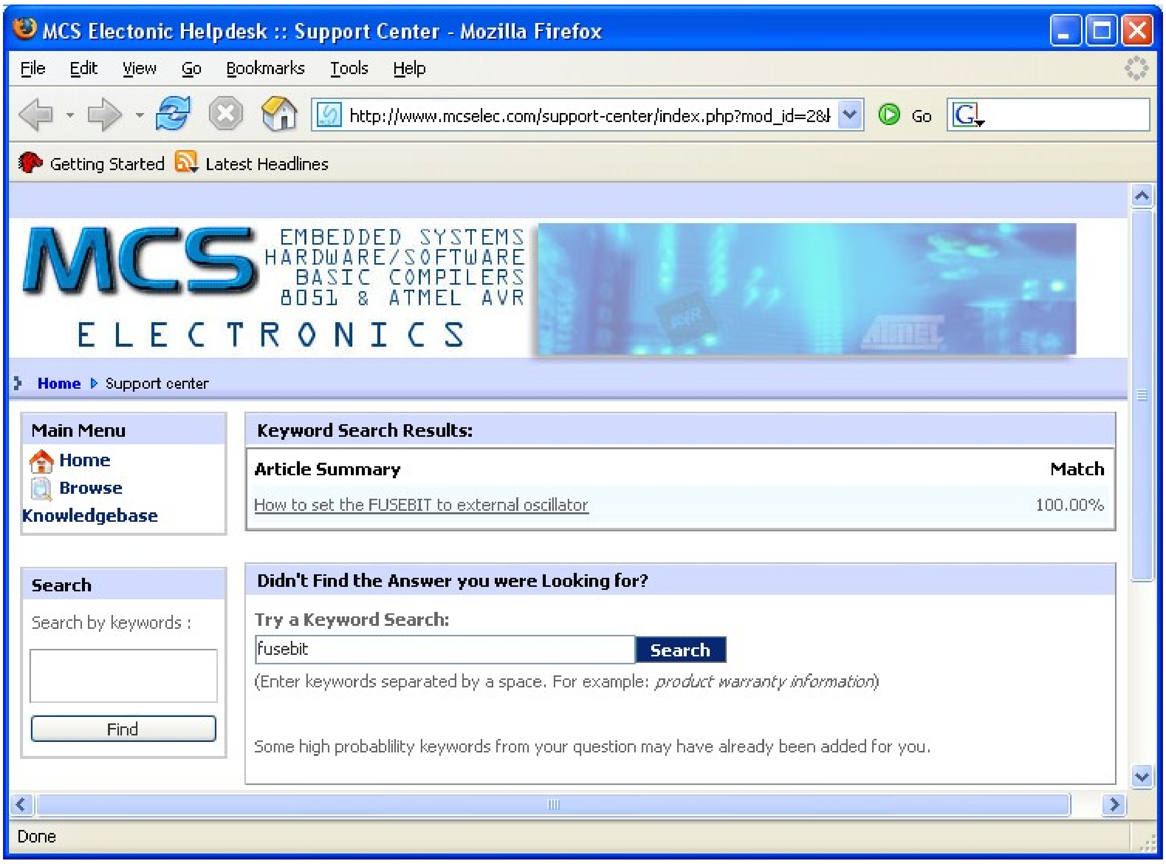
It depends from your browser settings if a new window or TAB will be created.
At the support site you can browse articles. You can also search on keywords.

3.59 Help Knowledge Base

This option will ask you to enter a search string.



This search string will be passed to the MCS support site.
The above example that searches for "FUSEBIT" will result in the following :



You can click one of the found articles to read it.

3.60 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.

3.61 BASCOM Editor Keys

Key	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 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
Select text	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.

3.62 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)

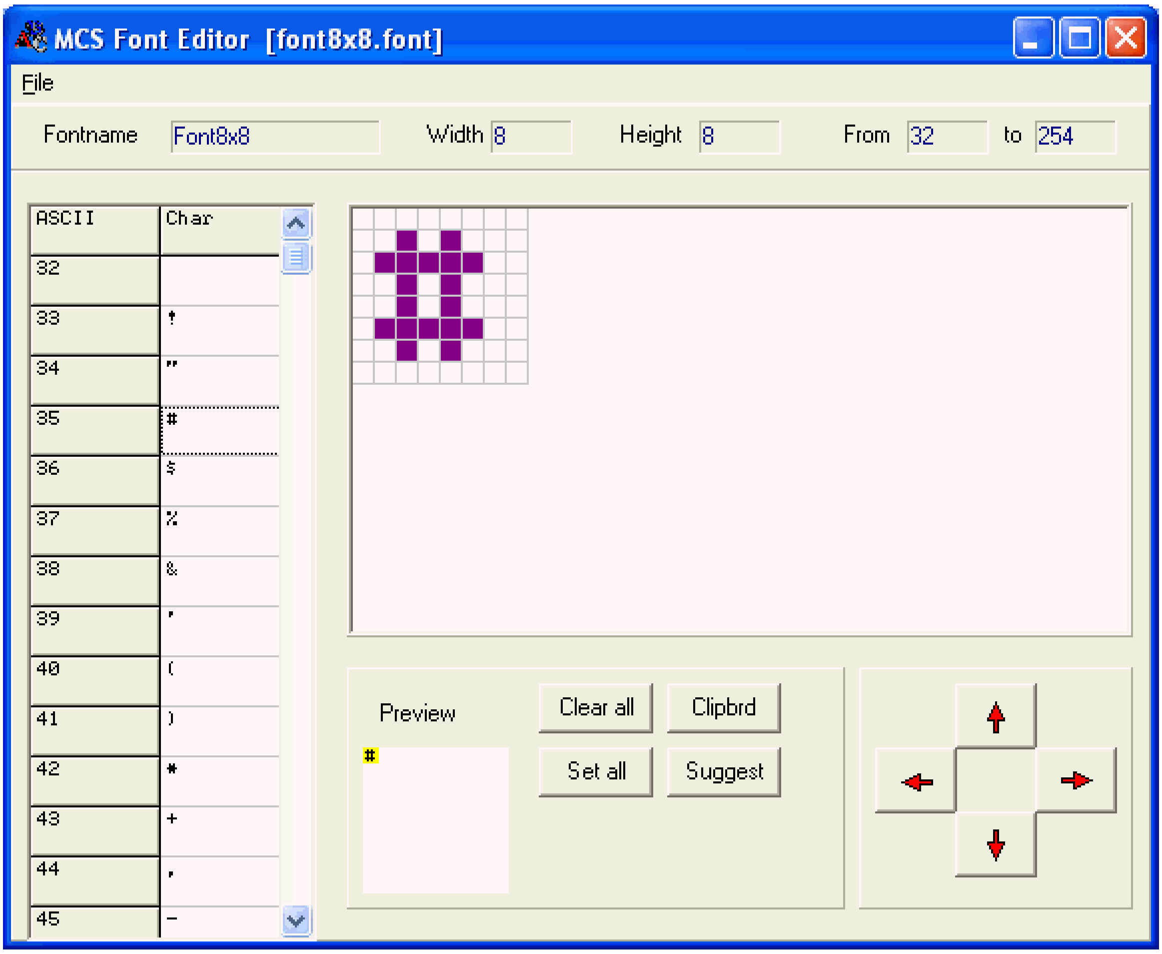
3.63 PlugIns

3.63.1 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:



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 SETFONT 723 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.

Part IV

4 BASCOM HARDWARE

4.1 Additional Hardware

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 talk about the chip.

The AVR internal hardware 129

Attaching an LCD display 139

Using the I2C protocol 150

Using the 1WIRE protocol 157

Using the SPI protocol 160

You can attach additional hardware to the ports of the microprocessor.
The following statements will then be able to be used:

I2CSEND 609 and I2CRECEIVE 608 and other I2C related statements.

CLS, 366 LCD, 629 DISPLAY 547 and other related LCD-statements.

1WRESET 314 , 1WWRITE 326 and 1WREAD 317

4.2 AVR Internal Hardware

The AVR chips all have internal hardware that can be used.

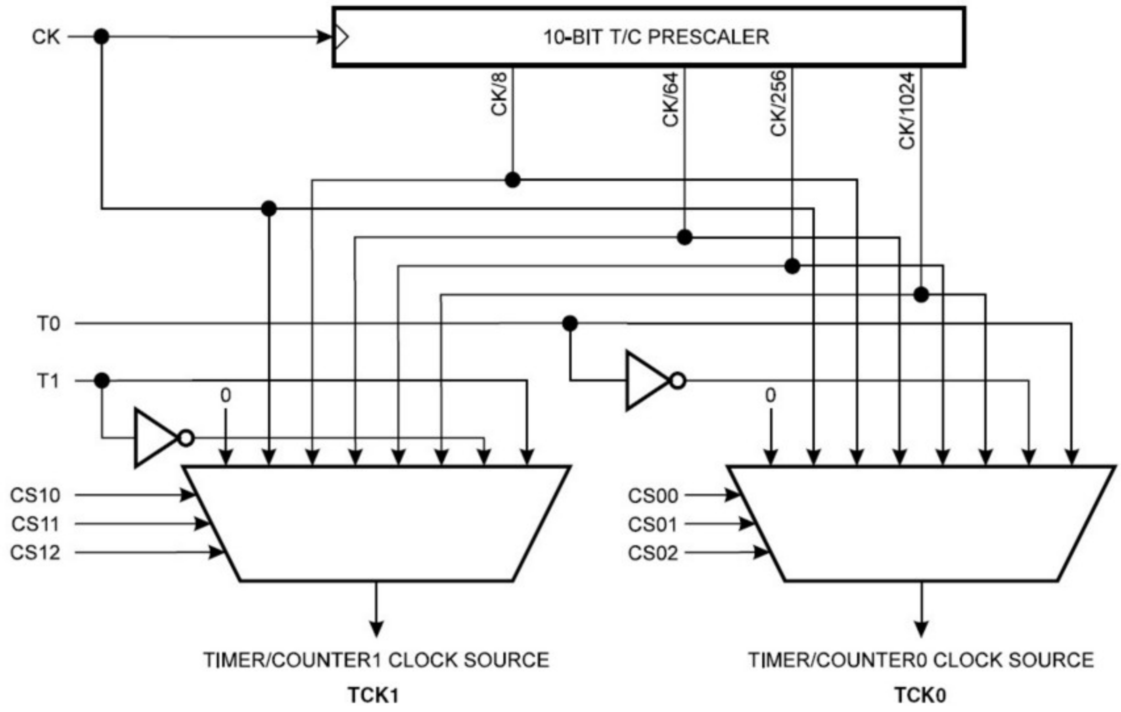
For this description of the hardware the 90S8515 was used. Newer chips like the Mega8515 may differ and have more or less internal hardware.

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.

Figure 28. Timer/Counter Prescaler



More about TIMER0 132

More about TIMER1 133

The WATCHDOG Timer 135

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 (XRAM 138). Since all ports are similar except that PORT B and PORT D have alternative functions, only these ports are described.

PORT B 135
PORT D 137

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

$3B	GIMSK INT1 INT0 - - - - - -
$3A	GIFR INTF1 INTF0
$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 COM1B0 - -PWM11 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	Reserved
$26	Reserved
$25	ICR1H Timer/Counter1 - Input Capture Register High Byte
$24	ICR1L Timer/Counter1 - Input Capture Register Low Byte
$23	Reserved
$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

$0E	SPSR SPIF WCOL - - - - - -
$0D	SPCR SPIE SPE DORD MSTR CPOL CPHA SPR1 SPR0
$0C	UDR UART I/O Data Register
$0B	USR RXC TXC UDRE FE OR - - -
$0A	UCR RXCIE TXCIE UDRIE RXEN TXEN CHR9 RXB8 TXB8
$09	UBRR UART Baud Rate Register
$08	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

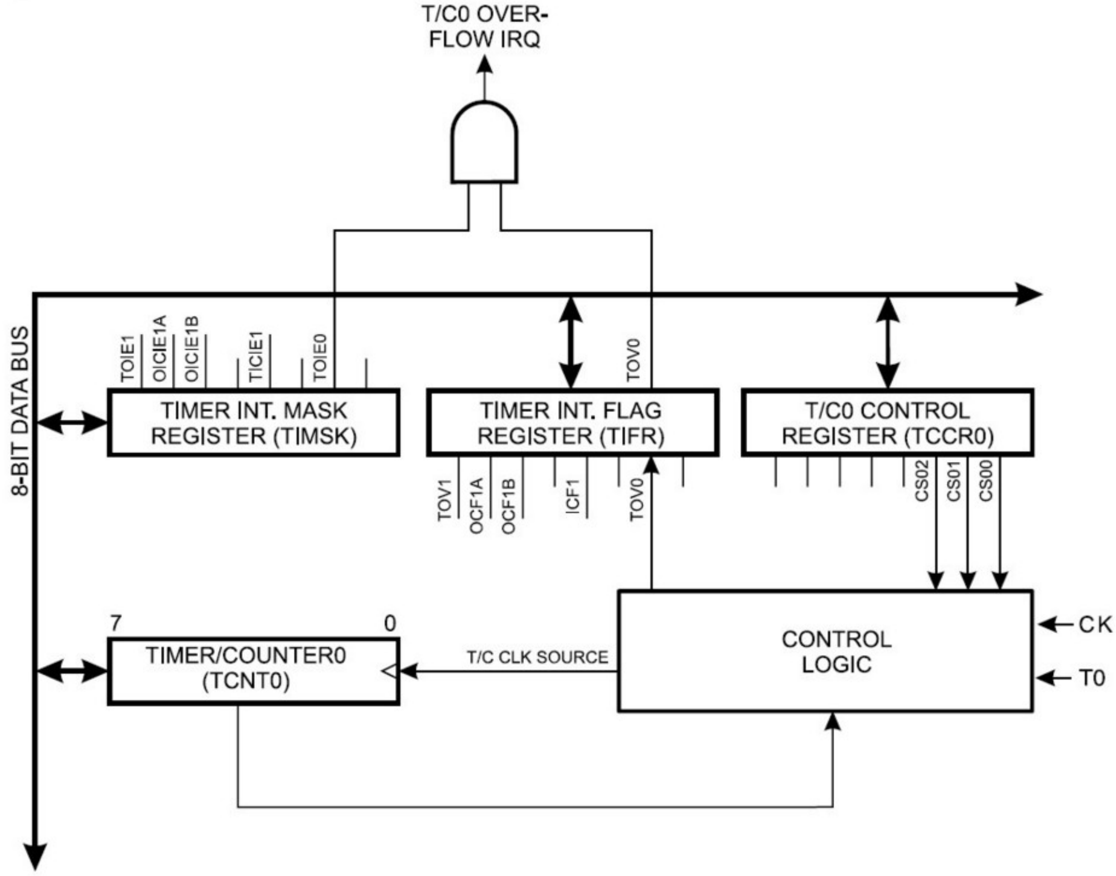
The 8-Bit Timer/Counter0

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.

Figure 29. Timer/Counter0 Block Diagram



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.

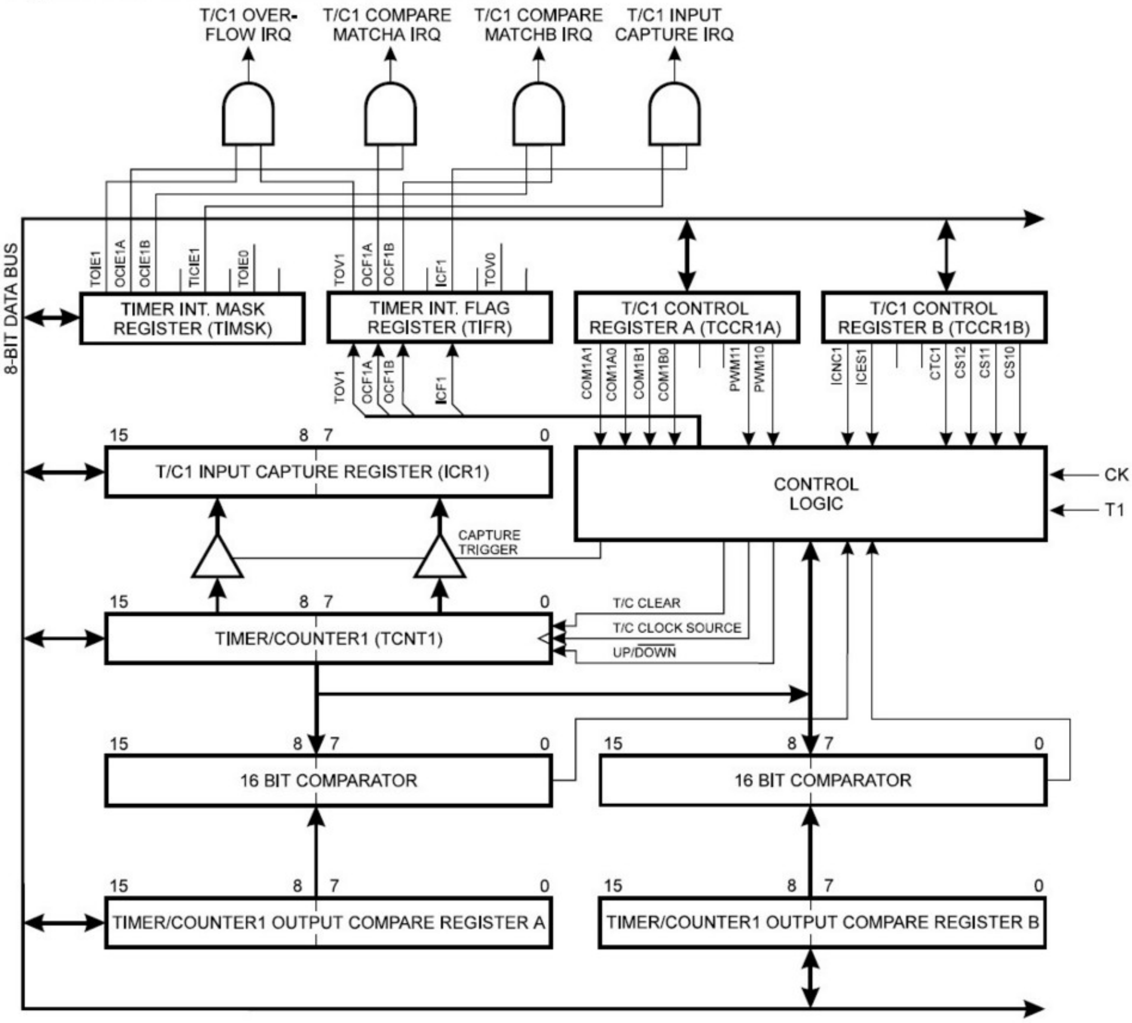
The Output Compare functions include optional clearing of the counter on compareA match, and can change the logic levels on the Output Compare pins on both compare matches.

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.

Figure 30. Timer/Counter1 Block Diagram



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 B Pins Alternate Functions

Port	**Pin**	**Alternate Functions**
PORTB.0	T0	(Timer/Counter 0 external counter input)
PORTB.1	T1	(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)

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 Effects on Port B Pins

DDBn	**PORTBn**	**I/O**	**Pull up**	**Comment**
0	0	Input	No	Tri-state (Hi-Z)

0	1	Input	Yes	PBn will source current if ext. pulled low.
1	0	Output	No	Push-Pull Zero Output
1	1	Output	No	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	INT0	(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)

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

INT0 - PORTD, Bit 2
INT0, 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).

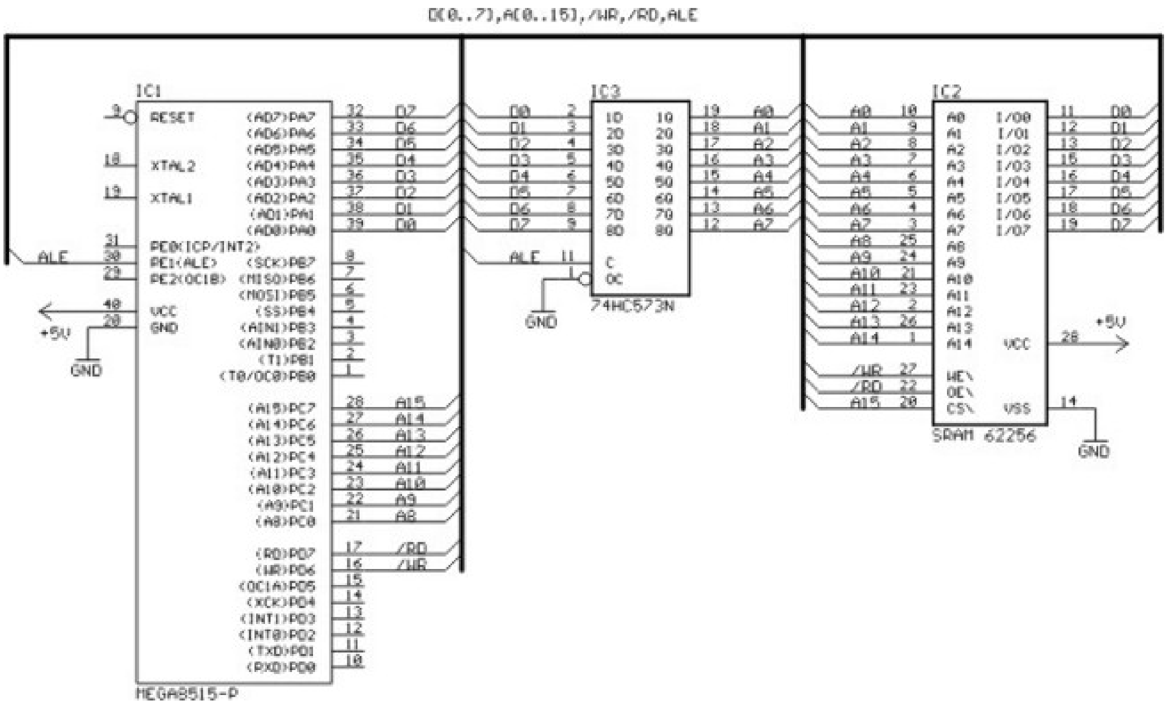
Here is some info from the BASCOM user list :

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.

The LCD-display can be connected in PIN mode as follows:

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

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 91 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.

```
$ASM
  Ldi _temp1, 5         'load register R24 with value
  Rcall _Lcd_control    'it is a control value to control the display
  Ldi _temp1,65         '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. This memory is also called SRAM.

The available memory depends on the 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.

Each 8 bits used occupy one byte.
Each byte variable occupies one byte.
Each integer/word variable occupies two bytes.
Each Long or Single variable occupies four bytes.
Each double variable occupies 8 bytes.
Each string variable occupies at least 2 byes.
A string with a length of 10. occupies 11 byes. The extra byte is needed to indicate 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, uses 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() etc. that convert from numeric to string and vice versa, you also need a frame. It should be 16 bytes in this case.
Add additional space for the local data.

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 16 bytes of frame space. This because these routines use the internal numeric<>string conversion routines.

XRAM

You can easy add external memory to an 8515. Then XRAM (extended memory) will become available. When you add a 32 KB RAM, the first address will be 0.
But because the XRAM can only start after the internal SRAM, which is &H0260 for the 8515, the lower memory locations of the XRAM will not be available for use.

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 needed, and never use it in a loop or the ERAM will become unusable.

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.

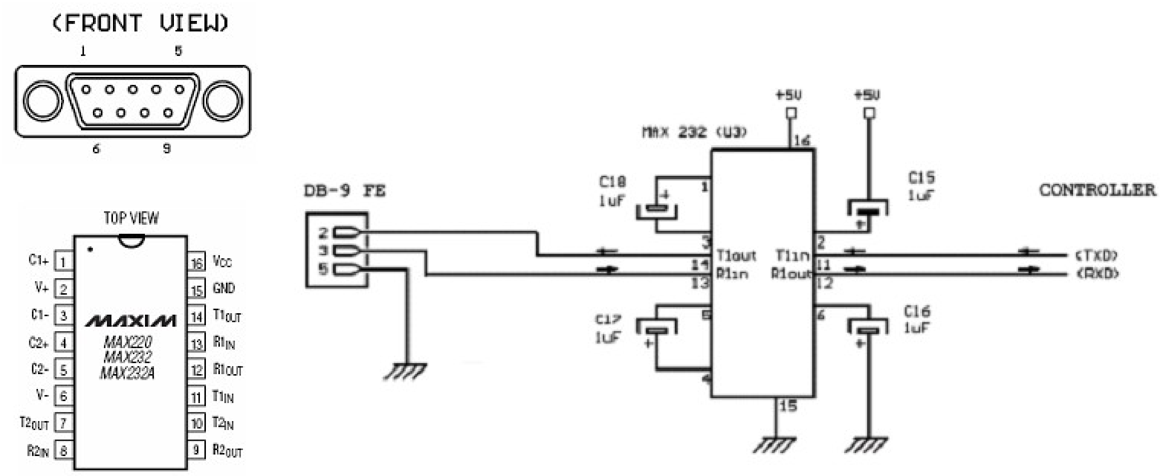
4.12 Using the UART

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 and add the following code in the beginning of your program.

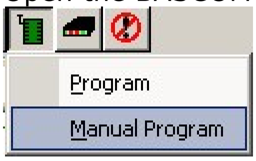
```
$regfile = "your   micro   here   def.dat"
$crystal =  8000000
$baud =  19200
```

Make sure to define your micro controller after $regfile for example if you use the ATMega32

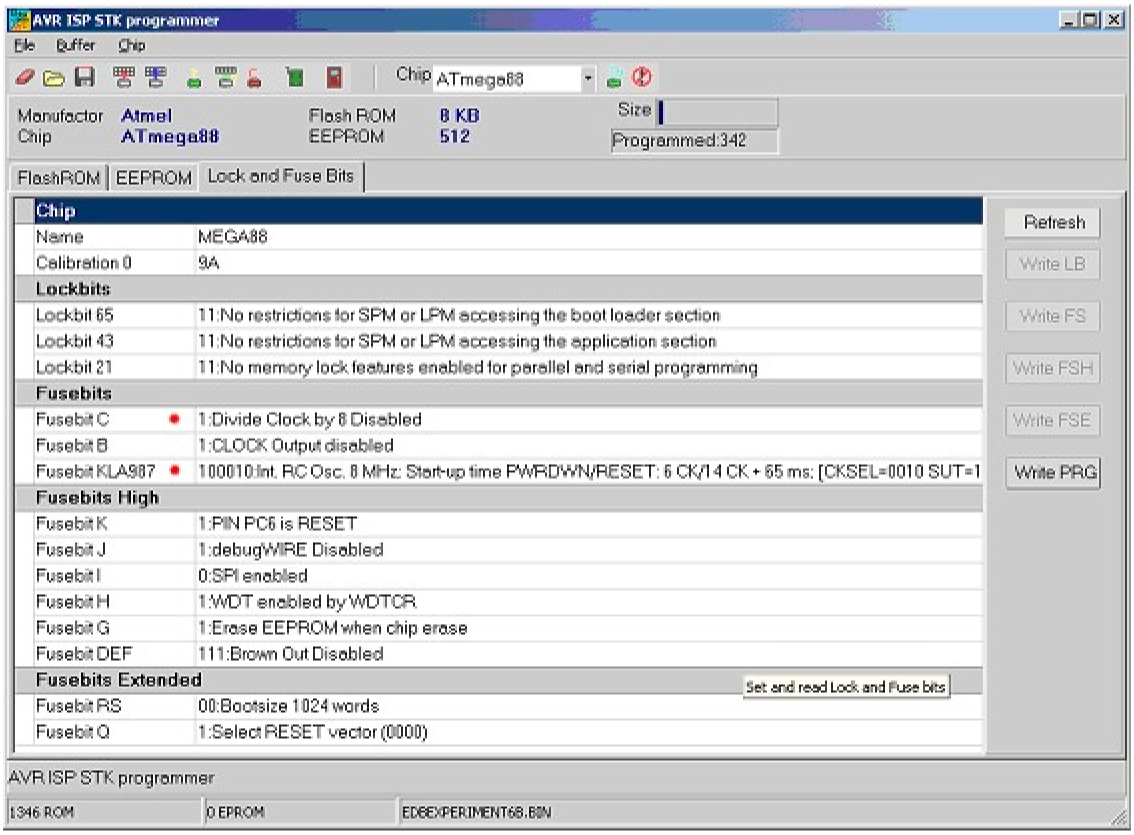
```
$regfile = "m32def.dat"
```

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]"



These options are not available for all AVR's, if you don't have the option do not change any fuse bits.

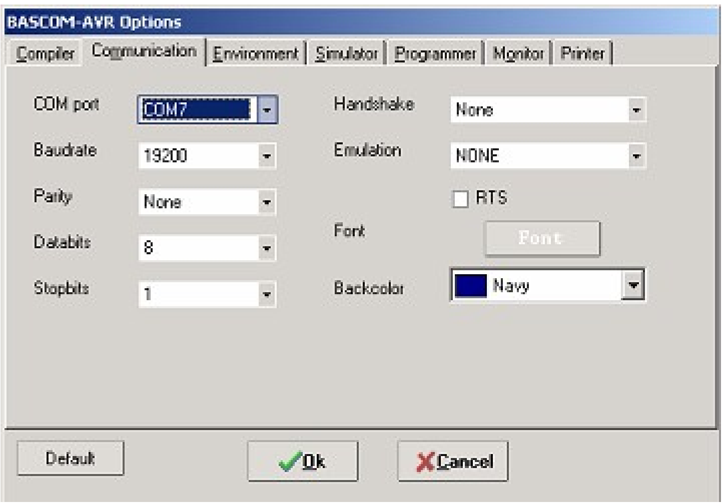
If these options are available, but in a wrong setting. Change the setting in the drop down box and click another Fuse section. Finally click the "Program FS" button. Click "Refresh" to see the actual setting.

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:

```
$regfile = "m32def.dat"   'Define   your   own
$crystal = 8000000
$baud = 19200
```

```
Do
   Print "Hello    World"
  Waitms 25
Loop

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.



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

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..."
Print     'Notice what this line does
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    008   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        B
067     043    01000011        C
```

You can find a complete ASCII table here 253

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.

Ischarwaiting( )

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, read it to a variable.

```
'Print "Press B key to start"
Dim Serialcharwaiting As Byte, Serialchar As Byte

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
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" 444.
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
```

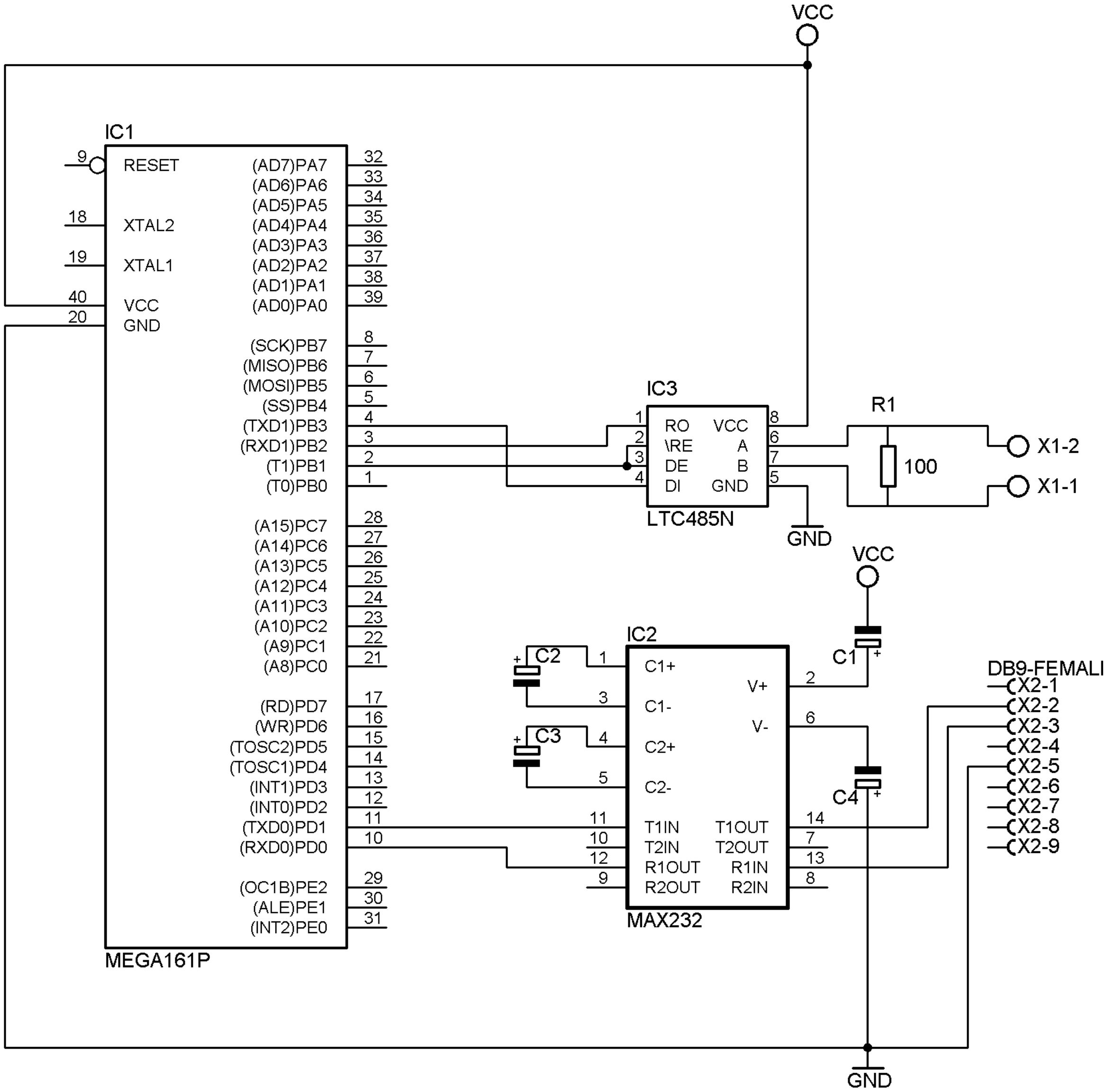
```
'To use it with the software UART you must provide the channel
Do
   'Store in byte
   B = Inkey(#2)
   'When the value > 0 we got something
   If B > 0 Then
      Print #1 , Chr(b)                          'Print the character
   End If
Loop
Close #2                                          'Close 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"                          ' specify the used micro
$crystal = 8000000
$baud = 19200                                     ' use baud rate
$hwstack = 42                                     ' 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

$lib "modbus.lbx"
Config Print1 = Portb.1 , Mode = Set              ' 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 = 0
Config 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 = &H12345678


Print "RS-485 MODBUS master"
Do
   If Pinc.0 = 0 Then                                    ' test button
      Waitms 500                                        ' delay since we want to send just 1
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

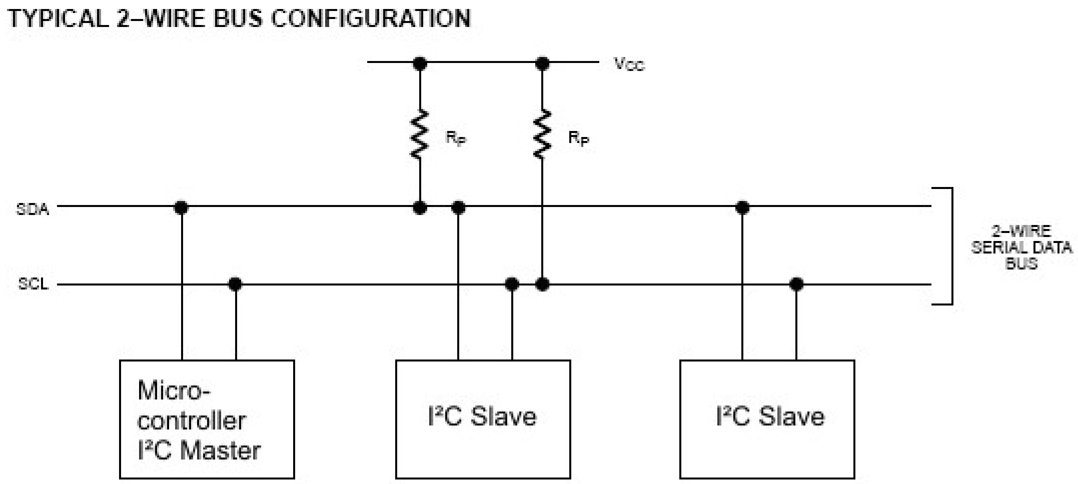
I²C bus

I²C 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 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.

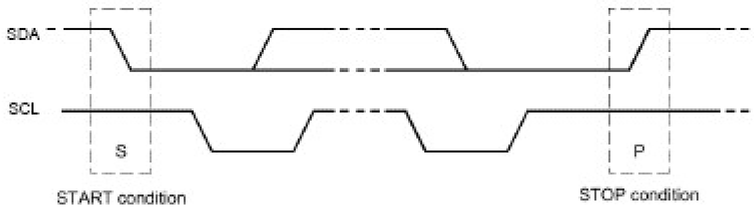
A typical hardware configuration is shown in the figure below:



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).

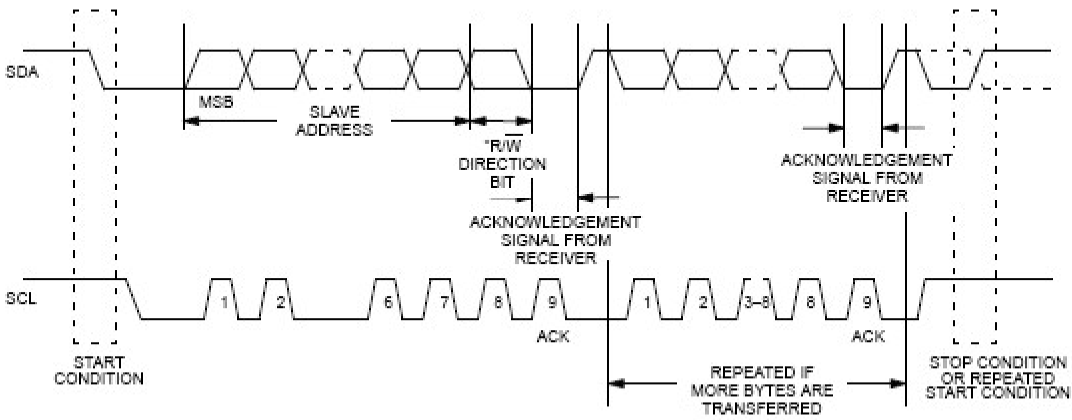
In most applications the micro controller is the I²C Master. Slave chips can be Real Time Clocks and Temperature sensors. For example the DS1307 and the DS1624 from www.maxim-ic.com. Of coarse you can also create your own slaves. In that case there is micro controller to micro controller communication.

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.

DATA TRANSFER ON 2–WIRE SERIAL BUS



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.

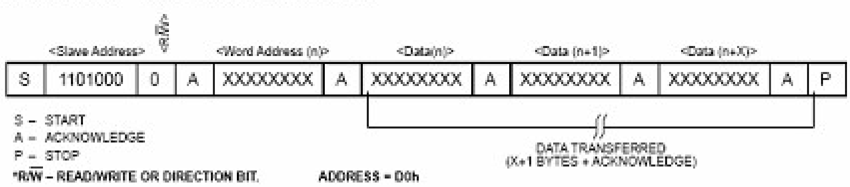
In other words if the master wishes to stop receiving, it sends a not acknowledge after the last received byte.

The master can stop any communication on the bus **at any time** by sending a stop condition.

BUS ADRESSING

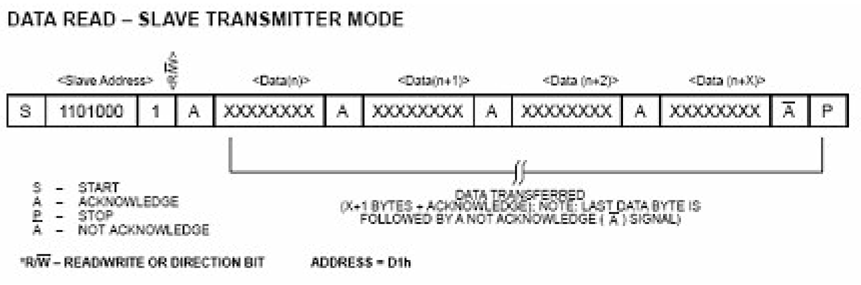
Let’s say we have a slave chip with the address “1101000” and that the master wishes to write to that slave, the slave would then be in receiver mode, like this:

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.

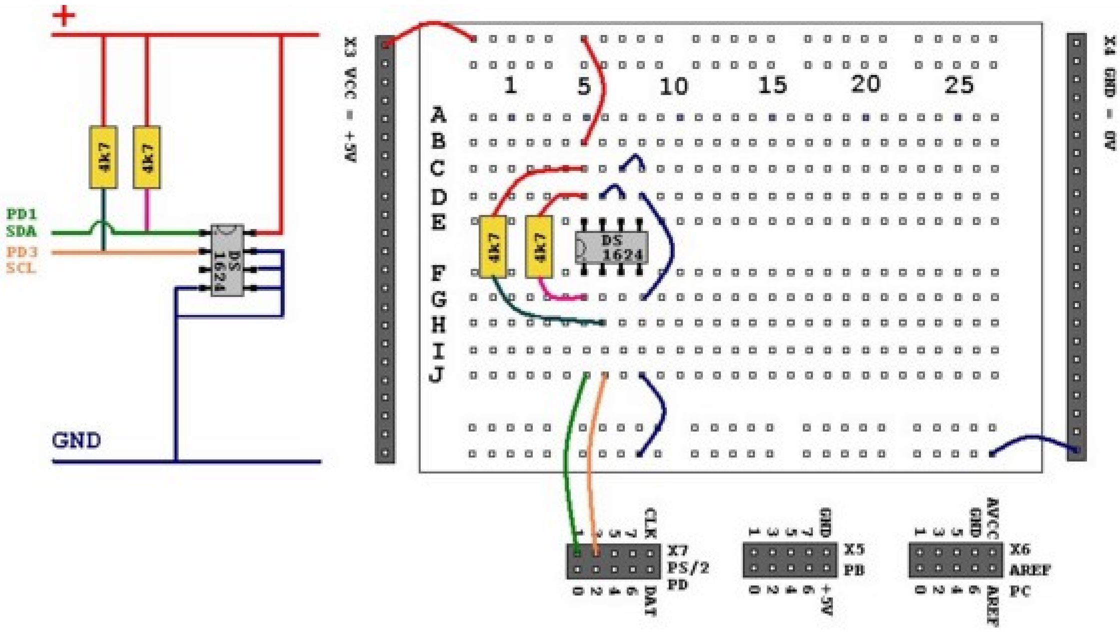
After that the master can send the data desired and stop the transfer with a stop condition.



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.

```
$regfile = "m88def.dat"                  'Define the chip you use
$crystal = 8000000                       'Define speed
$baud = 19200                            '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 I²C 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
   I2cstart                                'Sends start condition
   I2cwbyte Ds1624wr                       'Sends the address

'byte with r/w 0

'Access the CONFIG register (&HAC address byte)
   I2cwbyte &HAC
'Set continuous conversion  (&H00 command byte)
   I2cwbyte &H00
   I2cstop                                 'Sends stop condition
   Waitms 25         'We have to wait some time after a stop

   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
```

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.

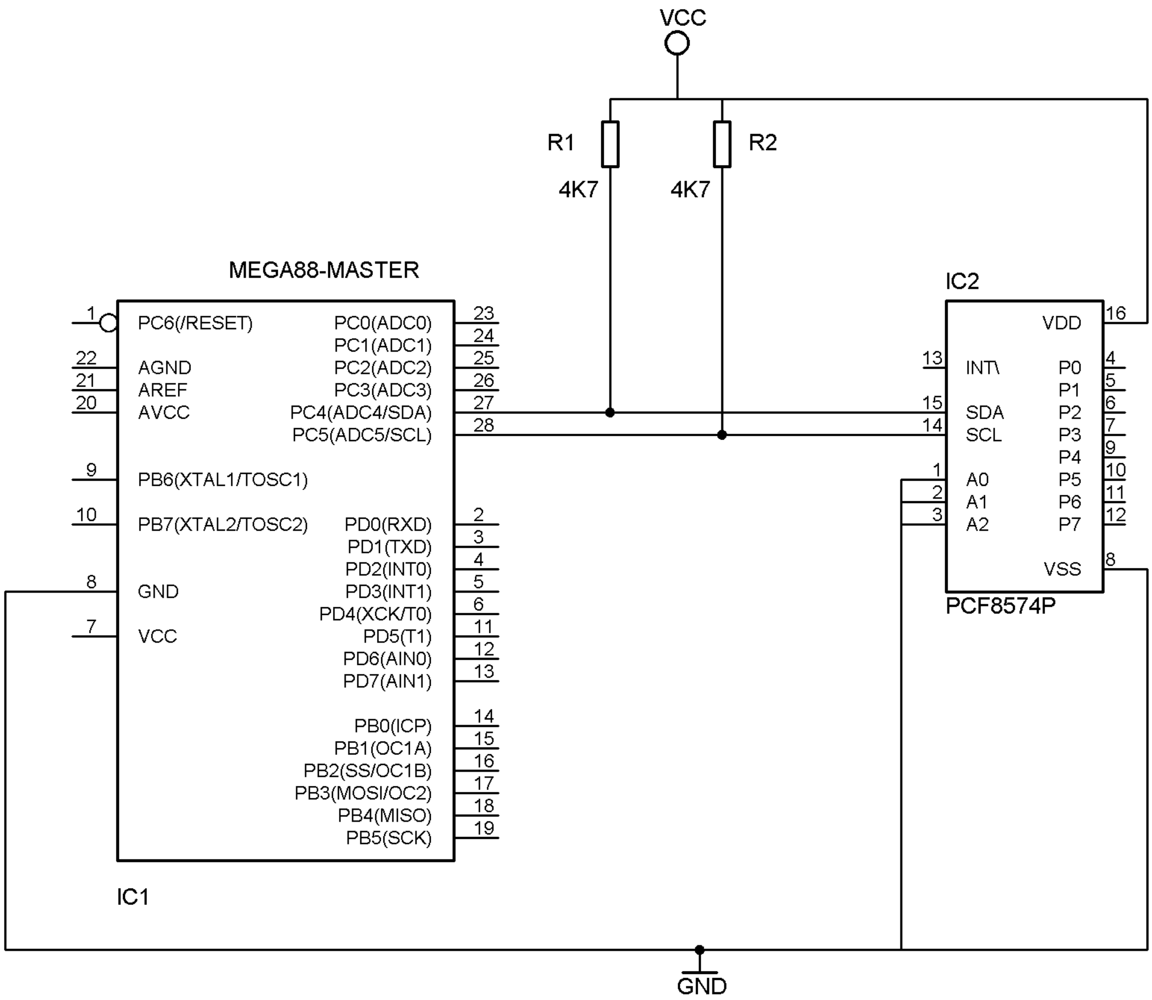
I2creceive
Reads a number of bytes from an I²Cslave.

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.

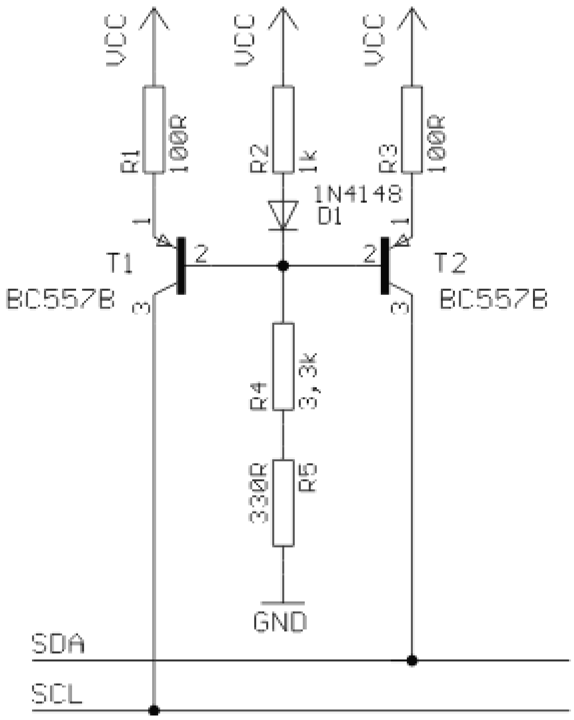
There are many I2C slave chips available. The example shows the PCF8574. With the additional TWI slave library you can make your own slave chips.



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 1-wire 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 time-slots 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 self-explanatory. 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 ^ 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.

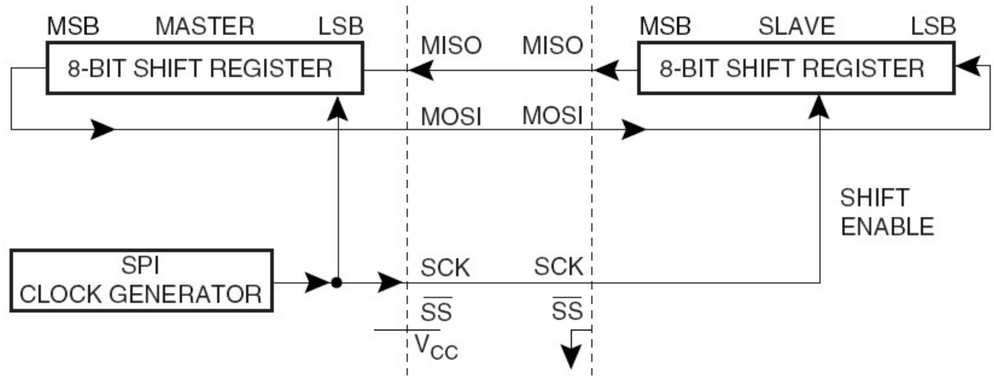
The mode the AVR is running in is specified by the settings of the master bit (MSTR) in the SPI control register (SPCR).

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.

Figure 61. SPI Master-Slave Interconnection



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.

Table 1. SPI Pin Overrides

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

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.

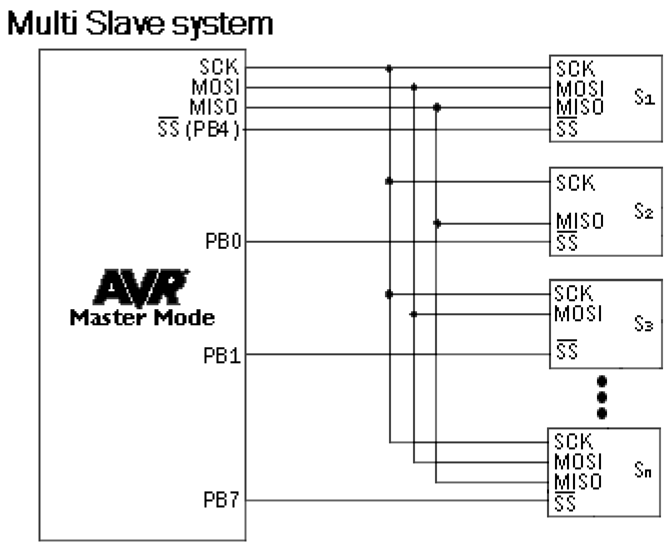
Table 2 shows an overview of the /SS Pin Functionality.

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.

TABLE 2. Overview of SS pin.

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	

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.



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.

Table 3. SPI Mode configuration

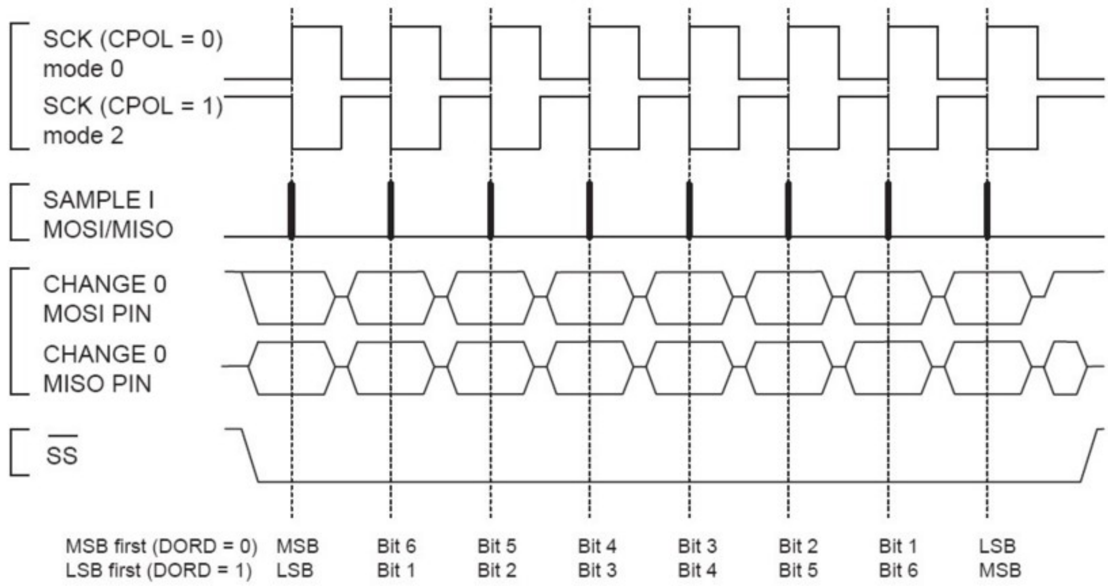
SPI Mode	CPOL	CPHA	Shift SCK edge	Capture SCK edge
0	0	0	Falling	Rising
1	0	1	Rising	Falling
2	1	0	Rising	Falling
3	1	1	Falling	Rising

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.

Figure 62. SPI Transfer Format with CPHA = 0



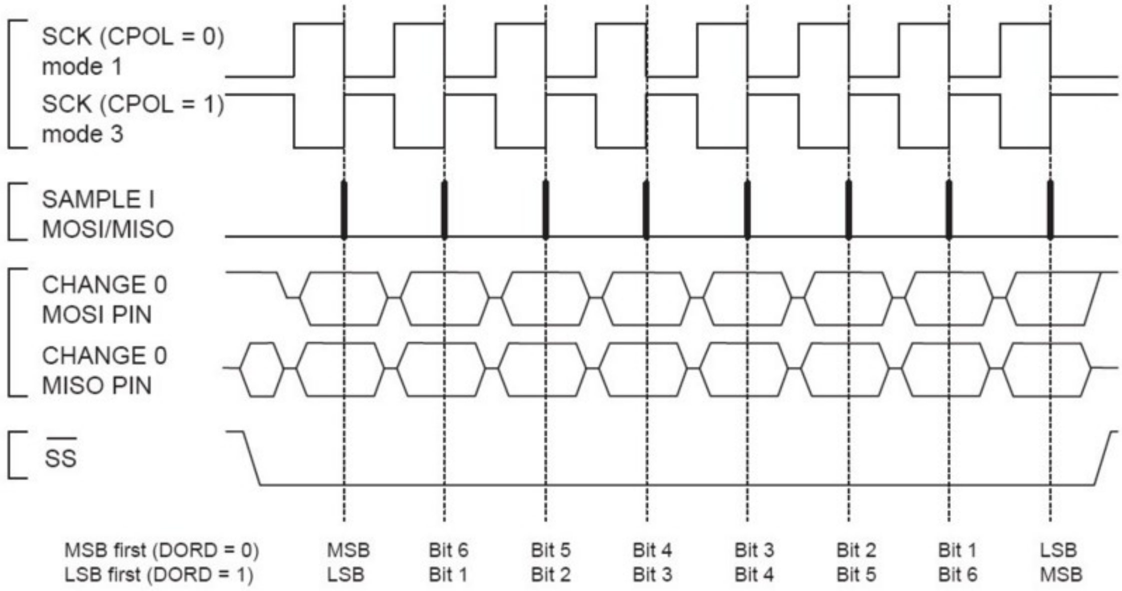
When the SPI is configured as a slave, the transmission starts with the falling edge of the /SS line. This activates the SPI of the slave and the MSB of the byte stored in its data register (SPDR) is output on the MISO line. The actual transfer is started by a software write to the SPDR of the master. 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.

Figure 63. SPI Transfer Format with CPHA = 1



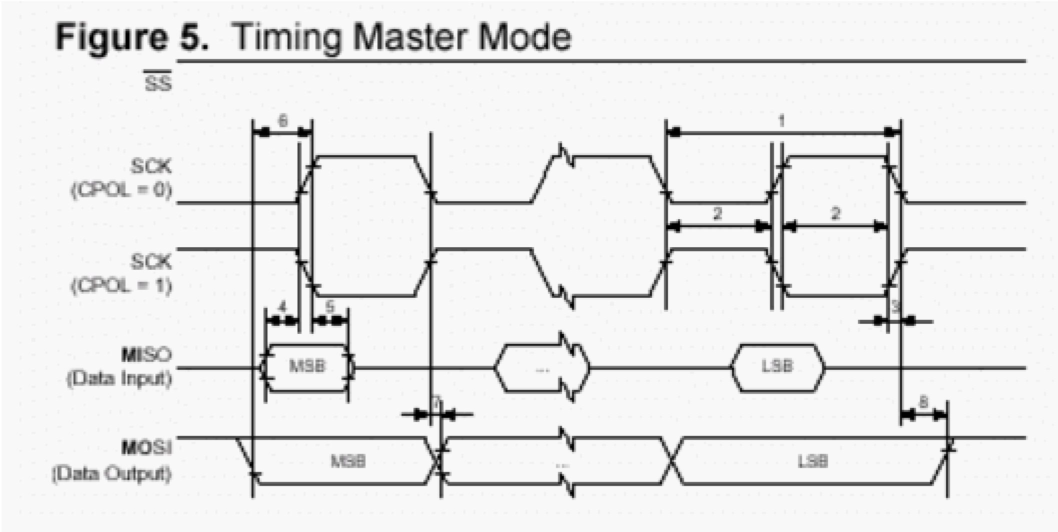
Like in the previous cases the falling edge of the /SS lines selects and activates the slave. Compared to the previous cases, where CPHA equals zero, the transmission is not started and the MSB is not output by the slave at this stage. The actual transfer is started by a software write to the SPDR of 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.



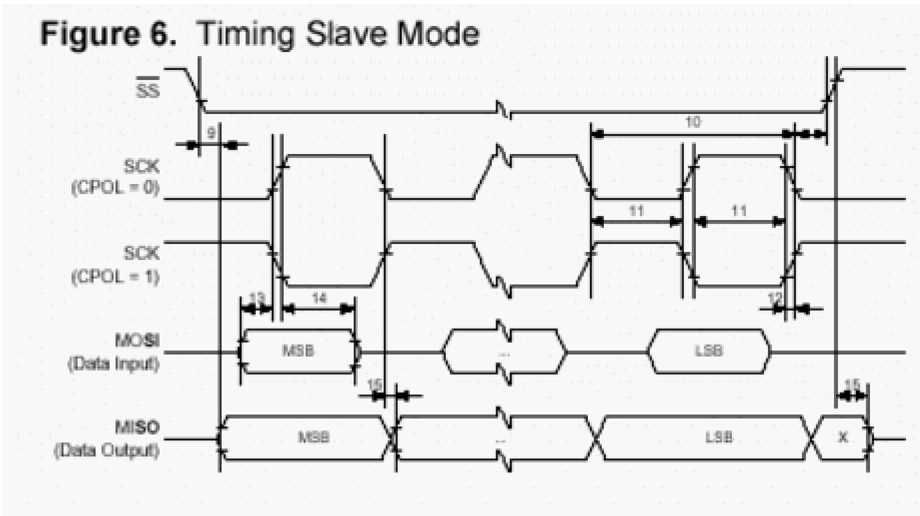
The minimum timing of the clock signal is given by the times "1" and "2". The value "1" specifies the SCK period while the value "2" specifies the high / low times of the clock signal. The maximum rise and fall time of the SCK signal is specified by the time "3". These are the first timings of the AVR 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 CONFIG SPI 453 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

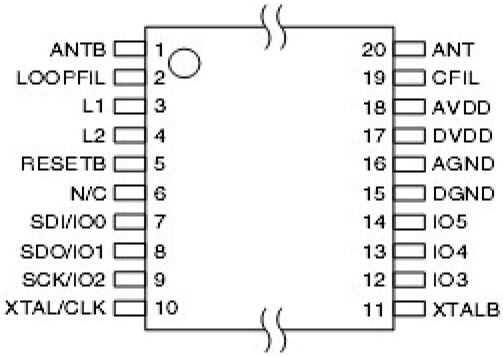
The internal RAM is cleared at power up or when a reset occurs. Use $NORAMCLEAR to disable this feature.

You may use $INITMICRO 274 to set a port level and direction immediately on startup.

4.18 Chips

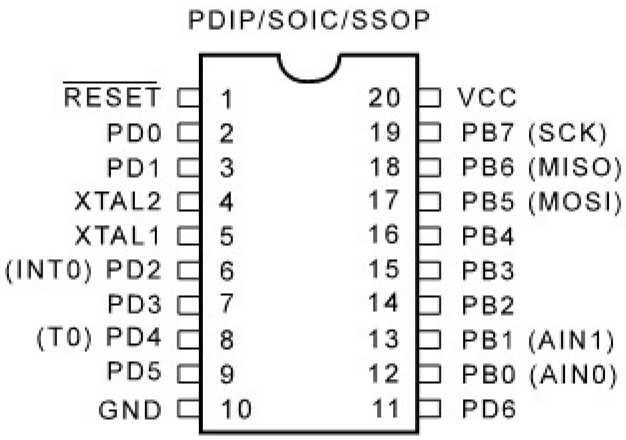
4.18.1 AT86RF401

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.



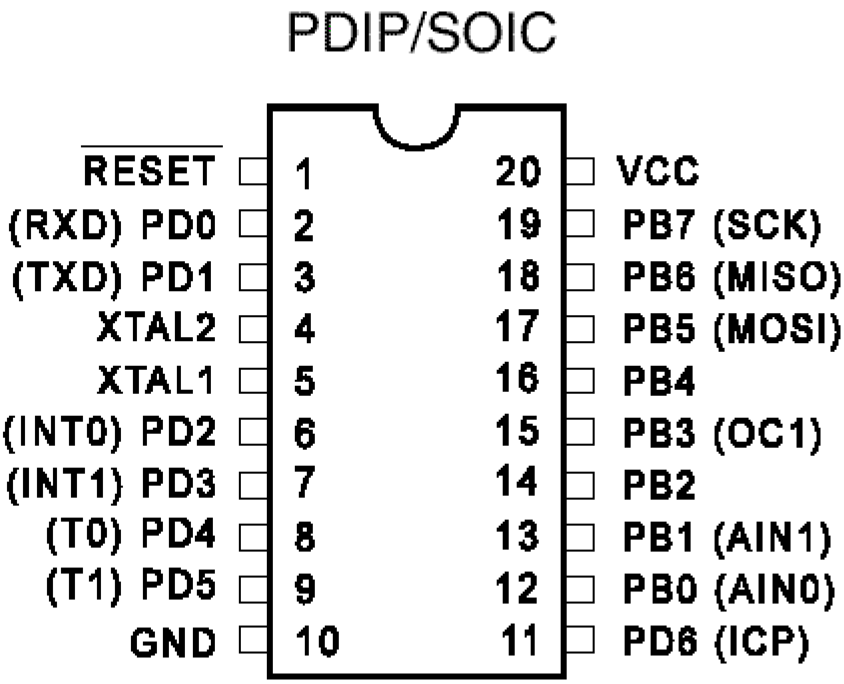
4.18.2 AT90S1200

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

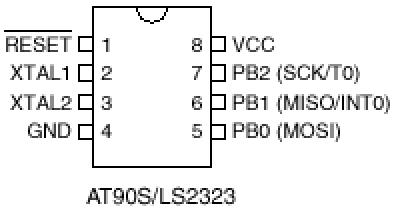
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The ATTiny2313 should be used for new designs.

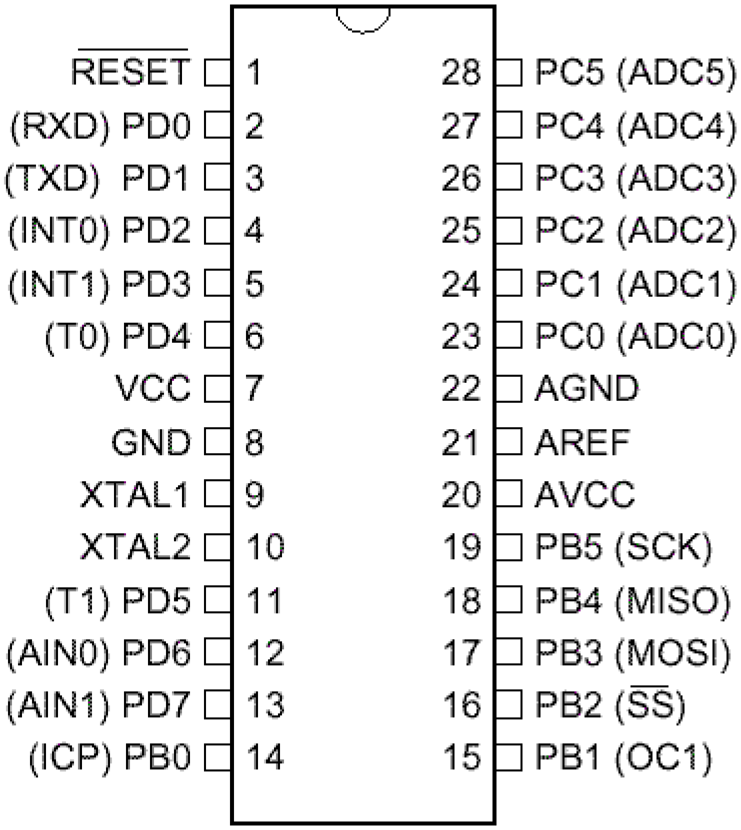
4.18.4 AT90S2323

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.



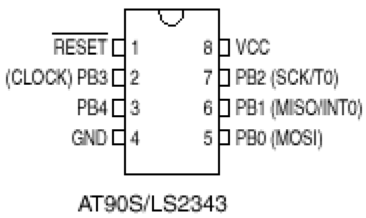
4.18.5 AT90S2333

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.

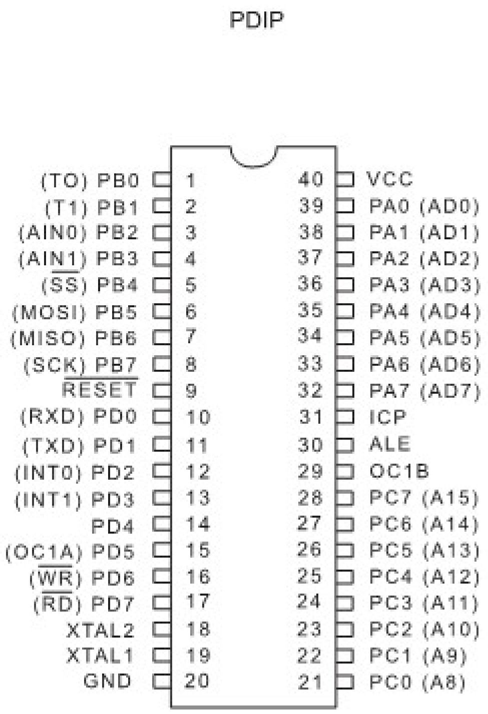
The current ATMEL data sheet specifies that all versions –1, -4 and –10 are supplied with a fuse bit set for the internal clock that operates at approximately 1Mhz. If using the internal clock make sure to enter 1000000 under Options\Compiler\Communication\frequency.

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.

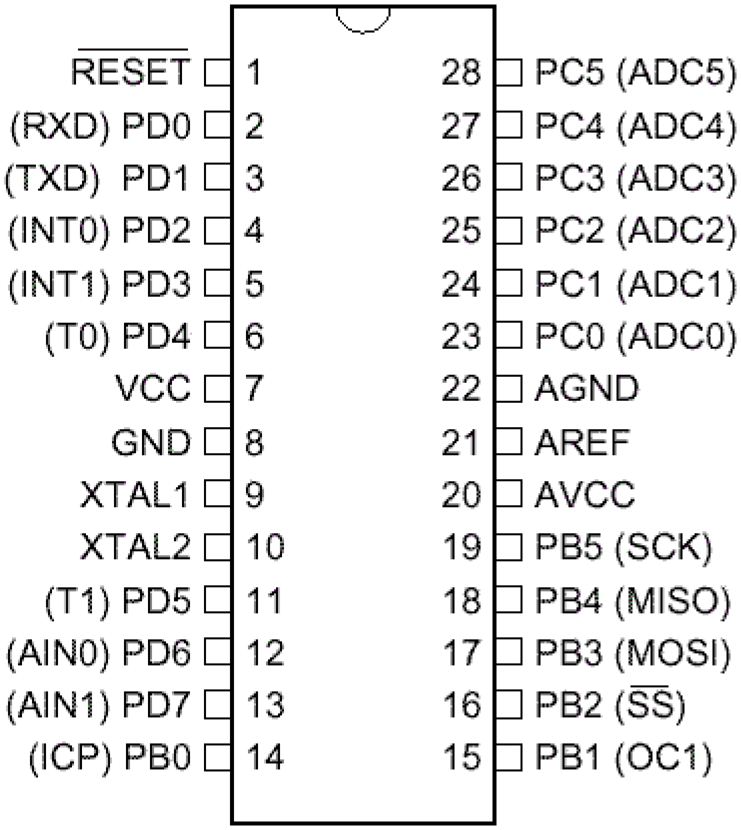
4.18.7 AT90S4414

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.



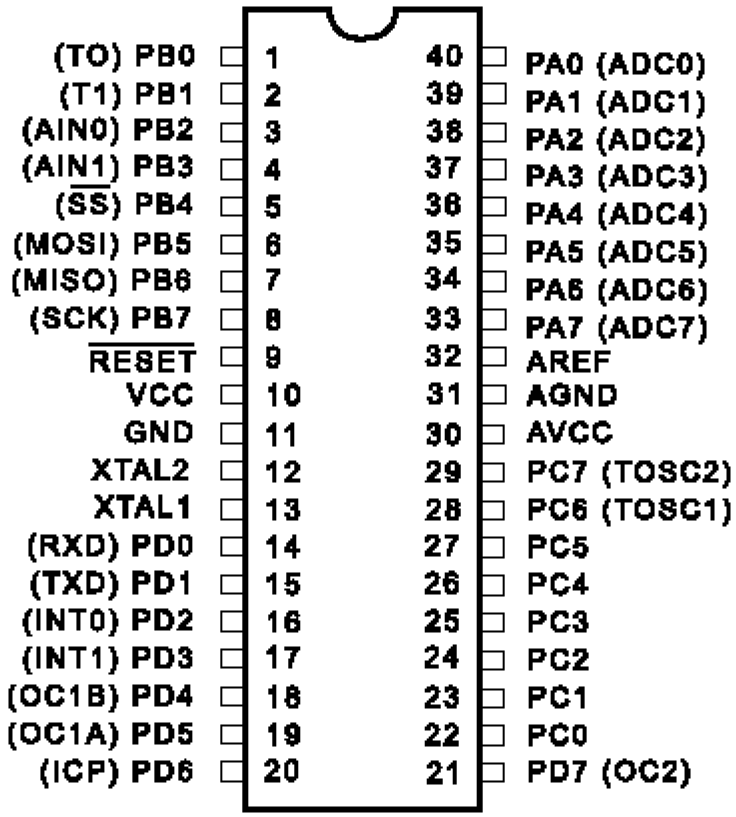
4.18.8 AT90S4433

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.


RESET 1
(RXD) PD0 2
(TXD) PD1 3
(INT0) PD2 4
(INT1) PD3 5
(T0) PD4 6
VCC 7
GND 8
XTAL1 9
XTAL2 10
(T1) PD5 11
(AIN0) PD6 12
(AIN1) PD7 13
(ICP) PB0 14
28 PC5 (ADC5)
27 PC4 (ADC4)
26 PC3 (ADC3)
25 PC2 (ADC2)
24 PC1 (ADC1)
23 PC0 (ADC0)
22 AGND
21 AREF
20 AVCC
19 PB5 (SCK)
18 PB4 (MISO)
17 PB3 (MOSI)
16 PB2 (SS)
15 PB1 (OC1)

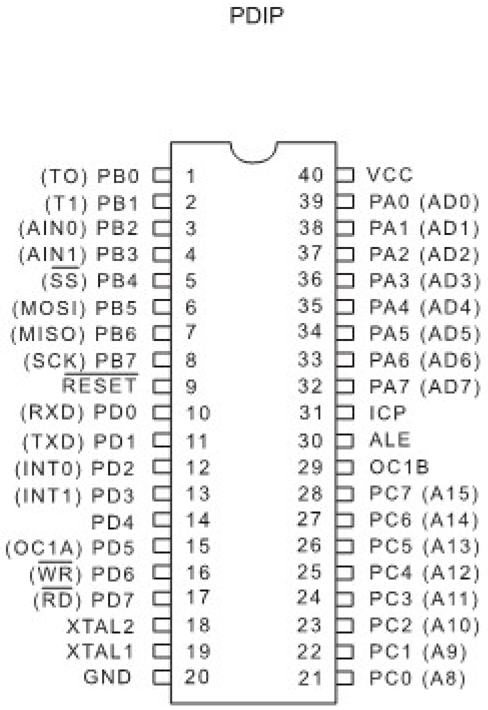
4.18.9 AT90S4434

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.



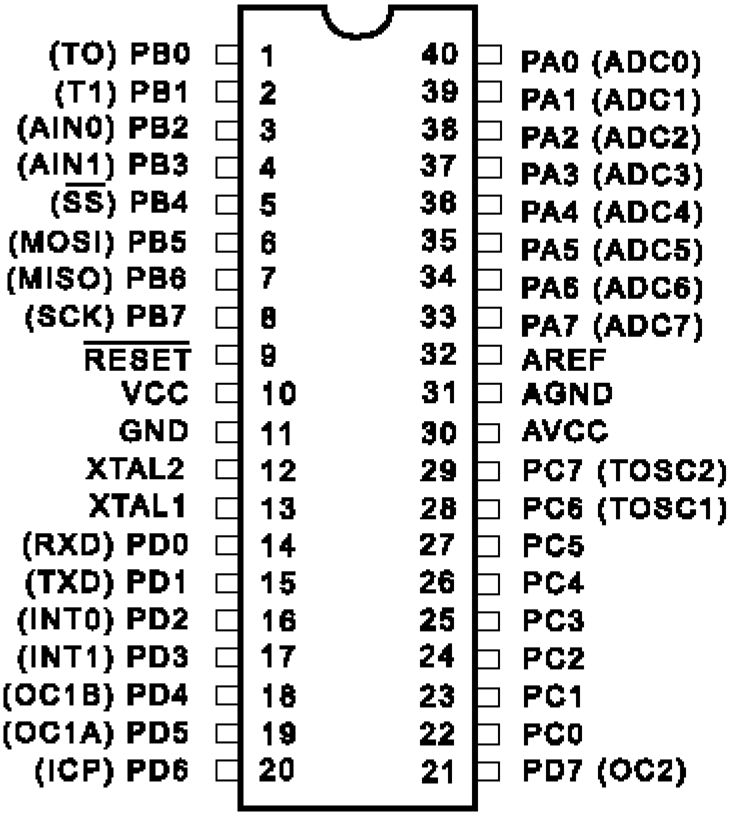
4.18.10 AT90S8515

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.



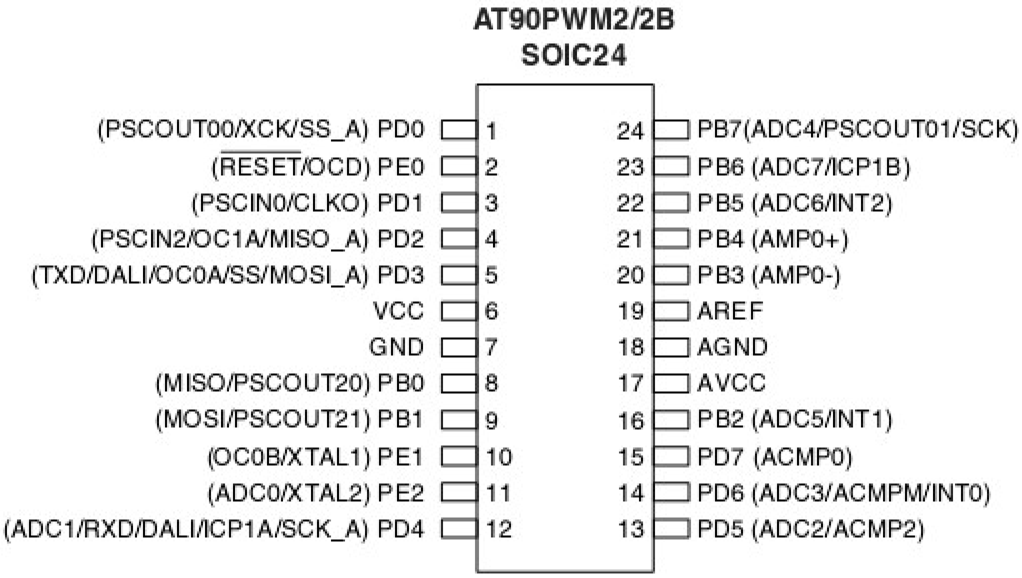
4.18.11 AT90S8535

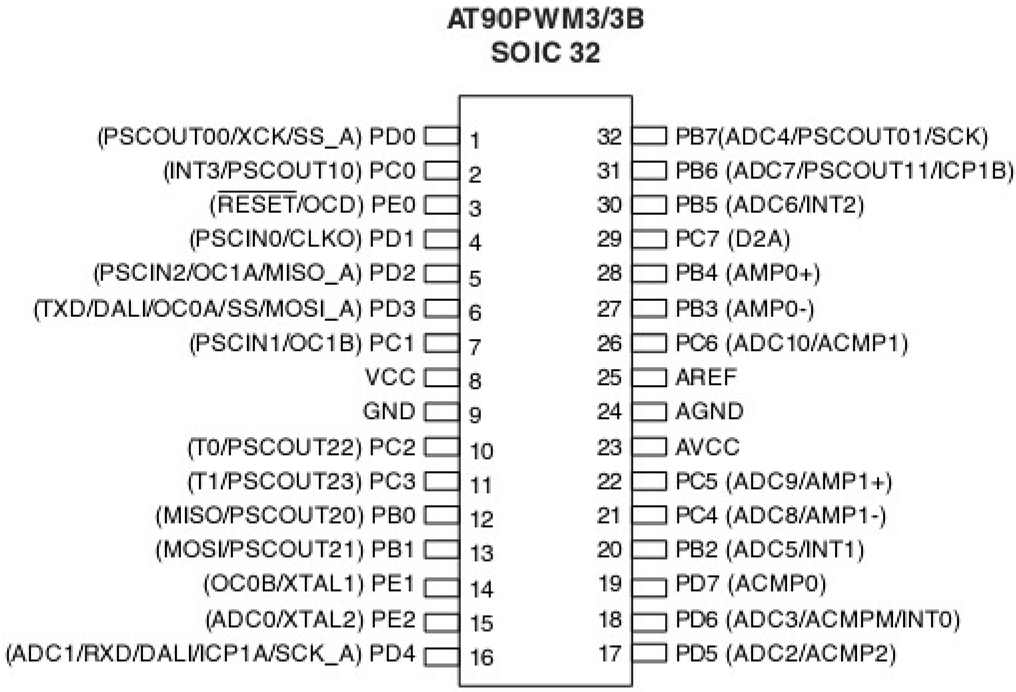
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.12 AT90PWM2-3

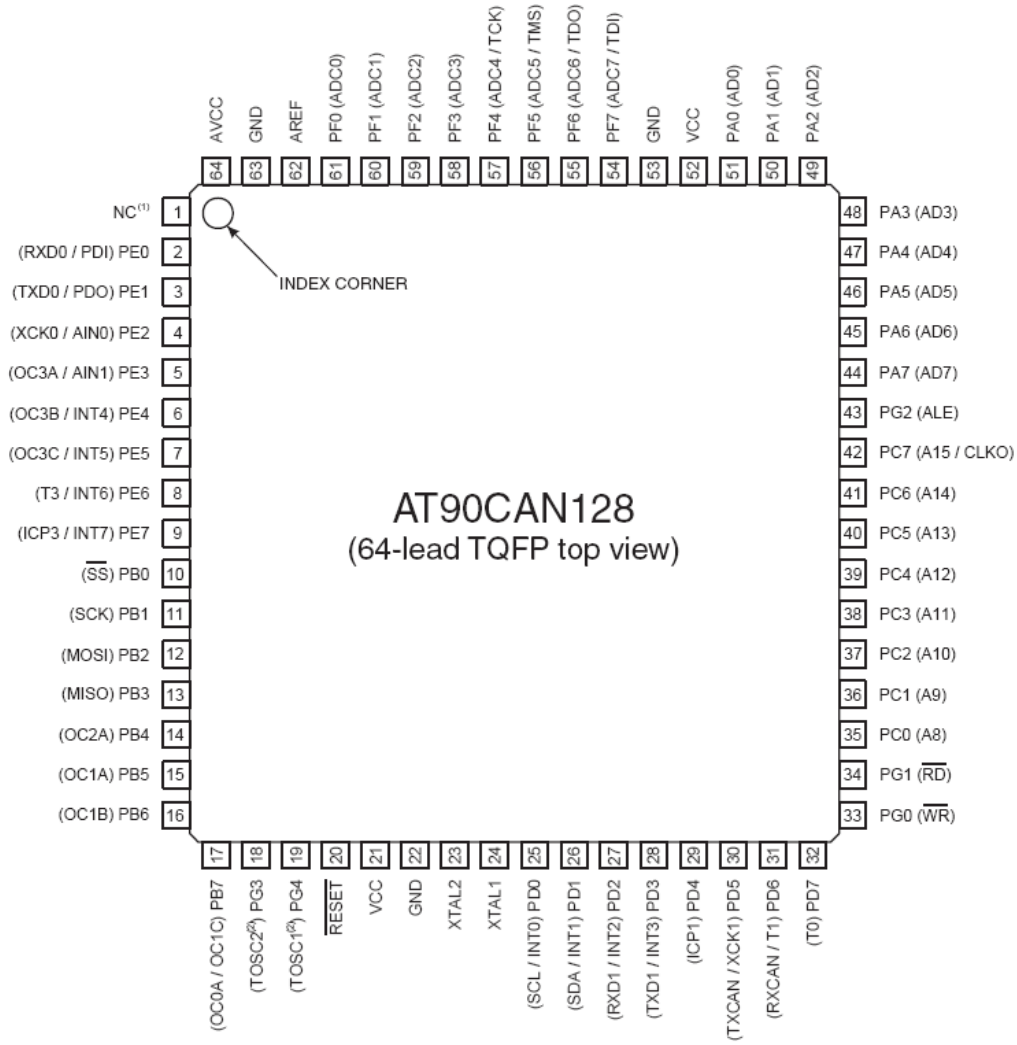
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.





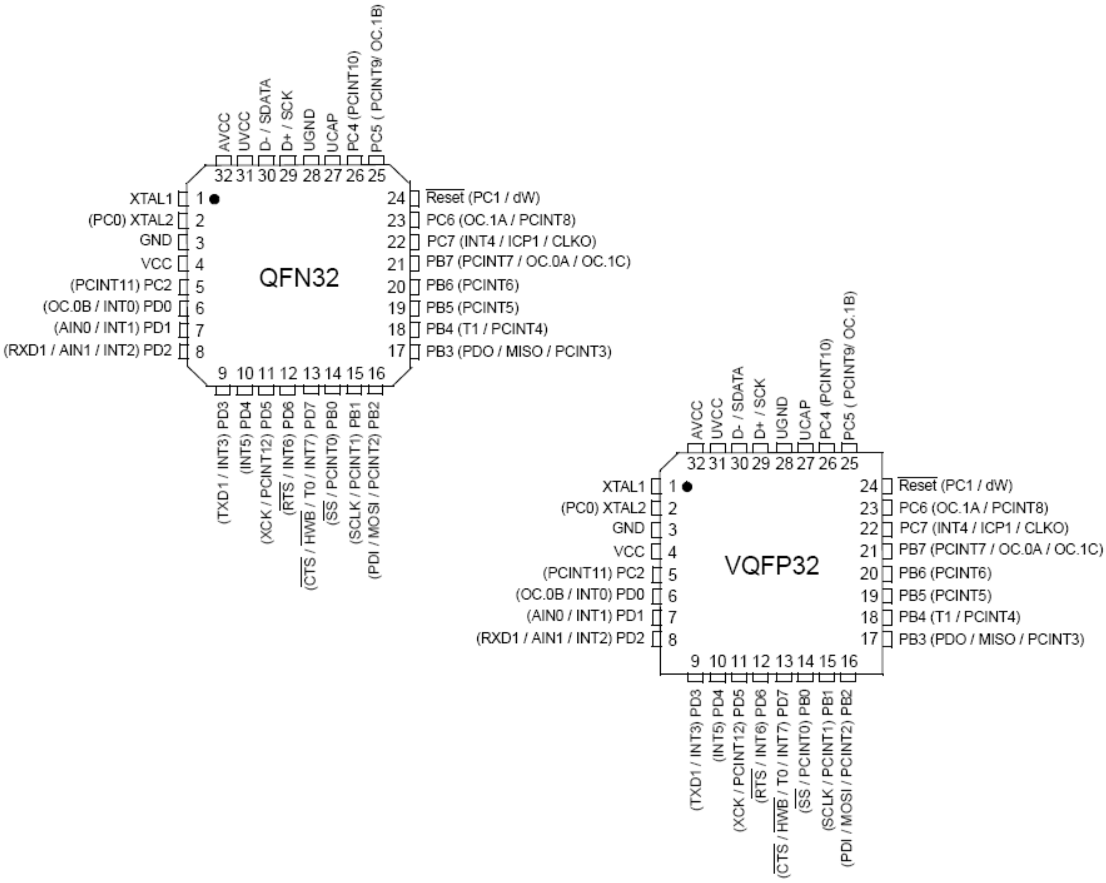
4.18.13 AT90CAN128

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.


AT90CAN128
(64-lead TQFP top view)
INDEX CORNER
NC(1) 1
(RXD0 / PDI) PE0 2
(TXD0 / PDO) PE1 3
(XCK0 / AIN0) PE2 4
(OC3A / AIN1) PE3 5
(OC3B / INT4) PE4 6
(OC3C / INT5) PE5 7
(T3 / INT6) PE6 8
(ICP3 / INT7) PE7 9
(SS) PB0 10
(SCK) PB1 11
(MOSI) PB2 12
(MISO) PB3 13
(OC2A) PB4 14
(OC1A) PB5 15
(OC1B) PB6 16
17 (OC0A / OC1C) PB7
18 (TOSC2(2)) PG3
19 (TOSC1(2)) PG4
20 RESET
21 VCC
22 GND
23 XTAL2
24 XTAL1
25 (SCL / INT0) PD0
26 (SDA / INT1) PD1
27 (RXD1 / INT2) PD2
28 (TXD1 / INT3) PD3
29 (ICP1) PD4
30 (TXCAN / XCK1) PD5
31 (RXCAN / T1) PD6
32 (T0) PD7
33 PG0 (WR)
34 PG1 (RD)
35 PC0 (A8)
36 PC1 (A9)
37 PC2 (A10)
38 PC3 (A11)
39 PC4 (A12)
40 PC5 (A13)
41 PC6 (A14)
42 PC7 (A15 / CLKO)
43 PG2 (ALE)
44 PA7 (AD7)
45 PA6 (AD6)
46 PA5 (AD5)
47 PA4 (AD4)
48 PA3 (AD3)
49 PA2 (AD2)
50 PA1 (AD1)
51 PA0 (AD0)
52 VCC
53 GND
54 PF7 (ADC7 / TDI)
55 PF6 (ADC6 / TDO)
56 PF5 (ADC5 / TMS)
57 PF4 (ADC4 / TCK)
58 PF3 (ADC3)
59 PF2 (ADC2)
60 PF1 (ADC1)
61 PF0 (ADC0)
62 AREF
63 GND
64 AVCC

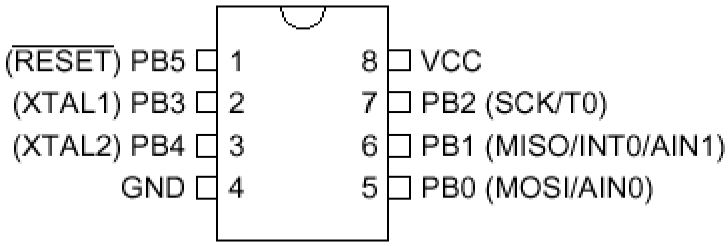
4.18.14 AT90USB162

See also the USB162 220 module for easy soldering of proto types.



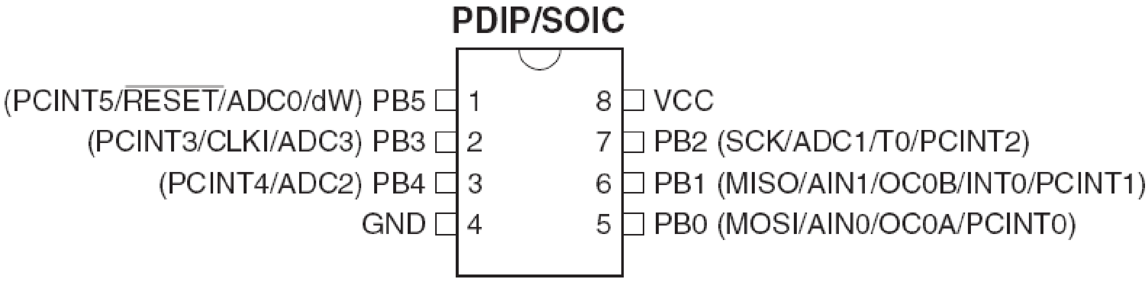
4.18.15 ATtiny12

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.



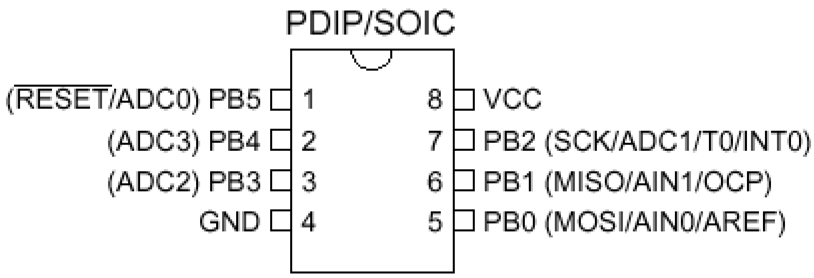
4.18.16 ATtiny13

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.



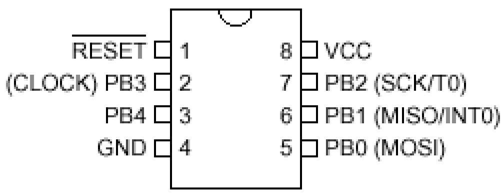
4.18.17 ATtiny15

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.



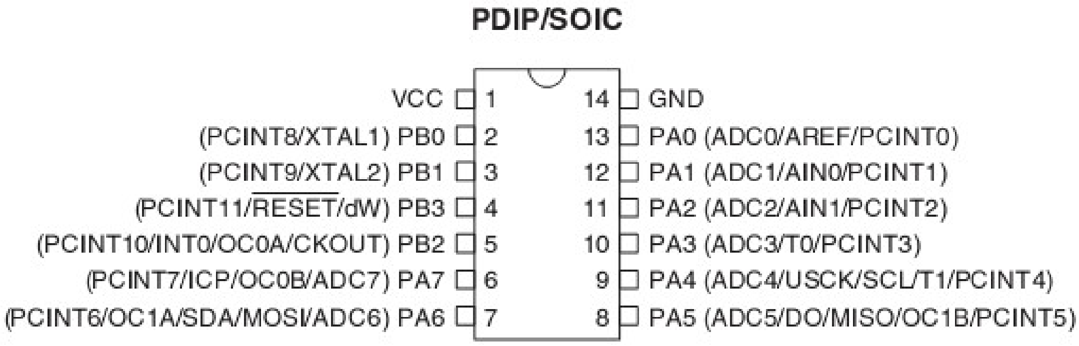
4.18.18 ATtiny22

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.



4.18.19 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.



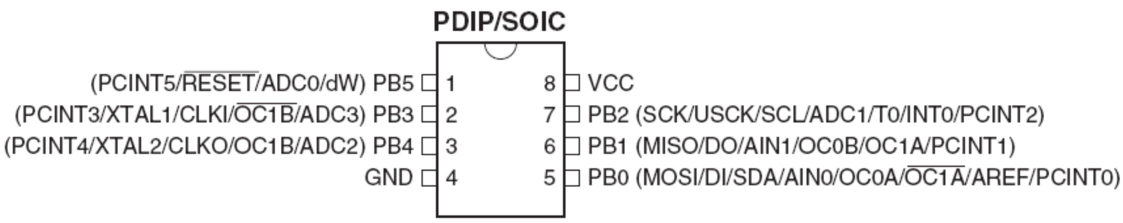
The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

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.

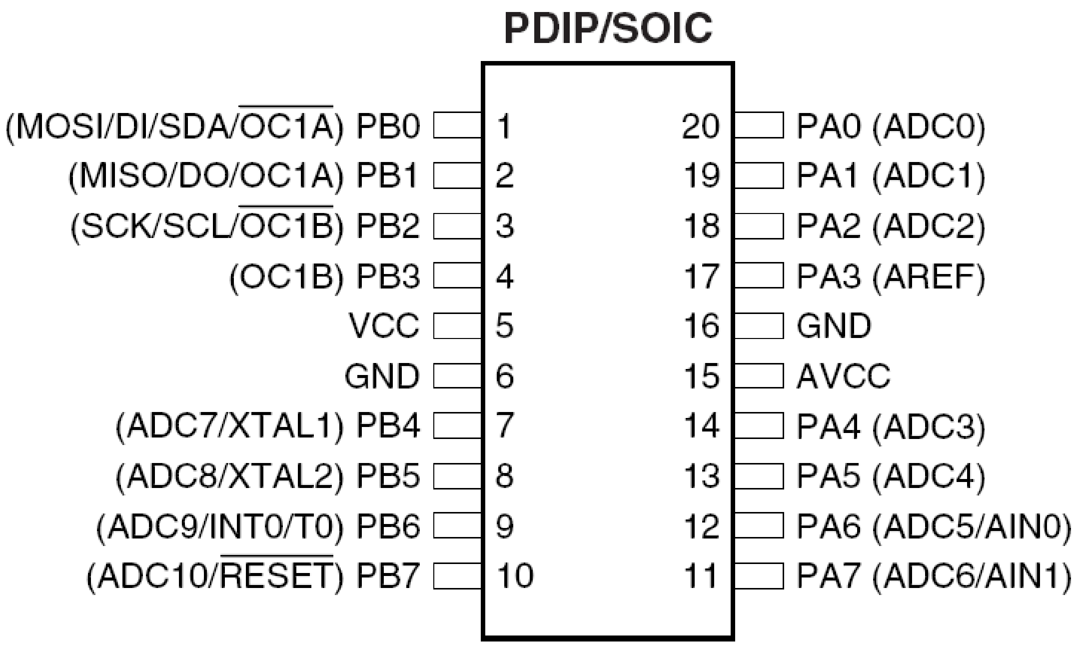
4.18.20 ATtiny25

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.



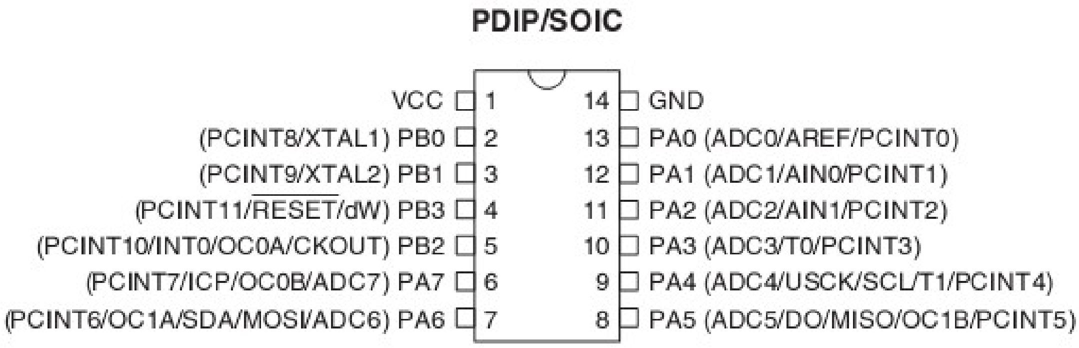
4.18.21 ATtiny26

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.



4.18.22 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.



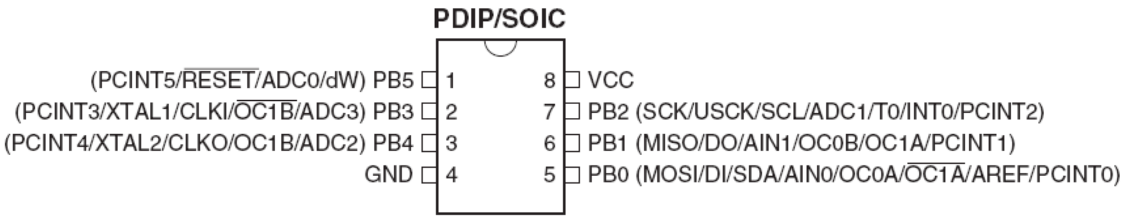
The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

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.

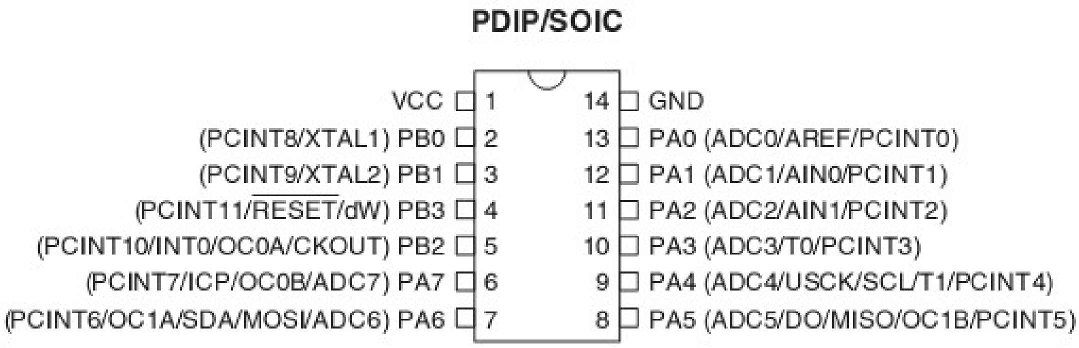
4.18.23 ATtiny45

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.



4.18.24 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.



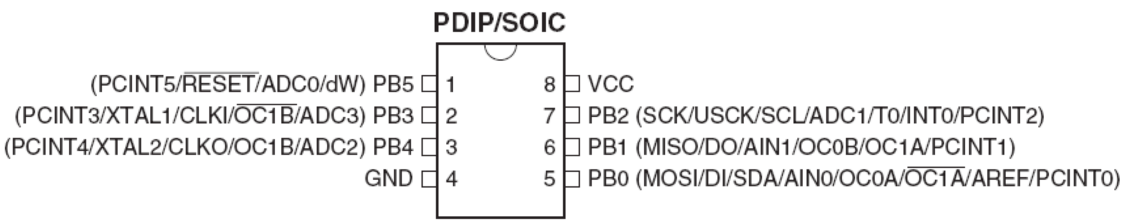
The data sheet does not specify that HWMUL is supported. The DAT file reflect this :

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.

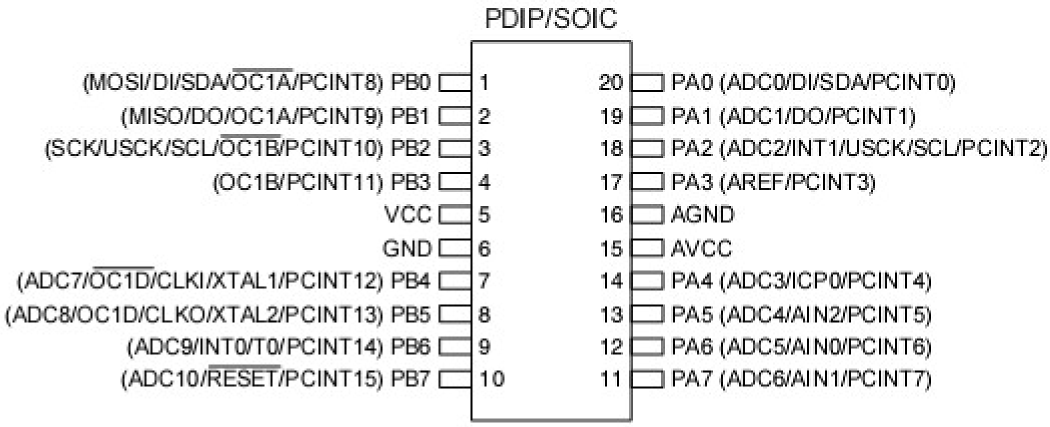
4.18.25 ATtiny85

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.



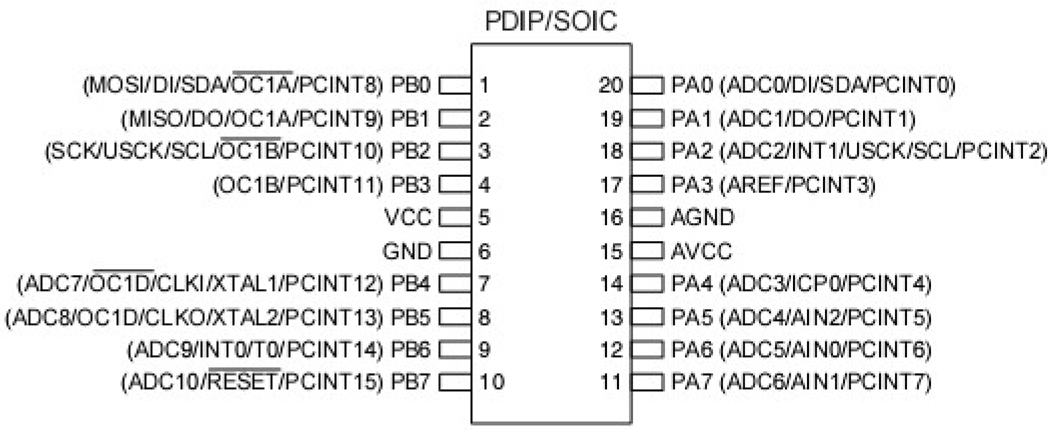
4.18.26 ATtiny261

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.



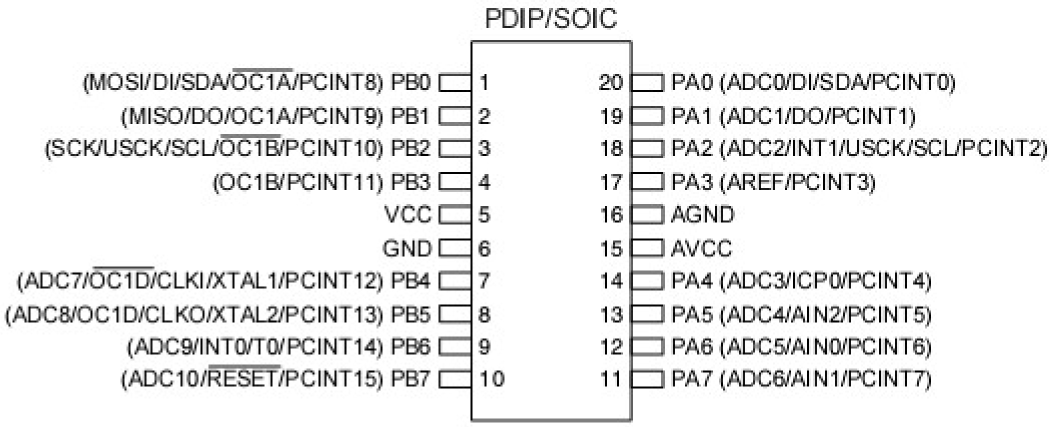
4.18.27 ATtiny461

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.



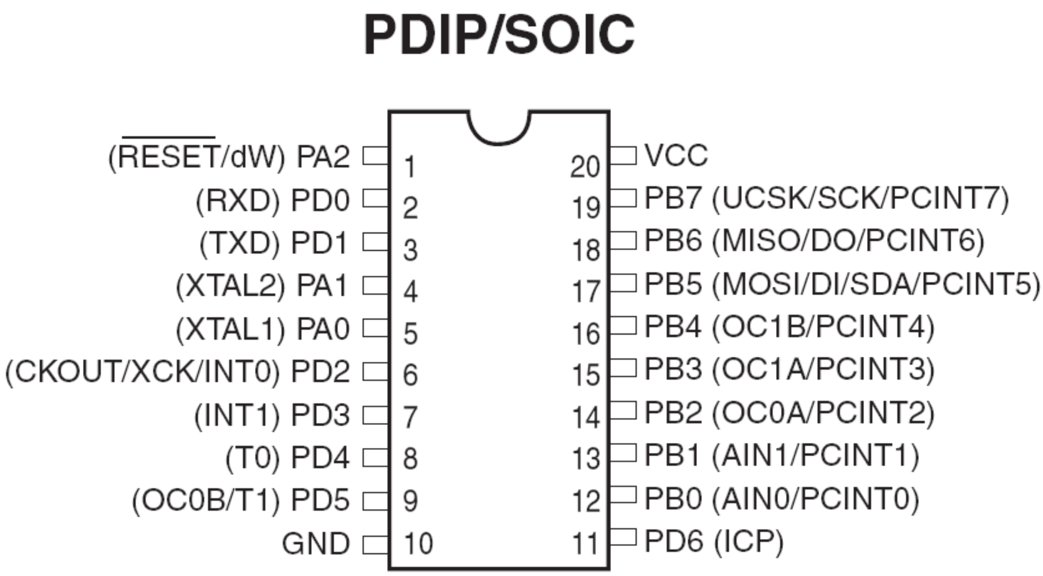
4.18.28 ATtiny861

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.



4.18.29 ATtiny2313

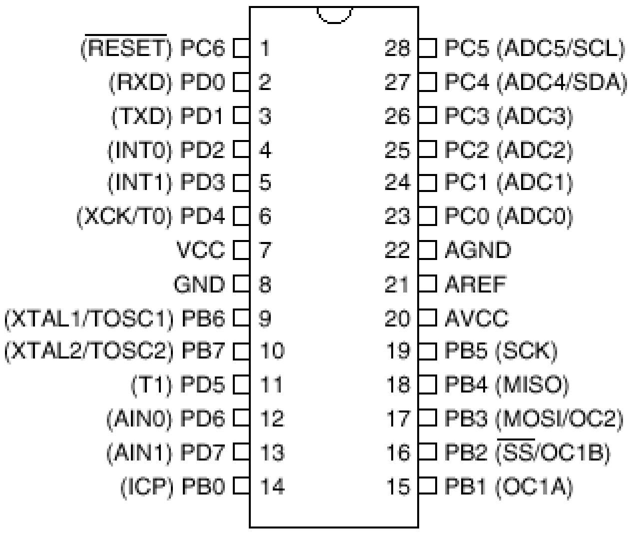
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



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.

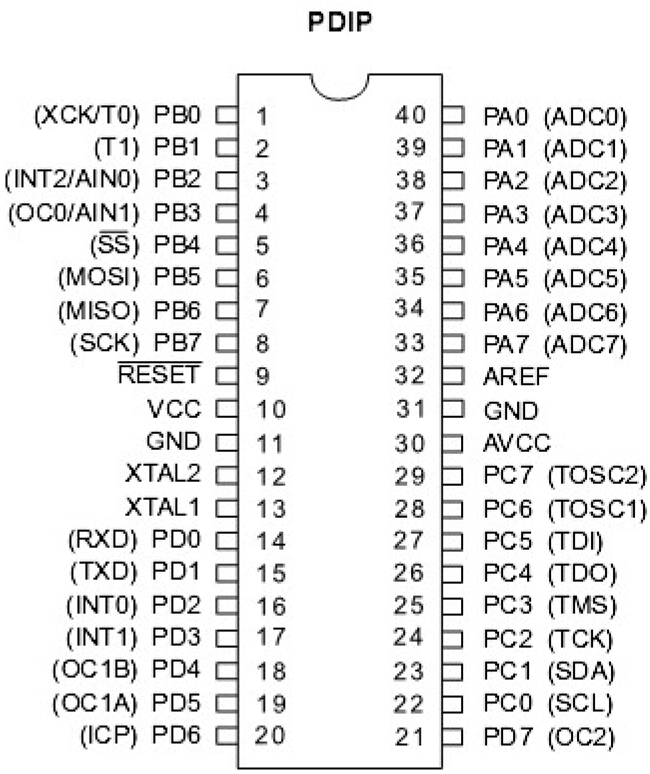
4.18.30 ATMEGA8

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.



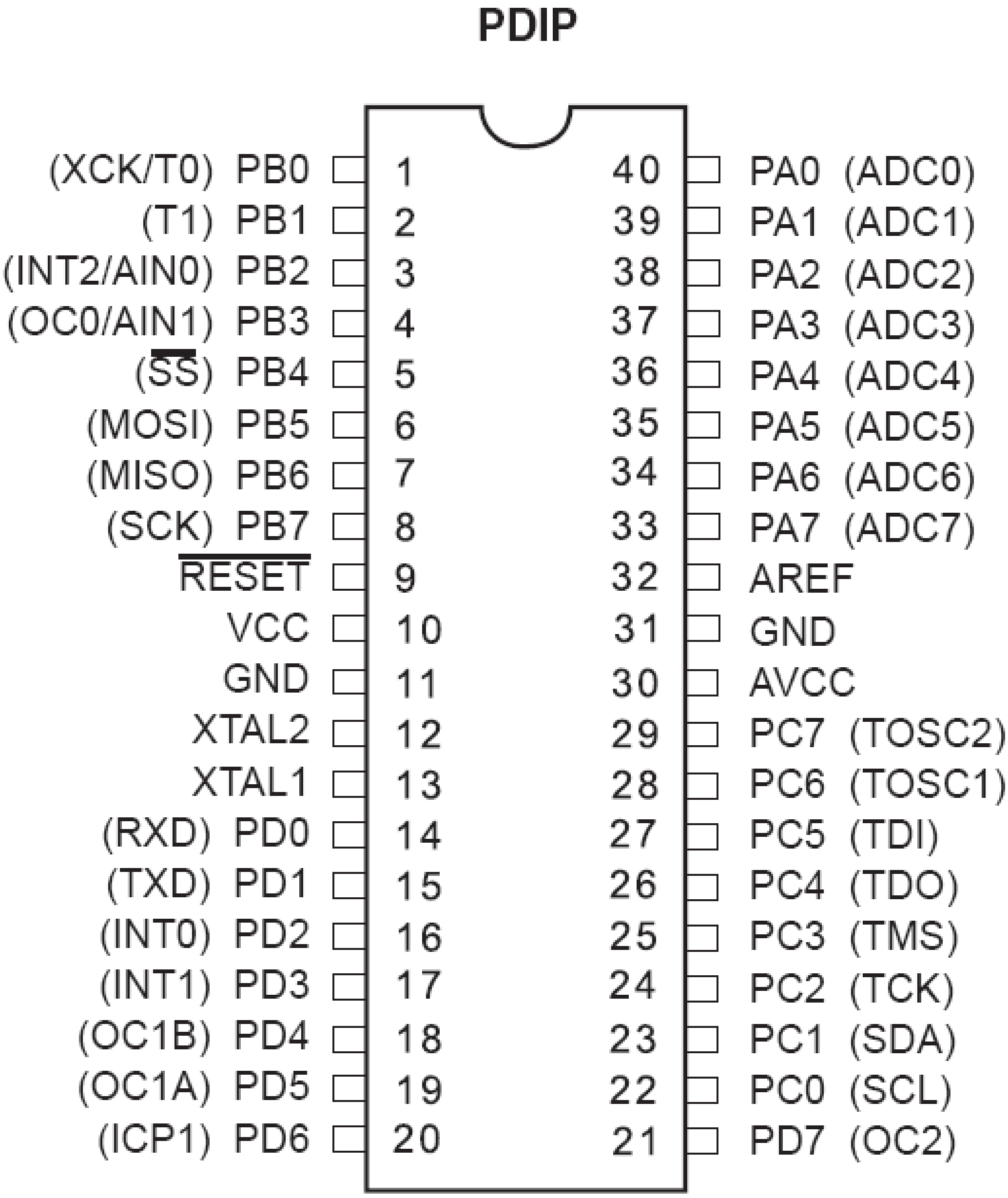
4.18.31 ATMEGA16

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.



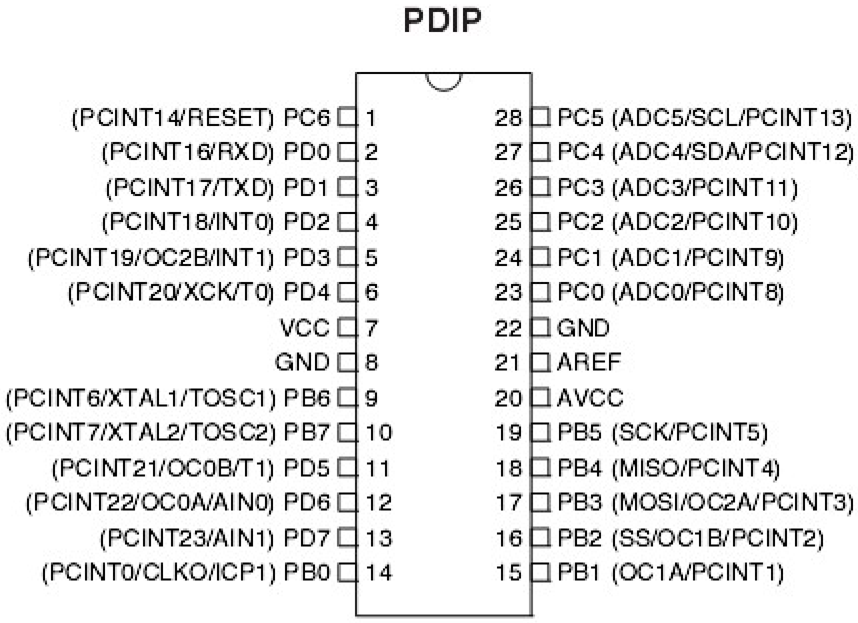
4.18.32 ATMEGA32

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.



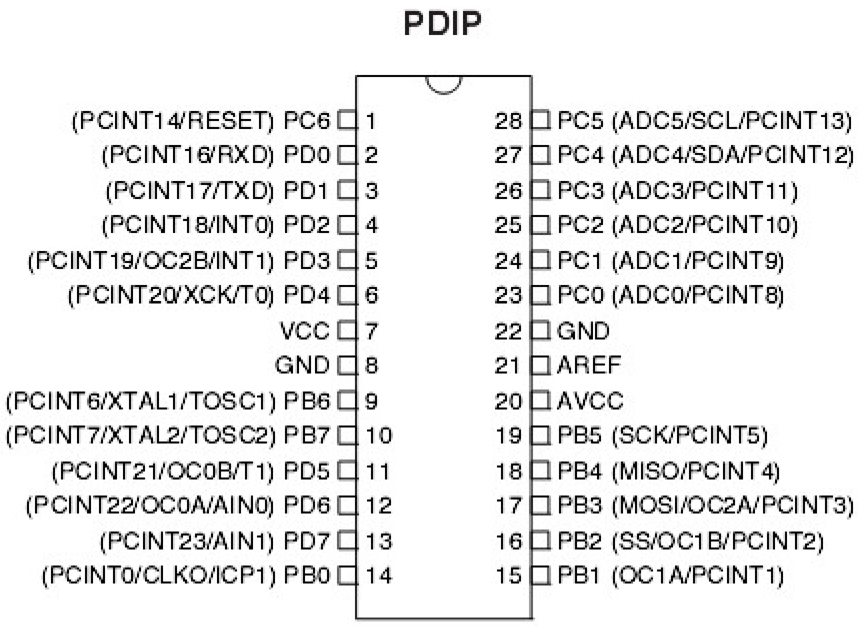
4.18.33 ATMEGA48

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.



4.18.34 ATMEGA88

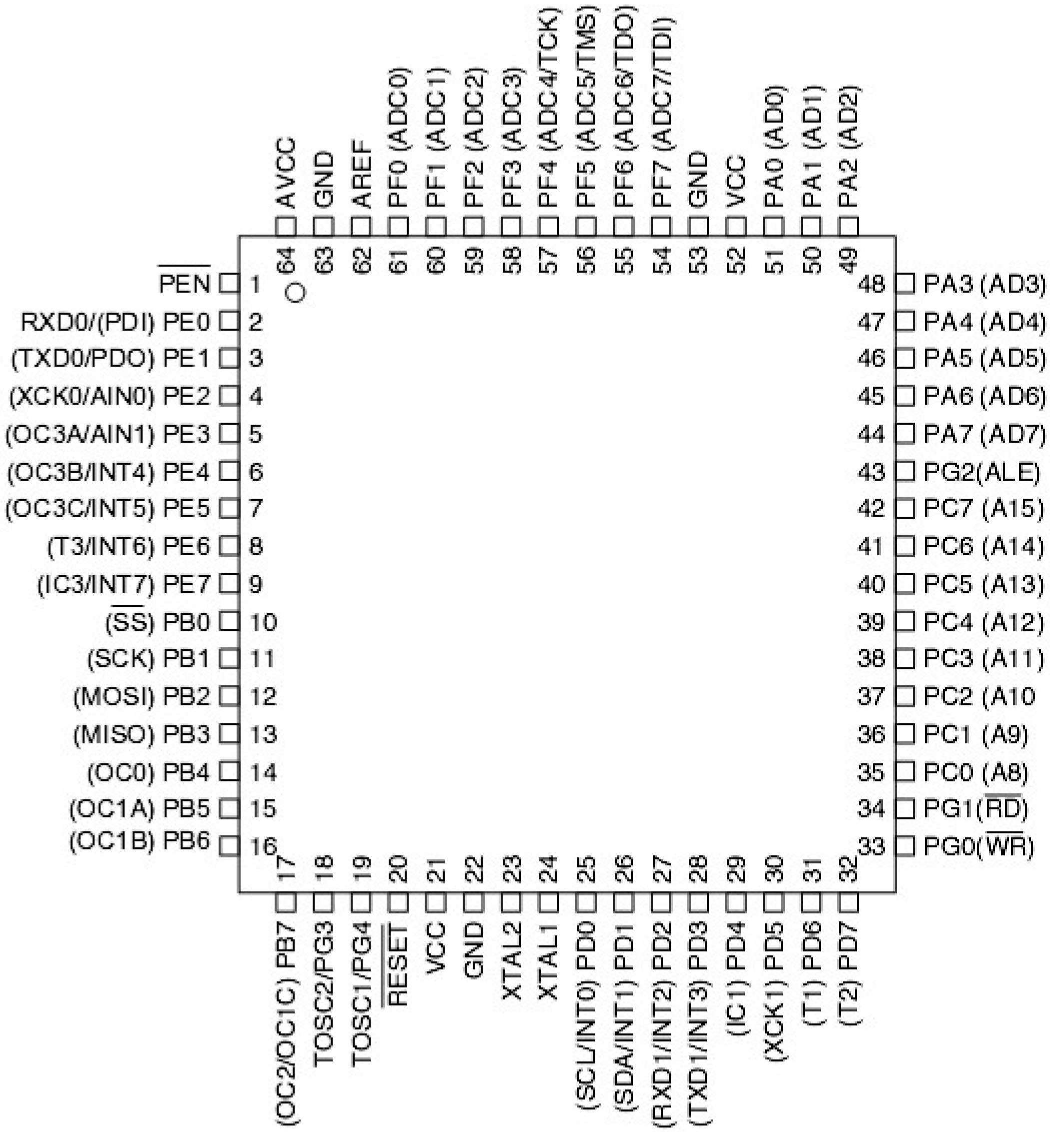
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



4.18.35 ATMEGA64

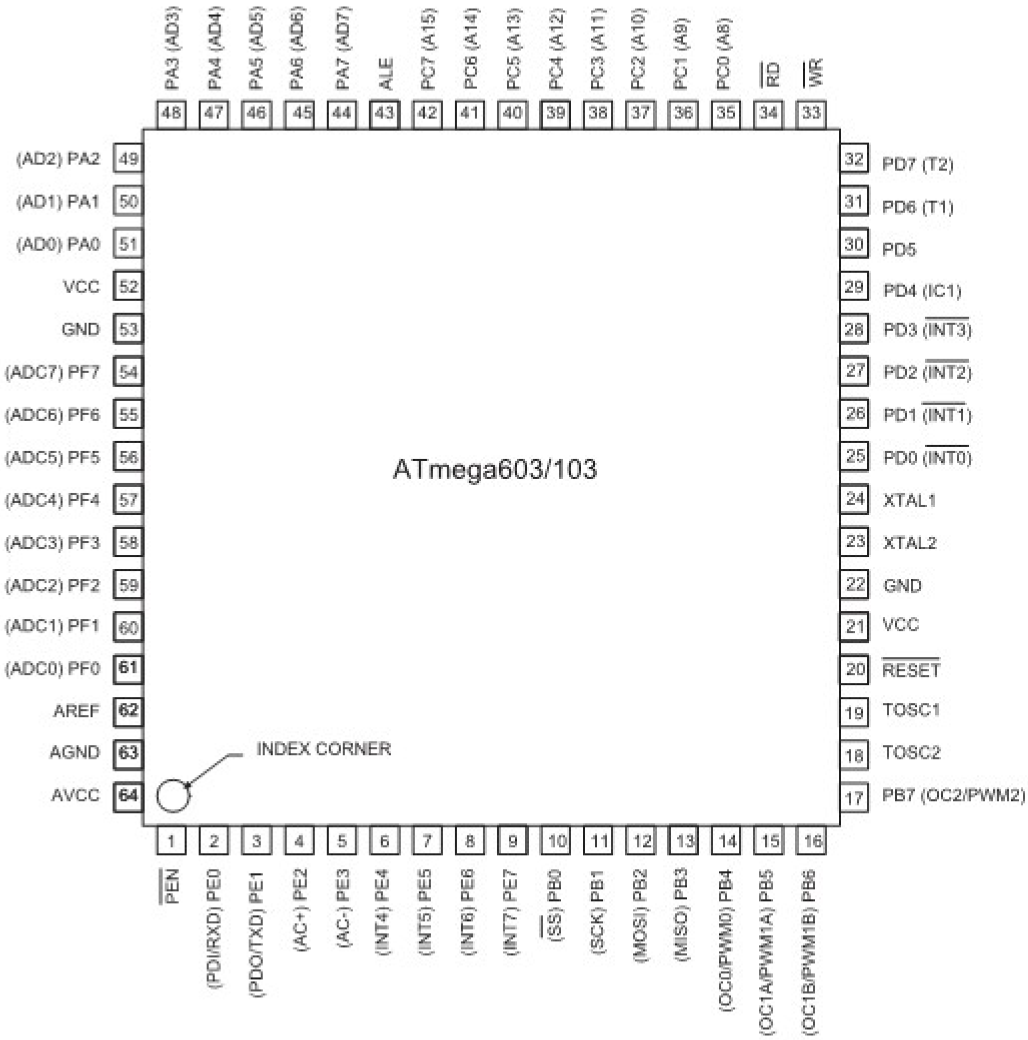
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.

Figure 1. Pinout ATmega64



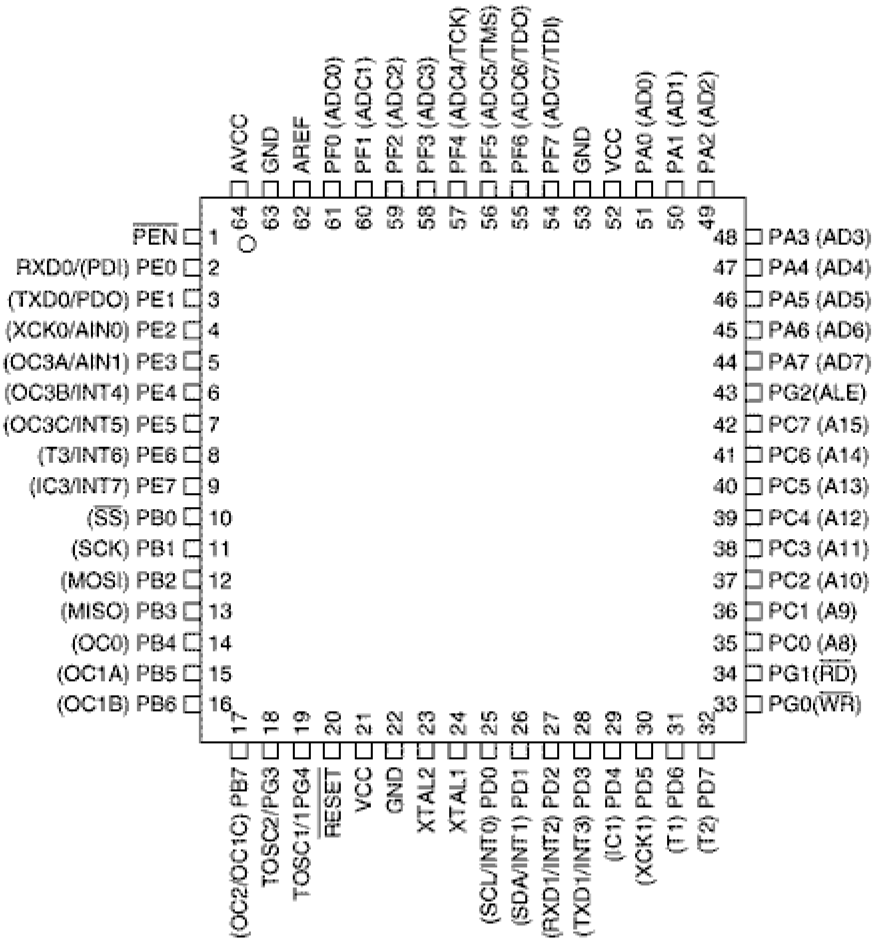
4.18.36 ATMEGA103

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.


ATmega603/103
INDEX CORNER
PA3 (AD3) 48
PA4 (AD4) 47
PA5 (AD5) 46
PA6 (AD6) 45
PA7 (AD7) 44
ALE 43
PC7 (A15) 42
PC6 (A14) 41
PC5 (A13) 40
PC4 (A12) 39
PC3 (A11) 38
PC2 (A10) 37
PC1 (A9) 36
PC0 (A8) 35
RD 34
WR 33
(AD2) PA2 49
(AD1) PA1 50
(AD0) PA0 51
VCC 52
GND 53
(ADC7) PF7 54
(ADC6) PF6 55
(ADC5) PF5 56
(ADC4) PF4 57
(ADC3) PF3 58
(ADC2) PF2 59
(ADC1) PF1 60
(ADC0) PF0 61
AREF 62
AGND 63
AVCC 64
32 PD7 (T2)
31 PD6 (T1)
30 PD5
29 PD4 (IC1)
28 PD3 (INT3)
27 PD2 (INT2)
26 PD1 (INT1)
25 PD0 (INT0)
24 XTAL1
23 XTAL2
22 GND
21 VCC
20 RESET
19 TOSC1
18 TOSC2
17 PB7 (OC2/PWM2)
1 PEN
2 (PDI/RXD) PE0
3 (PDO/TXD) PE1
4 (AC+) PE2
5 (AC-) PE3
6 (INT4) PE4
7 (INT5) PE5
8 (INT6) PE6
9 (INT7) PE7
10 (SS) PB0
11 (SCK) PB1
12 (MOSI) PB2
13 (MISO) PB3
14 (OC0/PWM0) PB4
15 (OC1A/PWM1A) PB5
16 (OC1B/PWM1B) PB6

4.18.37 ATMEGA128

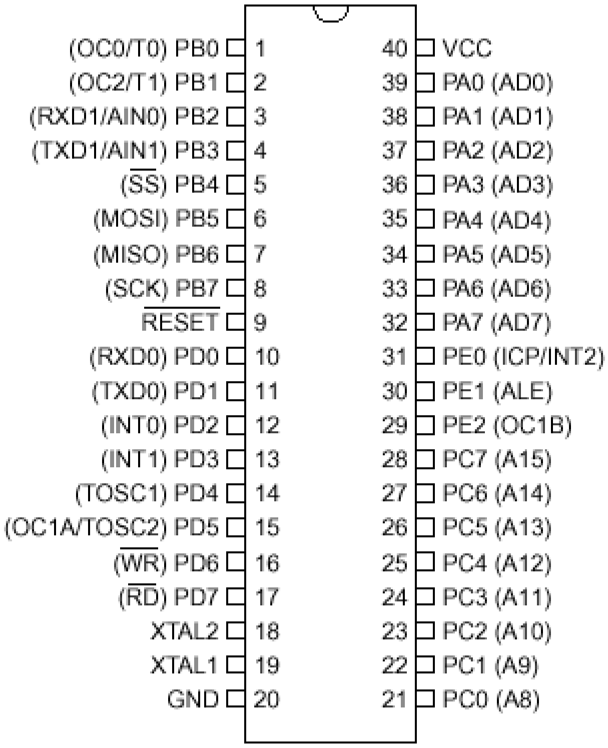
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



When using XRAM and IDLE, the micro need the CONFIG XRAM after returing from the power down mode.

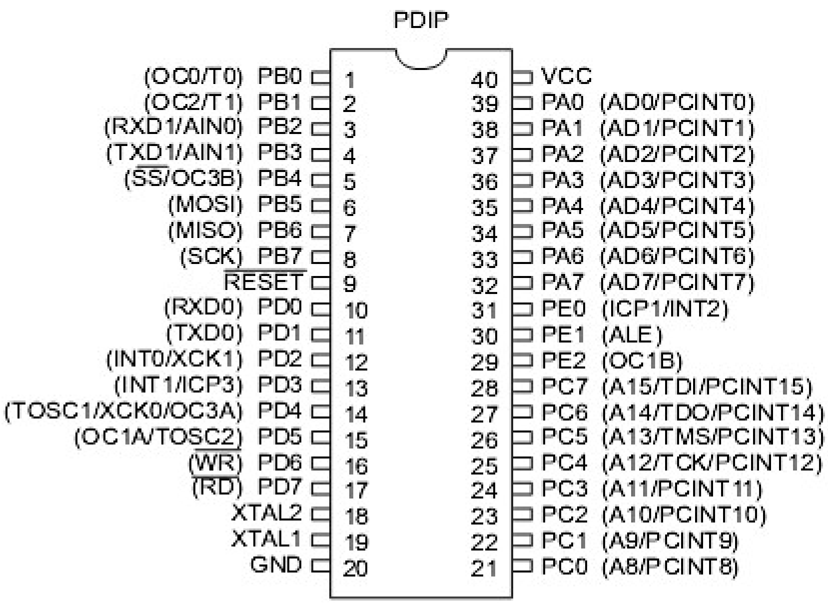
4.18.38 ATMEGA161

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.



4.18.39 ATMEGA162

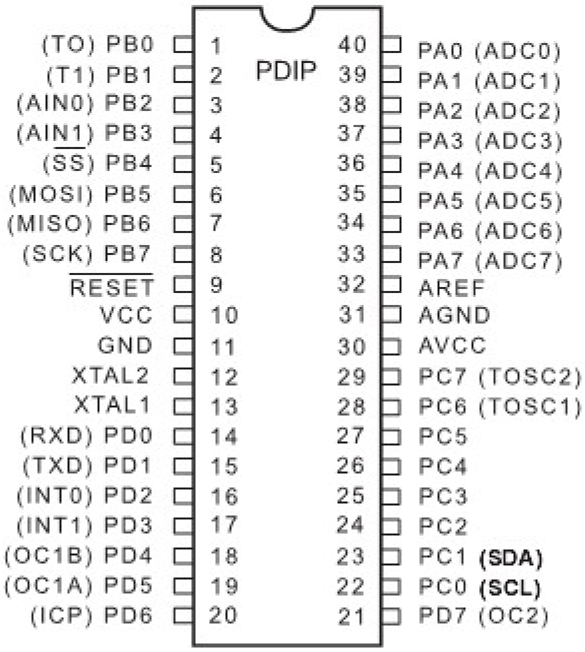
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The M162 has a clock-16 divider enabled by default. See the M162.bas sample file

4.18.40 ATMEGA163

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 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_0000

DDRB = &B1111_0000 'set upper bits for output.

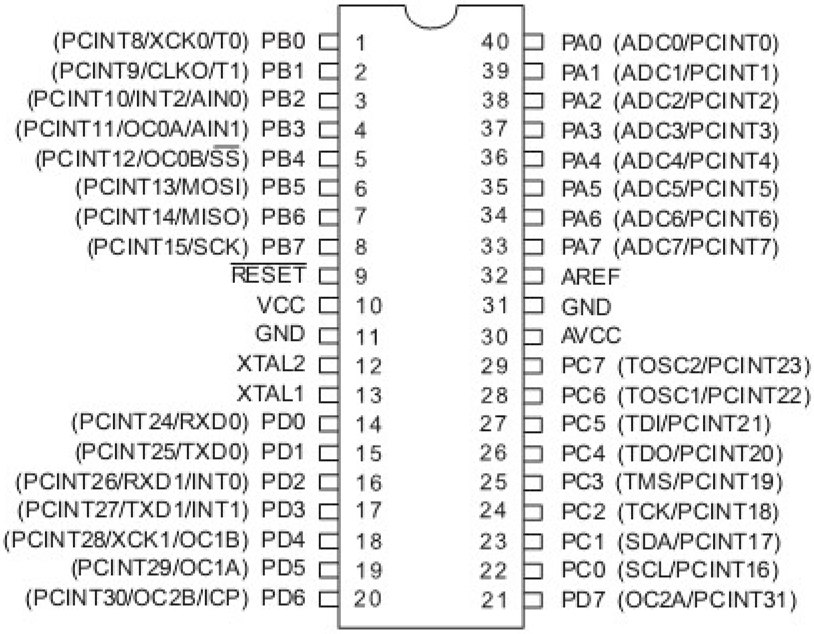
Spcr = &B0001_0000 ' set SPI to Master and Disable.

If The SPCR register is not set for Master, you cannot set the pins for

Output.

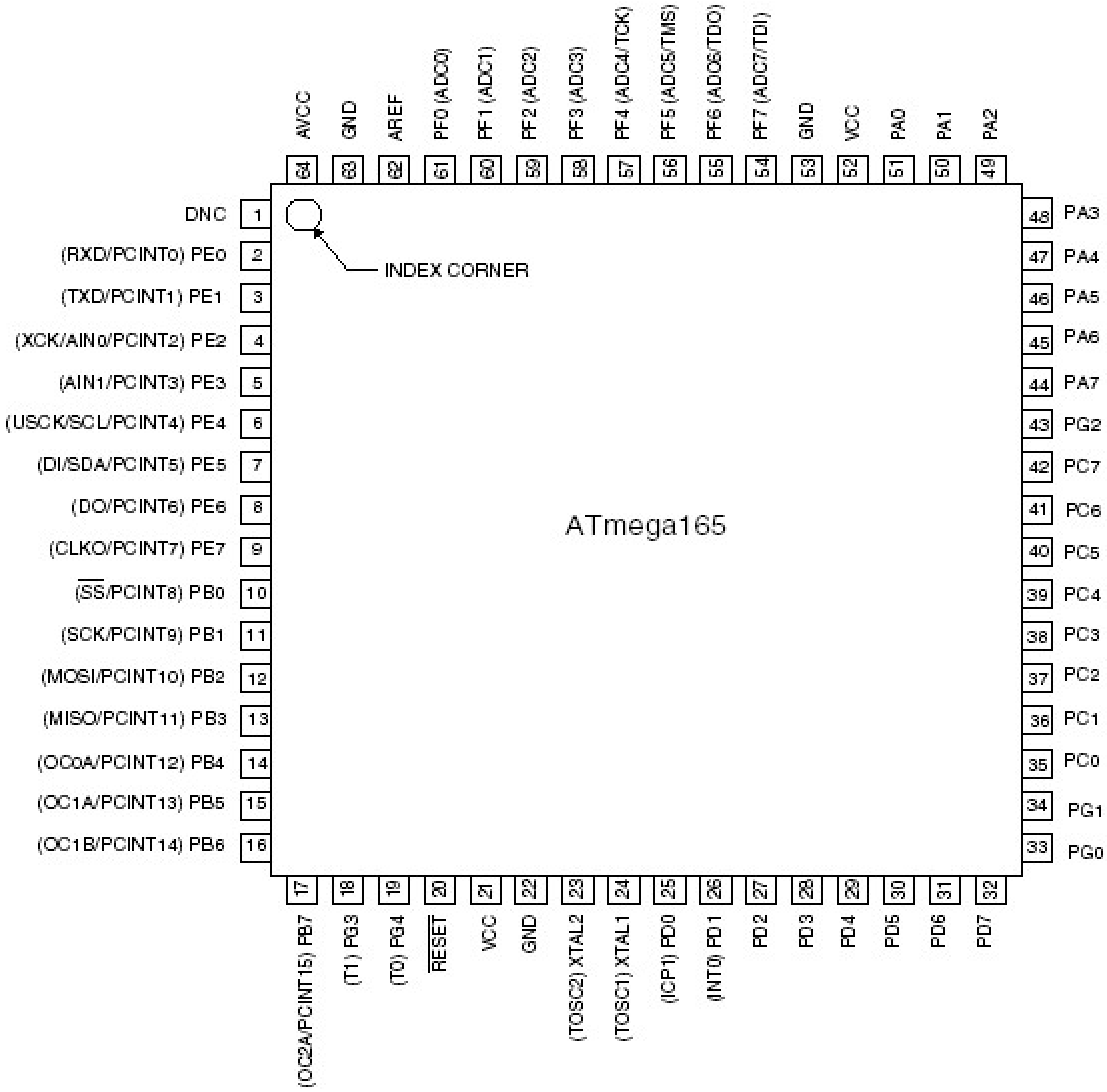
4.18.41 ATMEGA164P

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.



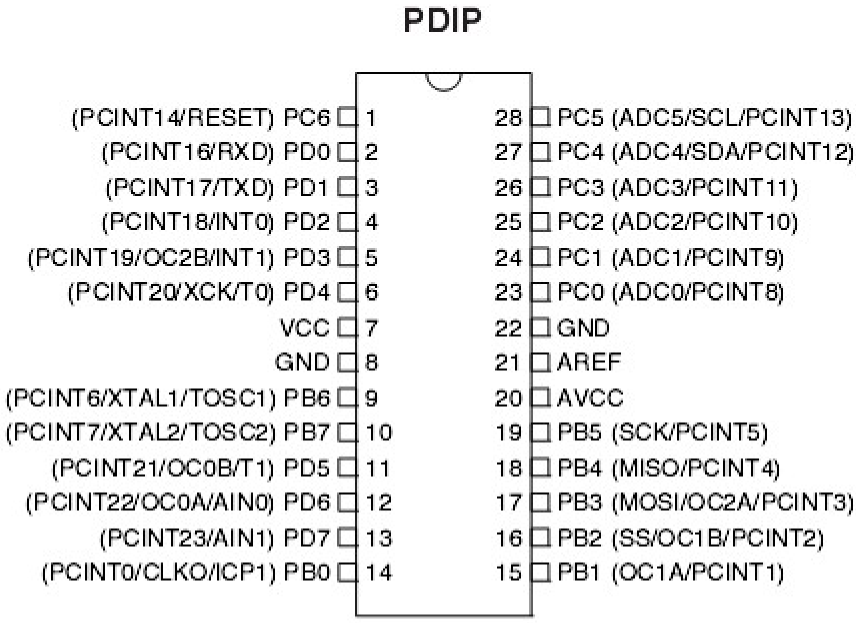
4.18.42 ATMEGA165

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.



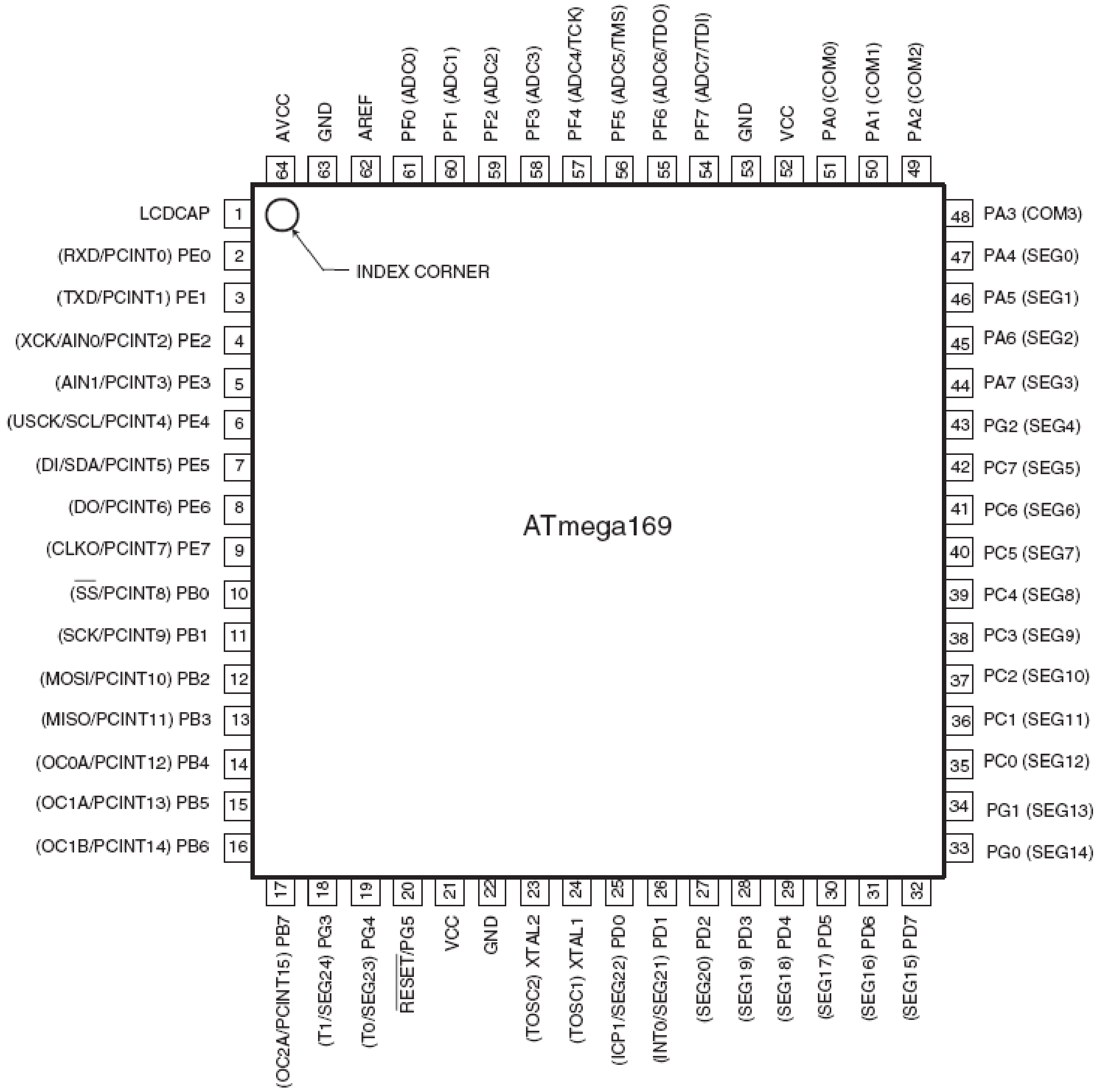
4.18.43 ATMEGA168

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.



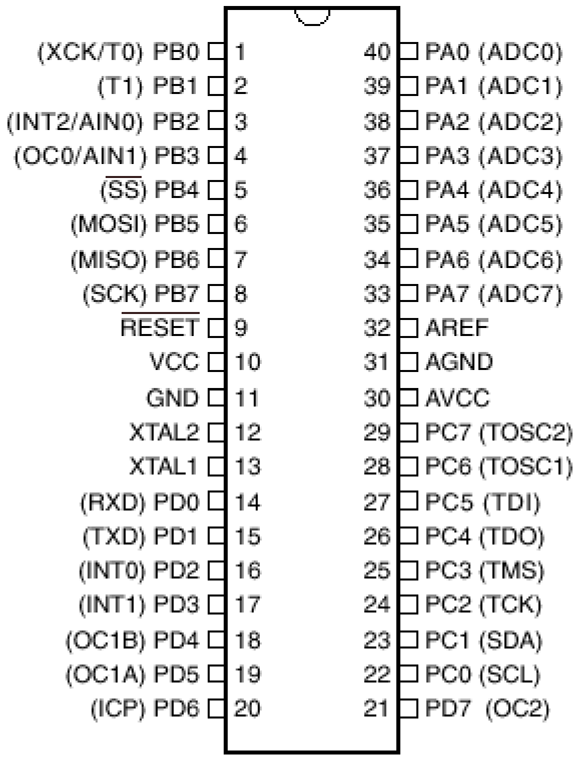
4.18.44 ATMEGA169

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.



4.18.45 ATMEGA323

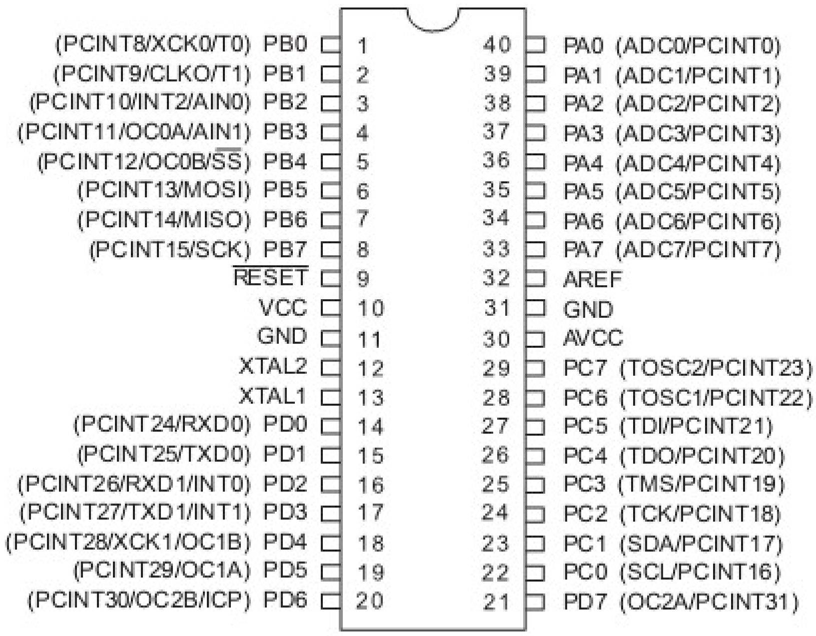
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



The 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.

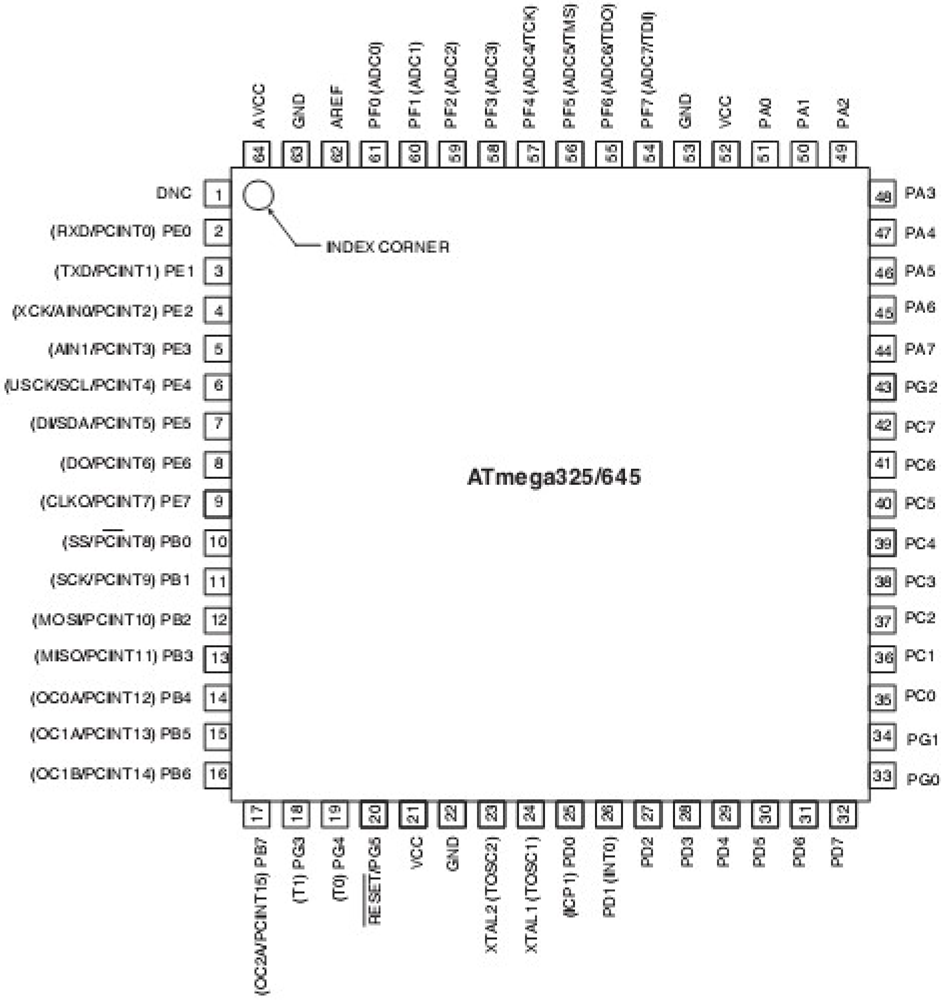
4.18.46 ATMEGA324P

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.



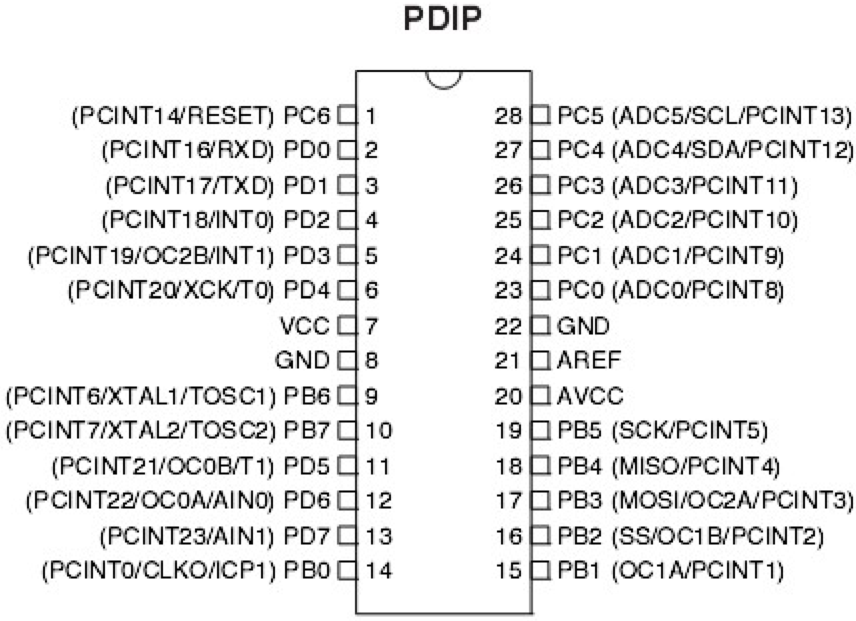
4.18.47 ATMEGA325

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.



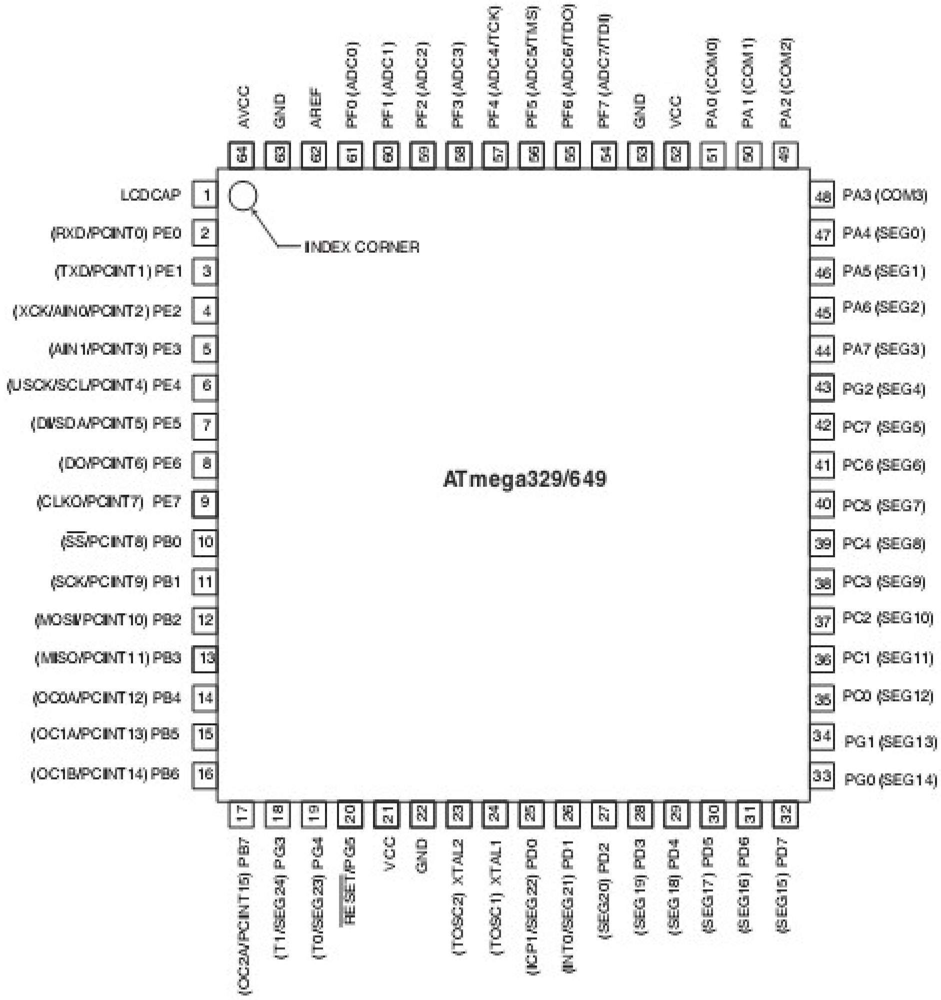
4.18.48 ATMEGA328P

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.



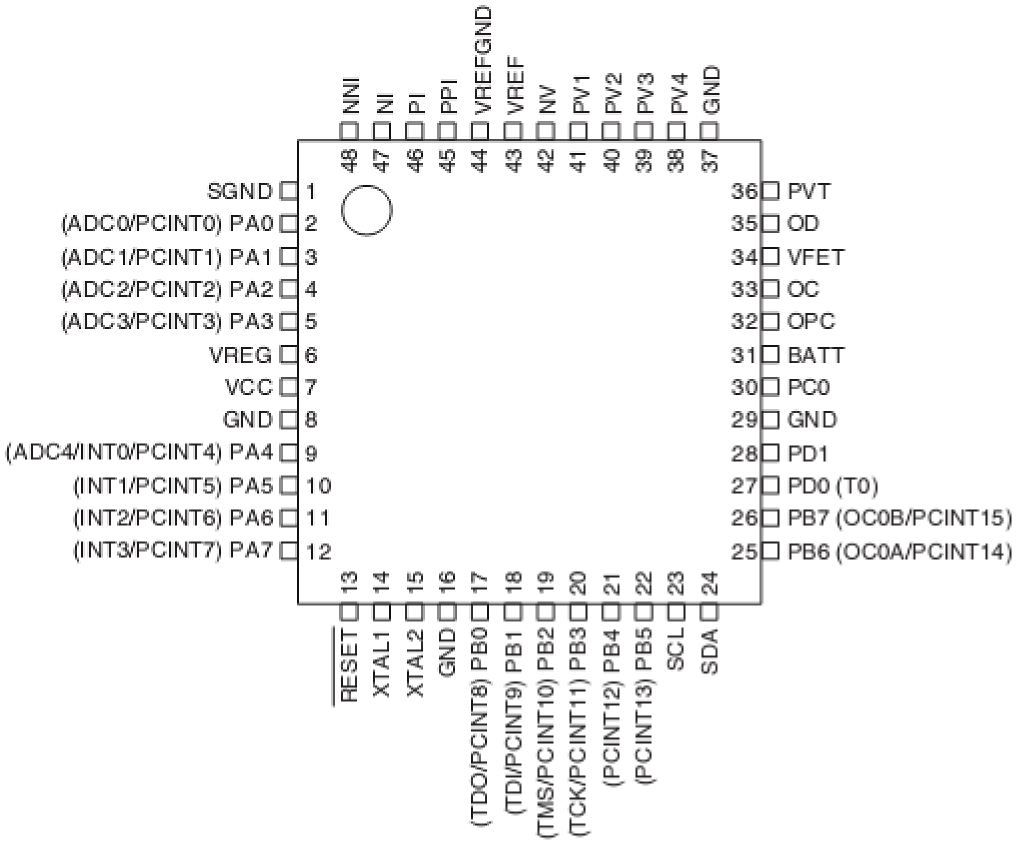
4.18.49 ATMEGA329

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.



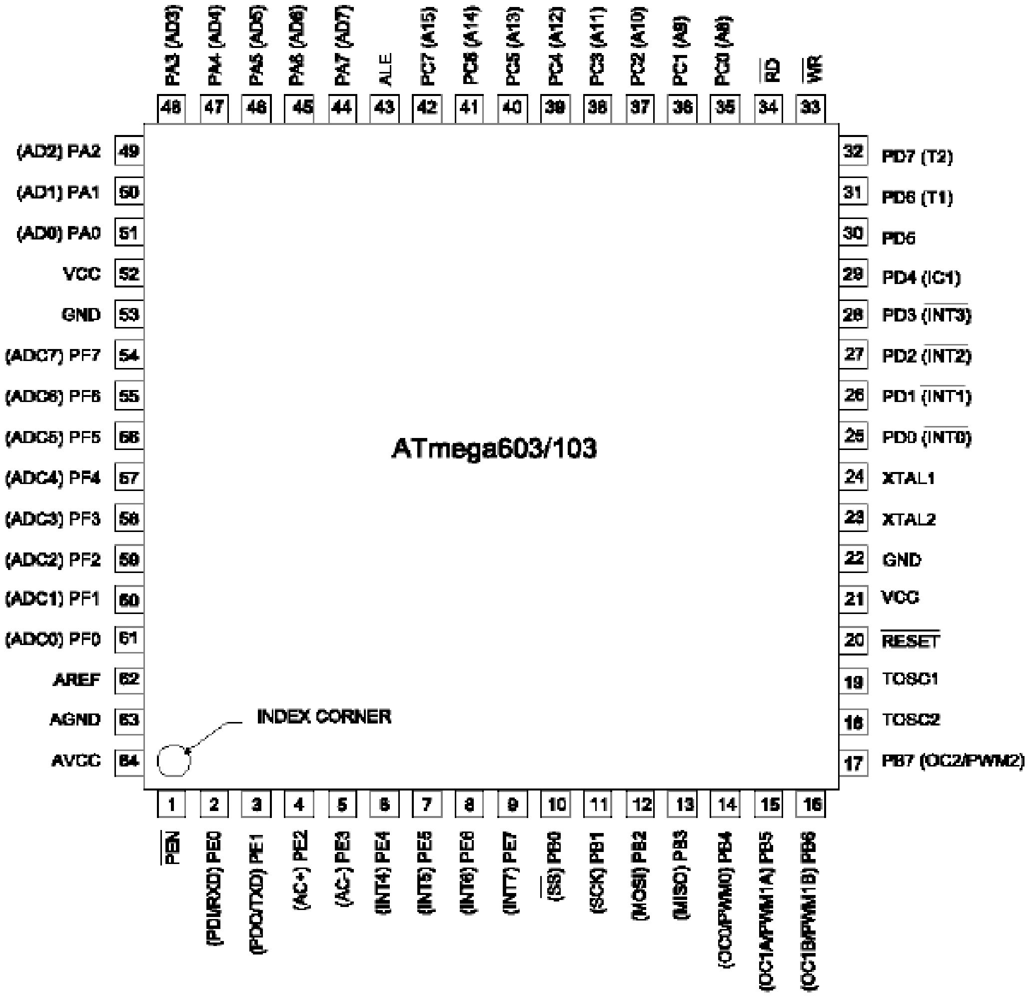
4.18.50 ATMEGA406

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.

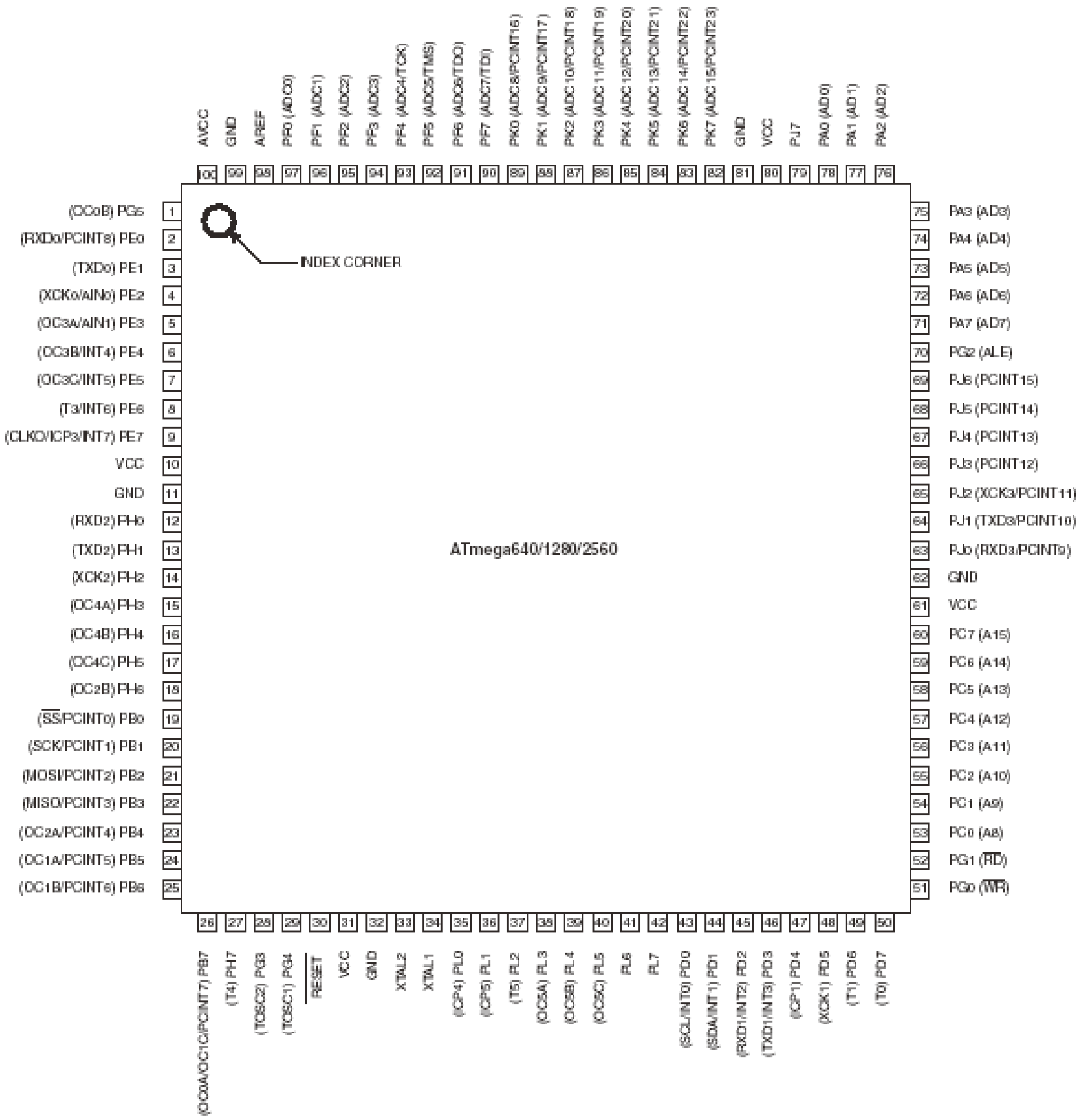


4.18.51 ATMEGA603

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


ATmega603/103
INDEX CORNER
PA3 (AD3)
PA4 (AD4)
PA5 (AD5)
PA6 (AD6)
PA7 (AD7)
ALE
PC7 (A15)
PC6 (A14)
PC5 (A13)
PC4 (A12)
PC3 (A11)
PC2 (A10)
PC1 (A9)
PC0 (A8)
RD
WR
(AD2) PA2
(AD1) PA1
(AD0) PA0
VCC
GND
(ADC7) PF7
(ADC6) PF6
(ADC5) PF5
(ADC4) PF4
(ADC3) PF3
(ADC2) PF2
(ADC1) PF1
(ADC0) PF0
AREF
AGND
AVCC
PD7 (T2)
PD6 (T1)
PD5
PD4 (IC1)
PD3 (INT3)
PD2 (INT2)
PD1 (INT1)
PD0 (INT0)
XTAL1
XTAL2
GND
VCC
RESET
TOSC1
TOSC2
PB7 (OC2/PWM2)
PEN
(PDI/RXD) PE0
(PDO/TXD) PE1
(AC+) PE2
(AC-) PE3
(INT4) PE4
(INT5) PE5
(INT6) PE6
(INT7) PE7
(SS) PB0
(SCK) PB1
(MOSI) PB2
(MISO) PB3
(OC0/PWM0) PB4
(OC1A/PWM1A) PB5
(OC1B/PWM1B) PB6

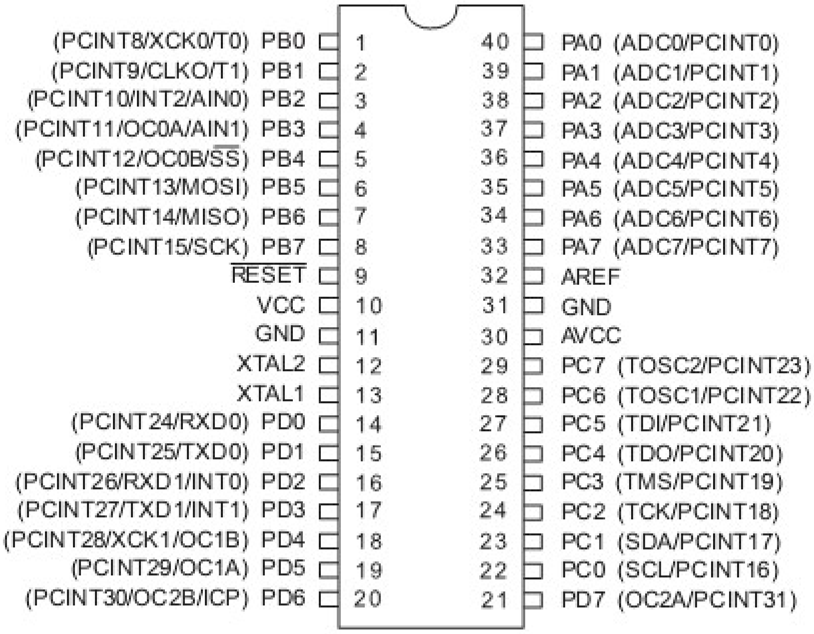
4.18.52 ATMEGA640



.

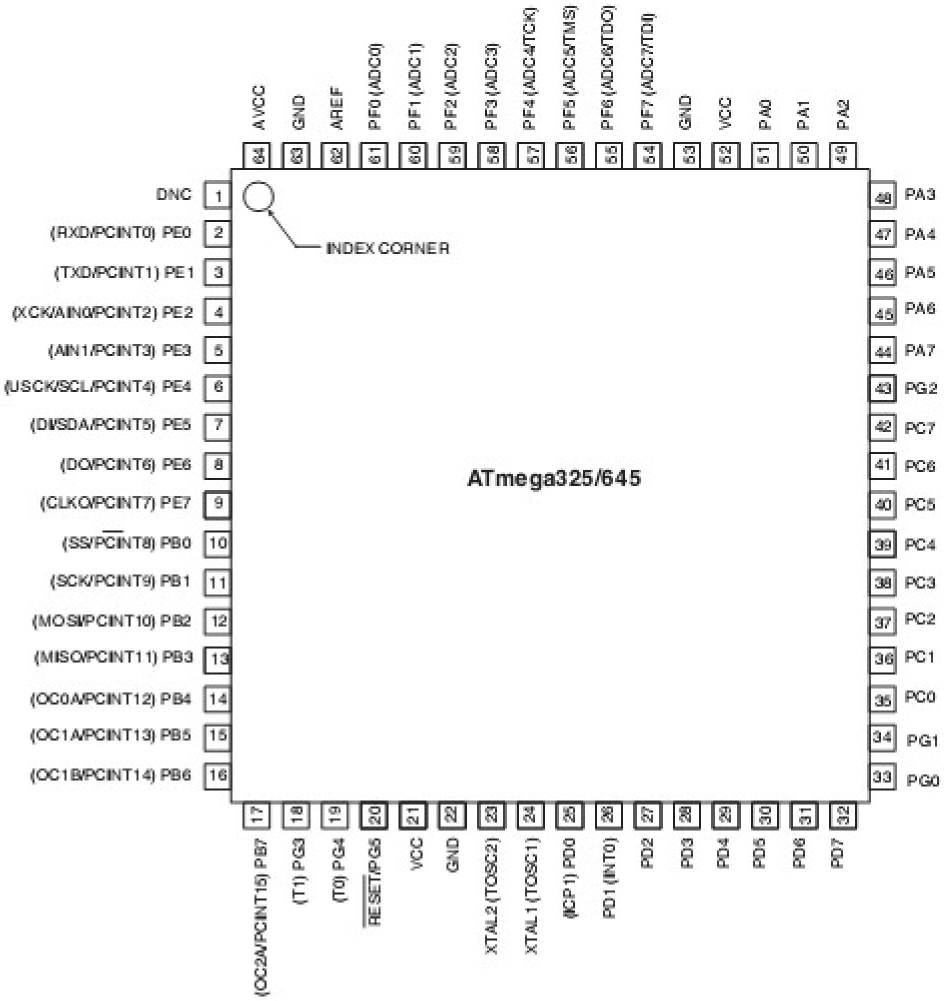
4.18.53 ATMEGA644P

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.



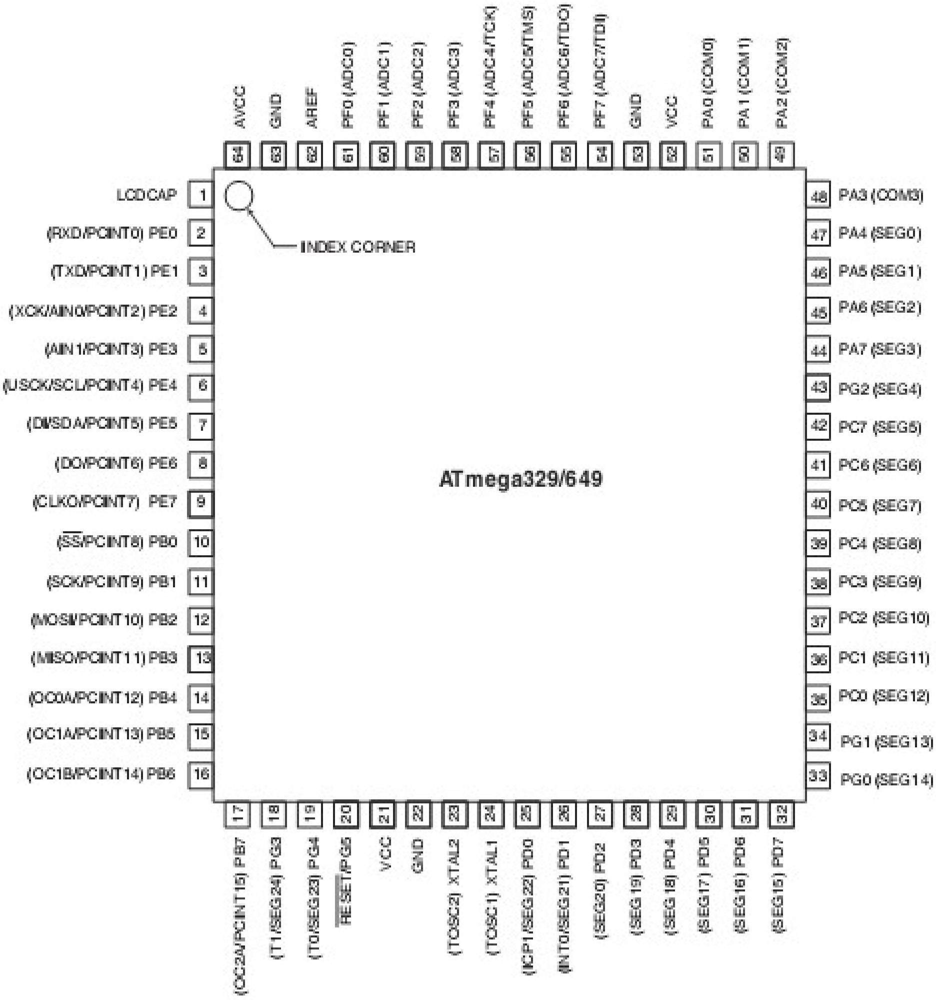
4.18.54 ATMEGA645

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.

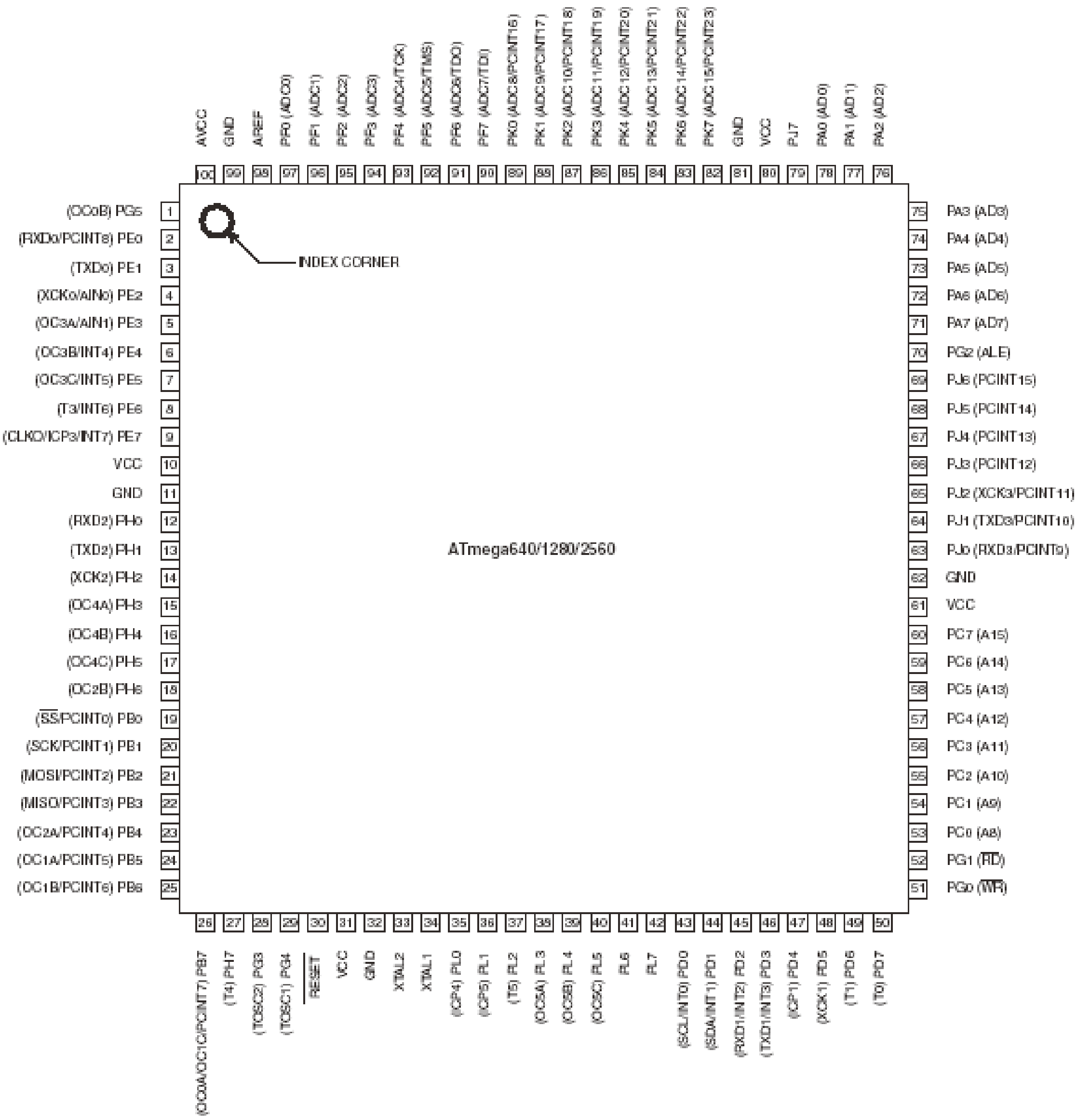


4.18.55 ATMEGA649

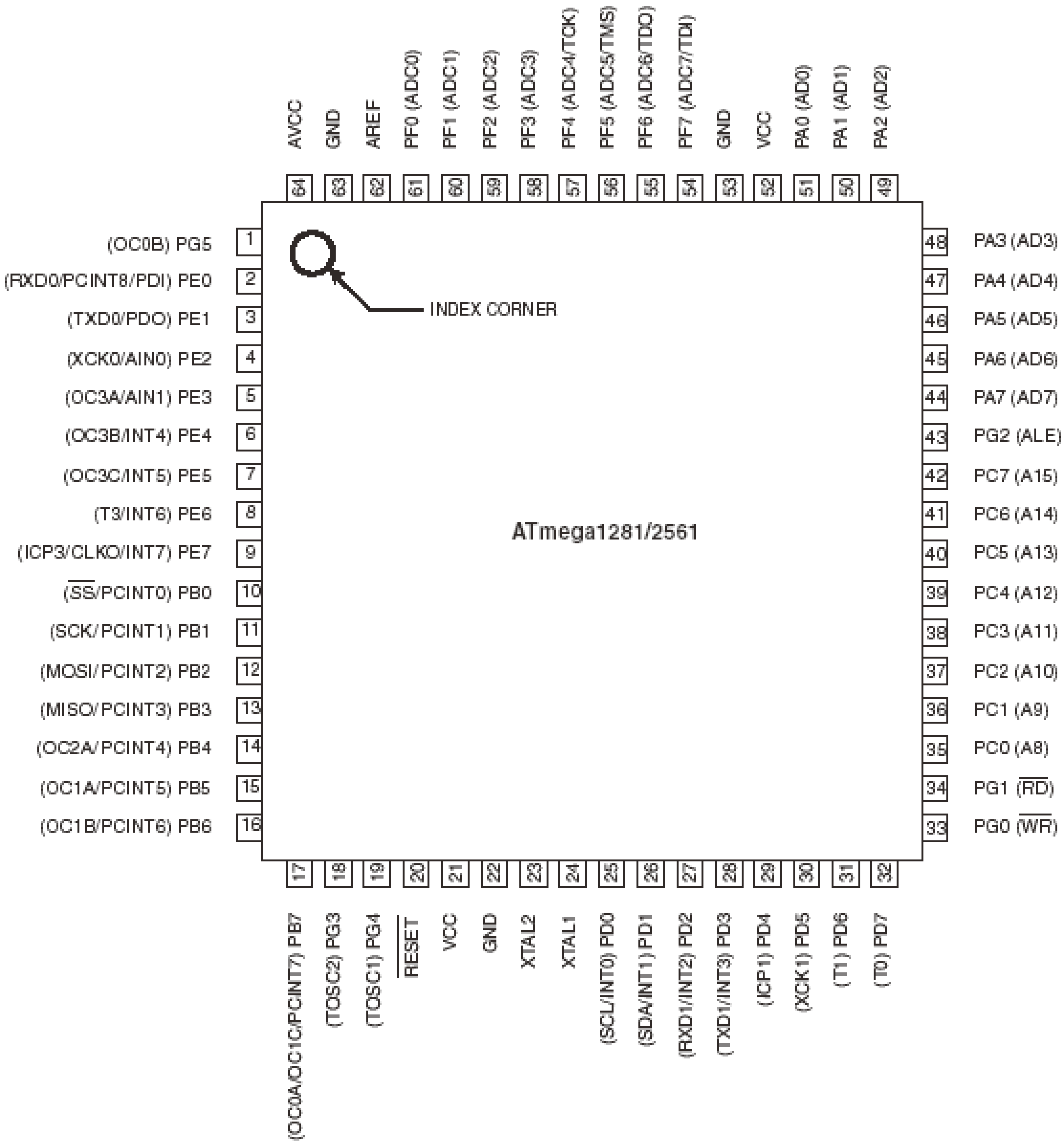
This page is intended to show the outline of the chip and to provide additional information that might not be clear from the data sheet.



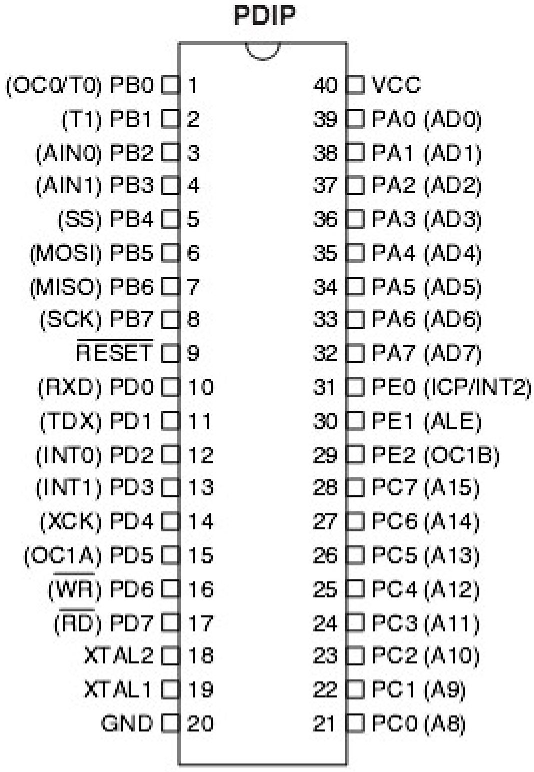
4.18.56 ATMEGA2560



4.18.57 ATMEGA2561

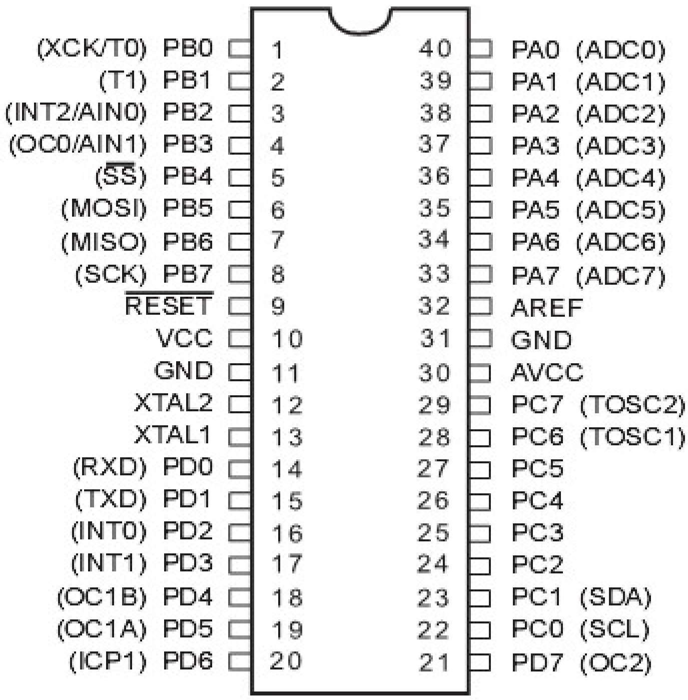


4.18.58 ATMEGA8515



4.18.59 ATMEGA8535

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.



4.19 Reference Designs

4.19.1 EM4095 RFID Reader

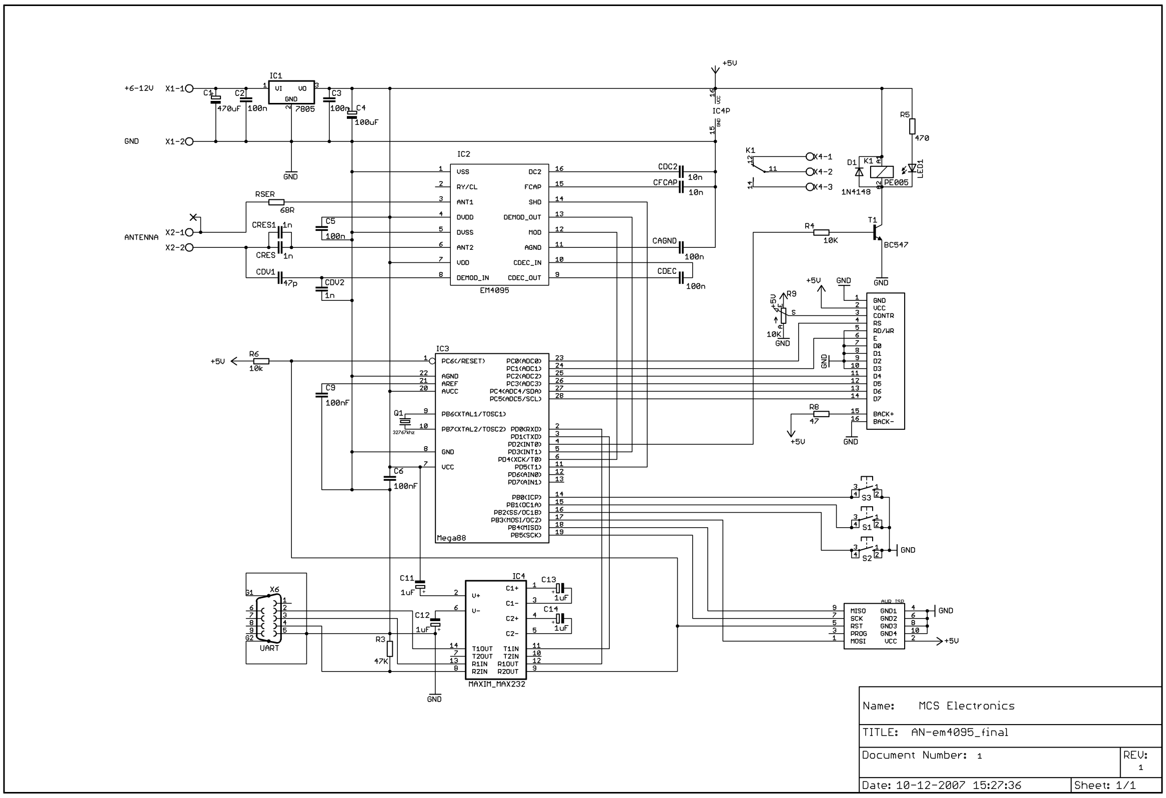
Introduction

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 www.mcselec.com

This topic describes the reference design.
The data sheets you can download from:

EM4095 (chip) , EM4102 (transponder)

The circuit

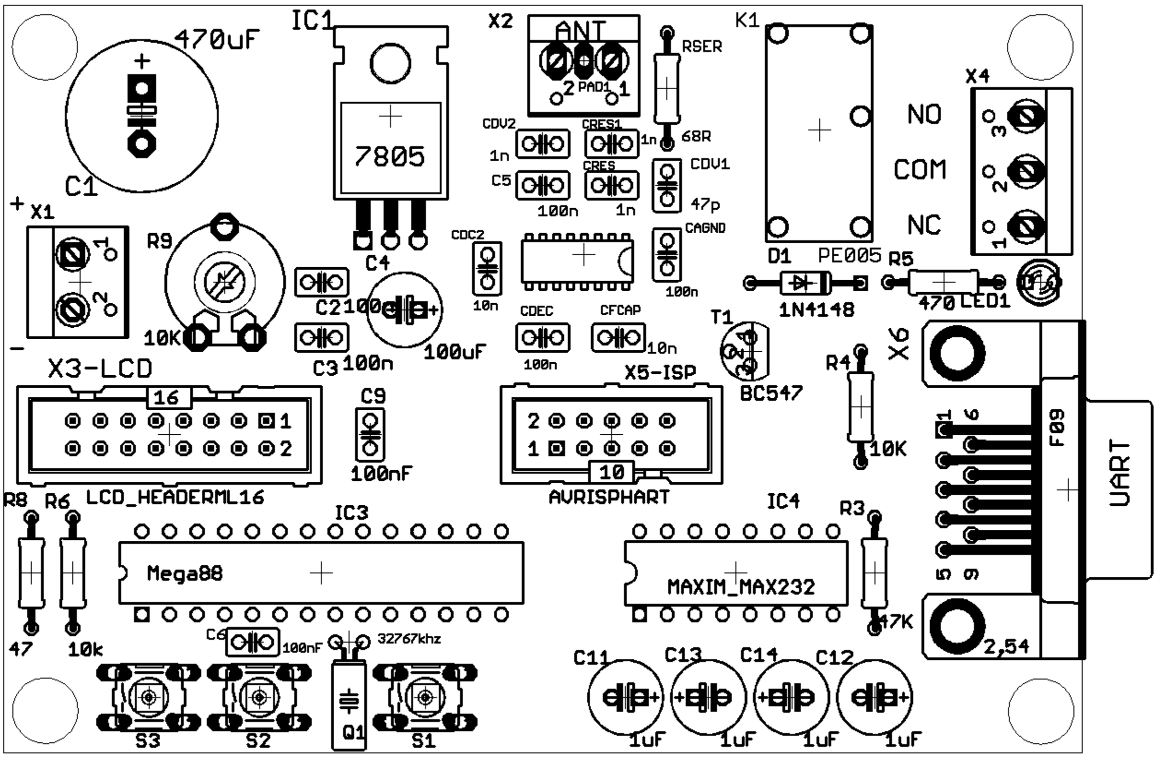


As you can see from the data sheets, the EM4095 needs little external hardware. A coil, capacitors that tune the coil for 125 KHz, are basically all that you need. IC1 is a voltage regulator that regulates the input voltage to 5V. (you can operate it from a 9V battery). The capacitors stabilize the output 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
X4	3-pin header
X5	10-pin boxed header

X6	DB-9 female connector
T1	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 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

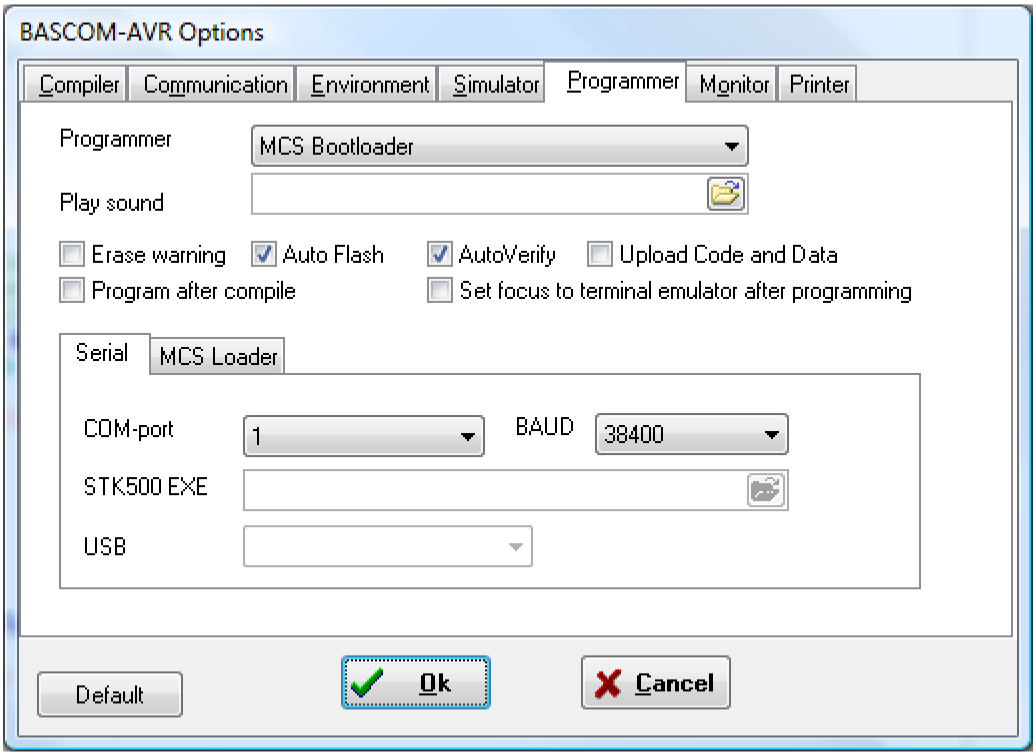
Now the PCB is ready. Make sure there are no solder drops on the PCB. You can measure with an Ohm-meter if there is a short circuit.
Measure pin 1 and pin 2 of IC1 (the voltage input) and pin 3 and pin 2 of IC1 (the voltage output).
When everything is ok, insert the MAX232 and the MEGA88.
You can connect 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.

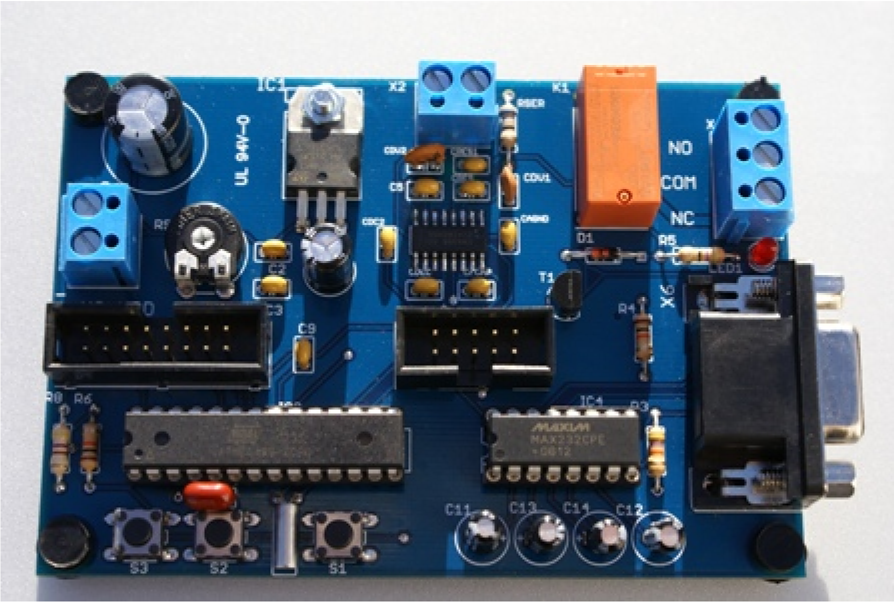
When you did not used the **MCS Bootloader** before, check the COM port settings and make sure the BAUD is set to 38400 as in the following screenshot:



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 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


Declare Function Havetag(b As Byte ) As Byte

'Make SHD and MOD low
_md Alias Portd.4
Config _md = Output
_md = 0

_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

Config Clock = Soft                                       'we use a clock
Config Date = Dmy , Separator = -
```

```
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 * 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 posi
Dim Etagcount As Eram Byte                                  ' number of stored tags
Dim Etags(100) As Eram Byte                                 'room for 20 tags
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 Int1 Checkints Nosave                                    'we use the INT1 pin al
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

Btags = Etagcount                                           ' get number of stored
If Btags = 255 Then                                         ' an empty cell is  FF
   Print "No tags stored yet"
   Btags = 0 : Etagcount = Btags                            ' reset and write to ee
Else                                                        ' we have some tags
   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"
   If Readhitag(tags(1)) = 1 Then                           'this will enable INT1
      Lowerline
      For J = 1 To 5
         Print Hex(tags(j)) ; ",";
         Lcd Hex(tags(j)) ; ","
```

```
        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 = 0                                           'relay off
        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
```

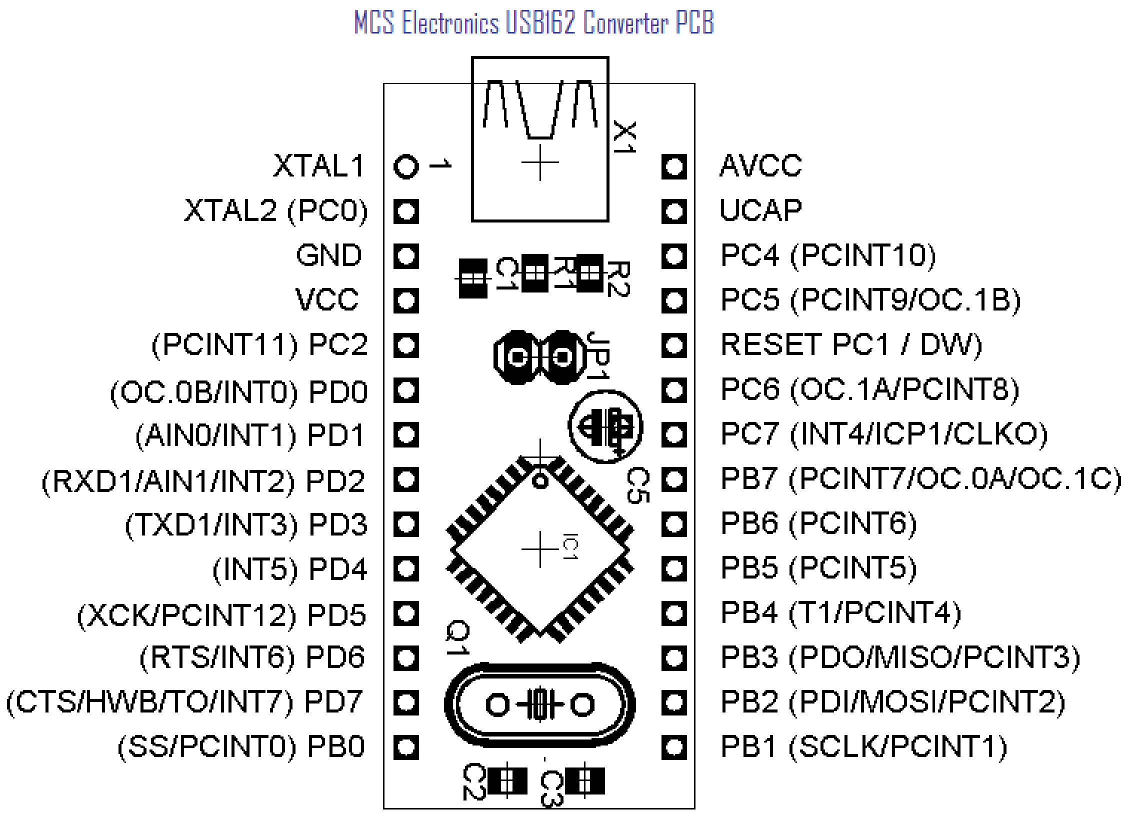
```
      Tel = 0
      For Idx = Tmp1 To Tmp2
         Incr Tel
         If Stags(idx) <> B(tel) Then                           'if they do not match
            Exit For                                            'exit and try next
         End If
      Next

      If Tel = 5 Then                                           'if we did found 5 matc
         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
  Return
```

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

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.



The USB module is available from the MCS Electronics online web shop.

Part V

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.

Statements that were removed

STATEMENT	DESCRIPTION
$LARGE	Not needed anymore.
$ROMSTART	Code always starts at address 0 for the AVR. Added again in 1.11.6.2
$LCDHEX	Use LCD Hex(var) instead.
$NOINIT	Not needed anymore. Added in 1.11.6.2
$NOSP	Not needed anymore
$NOBREAK	Can't be used anymore because there is no object code that can be used for it.
$OBJ	Removed.
BREAK	Can't be used anymore because there is no object code that can be used for it.
PRIORITY	AVR does no allow setting priority of interrupts
PRINTHEX	You can use Print Hex(var) now
LCDHEX	You can use Lcd Hex(var) now

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 ^ 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

STATEMENT	DESCRIPTION
ROTATE	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.
CONST	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.

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)
'	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 (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.

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.
- 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 0-byte. 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 BASCOM Reserved Words 246 , 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 DEFINT 535, DEFBIT 535, DEFBYTE 535 ,DEFWORD 535 ,DEFLNG 535 or DEFSNG 535.

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
3. Logical
4. Functional

Arithmetic

Arithmetic operators are +, - , * , \, / and ^.

- 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 \ 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

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 ... 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 ... 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 ... 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 ... 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 ... 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 ... 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:

1. 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:

.00000000000011010001101 * 2^0 <- Only 11 of 23 Bits retained
+.11111010111000010100011 * 2^0
____
.11111010111011100110000 * 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
A(c)= C
```

```
' print it
Print A(c)
Next
'you can add an offset to the index too
C = 0
A(c + 1)= 100
Print A(c + 1)
End
```

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 7. Bits are stored in bytes and the first dimensioned bit goes in the LS 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(lbl * 2)
LDI ZH , High(lbl * 2)
LPM ; get data into R0
Lbl:
```

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

* 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	Operands	Description	Operation	Flags	Clock
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,H	1

SBC	Rd, Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd = Rd - K - C	Z,C,N,V,H	1
AND	Rd, Rr	Logical AND	Rd = Rd · Rr	Z,N,V	1
ANDI	Rd, K	Logical AND with Immediate	Rd = Rd · 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
COM	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 · 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 Sbiw R24,K6	Rdl, K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K	Z,C,N,V,S	2
MUL	Rd,Rr	Multiply Unsigned	R1, R0 = Rd * Rr	C	2 *
BRANCH INSTRUCTIONS					
RJMP	K	Relative Jump	PC = PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC = Z	None	2
JMP	K	Jump	PC = k	None	3
RCALL	K	Relative Call Subroutine	PC = PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC = Z	None	3
CALL	K	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
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,H,	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H	1
CPI	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
SBRS	Rr, b	Skip if Bit in Register	If (Rr(b)=1) PC =	None	1 / 2

		Set	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	K	Branch if Equal	if (Z = 1) then PC = PC + k + 1	None	1 / 2
BRNE	K	Branch if Not Equal	if (Z = 0) then PC = PC + k + 1	None	1 / 2
BRCS	K	Branch if Carry Set	if (C = 1) then PC = PC + k + 1	None	1 / 2
BRCC	K	Branch if Carry Cleared	if (C = 0) then PC = PC + k + 1	None	1 / 2
BRSH	K	Branch if Same or Higher	if (C = 0) then PC = PC + k + 1	None	1 / 2
BRLO	K	Branch if Lower	if (C = 1) then PC = PC + k + 1	None	1 / 2
BRMI	K	Branch if Minus	if (N = 1) then PC = PC + k + 1	None	1 / 2
BRPL	K	Branch if Plus	if (N = 0) then PC = PC + k + 1	None	1 / 2
BRGE	K	Branch if Greater or Equal, Signed	if (N V= 0) then PC = PC+ k + 1	None	1 / 2
BRLT	K	Branch if Less Than, Signed	if (N V= 1) then PC = PC + k + 1	None	1 / 2
BRHS	K	Branch if Half Carry Flag Set	if (H = 1) then PC = PC + k + 1	None	1 / 2
BRHC	K	Branch if Half Carry Flag Cleared	if (H = 0) then PC = PC + k + 1	None	1 / 2
BRTS	K	Branch if T Flag Set	if (T = 1) then PC = PC + k + 1	None	1 / 2
BRTC	K	Branch if T Flag Cleared	if (T = 0) then PC = PC + k + 1	None	1 / 2
BRVS	K	Branch if Overflow Flag is Set	if (V = 1) then PC = PC + k + 1	None	1 / 2
BRVC	K	Branch if Overflow Flag is Cleared	if (V = 0) then PC = PC + k + 1	None	1 / 2
BRIE	K	Branch if Interrupt Enabled	if (I = 1) then PC = PC + k + 1	None	1 / 2
BRID	K	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	Load Indirect and Pre-Decrement	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	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	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	Z = Z - 1, Rd = (Z)	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	None	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	None	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		Load Program Memory	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	STACK = Rr	None	2
POP	Rd	Pop Register from Stack	Rd = STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
LSL	Rd	Logical Shift Left	Rd(n+1) =Rd(n), Rd(0)= 0,C=Rd	Z,C,N,V, H	1

			(7)		
LSR	Rd	Logical Shift Right	Rd(n) = Rd(n+1), Rd(7) =0, C=Rd (0)	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0) =C, Rd (n+1) =Rd(n), C=Rd(7)	Z,C,N,V, H	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=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) « Rd (7..4)	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	I/O(P, b) = 1	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)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) = T	None	1
SEC		Set Carry	C = 1	C	1
CLC		Clear Carry	C = 0	C	1
SEN		Set Negative Flag	N = 1	N	1
CLN		Clear Negative Flag	N = 0	N	1
SEZ		Set Zero Flag	Z = 1	Z	1
CLZ		Clear Zero Flag	Z = 0	Z	1
SEI		Global Interrupt Enable	I = 1	I	1
CLI		Global Interrupt 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	T	1
CLT		Clear T in SREG	T = 0	T	1
SHE		Set Half Carry Flag in SREG	H = 1	H	1
CLH		Clear Half Carry Flag in SREG	H = 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog Reset		None	1

*) 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

BAUD, BCD , BIN , BIN2GRAY , BINVAL , BIT , BITWAIT , BLINK , 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, INT0, 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

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
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
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
218	Register must be in range from R16-R31

219	INT0-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
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

Decimal	Octal	Hex	Binary	Value
000	000	000	00000000	NUL (Null char.)
001	001	001	00000001	SOH (Start of Header)
002	002	002	00000010	STX (Start of Text)
003	003	003	00000011	ETX (End of Text)
004	004	004	00000100	EOT (End of Transmission)
005	005	005	00000101	ENQ (Enquiry)
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	DC1 (XON) (Device Control 1)
018	022	012	00010010	DC2 (Device Control 2)
019	023	013	00010011	DC3 (XOFF)(Device Control 3)
020	024	014	00010100	DC4 (Device Control 4)
021	025	015	00010101	NAK (Negative Acknowledgement)
022	026	016	00010110	SYN (Synchronous Idle)
023	027	017	00010111	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	" (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	01000000	@	(AT symbol)
065	101	041	01000001	A	
066	102	042	01000010	B	
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	J	
075	113	04B	01001011	K	
076	114	04C	01001100	L	
077	115	04D	01001101	M	
078	116	04E	01001110	N	
079	117	04F	01001111	O	
080	120	050	01010000	P	
081	121	051	01010001	Q	
082	122	052	01010010	R	
083	123	053	01010011	S	
084	124	054	01010100	T	
085	125	055	01010101	U	
086	126	056	01010110	V	
087	127	057	01010111	W	
088	130	058	01011000	X	
089	131	059	01011001	Y	
090	132	05A	01011010	Z	
091	133	05B	01011011	[	(left/opening bracket)
092	134	05C	01011100	\	(back slash)
093	135	05D	01011101	]	(right/closing bracket)
094	136	05E	01011110	^	(caret/circumflex)
095	137	05F	01011111	_	(underscore)
096	140	060	01100000	`	
097	141	061	01100001	a	
098	142	062	01100010	b	
099	143	063	01100011	c	
100	144	064	01100100	d	
101	145	065	01100101	e	
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	l	
109	155	06D	01101101	m	
110	156	06E	01101110	n	
111	157	06F	01101111	o	
112	160	070	01110000	p	
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	x	
121	171	079	01111001	y	
122	172	07A	01111010	z	
123	173	07B	01111011	{	(left/opening brace)
124	174	07C	01111100	|	(vertical bar)
125	175	07D	01111101	}	(right/closing brace)
126	176	07E	01111110	~	(tilde)
127	177	07F	01111111	DEL	(delete)

Part VI

6 BASCOM Language Reference

6.1 $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 236 and assembler mnemonics 241

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
```

6.2 $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 Compiler Settings 89. It is stored in a configuration file. The $BAUD directive overrides the setting 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 UART0-TAB to view the output of the UART0, or to send data to this UART.

See also

$CRYSTAL 262 , BAUD 343

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
```

6.3 $BAUD1

Action

Instruct the compiler to set the baud rate for the second hardware UART.

Syntax

$BAUD1 = var

Remarks

Var	The baud rate that you want to use. This must be a 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 262 , BAUD 343 , $BAUD 257

Example

```
'-----------------------------------------------------------------------
--------
'copyright                  : (c) 1995-2005, MCS Electronics
'micro                      : Mega162
'suited for demo            : yes
'commercial addon needed    : no
'purpose                    : demonstrates BAUD1 directive and BAUD1
statement

'-----------------------------------------------------------------------
--------
$regfile = "M162def.dat"
$baud1 = 2400
$crystal= 14000000 ' 14 MHz crystal

Open "COM2:" For BINARY As #1

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
```

6.4 $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 749 , PSET 682 , CONFIG GRAPHLCD 416

Example

```
'-----------------------------------------------------------------
'                       (c) 1995-2005 MCS Electronics
'                   T6963C graphic display support demo
'-----------------------------------------------------------------

'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
'10       RESET          PORTC.4conneted
'11-18    D0-D7           PA
'19       FS             PORTC.5
'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
on
For X = 0 To 140
   Pset X , 20 , 255                                          ' set the
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
```

6.5 $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
$LOADER 285

Example

See BOOT.BAS from the samples dir. But better look at the $LOADER directive.

6.6 $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 Compiler Settings 89. 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 $BAUD 257, the compiler can calculate the proper settings for the UBR register. And second there are a number of routines like WAITMS 811, 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 257 , BAUD 343 , CONFIG CLOCKDIV 390

Example

```
$regfile = "m48def.dat"
$crystal = 4000000
$baud = 19200
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0

Print "Hello world"
End
```

6.7 $DATA

Action

Instruct the compiler to store the data in the DATA lines following the $DATA directive, in code memory.

Syntax
$DATA

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 267 , READEEPROM 699 , WRITEEEPROM 815 , DATA 501

ASM
NONE

Example
```
'--------------------------------------------------------------------------
--------
'copyright                    : (c) 1995-2005, MCS Electronics
'micro                        : AT90S2313
'suited for demo              : yes
'commercial addon needed      : no
'purpose                      : demonstrates $DATA directive

'--------------------------------------------------------------------------
--------
$regfile = "2313def.dat"
$baud = 19200
$crystal = 4000000                                            ' 4 MHz
crystal

Dim B As Byte
Readeeprom B , 0                                              'now B will
```

```
be 1
End


Dta:
$eeprom
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
$data
End
```

6.8 $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:

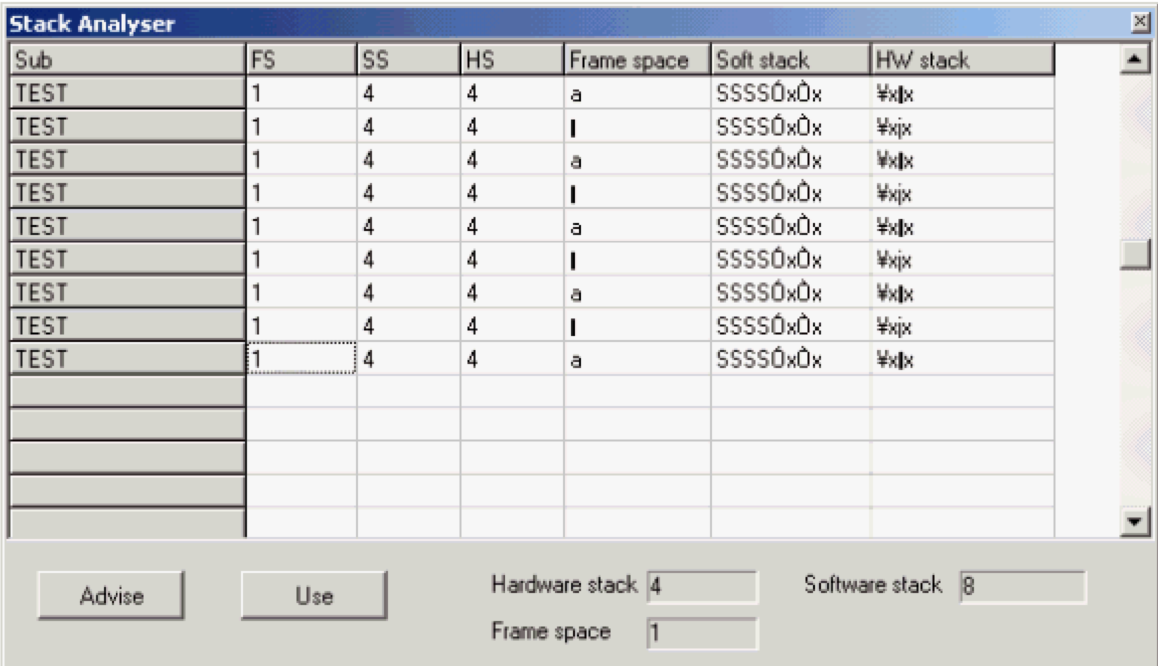
- Make the frame space 40, the soft stack 20 and the HW stack 50
- Add $DBG to the top of your program
- Add a DBG statement to every Subroutine or Function
- Open the terminal emulator and open a new log file. By default it will have the name of your current program with the .log extension
- Run your 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.



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
FS	Used frame space
SS	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 525

6.9 $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
```

6.10 $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 268

Example

NONE

6.11 $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 $EEPROM 267 and $DATA will cause the data to be stored into the EEPROM. The EEP file is a binary file. The $EEPROMHEX 268 directive can be used to create Intel HEX records in the EEP file

See also

$EEPROM 267 , READEEPROM 699 , WRITEEEPROM 815 , DATA 501 , $EEPROMHEX 268

ASM

NONE

Example

```
'--------------------------------------------------------------------------
--------
'copyright                    : (c) 1995-2005, MCS Electronics
'micro                        : AT90S2313
'suited for demo              : yes
'commercial addon needed      : no
'purpose                      : demonstrates $DATA directive

'--------------------------------------------------------------------------
--------
$regfile = "2313def.dat"
$baud = 19200
$crystal = 4000000                                          ' 4 MHz
crystal

Dim B As Byte
Readeeprom B , 0                                            'now B will
be 1
End


Dta:
$eeprom
Data 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
$data
End
```

6.12 $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

$EEPROMLEAVE 267

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 $eeprom hex in order to create an Intel Hex file.

This is how the EEP file content looks when using $eeprom hex

:0A00000001020304050A141E283251
:00000001FF

6.13 $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 283

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
```

6.14 $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

See also

$SWSTACK 306, $HWSTACK 271

Example

```
'--------------------------------------------------------------------------
---------
'name                     : adc.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstration of GETADC() function for 8535
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
```

6.15 $HWSTACK

Action

Sets the available space for the Hardware stack.

Syntax

$HWSTACK = var

Remarks

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

$SWSTACK 306 , $FRAMESIZE 270

Example

```
'--------------------------------------------------------------------------
---------
'name                     : adc.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstration of GETADC() function for 8535
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
```

6.16 $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.
Nosize	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 709, DATA 501 , READ 697

Example

```
$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

Restore L1                                                    ' set
pointer to label
Read Size                                                     ' 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
```

6.17 $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

$INC 272

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
```

6.18 $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.
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 Version()                                              'show date
and time of compilation

Print Portb
Do
  nop
Loop
End


'do not write a complete application in this routine.
'only perform needed init functions
_init_micro:
  Config Portb = Output
  Portb = 3
Return
```

6.19 $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.

See also

$LCDRS 280 , CONFIG LCD 430

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          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
'       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 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
!rCall _write_lcd                                            'put it on
LCD
End
```

6.20 $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 279

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
```

6.21 $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 277

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
```

6.22 $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 275 , CONFIG LCDBUS 430

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         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
'       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 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
!rCall _write_lcd                                             'put it on
LCD
End
```

6.23 $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

NONE

Example

NONE

6.24 $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:
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]
```

After you have saved your library in the **LIB** subdirectory you must compile it with the LIB Manager 78. 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 269

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
```

6.25 $LOADER

Action

Instruct the compiler to create a boot loader at the specified address.

Syntax

$LOADER = address

Remarks

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 261 , $LOADERSIZE 291

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

'$regfile = "m88def.dat"
'Const Loader = 88

'$regfile = "m32def.dat"
'Const Loaderchip = 32


'$regfile = "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
    $loader = $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
    $loader = $c00                                                 ' 1024 words
    Const Maxwordbit = 5                                           'Z5 is
maximum bit                                     '
    Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
#endif
#if Loaderchip = 161                                               ' Mega161
    $loader = $1e00                                                ' 1024 words
    Const Maxwordbit = 6                                           'Z5 is
maximum bit                                     '
#endif
#if Loaderchip = 162                                               ' Mega162
    $loader = $1c00                                                ' 1024 words
    Const Maxwordbit = 6                                           'Z5 is
maximum bit                                     '
    Config Com1 = 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
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
  Bstatus = Waitkey()                                             'wait for
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 _reset                                                       'goto the
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
    Bstatus = Waitkey()                                     'wait for
statuse byte
    Select Case Bstatus
       Case 1:                                              ' start of
heading, PC is ready to send
            Incr Bblocklocal                                'increase
local block count
            Csum = 1                                        'checksum is
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
               Else                                         'no match so
send nak
                  Print Chr(nak);
               End If
            Else
               Print Chr(nak);                              'blocks do
not match
            End If
       Case 4:                                              ' end of
transmission , file is transmitted
             Print Chr(ack);                                ' send ack
and ready

             Portb.3 = 0                                    ' simple
indication that we are finished and ok
             Goto _reset                                    ' start new
program
       Case &H18:                                           ' PC aborts
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
        Goto _reset                                                  'reset chip
     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
      lds r1, {vh}
      Spmcrval = 1 : Gosub Do_spm                                    'write value
into page at word address
      Wrd = Wrd + 2                                                  ' word
address increases with 2 because LS bit of Z is not used
      If Wrd = Maxword Then                                          ' page is
full
          Wrd = 0                                                    'Z pointer
needs wrd to be 0
          Spmcrval = 5 : Gosub Do_spm                                'write page
          Page = Page + 1                                            'next page
          Spmcrval = 3 : Gosub Do_spm                                ' erase
next page
          Spmcrval = 17 : Gosub Do_spm                               ' re-enable
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
  Bitwait Eecr.eepe , Reset                                          'wait for
eeprom

  Z = Page                                                           'make equal
to page
  Shift Z , Left , Maxwordshift                                      'shift to
proper place
  Z = Z + Wrd                                                        'add word
  lds r30,{Z}
  lds r31,{Z+1}

  Spmcsr = Spmcrval                                                  'assign
register
```

```
  spm                                                         'this is an
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
```

6.26 $LOADERSIZE

Action

Instruct the compiler that a boot loader is used so it will not overwrite the boot space.

Syntax

$LOADERSIZE = size

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 285

ASM

NONE

Example

NONE

6.27 $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
```

6.28 $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 81

Example

```
$NOCOMPILE
```

6.29 $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

$LOADER 285

Example

NONE

6.30 $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
$NOINIT 293

6.31 $PROG

Action
Directive to auto program the lock and fuse bits.

Syntax
$PROG LB, FB , FBH , FBX

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.
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.

Once 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.

See also

Programmers 99 , $PROG 294

6.32 $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

$PROG 294

ASM

NONE

Example

```
$REGFILE
```

6.33 $REGFILE

Action

Instruct the compiler to use the specified register file instead of the selected dat file.

Syntax

$REGFILE = "name"

Remarks

Name	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 51 viewer or the PDF 55 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 306 , $HWSTACK 271 , $FRAMESIZE 270

ASM

NONE

Example

$REGFILE = "8515DEF.DAT"

6.34 $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

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 Resource Editor 85 can import 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 85

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 lan
'--------------------------------------------------------------------------------
$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 $RESOURCE DUMP directi
'$resource Dump , "English" , "Dutch" , "German" , "Italian" 'we will use 4 languag
'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 D

$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
```

6.35 $ROMSTART

Action

Instruct the compiler to generate a hex file that starts at the specified address.

Syntax

$ROMSTART = address

Remarks

Address	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

$LOADER 285

ASM

NONE

Example

$ROMSTART = &H4000

6.36 $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 you can write a custom routine that puts the data into register R24 once it needs this data.

See also

$SERIALOUTPUT 304

Example

```
'--------------------------------------------------------------------------
---------
'name                     : $serialinput.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates $SERIALINPUT redirection of
serial input
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------
---------

$regfile = "m48def.dat"

'define used crystal
$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

'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
   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
  Err = 1                                                     'set error
variable
  ldi R24, 13                                                 'fake enter
so INPUT will end
Return
```

6.37 $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

$SERIALOUTPUT1 304 , $SERIALINPUT 300 , $SERIALOUTPUT 304

Example

See the $SERIALINPUT 300 sample

6.38 $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 $SERIALINPUT 300 and $SERIALOUTPUT 304 statements, but the $SERIALINPUT2LCD 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

$SERIALINPUT 300 , $SERIALOUTPUT 304 , $SERIALINPUT1 302 , $SERIALOUTPUT1 304

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                                   'this will
go to the LCD display
Loop
```

6.39 $SERIALOUTPUT

Action
Specifies that serial output must be redirected.

Syntax
$SERIALOUTPUT = label

Remarks

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
$SERIALINPUT 300 , $SERIALINPUT2LCD 303 , $SERIALINPUT1 302 , $SERIALOUTPUT1 304

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
```

6.40 $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

$SERIALINPUT1 302 , $SERIALINPUT 300 , $SERIALINPUT2LCD 303 , $SERIALOUTPUT 304

Example

See the $SERIALOUTPUT 304 example

6.41 $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
```

6.42 $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 271 , $FRAMESIZE 270

Example

```
'--------------------------------------------------------------------------
---------
'name                     : adc.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstration of GETADC() function for 8535
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
```

6.43 $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.

$TIMEOUT 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.

See Also

INPUT 622 , WAITKEY 810

Example

```
'--------------------------------------------------------------------------
------------------
'name                          : timeout.bas
```

```
'copyright                   : (c) 1995-2005, MCS Electronics
'purpose                     : demonstration of the $timeout option
'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

'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
```

6.44 $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

```
'--------------------------------------------------------------------------
------------------
'name                     : tiny15.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrate using ATtiny15
'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
A = A + B
nop
```

6.45 $WAITSTATE

Action

Compiler directive to activate external SRAM and to insert a WAIT STATE for a slower ALE signal.

CONFIG XRAM 482 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
$XA 310 , CONFIG XRAM 482

Example
$WAITSTATE

6.46 $XA

Action
Compiler directive to activate external memory access.

CONFIG XRAM 482 should 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
$WAITSTATE 309 , CONFIG XRAM 482

Example

$XA

6.47 $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 Options Compiler Chip 87 menu.
The $XRAMSIZE overrides this setting. It is important that $XRAMSTART precedes $XRAMSIZE

See also
$XRAMSTART 311

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : m128.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrate using $XRAM directive
'micro                    : Mega128
'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
$framesize = 40                                          ' default
use 40 for the frame space

$xramstart = &H1000

$xramsize = &H1000
Dim X As X
```

6.48 $XRAMSTART

Action

Specifies the location of the external RAM memory.

Syntax

$XRAMSTART = [&H]address

Remarks

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

$XRAMSIZE 310

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : m128.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrate using $XRAM directive
'micro                    : Mega128
'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
$framesize = 40                                             ' default
use 40 for the frame space

$xramstart = &H1000

$xramsize = &H1000
Dim X As X
```

6.49 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.
pin	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 326 , 1WRESET 314 , 1WREAD 317, 1WSEARCHFIRST 319, 1WSEARCHNEXT 321 , Using the 1wire protocol 157

Example

```
'-----------------------------------------------------------------------
---------
'name                     : 1wireSearch.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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

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
'       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) = 1wsearchfirst()

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 1wirecount 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
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line  1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)                                   'for
example look at pin PINB.1
End
```

6.50 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

1WRESET	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.
Pin	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 317 , 1WWRITE 326

Example

```
'--------------------------------------------------------------------------
---------
'name                     : 1wire.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates 1wreset, 1wwrite and 1wread()
'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
  1wreset                                                     '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
  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
1wwrite &H33 , 1 , Pinb , 2                                   'note that
now the number of bytes must be specified!
'1wwrite 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
```

```
    1wreset Pinb , I
    1wwrite &H33 , 1 , Pinb , I
    Ar(1) = 1wread(8 , Pinb , I)
    For A = 1 To 8
      Print Hex(ar(a));
    Next
    Print
  Next
  End
```

6.51 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.
Pin	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 old syntax. And the pin can be configured from the compiler options or with the CONFIG 1WIRE statement 377.

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 326 , 1WRESET 314

Example

```
'--------------------------------------------------------------------------
---------
'name                     : 1wire.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates 1wreset, 1wwrite and 1wread()
'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
  1wreset                                                                '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
  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
```

```
1wwrite &H33 , 1 , Pinb , 2                                   'note that
now the number of bytes must be specified!
'1wwrite 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
  1wreset Pinb , I
  1wwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8 , Pinb , I)
  For A = 1 To 8
    Print Hex(ar(a));
  Next
  Print
Next
End
```

6.52 1WSEARCHFIRST

Action

This statement reads the first ID from the 1wire bus into a variable(array).

Syntax

var2 = **1WSEARCHFIRST**()
var2 = **1WSEARCHFIRST**(port , pin)

Remarks

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 1wire 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 321 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 326 , 1WRESET 314 , 1WREAD 317, 1WSEARCHNEXT 321, 1WIRECOUNT 312

Example

```
'--------------------------------------------------------------------------
---------
'name                     : 1wireSearch.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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

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
'      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) = 1wsearchfirst()

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 1wirecount 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
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line  1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)                                   'for
example look at pin PINB.1
End
```

6.53 1WSEARCHNEXT

Action

This statement reads the next ID from the 1wire bus into a variable(array).

Syntax

var2 = **1WSEARCHNEXT**()
var2 = **1WSEARCHNEXT**(port , pin)

Remarks

var2	A variable or array that should be at least 8 bytes long that will be assigned with the 8 byte ID from the next 1wire 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. 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 326 , 1WRESET 314 , 1WREAD 317, 1WSEARCHFIRST 319, 1WIRECOUNT 312

Example

```
'--------------------------------------------------------------------------
---------
'name                       : 1wireSearch.bas
'copyright                  : (c) 1995-2005, MCS Electronics
'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

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
'      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) = 1wsearchfirst()

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 1wirecount 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
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
```

```
' optinal call it with pinnumber line  1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)                                   'for
example look at pin PINB.1
End
```

6.54 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 326 , 1WRESET 314 , 1WREAD 317, 1WSEARCHFIRST 319, 1WIRECOUNT 312

Example

```
'--------------------------------------------------------------------------
---------
'name                     : 1wireSearch.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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

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
'      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) = 1wsearchfirst()

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 1wirecount 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
1wverify Reg_no(1)
Print Err
'err =1 when the ID passed n reg_no() does NOT exist
' optinal call it with pinnumber line  1wverify reg_no(1),pinb,1

'As for the other 1wire statements/functions, you can provide the port
and pin number as anoption
'W = 1wirecount(pinb , 1)                                        'for
example look at pin PINB.1
```

End

6.55 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.
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 CONFIG 1WIRE 377 statement.

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 317 , 1WRESET 314

Example

```
'--------------------------------------------------------------------------
---------
'name                     : 1wire.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates 1wreset, 1wwrite and 1wread()
'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
  1wreset                                                    '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
  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
1wwrite &H33 , 1 , Pinb , 2                                  'note that
now the number of bytes must be specified!
'1wwrite 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
  1wreset Pinb , I
  1wwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8 , Pinb , I)
  For A = 1 To 8
    Print Hex(ar(a));
  Next
  Print
Next
End
```

6.56 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

6.57 ACOS

Action

Returns the arccosine of a single in radians.

Syntax

var = **ACOS**(x)

Remarks

Var	A floating point variable such as single or double, that is assigned with the ACOS of variable x.
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 690 , DEG2RAD 537 , COS 485 , SIN 751 , TAN 782 , ATN 339 , ASIN 338 , ATN2 340

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
x= 0.5 : S = Acos(x)
Print S
End
```

6.58 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

Example

```
'===============================================================================
' 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
$framesize = 64                                            ' specify the framesize

$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 Local1 As Byte
Dim Local_loop1 As Byte

Const Menu_id = &HAA                                       ' sub-menu ID byte
Const Routine_id = &H55                                    ' service routine ID by

'------------------------------------------------------------------------------

 Restore Main_menu                                         ' point to the start of
 sts {MenuPointer}, R8                                     ' }
 sts {MenuPointer + 1}, R9                                 ' } store the pointer t

Display_new_menu:

 lds R8, {MenuPointer}                                     ' }
 lds R9, {MenuPointer + 1}                                 ' } restore the pointer

 Read Entries                                              ' get the number of ent
 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 th
 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                         '
   ldI R30,4                                               ' }
   clr R1                                                  ' }
   add R8,R30                                              ' }
   adc R9,R1                                               ' }
     Next                                                  ' } calculate the locat
 End If

 Read Local1                                               ' get the menu entry's
 Read Dummy                                                ' to handle the uP expe

 If Local1 = Menu_id Then                                  ' did the user select a
  Read Dataptr
  sts {MenuPointer}, R8                                    ' }
  sts {MenuPointer + 1}, R9                                ' } store the start of
  Goto Display_new_menu
 End If

 Read Dataptr                                              ' get the address of th
 movw R30,R8
 icall                                                     ' pass control to the e

 Goto Display_new_menu                                     ' re-display the last m
```

```
'-------------------------------------------------------------------------------
'   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
 Data "2. Go to Menu 3"                                     ' } 2nd menu entry
 Data "3. Print 'Hello' message"                            ' } 3rd menu entry

 Adr2 Mainmenu_supporttable                                 ' point to this menu su

'-------------------------------------------------------------------------------

Mainmenu_supporttable:

 Data Menu_id                                               ' identify this menu en
 Adr2 Second_menu                                           ' address of next menu
```

```
 Data Menu_id                                              ' identify this menu en
 Adr2 Third_menu                                           ' address of next menu

 Data Routine_id                                           ' identify this menu en
 Adr Hello_message                                         ' address of the suppor

'--------------------------------------------------------------------------------
' Second Menu
'--------------------------------------------------------------------------------

Second_menu:

 Data 4                                                    ' number of entries in

 Data "SECOND MENU"                                        ' } menu title
 Data "1. 2nd Menu Entry #1"                               ' } 1st menu entry
 Data "2. 2nd Menu Entry #2"                               ' } 2nd menu entry
 Data "3. Go to previous menu"                             ' } 3rd menu entry

 Adr2 Secondmenu_supporttable                              ' point to this menu su

'--------------------------------------------------------------------------------

Secondmenu_supporttable:

 Data Routine_id                                           ' identify this menu en
 Adr 2nd_menu_1st_entry_routine                            ' support routine for 1

 Data Routine_id                                           ' identify this menu en
 Adr 2nd_menu_2nd_entry_routine                            ' support routine for 2

 Data Menu_id                                              ' identify this menu en
 Adr2 Main_menu                                            ' support routine for 3

'--------------------------------------------------------------------------------
' Third Menu
'--------------------------------------------------------------------------------

Third_menu:

 Data 4                                                    ' number of entries in

 Data "THIRD MENU"                                         ' } menu title
 Data "1. 3rd Menu Entry #1"                               ' } 1st menu entry
 Data "2. 3rd Menu Entry #2"                               ' } 2nd menu entry
 Data "3. Go to previous menu"                             ' } 3rd menu entry

 Adr2 Thirdmenu_supporttable                               ' point to this menu su

'--------------------------------------------------------------------------------

Thirdmenu_supporttable:

 Data Routine_id                                           ' identify this menu en
 Adr 3rd_menu_1st_entry_routine                            ' support routine for 1

 Data Routine_id                                           ' identify this menu en
 Adr 3rd_menu_2nd_entry_routine                            ' support routine for 2

 Data Menu_id                                              ' identify this menu en
 Adr2 Main_menu                                            ' support routine for 3
```

6.59 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.

See also

CONST 483

Example

```
'-------------------------------------------------------------------------------
'copyright                   : (c) 1995-2005, MCS Electronics
'micro                       : Mega48
'suited for demo             : yes
'commercial addon needed     : no
'purpose                     : demonstrates ALIAS

'-------------------------------------------------------------------------------
$regfile = "m48def.dat"
$crystal = 4000000                                          ' 4 MHz
crystal

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
Relais2 = 0
Relais3 = On
Relais4 = Off
```

End

6.60 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	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
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	T	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	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Extended ASCII

As people gradually required computers to understand additional characters and non-printing 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.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	□

See also

CHR 361

ASM

NONE

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 A As Byte , S As String * 10
s ="ABC"
A = Asc(s)
Print A                                                    'will print
65
End
```

6.61 ASIN

Action

Returns the arcsine of a single in radians.

Syntax

var = **ASIN**(x)

Remarks

Var	A float variable such as single or double that is assigned with the ASIN of variable x.
X	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 690 , DEG2RAD 537 , COS 485 , SIN 751 , TAN 782 , ATN 339 , ACOS 329 , ATN2 340

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
X = 0.5 : S = Asin(x)
Print S    '0.523595867

End
```

6.62 ATN

Action

Returns the Arctangent of a single in radians.

Syntax

var = **ATN**(single)

Remarks

Var	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 690 , DEG2RAD 537 , COS 485 , SIN 751 , TAN 782 , ATN2 340

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 = Atn(1) * 4
Print S ' prints 3.141593 PI
End
```

6.63 ATN2

Action

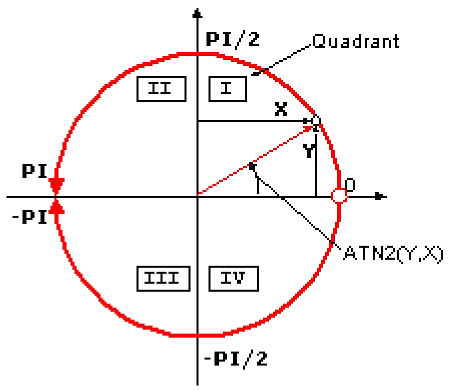
ATN2 is a four-quadrant arc-tangent.
While the ATN-function returns from -p/2 (-90°) to p/2 (90°), the ATN2 function returns the whole range of a circle from -p (-180°) to +p (180°). The result depends on the ratio of Y/X and the signs of X and Y.

Syntax

var = **ATN2**(y, x)

Remarks

Var	A single variable that is assigned with the ATN2 of variable single.
X	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
I	+	+	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 690 , DEG2RAD 537 , COS 485 , SIN 751 , TAN 782 , ATN 339

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
X = 0.5 : S = 1.1
S = Atn2(s , X)
Print S ' prints 1.144164676

End
```

6.64 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

CONFIG TCPIP 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, CLOSESOCKET 372 , SOCKETLISTEN 755 , BASE64ENC 342

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
$lib "tcpip.lbx"
Config Com1 = 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
```

6.65 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
CONFIG TCPIP 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, CLOSESOCKET 372 , SOCKETLISTEN 755 , BASE64DEC 341

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
$lib "tcpip.lbx"
Config Com1 = 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
```

6.66 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.
X	The channel number of the software UART.
Const	A numeric constant for the baud rate that you want to use.

Do not confuse the BAUD statement with the $BAUD 257 compiler directive.

And do not confuse $CRYSTAL 262 and CRYSTAL 497

$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 262 , $BAUD 257 , BAUD1 344

ASM

NONE

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
```

6.67 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.
X	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 262 and CRYSTAL 497

$BAUD1 overrides the compiler setting for the baud rate and BAUD1 will change the current baud rate.
BAUD1 = ... will work on the hardware UART.
BAUD #x, yyyy will work on the software UART.

See also

$CRYSTAL 262 , $BAUD 257 , $BAUD1 258 , BAUD 343

ASM

NONE

Example

```
'--------------------------------------------------------------------------
--------
'copyright                    : (c) 1995-2005, MCS Electronics
'micro                        : Mega162
'suited for demo              : yes
```

```
'commercial addon needed  : no
'purpose                  : demonstrates BAUD1 directive and BAUD1
statement

'-----------------------------------------------------------------------
--------
$regfile = "M162def.dat"
$baud1 = 2400
$crystal= 14000000 ' 14 MHz crystal

Open "COM2:" For BINARY As #1

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
```

6.68 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 655 , MAKEBCD 654

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
```

```
'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 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
```

6.69 BIN

Action

Convert a numeric variable into the binary string representation.

Syntax

Var = **Bin**(source)

Remarks

Var	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 604 , STR 775 , VAL 806 , HEXVAL 605 , BINVAL 348

ASM

NONE

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 B As Byte
' assign value to B
B = 45

Dim S As String * 10
'convert to string
S = Bin(b)

'assign value to portb
Portb = 33
```

```
Print Bin(portb)

'of course it also works for other numerics
End
```

6.70 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

STR 775 , HEXVAL 605 , HEX 604 , BIN 347 , VAL 806

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 String * 8
S = "11001100"

Dim B As Byte
' assign value to B
B = Binval(s)

Print B
End
```

6.71 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 603 , ENCODER 563

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.

Example

```
'--------------------------------------------------------------------------
------------------
'name                      : graycode.bas
'copyright                 : (c) 1995-2005, MCS Electronics
'purpose                   : show the Bin2Gray and Gray2Bin functions
'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
```

```
Dim B As Byte                                                    ' could be
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
```

6.72 BITWAIT

Action
Wait until a bit is set or reset.

Syntax
BITWAIT x , SET/RESET

Remarks

X	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 micr
```

```
$crystal = 8000000                                          ' used crystal frequenc
$baud = 19200                                               ' use baud rate
$hwstack = 32                                               ' default use 32 for th
$swstack = 10                                               ' default use 10 for th
$framesize = 40                                             ' default use 40 for th

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
```

6.73 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 663

Example

```
'--------------------------------------------------------------------------
---------
'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
$swstack = 10                                                    ' default
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
```

6.74 BLOAD

Action

Writes the Content of a File into SRAM

Syntax

BLoad sFileName, wSRAMPointer

Remarks

sFileName	(String) Name of the File to be read
wSRAMPointer	(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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

ASM

Calls	_BLoad	
Input	X: Pointer to string with filename	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
  Ar(i) = I                                                     ' fill the
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
```

6.75 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	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

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 635, CIRCLE 362 , BOXFILL 355

ASM

NONE

Example

```
'
--------------------------------------------------------------------------
----------------
' 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
$crystal = 8000000                                          '8 MHz

'First we define that we use a graphic LCD
Config Graphlcd = 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
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"
```

6.76 BOXFILL

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	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 635, CIRCLE 362 , BOX 353

ASM

NONE

Example

```
'create a bargraph effect
Boxfill(0 , 0) - (60 , 10) , 1
Boxfill(2 , 2) - (40 , 8) , 0
```

6.77 BSAVE

Action

Save a range in SRAM to a File

Syntax

BSave sFileName, wSRAMPointer, wLength

Remarks

sFileName	(String) Name of the File to be written
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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

ASM

Calls	_BSave	
Input	X: Pointer to string with filename	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

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
  Ar(i) = I                                            ' fill the
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
```

6.78 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.
N	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 449 , CLEAAR 365

Example

```
'-----------------------------------------------------------
NONE
```

6.79 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 358 , DECLARE 532 , SUB 777 , FUNCTION 530

ASM

NONE

Example

```
Declare Sub Test(Byval X As Byte, Byref Y As Byte, Z As Byte)
```

6.80 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](532), [SUB](777), [EXIT](567), [FUNCTION](530), [LOCAL](644)

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 A As Byte , B As Byte                                  'dimension
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
Test A , 5                                                 'alternative
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
  B2 = 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.

6.81 CHECKSUM

Action

Returns a checksum of a string.

Syntax

PRINT **Checksum**(var)
b = **Checksum**(var)

Remarks

Var	A string variable.
B	A numeric variable that is assigned with the checksum.

The checksum is computed by counting 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 = 255
For x = 1 To Len(s$)
  Check = check – ASC(mid$(s$,x,1))
Next
```

See also

CRC8 490 , CRC16 491 , CRC32 496

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 String * 10                                     'dim
variable
S = "test"                                               'assign
variable
Print Checksum(s)                                        'print value
(192)
End
```

6.82 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 335

Example

```
'--------------------------------------------------------------------------
-----------------
'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
'--------------------------------------------------------------------------
```

```
------------------

$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 Com1 = 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
```

6.83 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 635

Example

```
'-----------------------------------------------------------------------
------------------
'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
'-----------------------------------------------------------------------
------------------

$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            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        RESET          PORTC.4
'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 ,
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
   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
```

6.84 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 **R** for **R**eceive, and **W** for **W**rite, and y is 0 for the first UART, and 1 for the second UART.
The

See also

CONFIG SERIALIN 444, CONFIG SERIALOUT 449

ASM

Calls _BUF_CLEAR from MCS.LIB

Example

```
CLEAR SERIALIN
```

6.85 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
$LCD 275 , $LCDRS 280 , LCD 629 , SHIFTLCD 748 , SHIFTCURSOR 743 , SHIFTLCD 748 , INITLCD 616

Example
```
'--------------------------------------------------------------------------
------------------
'name                     : lcd.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
$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


$sim
'REMOVE the above command for the real program !!
'$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

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
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
!rCall _write_lcd                                           'put it on
LCD
End
```

6.86 CLOCKDIVISION

Action

Will set the system clock division available in the MEGA chips.

Syntax

CLOCKDIVISON = var

Remarks

Var	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 679

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
```

6.87 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. 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 INPUTHEX , INKEY, WAITKEY.

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.

See also

OPEN 669 , PRINT 679

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
```

6.88 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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, TCPREAD 786, SOCKETLISTEN 755

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : clienttest.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : start the easytcp.exe program and listen to
port 5000
'micro                    : Mega161
'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
Const Sel_control = 0                                        ' Confirm
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
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
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
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


$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 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

  If Ischarwaiting() <> 0 Then                             ' is there a
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
```

6.89 CONFIG

The CONFIG statement is used to configure the various hardware devices.

DIRECTIVE	RE-USABLE
CONFIG 1WIRE 377	NO
CONFIG ACI 379	YES
CONFIG ADC 380	NO
CONFIG ATEMU 382	NO
CONFIG BCCARD 384	NO
CONFIG CLOCK 387	NO
CONFIG CLOCKDIV 390	YES
CONFIG COM1 390	YES
CONFIG COM2 392 also COM3, COM4	YES
CONFIG DATE 395	NO
CONFIG DCF77 398	NO
CONFIG DEBOUNCE 403	NO
CONFIG GRAPHLCD 416	NO

CONFIG HITAG 405	NO
CONFIG I2CDELAY 408	NO
CONFIG I2CSLAVE 411	NO
CONFIG INPUT 413	NO
CONFIG INTx 414	YES
CONFIG KBD 421	NO
CONFIG KEYBOARD 421	NO
CONFIG LCD 426	NO
CONFIG LCDBUS 430	NO
CONFIG LCDMODE 433	NO
CONFIG LCDPIN 433	NO
CONFIG RC5 443	NO
CONFIG PORT 436	YES
CONFIG PRINT 438	NO
CONFIG PRINTBIN 439	NO
CONFIG SERIALIN 444	NO
CONFIG SERIALIN1 444	NO
CONFIG SERIALIN2 444	NO
CONFIG SERIALIN3 444	NO
CONFIG SERIALOUT 449	NO
CONFIG SERIALOUT1 449	NO
CONFIG SERIALOUT2 449	NO
CONFIG SERIALOUT3 449	NO
CONFIG SERVOS 454	NO
CONFIG PS2EMU 440	NO
CONFIG SINGLE 451	NO
CONFIG SDA 443	NO
CONFIG SCL 444	NO
CONFIG SPI 453	NO
CONFIG SHIFTIN 452	NO
CONFIG TCPIP 456	NO
CONFIG TWI 466	YES
CONFIG TWISLAVE 467	NO
CONFIG TIMER0 459	YES
CONFIG TIMER1 461	YES
CONFIG TIMER2 and 3 464	YES
CONFIG USB 471	NO
CONFIG WATCHDOG 478	YES
CONFIG WAITSUART 478	NO
CONFIG X10 480	NO
CONFIG XRAM 482	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.

6.90 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 314 , 1WREAD 317 , 1WWRITE 326 , 1WIRECOUNT 312, 1WRESET 314 , 1WSEARCHFIRST 319 , 1WSEARCHNEXT 321

Example

```
'--------------------------------------------------------------------------------
```

```
'name                     : 1wire.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates 1wreset, 1wwrite and 1wread()
'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 = 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) = 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
  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
1wwrite &H33 , 1 , Pinb , 2                              'note that
now the number of bytes must be specified!
'1wwrite 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
  1wreset Pinb , I
  1wwrite &H33 , 1 , Pinb , I
  Ar(1) = 1wread(8 , Pinb , I)
  For A = 1 To 8
    Print Hex(ar(a));
  Next
  Print
Next
End
```

6.91 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
COMPARE	Can be on or off. When switched ON, the TIMER1 in capture mode will trigger on ACI too.
TRIGGER	Specifies which comparator events trigger the analog comparator interrupts.

See also
NONE

Example
NONE

6.92 CONFIG ADC

Action

Configures the A/D converter.

Syntax

CONFIG ADC = single, PRESCALER = AUTO, REFERENCE = opt

Remarks

ADC	Running mode. May be SINGLE or FREE.
PRESCALER	A numeric constant for the clock divider. Use AUTO to let the compiler generate the best value depending on the XTAL
REFERENCE	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, m603, m128103	OFF AVCC	0
m165, m169, m325,m3250, m645, m6450, m329,m3290, m649, m6490,m48,m88,m168	OFF AVCC INTERNAL or INTERNAL_1.1	1
tiny15,tiny26	AVCC OFF INTERNAL INTERNALEXTCAP	2
tiny13	AVCC INTERNAL	3
tiny24,tiny44,tiny85	AVCC EXTERNAL or OFF INTERNAL or INTERNAL_1.1	4
m164,m324,m644,m640,m1280, m1281,m2561,m2560	AREF or OFF AVCC INTERNAL1.1 INTERNAL_2.56	5
tiny261,tiny461,tiny861, tiny25, tiny45,tiny85	AVCC EXTERNAL or OFF INTERNAL_1.1 INTERNAL_2.56_NOCAP INTERNAL_2.56_EXTCAP	7
CAN128, PWM2_3,USB1287, m128, m16, m163, m32, m323, m64	AREF or OFF AVCC INTERNAL or INTERNAL_2.56	8
	You may also use VALUE=value	

When you use VALUE=value, you may specify any value. The disadvantage is that 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.

See also

GETADC 585

Example

```
'--------------------------------------------------------------------------
---------
'name                     : adc.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstration of GETADC() function for 8535
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  ' 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 :
'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.
```

6.93 CONFIG ATEMU

Action

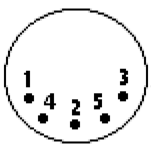
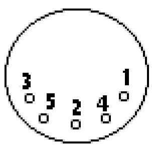
Configures the PS/2 keyboard data and clock pins.

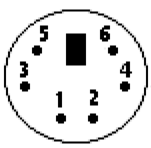
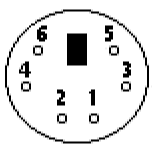
Syntax

CONFIG ATEMU = int , DATA = data, CLOCK=clock [,INIT=VALUE]

Remarks

Int	The interrupt used such as INT0 or INT1.
DATA	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.
INIT	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  (Plug)	Female  (Socket)	5-pin DIN (AT/XT): 1 - Clock 2 - Data 3 - Not Implemented 4 - Ground 5 - +5v

Male  (Plug)	Female  (Socket)	6-pin Mini-DIN (PS/2): 1 - Data 2 - Not Implemented 3 - Ground 4 - +5v

		5 - Clock 6 - Not Implemented

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 730

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : ps2_kbdemul.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : PS2 AT Keyboard emulator
'micro                    : 90S2313
'suited for demo          : no, ADD ONE 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 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)

Mark:                                                    ' send mark
Data 12 , &H3A , &HF0 , &H3A , &H1C , &HF0 , &H1C , &H2D , &HF0 , &H2D ,
 &H42 , &HF0 , &H42
'    ^ send 12 bytes
'           m                    a                   r
 k
```

6.94 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 B or D for most micro's. (PORTB and PORTD)
IO	The pin number that is connected to the IO of the BasicCard. Must be in the range from 0-7
RESET	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 861 , BCDEF 855 , BCCALL 855

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 = D , 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
```

```
'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)
S = "1+1+3"                                                   ' we want to
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
'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
```

```
'          B  a  s  i  c  C  a  r  d     Z  C   1  2  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(l)


'try the echotest command
Bcdef Echotest(byte)
Bccall Echotest(0 , &HC0 , &H14 , 1 , 0 , B)
Print B
End                                                            'end program
```

6.95 CONFIG CLOCK

Action

Configures the timer to be used for the TIME$ and DATE$ variables.

Syntax

CONFIG CLOCK = soft | USER [, GOSUB = SECTIC]

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.

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 TIME$ 793 and DATE$ 514.

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$ 793 , DATE$ 514 , CONFIG DATE 395

ASM

The following ASM routines are called from datetime.lib
_soft_clock. This is the ISR that gets called once per second.

Example

```
'--------------------------------------------------------------------------------
'name                     : megaclock.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : shows the new TIME$ and DATE$ reserved
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 * 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
  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
```

6.96 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

$CRYSTAL 262

Example

```
CONFIG CLOCKDIV = 8 'we divide 8 MHz crystal clock by 8 resulting in 1
MHz speed
```

6.97 CONFIG COM1

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 390 , CONFIG COMx 394

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
$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 m128def.dat file when you wanto to use the M128 in M128 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
```

6.98 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	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 390 , CONFIG COMx 394

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
$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 m128def.dat file when you wanto to use the M128 in M128 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
```

6.99 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

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
Stopbits	The number of stopbits : 1 or 2
Databits	The number of databits : 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.

See Also

CONFIG COM1 390 , CONFIG COM2 390

Example

```
'--------------------------------------------------------------------------------
'name                     :
'copyright                : (c) 1995-2008, MCS Electronics
'purpose                  : test for M2560 support
'micro                    : Mega2560
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------------

$regfile = "m2560def.dat"                                   ' specify the used micr
$crystal = 8000000                                          ' used crystal frequenc
$hwstack = 40                                               ' default use 32 for th
$swstack = 40                                               ' default use 10 for th
$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
```

6.100 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

CONFIG CLOCK 387 , DATE TIME functions 852 , DayOfWeek 504 , DayOfYear 513 , SecOfDay 717 , SecElapsed 716 , SysDay 780 , SysSec 777 , SysSecElapsed 779 , Time 794 , Date 516

Example

```
'--------------------------------------------------------------------------
-----------------
'name                     : megaclock.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : shows the new TIME$ and DATE$ reserved
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 * 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
  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

6.101 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 : **0** - 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.
UPDATE	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.
UPDATETIME	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.
POWERLEVEL	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.
SECONDTICKS	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.
DEBUG	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 CONFIG CLOCK 387

The DCF decoding routines use a status byte. This byte can be examined as in the example.
The bits have the following meaning.

Bit	Explanation
0	The last reading of the DCF pin.
1	This bit is reserved.
2	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.
3	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 526

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 RESET 707 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 : http://www.ptb.de/en/org/4/44/_index.htm

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".

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 : http://en.wikipedia.org/wiki/

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 standard-frequency 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.

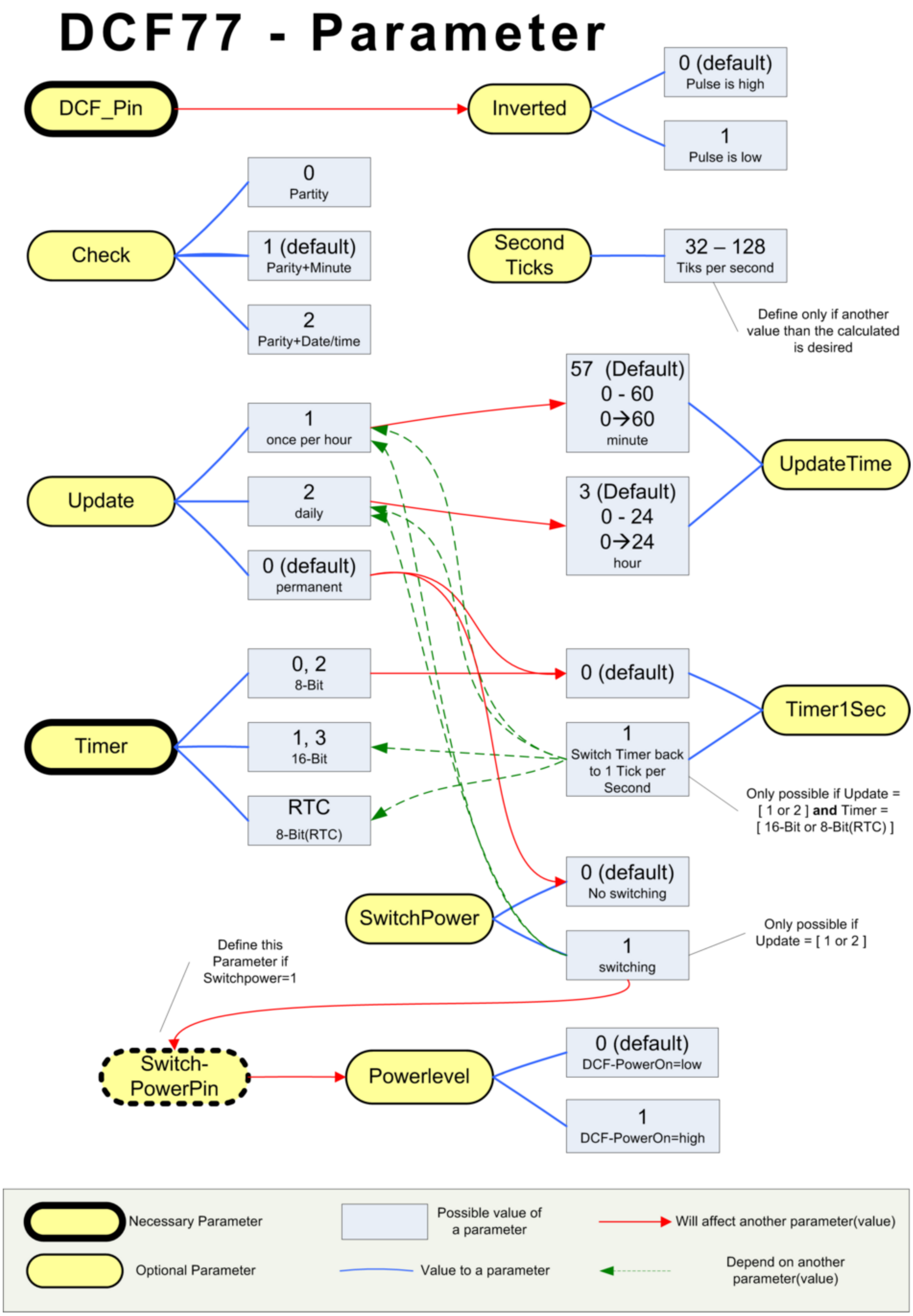
Since 2003, 14 previously unused bits of the time code have been used for civil defence emergency signals. This is still an experimental service, aimed to replace one day the German network of civil defense sirens.

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:



See also

[CONFIG DATE](395)

ASM

_DCF77 from DCF77.LBX is included by the compiler when you use the CONFIG statement.

Example

```
$regfile = "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

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
```

6.102 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 527

Example

```
'--------------------------------------------------------------------------------
'name                     : deboun.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates DEBOUNCE
'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

Config Debounce = 30                                        'when the
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

  '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
```

6.103 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 701

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 f
'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 HTRC
'
'the config statement will generate a number of constante and internal variables us
```

```
'the htrc110.lbx 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 INT0
Config Int0 = Change                                          'you must configure the

Enable Interrupts                                             ' enable global interru

Do
 If Readhitag(tags(1)) = 1 Then                               'check if there is a ne
     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 lab
 '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


Dim Tags(5) As Byte                                           'make sure the array is
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 Int1 Checkints Nosave                                    'we use the INT1 pin al
Config Int1 = Change                                        'we have to config so t
Enable Interrupts                                           'as last we have to ena


Do
   Print "Check..."

   If Readhitag(tags(1)) = 1 Then                           'this will enable INT1
      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
```

6.104 CONFIG I2CDELAY

Action

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 444 , CONFIG SDA 443

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : i2c.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: I2CSEND and I2CRECEIVE
'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

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 -------
I2csend &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
    I2cwbyte Addressw                                           'slave
address
    I2cwbyte Adres                                              'asdress of
EEPROM
    I2cwbyte 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
   I2cwbyte Addressw                                            'slave
adsress
   I2cwbyte Adres                                               'address of
EEPROM
   I2cstart                                                     'repeated
start
   I2cwbyte 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!

'   I2cstart                                                    'start
condition
'   I2cwbyte &B1010_0000                                        'slave
address
'   I2cwbyte H                                                  'high
address
'   I2cwbyte L                                                  'low address
'   I2cwbyte Value                                              'value to
write
'   I2cstop                                                     'stop
condition
'   Waitms 10
```

6.105 CONFIG I2CSLAVE

Action

Configures the I2C slave mode.

Syntax

CONFIG I2CSLAVE = address , INT = interrupt , TIMER = tmr

Remarks

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.
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 466

Example

```
'--------------------------------------------------------------------------
------------------
'name                       : i2c_pcf8574.bas
'copyright                  : (c) 1995-2005, MCS Electronics
'purpose                    : shows how you could use the I2C slave
library to create a PCF8574
'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
$framesize = 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 INT0 and TIMER0 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 = &B01000000                                    ' 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

'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
Do
   ' 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

'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!
'              The following labels are called from the slave library
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
!!!!!!!

'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
  _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)
Return


'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!

'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
'switches = PIND         ' get dip switch value
'switches = switches and &H1110 ' we only need the lower nibble without
the LS bit
'_i2c_slave_address = &H40 + switches ' set the proper address
```

6.106 CONFIG INPUT

Action

Instruct the compiler to modify serial input line terminator behaviour

Syntax

CONFIG INPUT = 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 622 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

See also

INPUT 622

ASM

NONE

Example

```
Config Input0 = CR , Echo = CRLF
Dim S as String * 20
Input "Hello ",s
```

6.107 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

state	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 INT0-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                     : spi-softslave.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : shows how to implement a SPI SLAVE with
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.
'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 INT0 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 Int0 Isr_sspi Nosave
'we enable the int0 interrupt
Enable Int0
'we configure the INT0 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

_ssspdr = 0                                                     ' we send a
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
```

6.108 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

Type	This must be 240 * 64, 128* 128, 128 * 64 , 160 * 48 , 240 * 128 or SED180*32. For SED displays use 128 * 64sed or 120* 64SED or SED180*32 For 132x132 color displays, use COLOR For EADOG128x64 use 128 * 64EADOGM
Dataport	The name of the port that is used to put the data on the LCD data pins db0-db7.

	PORTA for example.
Controlport	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.
	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.

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 366, 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

LOCATE 647 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 498 ON/OFF BLINK/NOBLINK can be used the same way as for text displays.

LCD 629 : can be handled the same way as for text displays.

SHOWPIC 749 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 682 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.

$BGF 259 "file.bgf" : inserts a BGF file at the current location

LINE 635 (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 353 (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.

BOXFILL 355 (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.

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.
CS	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

As the color display does not have a built in font, you need to generate the fonts yourself.
You can use the Fonteditor 125 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 749 , PSET 682 , $BGF 259 , LINE 635 , LCD 275 , BOX 353 , BOXFILL 355

Example

```
'-------------------------------------------------------------------------
------------------
'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
'-------------------------------------------------------------------------
------------------

$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) 2001-2008 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
'5         /WR            PORTC.0
'6         /RD            PORTC.1
'7         /CE            PORTC.2
'8         C/D            PORTC.3
'9         NC             not conneted
'10        RESET          PORTC.4
'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 ,
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
   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
```

6.109 CONFIG KBD

Action

Configure the GETKBD() function and tell which port to use.

Syntax

CONFIG KBD = PORTx , DEBOUNCE = value [, DELAY = value]

Remarks

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.
Delay	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.

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 593

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : getkbd.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : GETKBD
'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

'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
```

6.110 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.
KEYDATA	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 INT0 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.

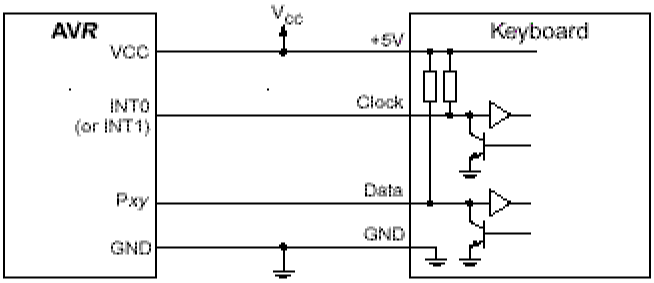
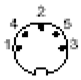
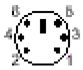


Table 1. AT Keyboard Connector Pin Assignments

AT Computer		
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 587

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
```

```
'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
  '    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 Kbdinput1      ; yes so try again
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

Keydata:
'normal keys lower case
Data 0 , 0 , 0 , 0 , 0 , 200 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , &H5E , 0
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 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
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
```

6.111 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.
Chipset 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.
CHIPSET 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
CONTRAST	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 INITLCD 616 statement.

See Also

CONFIG LCDPIN 433 , CONFIG LCDBUS 430

Example1

```
'--------------------------------------------------------------------------
------------------
'name                     : lcd.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
$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


$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

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
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
!rCall _write_lcd                                           'put it on
LCD
End
```

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                    '16*3 type LCD display
'other options for chipset are DOG163V3 for 3Volt operation


'Config Lcd = 16 * 3 , Chipset = Dogm163v3 , Contrast = &H702          '16*3 type LCD display
'The CONTRAST can be specified when the default value is not what you need


'The EADOG-M162 is also supported :
'Chipset params for the DOGM162 : DOG162V5, DOG162V3

Cls                                                          'Dit maakt het scherm leeg
Locate 1 , 1 : Lcd "Hello World"
Locate 2 , 1 : Lcd "line 2"
Locate 3 , 1 : Lcd "line 3"
End
```

6.112 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 426

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         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
'       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 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
!rCall _write_lcd                                   'put it on
LCD
```

6.113 CONFIG LCDMODE

Action

Configures the LCD operation mode and overrides the compiler setting.

Syntax

CONFIG LCDMODE = type

Remarks

Type	**PORT** Will drive the LCD in 4-bit port mode and is the default. In PORT mode you can choose different PIN's from different PORT's to connect to the upper 4 data lines of the LCD display. The RS and E can also be connected to a user selectable pin. This is very flexible since you can use pins that are not used by your design and makes the board layout simple. On the other hand, more software is necessary to drive the pins. **BUS** will drive the LCD in bus mode and in this mode is meant when you have external RAM and so have an address and data bus on your system. The RS and E line of the LCD display can be connected to an address decoder. Simply writing to an external memory location select the LCD and the data is sent to the LCD display. This means the data-lines of the LCD display are fixed to the data-bus lines. Use $LCD 275 = address and $LCDRS 280 = address, to specify the addresses that will enable the E and RS lines.

See also

CONFIG LCD 426 , $LCD 275 , $LCDRS 280

Example

```
Config LCDMODE = PORT 'the report will show the settings
Config LCDBUS = 4  '4 bit mode
LCD "hello"
```

6.114 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
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.

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 menu settings since the code makes clear which pins are used. The CONFIG statement overrides the Options setting.

See also

CONFIG LCD 426

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : lcd.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
$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


$sim
'REMOVE the above command for the real program !!
'$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

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
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
!rCall _write_lcd                                           'put it on
LCD
End
```

6.115 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. &B0001111, 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.

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.

See Also

AVR Internal hardware ports 135

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : port.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: PortB and PortD
'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 , 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 = 0
Do
  Incr Count
  Portb = 1
  For A = 1 To 8
    Rotate Portb , Left                                  'rotate bits
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 =&B11110000  'this will set portb1.0,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
```

6.116 CONFIG PRINT

Action

Configure the UART to be used for RS-485

Syntax

CONFIG PRINT0 = pin
CONFIG PRINT1 = pin

Remarks

pin	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.

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 439

Example

```
'--------------------------------------------------------------------------
'name                     : rs485.bas
'copyright                : (c) 1995-2006, MCS Electronics
'purpose                  : demonstrates
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------
$regfile = "m48def.dat"                                  ' we use the M48
$crystal = 8000000
$baud = 19200

$hwstack = 32
$swstack = 32
$framesize = 32

Config Print0 = Portb.0 , Mode = Set
Config Pinb.0 = Output                                   'set the direction yourself

Dim Resp As String * 10
Do
   Print "test message"
   Input Resp                                            ' get response
Loop
```

6.117 CONFIG PRINTBIN

Action

Configure PRINTBIN behavior

Syntax

CONFIG PRINTBIN = extended

Remarks

extended	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 438

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

Config Printbin = Extended
Dim A(1000)
Printbin A(1) ; 1000
```

6.118 CONFIG PS2EMU

Action

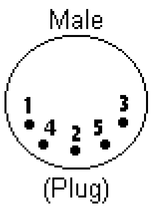
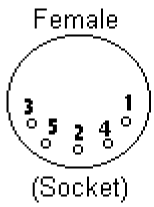
Configures the PS2 mouse data and clock pins.

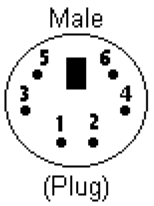
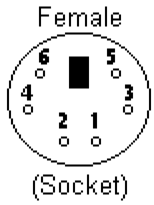
Syntax

CONFIG PS2EMU= int , DATA = data, CLOCK=clock

Remarks

Int	The interrupt used such as INT0 or INT1.
DATA	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.

		5-pin DIN (AT/XT): 1 - Clock 2 - Data 3 - Not Implemented 4 - Ground 5 - +5v

		6-pin Mini-DIN (PS/2): 1 - Data 2 - Not Implemented 3 - Ground 4 - +5v 5 - Clock 6 - Not Implemented

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 728, PS2MOUSEXY 685

Example

```
'-------------------------------------------------------------------------
-----------------
'name                     : ps2_emul.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : PS2 Mouse emulator
'micro                    : 90S2313
'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 "l" : 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:
Data 3 , &H08 , &H00 , &H01                                 ' mouse up
by 1 unit
```

6.119 CONFIG RC5

Action

Overrides the RC5 pin assignment from the Option Compiler Settings 90.

Syntax

CONFIG RC5 = pin [,TIMER=2]

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.

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 596

Example

```
CONFIG RC5 = PIND.5  'PORTD.5 is the RC5 input line
```

6.120 CONFIG SDA

Action

Overrides the SDA pin assignment from the Option Compiler Settings 90.

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.

See also

CONFIG SCL 444 , CONFIG I2CDELAY 408

Example

```
CONFIG SDA = PORTB.7  'PORTB.7 is the SDA line
```

6.121 CONFIG SCL

Action

Overrides the SCL pin assignment from the Option Compiler Settings 90.

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.

See also

CONFIG SDA 443 , CONFIG I2CDELAY 408

Example

```
CONFIG SCL = PORTB.5  'PORTB.5 is the SCL line
```

6.122 CONFIG SERIALIN

Action

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: • Serial**0**CharMatch (for SERIALIN or the first UART/UART**0**) • Serial**1**CharMatch (for SERIALIN**1** or the second UART/UART**1**) • Serial**2**CharMatch (for SERIALIN**2** or the third UART/UART**2**) • Serial**3**CharMatch (for SERIALIN**3** or the fourth UART/UART**3**) The following label names must be used when you check for any value: • Serial**0**ByteReceived (for SERIALIN or the first UART/UART**0**) • Serial**1**ByteReceived (for SERIALIN**1** or the second UART/UART**1**) • Serial**2**ByteReceived (for SERIALIN**2** or the third UART/UART**2**) • Serial**3**ByteReceived (for SERIALIN**3** or the fourth UART/UART**3**) 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_empty	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 UART**0**:
_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.

To clear the buffer, use CLEAR 365 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 make RTS '1' to signal to the sender, that the buffer is full. The sender 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 449 , ISCHARWAITING 626 , CLEAR 365

Example

```
'-----------------------------------------------------------------------------------
'name                     : rs232buffer.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : example shows the difference between normal
```

```
and buffered
'                           serial INPUT
'micro                    : Mega161
'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
$framesize = 40                                                 ' default
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"
Do
   'get a char from the UART

   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                          :
'copyright                        :  (c)  1995-2008,  MCS  Electronics
'purpose                          :  test  for  M2560  support
'micro                         :  Mega2560
'suited  for  demo             :  yes
'commercial  addon  needed     :  no
'-----------------------------------------------------------------------------

$regfile = "m2560def.dat"                                ' specify the used micro
$crystal = 8000000                                       ' used crystal frequency
$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


'$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 ,
Clockpol = 0
Config Com2 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Clockpol = 0
Config Com3 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Clockpol = 0
Config Com4 = 19200 , Synchrone = 0 , Parity = None , Stopbits = 1 , Databits = 8 ,
Clockpol = 0

Enable Interrupts
Config   Serialin = Buffered , Size = 20
Config   Serialin1 = Buffered , Size = 20 , Bytematch = 65
Config   Serialin2 = Buffered , Size = 20 , Bytematch = 66
Config   Serialin3 = Buffered , Size = 20 , Bytematch = All


'Open all UARTS
Open "COM2:" For Binary As #2
Open "COM3:" For Binary As #3
Open "COM4:" For Binary As #4


Print "Hello"                                            'first uart
Dim B1 As Byte , B2 As Byte , B3 As Byte , B4 As Byte
Dim Tel As Word , Nm As String * 16

'unremark to test second UART
'Input #2 , "Name ?" , Nm
'Print #2 , "Hello " ; Nm


Do
  Incr Tel
  Print Tel ; " test serial port 1"
  Print #2 , Tel ; " test serial port 2"
  Print #3 , Tel ; " test serial port 3"
  Print #4 , Tel ; " test serial port 4"

  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 If
   If B4 <> 0 Then
      Print #4 , B4 ; " from port 4"
   End If

   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
```

6.123 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 • SERIALOUT**1** : second UART/UART1 • SERIALOUT**2** : third UART/UART2 • SERIALOUT**3** : 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_PTRW**0** , byte that stores the head of the buffer
_RS_TAIL_PTRW**0** , byte that stores the tail of the buffer
_RS232OUTBUF**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.
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 444

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : rs232bufferout.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates how to use a serial output
buffer
'micro                    : Mega128
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------
------------------

$regfile = "m128def.dat"                                    ' specify
the used micro
$crystal = 4000000                                          ' used
crystal 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 = 0
Config 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
```

6.124 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
'line below remarked and active
Config Single = Scientific , Digits = 7

Dim S As Single
S = 1
Do
   S = S / 10
   Print S
Loop
```

6.125 CONFIG SHIFTIN

Action

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 743

6.126 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

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 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 SPI 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 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.

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
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.

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 764 , SPIOUT 766 , SPIINIT 765 , SPI 160 , SPIMOVE 765

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
```

6.127 CONFIG SERVOS

Action

Configures how much servo's will be controlled.

Syntax

CONFIG SERVOS = X , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = rl

Remarks

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.

X	The number of servo's you want to control. Each used servo will use one byte of SRAM.
PORT	The port pin the servo is attached too.
RL	The reload value for the ISR in uS.

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.

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
TIMER0
'This means that you can not use TIMER0 anymore
'The reload value specifies the interval of the timer in uS
'Config Servos = 2 , Servo1 = Portb.0 , Servo2 = Portb.1 , Reload = 10

Config Servos = 1 , Servo1 = Portb.0 , Reload = 10
'as an option you can use TIMER1
'Config Servos = 2 , Servo1 = 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 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
```

6.128 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]

Remarks

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.
TX	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 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 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755

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.

6.129 CONFIG TIMER0

Action

Configure TIMER0.

Syntax

CONFIG TIMER0 = COUNTER , PRESCALE= 1|8|64|256|1024 , EDGE=RISING/FALLING , CLEAR TIMER = 1|0
CONFIG TIMER0 = TIMER , PRESCALE= 1|8|64|256|1024

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:

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

Note that some new AVR chips have different pre scale values. You can use these.

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 START 769 and STOP 775 TIMER0 statements, use the CONFIG statement first.

Example

```
'--------------------------------------------------------------------------
------------------
'name                      : timer0.bas
'copyright                 : (c) 1995-2005, MCS Electronics
'purpose                   : shows how to use TIMER0 related statements
'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
'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 TIMER0 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 TIMER0 for OVF0, 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
```

6.130 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

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.

Here is the effect of the various options.

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

The TIMER1 also has two compare registers A and B

When the timer value matches a compare register, an action can be performed

COMPARE A	The action can be: 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 wanted 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

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : timer1.bas
'copyright                : (c) 1995-2005, MCS Electronics
```

```
'purpose                    : show using Timer1
'micro                      : 90S8515
'suited for demo            : yes
'commercial addon needed    : no
'------------------------------------------------------------------------
------------------

$regfile = "8515def.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 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 , Prescale = 1
'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 ,
Prescale = 1024
'Config Timer1 = Counter , Edge = Falling , Capture Edge = Rising

'To allow noise canceling you can also provide :
Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling ,
Noise Cancel = 1 , Prescale = 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
Compare1a = W
W = Compare1a


'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
Compare1a = 100
Compare1b = 200

'Or for better reading :
Pwm1a = 100
Pwm1b = 200
End
```

6.131 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

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

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.

EDGE	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

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

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 = Timer2
Timer2 = W
```

6.132 CONFIG TWI

Action

Configure the TWI (two wire serial interface).

Syntax

CONFIG TWI = 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 $CRYSTAL 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.

It is important that you specify the proper crystal frequency. Otherwise it will result in a wrong TWI clock frequency.

See also

$CRYSTAL 262

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
```

6.133 CONFIG TWISLAVE

Action

Configure the TWI Slave address and bit rate

Syntax

CONFIG TWISLAVE = address , BTR = value , BITRATE = value , SAVE=option [, GENCALL=value]

Remarks

Address	The slave address that is assigned to the slave chip. This must be an Even number. The address 0 is the general call address and may not be used. While a slave address is 7 bit since bit 0 is used to indicate read/write, BASCOM uses byte notation where you can ignore the last bit. The last bit will be set by BASCOM automatically.
BTR	Bytes to receive. With this constant you specify how many bytes will be expected when the slave receives bytes.
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.
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.

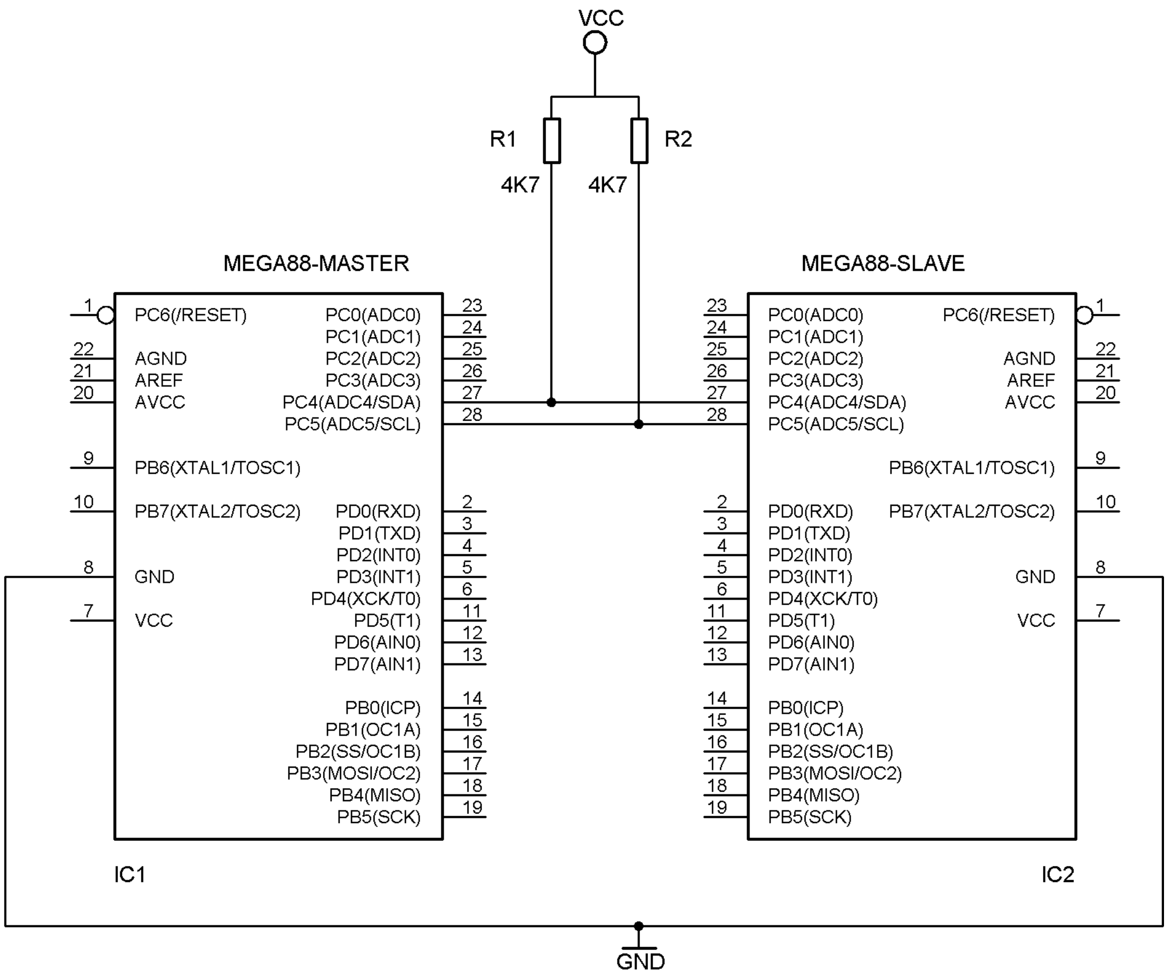
The variables Twi , Twi_btr and Twi_btw are created by the compiler. These are all bytes
The TWI interrupt is enabled but you need to enabled the global interrupt

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 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.

The TWI Slave code will save all used registers. But as it will call your BASIC application as the TWI interrupt occurs, your BASIC code could be in the middle of a PRINT statements.
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.
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.



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.

See also

CONFIG TWI 466

ASM

NONE

Example1(master)

```
'-----------------------------------------------------------------------------
'                        (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 = "M88def.dat"                                     ' the used chip
$crystal = 8000000                                          ' frequency used
$baud = 19200                                               ' baud rate
```

```
$lib "i2c_twi.lbx"                                          ' we do not use softwar

Config Scl = Portc.5                                        ' we need to provide th
Config Sda = Portc.4                                        ' we need to provide th

'On the Mega88,          On the PCF8574A
'scl=PC5 , pin 28            pin 14
'sda=PC4 , pin 27            pin 15


I2cinit                                                     ' we need to set the pi


Config Twi = 100000                                         ' wanted clock frequenc
'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
  Print X ; " " ; Err                                       ' show error
  Waitms 500                                                'wait a bit
Loop
End
```

Example2(slave)

```
'--------------------------------------------------------------------------------
'                             (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 = 8000000                                          ' 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 &H01
'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 wi
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
```

6.134 CONFIG USB

Action

Create settings related to USB.

Syntax

CONFIG USB = dev, Language= lang, Manufact= "man", Product="prod" , Serial="serial"

Remarks

Dev	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"
$crystal = 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
ID
Const Product_id = &H201D                                 ' MCS
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
Const Interface_class = 3                                   ' HID
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
Const Interface_class_mouse = 3                             ' HID Class
Const Interface_sub_class_mouse = 1                         ' Sub Class
is Mouse
Const Interface_protocol_mouse = 2                          ' Mouse
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                                   ' BULK =
0x02, INTERUPT = 0x03
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 =
0x02, INTERUPT = 0x03
Const Ep_out_length = 8
Const Ep_size_2 = Ep_out_length
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
         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                                   '
Is_usb_in_ready())
         Uedatx = 1
         Uedatx = 2
         Uedatx = 3
         Uedatx = 4
         Uedatx = 5
         Uedatx = 6
         Uedatx = 7
         Uedatx = 8
         Usb_ack_fifocon                                         ' Send data
over the USB
      End If
    End If
```

```
End Sub


Function Usb_user_read_request(type As Byte , Request As Byte) As Byte
  #if Mdbg
    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:
        '//      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)
Data Release_number%                                        '
Usb_write_word_enum_struc(RELEASE_NUMBER)
Data Man_index , Prod_index                                 ' MAN_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
Data Max_power                                              ' maximum
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                                     'length  ,
HID descriptor type
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%                                        'HID
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_1%                                                 ' Maximum
packet size for this EP   ,   Interval for polling EP in ms
Data Ep_interval_1

Data 7 , Endpoint_descriptor                                    ' Size Of
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 &H06 , &HFF , &HFF                                         ' 04|2   ,
Usage Page (vendordefined?)
Data &H09 , &H01                                                ' 08|1   ,
Usage      (vendordefined
Data &HA1 , &H01                                                ' A0|1   ,
Collection (Application)
'       // IN report
Data &H09 , &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 &H09 , &H04                                                ' 08|1   ,
Usage      (vendordefined)
Data &H09 , &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?)
Data &H75 , &H08                                                ' 74|1   ,
Report Size(8) = field size in bits = 1 byte
Data &H95 , Length_of_report_out                                '
94|1:ReportCount(size) = repeat count of previous item
Data &H91 , &H02                                                ' 90|1: OUT
report (Data,Variable, Absolute)
'       // Feature report
Data &H09 , &H06                                                ' 08|1   ,
Usage      (vendordefined)
Data &H09 , &H07                                                ' 08|1   ,
Usage      (vendordefined)
Data &H15 , &H00                                                ' 14|1   ,
LogicalMinimum(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 , &H04                                                       '
94|1:ReportCount
Data &HB1 , &H02                                                       ' B0|1:
Feature report
Data &HC0                                                              ' C0|0    ,
End Collection
```

6.135 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 669

Example
See OPEN 669 example for more details.

6.136 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.

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

START WATCHDOG 769, STOP WATCHDOG 775, RESET WATCHDOG 707

Example

```
'--------------------------------------------------------------------------
'name                     : watchd.bas
'copyright                : (c) 1995-2008, MCS Electronics
'purpose                  : demonstrates the watchdog timer
'micro                    : Mega88
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------

$regfile = "m88def.dat"                                     ' specify the used micr
$crystal = 8000000                                          ' used crystal frequenc
$baud = 19200                                               ' use baud rate
$hwstack = 32                                               ' default use 32 for th
$swstack = 32                                               ' default use 32 for th
$framesize = 40                                             ' default use 40 for th
Dim B As Byte
Dim Wdbit As Bit

Print "Watchdog test"
If Mcusr.wdrf = 1 Then                                      ' there was a WD overfl
   Wdbit = 1                                                'store the flag
End If
```

```
Config Watchdog = 2048                                          'reset after 2048 mSec
If Wdbit = 1 Then                                               'just print it now sinc
    Print "Micro was reset by Watchdog overflow"
End If

Start Watchdog                                                  'start the watchdog tim
Dim I As Word
For I = 1 To 1000
  Waitms 100
  Print I                                                       'print value
  B = Inkey()                                                   ' get a key from the se
  If B = 65 Then                                                'letter A pressed
     Stop Watchdog                                              ' test if the WD will s
  Elseif B = 66 Then                                            'letter B pressed
     Config Watchdog = 4096                                     'reconfig to 4 sec
     Start Watchdog                                             'CONFIG WATCHDOG will d
  Elseif B = 67 Then                                            'C pressed
     Config Watchdog = 8192                                     ' some have 8 sec timer
     '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

6.137 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 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	TX	Output pin. Add 1K pull up.

See also

X10DETECT 817 , X10SEND 819

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : x10.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : example needs a TW-523 X10 interface
'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


'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()
Print X                                                     ' 0 means
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
```

6.138 CONFIG XRAM

Action

Instruct the compiler to set options for external memory access.

Syntax

CONFIG XRAM = mode [, WaitstateLS=wls , WaitStateHS=whs]

Remarks

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.

See also

$XA 310 , $WAITSTATE 309

ASM

NONE

Example

```
CONFIG XRAM = Enabled, WaitstateLS=1 , WaitstateHS=2
```

6.139 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 334

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : const.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo for constants
'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 Z As String * 10
Dim B As Byte

'assign some constants
'constants dont use program memory
Const S = "test"
Const A = 5                                                 'declare a
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                                                      ' else is
optional
  #endif
#else
#endif
End
```

6.140 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 690 , DEG2RAD 537 , ATN 339 , SIN 751 , TAN 782

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
```

6.141 COSH

Action

Returns the cosine hyperbole of a single

Syntax

var = **COSH**(single)

Remarks

Var	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 690 , DEG2RAD 537 , ATN 339 , COS 485 , SIN 751 , TANH 792 , SINH 752

Example

Show sample 842

6.142 COUNTER0 and COUNTER1

Action

Set or retrieve the internal 16 bit hardware register.

Syntax

COUNTER0 = var var = COUNTER0	TIMER0 can also be used
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.

6.143 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 674 , POKE 675 , INP 618 , OUT 673

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
'--------------------------------------------------------------------------
------------------

$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 ofthe 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
```

6.144 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

Var	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 674 , POKE 675 , INP 618 , OUT 673

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
'--------------------------------------------------------------------------
------------------

$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 ofthe 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
```

6.145 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 360 , CRC16 491, CRC16UNI 494 , CRC32 496 , TCPCHECKSUM 783

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
$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

Ar(1) = 1
Ar(2) = 2
Ar(3) = 3

J = Crc8(ar(1) , 3)                                  'calculate
value which is 216
Print J
End
```

6.146 CRC16

Action

Returns the CRC16 value of a variable or array.

Syntax

Var = **CRC16**(source , L)

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.
L	The number of bytes to check.

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) = 1
ar(2) = 2
ar(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 \ 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
  shl = L And &HFFFF&
End Function
```

See also

CHECKSUM 360 , CRC8 490, CRC16UNI 494 , CRC32 496 , TCPCHECKSUM 783

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)                                              'calculate
value which is 216
W = Crc16(ar(1) , 3)                                             '24881
L = Crc32(ar(1) , 3)                                             '494976085
End
```

6.147 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.
polynomial	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 http://www.ross.net/crc/download/crc_v3.txt 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 360 , CRC8 490, CRC16 491 , CRC32 496 , TCPCHECKSUM 783

Example

```
'-------------------------------------------------------------------------------
'name                     : crc8-16-32.bas
'copyright                : (c) 1995-2008, MCS Electronics
'purpose                  : demonstrates CRC
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-------------------------------------------------------------------------------

$regfile = "m48def.dat"                                   ' specify the used micr
$crystal = 8000000                                        ' used crystal frequenc
$baud = 19200                                             ' use baud rate
$hwstack = 32                                             ' default use 32 for th
$swstack = 10                                             ' default use 10 for th
$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 outpu
```

```
Print Hex(crc16uni(s , 9 , 0 , &H1021 , 0 , 0))              'CRC-CCITT (0x0000)   3
Print Hex(crc16uni(s , 9 , &HFFFF , &H1021 , 0 , 0))         'CRC-CCITT (0xFFFF)   2
Print Hex(crc16uni(s , 9 , &H1D0F , &H1021 , 0 , 0))         'CRC-CCITT (0x1D0F)   E
Print Hex(crc16uni(s , 9 , 0 , &H8005 , 1 , 1))              'crc16                E
Print Hex(crc16uni(s , 9 , &HFFFF , &H8005 , 1 , 1))         'crc16-modbus         4


End
```

6.148 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.
Source	The source variable or first element of the array to get the CRC 32 value from.
L	The number of bytes to check.

CRC32 is used in communication protocols to check if there are no transmission errors.

See also

CHECKSUM 360 , CRC8 490, CRC16 491 , CRC16UNI 494 , TCPCHECKSUM 783

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)                                        'calculate
value which is 216
W = Crc16(ar(1) , 3)                                       '24881
L = Crc32(ar(1) , 3)                                       '494976085
End
```

6.149 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 669 , CLOSE 669

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
```

6.150 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 547 , LCD 629 , SHIFTLCD 748 , SHIFTCURSOR 743

Example

```
'--------------------------------------------------------------------------
------------------
'name                        : lcd.bas
'copyright                   : (c) 1995-2005, MCS Electronics
'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
$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


$sim
'REMOVE the above command for the real program !!
'$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

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
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
!rCall _write_lcd                                           'put it on
LCD
End
```

6.151 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 262 and $EEPROM 267 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 697 , RESTORE 709 , $DATA 262 , $EEPROM 267

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : readdata.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : READ,RESTORE
'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 Dta1                                               '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
W = Loadlabel(dta1)                                        ' loadlabel
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
```

6.152 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**(lSysSec)

Remarks

Target	A Byte – variable, that is assigned with the day of the week
BDayMonthYear	A Byte – variable, which holds the Day-value followed by Month (Byte) and Year (Byte)
StrDate	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 852 , CONFIG DATE 395 , CONFIG CLOCK 387, SYSDAY 780, SYSSEC 777

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"                                    ' 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

Const Clockmode = 1
'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
Config Date = Ymd , Separator = -                           ' 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
Strdate = "04-11-02"                                        ' we have
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

Lssyssec = 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


' =================== System Day ======================================
' 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



' ============================= 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
  I2crbyte _min , Ack                                   ' MINUTES
  I2crbyte _hour , Ack                                  ' Hours
  I2crbyte Weekday , Ack                                ' Day of
Week
  I2crbyte _day , Ack                                   ' Day of
Month
  I2crbyte _month , Ack                                 ' Month of
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
  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
  I2cwbyte _hour                                                ' Hours
  I2cstop
Return

#endif


Weekdays:
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" ,
"Saturday" , "Sunday"
```

6.153 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**(lSysSec)

Remarks

Target	A Integer, that is assigned with the Day of the Year
BDayMonthYear	A Byte, which holds the Day-value followed by Month(Byte) and Year (Byte)
StrDate	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 852 , SysSec 777 , SysDay 780

Example

See DayOfWeek 504

6.154 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.

Do 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$ 793 , CONFIG CLOCK 387 , DATE 516

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : megaclock.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : shows the new TIME$ and DATE$ reserved
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 * 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
  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
```

6.155 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**(lSysSec)
bDayMonthYear = **Date**(lSysDay)
bDayMonthYear = **Date**(strDate)

strDate = **Date**(lSysSec)
strDate = **Date**(lSysDay)
strDate = **Date**(bDayMonthYear)

Remarks

StrDate	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)
BDayMonthYear	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 variable-name of the first Byte, the one for Day must be passed to the function.

See also

Date and Time Routines 852 , DAYOFYEAR 513, SYSDAY 780

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"                                    ' 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

Const Clockmode = 1
'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
Config Date = Ymd , Separator = -                           ' 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
Strdate = "04-11-02"                                      ' we have
```

```
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



' =================== System Day ======================================
' 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
```

```
' ============================== 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()
Lssyssec = 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
  I2crbyte _min , Ack                                      ' MINUTES
  I2crbyte _hour , Ack                                     ' Hours
  I2crbyte Weekday , Ack                                   ' Day of
Week
  I2crbyte _day , Ack                                      ' Day of
Month
  I2crbyte _month , Ack                                    ' Month of
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
  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
  I2cwbyte _hour                                            ' Hours
  I2cstop
Return

#endif


Weekdays:
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" ,
"Saturday" , "Sunday"
```

6.156 DBG

Action

Prints debug info to the hardware UART

Syntax

DBG

Remarks

See $DBG 264 for more information

6.157 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 398

Example
```
Print = DCF77TimeZone()
```

6.158 DEBUG

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
```

6.159 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 403 , BITWAIT 350

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : deboun.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates DEBOUNCE
'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

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
```

6.160 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 615

Example

```
'--------------------------------------------------------------------------
```

```
------------------
'name                     : decr.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demostrate decr
'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 , I As Integer

A = 5                                                       'assign
value to a
Decr A                                                      'decrease
(by one)
Print A                                                     'print it


I = 1000
Decr I
Print I
End
```

6.161 DECLARE FUNCTION

Action

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).
Type	Type of the variable(s) and of the result. Byte,Word, Integer, Long, Single 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.

See the CALL 358 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 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

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 358, SUB 777

Example

```
'--------------------------------------------------------------------------------------
```

```
'name                     : function.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstration of user function
'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

'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
```

6.162 DECLARE SUB

Action

Declares a subroutine.

Syntax

DECLARE SUB TEST[([BYREF/BYVAL] var as type)]

Remarks

test	Name of the procedure.
Var	Name of the variable(s).
Type	Type of the variable(s). Byte, Word, Integer, Long, Single 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.
Use BYVAL to pass a copy of the variable.

See the CALL 358 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 358, SUB 777 , FUNCTION 530

Example

```
'-------------------------------------------------------------------------
------------------
'name                       : declare.bas
'copyright                  : (c) 1995-2005, MCS Electronics
'purpose                    : demonstrate using declare
'micro                      : Mega48
'suited for demo            : yes
'commercial addon 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
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
array
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 Test2                                                     'sub without
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
    B = X
    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
    B1(1) = 3                                                 'specify the
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
```

6.163 DEFxxx

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 l	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
```

6.164 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 LCD designer 76 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 Chr 361 () function.

LCD Text displays have a 64 byte memory that can be used to show your own custom 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 Û.
You can also use custom characters to create a bar graph or a music note.

See also
Tools LCD designer 76

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
```

6.165 DEG2RAD

Action

Converts an angle in to radians.

Syntax

var = **DEG2RAD**(Source)

Remarks

Var	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 690

Example

```
'--------------------------------------------------------------------------
--------
'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
```

6.166 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 809 , WAITMS 811

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : delay.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: DELAY, WAIT, WAITMS
'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

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
```

6.167 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.
Type	Bit, Byte, Word, Integer, Long, 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 $TINY 308 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 644.
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.

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.

ERAM variables only can be assigned to SRAM variables, and ERAM variables can be assigned to SRAM variables. You can not use an ERAM variable as you would use a normal variable.

Dim b as byte, bx as ERAM byte
B= 1
Bx=b ' write to EEPROM

B=bx ' read from EEPROM

See Also

CONST 483 , LOCAL 644

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
$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 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
```

6.168 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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688, FILELEN 571 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , WRITE 814 , INPUT 622

ASM

Calls	_Dir ; with file mask	_Dir0 ; without file mask
Input	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
```

6.169 DISABLE

Action

Disable specified interrupt.

Syntax

DISABLE interrupt

Remarks

Interrupt	Description
INT0	External Interrupt 0
INT1	External Interrupt 1
OVF0,TIMER0, COUNTER0	TIMER0 overflow interrupt
OVF1,TIMER1, COUNTER1	TIMER1 overflow interrupt
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.

See also

ENABLE 562

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : serint.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : serial interrupt example for AVR
'micro                    : 90S8535
'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

Const Cmaxchar = 20                                         'number of
characters

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 ISR
'we need to fill the buffer with spaces otherwise it will contain
garbage

Print "Start"

On Urxc Rec_isr                                             'define
serial receive ISR
Enable Urxc                                                 'enable
receive isr



Enable Interrupts                                           'enable
interrupts to occur

Do
  If B = 1 Then                                             'we received
something
     Disable Serial
     Print Buf                                              'print
buffer
     Print Bc                                               'print
character counter
```

```
      'now check for buffer full
      If Bc = Cmaxchar Then                                     'buffer full
         Buf = ""                                               'clear
         Bc = 0                                                 '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
```

6.170 DISKFREE

Action

Returns the free size of the Disk

Syntax

lFreeSize = **DISKFREE**()

Remarks

lFreeSize	A Long Variable, which is assigned with the available Bytes on the Disk in Bytes

This functions returns the free size of the disk in Bytes.

See also

INITFILESYSTEM 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 ,

KILL 627 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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
```

6.171 DISKSIZE

Action

Returns the size of the Disk

Syntax

lSize = **DISKSIZE**()

Remarks

lSize	A Long Variable, which is assigned with the capacity of the disk in Bytes

This functions returns the capacity of the disk.

See also

INITFILESYSTEM 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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
```

6.172 DISPLAY

Action

Turn LCD display on or off.

Syntax

DISPLAY ON / OFF

Remarks

The display is turned on at power up.

See also

LCD 629

Example

```
'--------------------------------------------------------------------------------
'name                     : lcd.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
$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


$sim
'REMOVE the above command for the real program !!
'$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
```

```
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
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
!rCall _write_lcd                                           'put it on
LCD
End
```

6.173 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 567 , WHILE-WEND 813 , FOR-NEXT 576

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : do_loop.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: DO, LOOP
'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

A = 1                                                     'assign a
var
Do                                                        'begin a
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
Do
  'Your code goes here
Loop
```

6.174 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

DriveReset 553 , DriveInit 553 , DriveGetIdentity 552 , DriveWriteSector 555 , DriveReadSector 554

ASM

Calls	_DriveCheck	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

Dim bError as Byte
bError = DriveCheck()

6.175 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
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer) , to which 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 551, DriveReset 553 , DriveInit 553 , DriveWriteSector 555 , DriveReadSector 554

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)
```

6.176 DriveInit

Action
Sets the AVR-Hardware (PORTs, PINs) attached to the Drive and resets the Drive.

Syntax
bErrorCode = **DRIVEINIT**()

Remarks

BErrorCode	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
DriveCheck 551, DriveReset 553 , DriveGetIdentity 552 , DriveWriteSector 555 , DriveReadSector 554

ASM

Calls	_DriveInit	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example
Dim bError as Byte
bError = DriveInit()

6.177 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

DriveCheck 551, DriveInit 553 , DriveGetIdentity 552 , DriveWriteSector 555 , DriveReadSector 554

ASM

Calls	_DriveReset	
Input	none	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

Dim bError as Byte
bError = DriveReset()

6.178 DriveReadSector

Action

Read a Sector (512 Bytes) from the (Compact Flashcard) Drive

Syntax

bErrorCode = **DRIVEREADSECTOR**(wSRAMPointer, lSectorNumber)

Remarks

bErrorCode	A Byte Variable, which is assigned with the error code of the function
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer) , to which the Sector from the Drive should be written
lSectorNumber	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 drive-sector 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 551, DriveReset 553 , DriveInit 553 , DriveGetIdentity 552 , DriveWriteSector 555

ASM

Calls	_DriveReadSector	
Input	Z: SRAM-Address of	X: Address of Long-variable with

	buffer *)	sectornumber
Output	r25: Errorcode	C-Flag: Set on Error

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
Dim lSectorNumber 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
lSectorNumber = 32

' Now read in sector 32 from CF-Card
bError = DriveReadSector(wSRAMPointer , lSectorNumber)
' Now Sector number 32 is in Byte-Array bBuffer

6.179 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
wSRAMPointer	A Word Variable, which contains the SRAM address (pointer), from which the Sector to the Drive should be written
lSectorNumber	A Long Variable, which give the sector number on the drive to 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 lSectornumber. 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 551, DriveReset 553 , DriveInit 553 , DriveGetIdentity 552 , DriveReadSector 554

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 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 read
Dim lSectorNumber as Long' Sector Number


' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer =VarPtr(aBuffer(1))

' Set Sectornumber

lSectorNumber = 3

' Now Write in sector 3 from CF-Card
bError = DriveWriteSector( wSRAMPointer , lSectorNumber)
```

6.180 DTMFOUT

Action

Sends a DTMF tone to the compare1 output pin of timer 1.

Syntax

DTMFOUT number, duration
DTMFOUT string , duration

Remarks

Number	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 562. 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

```
'--------------------------------------------------------------------------
------------------
'name                     : dtmfout.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
   Dtmfout Btmp , 50                                        ' dtmf out
on PORTB.3 for the 2313 for 500 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                 B
'7  8  9                 C
'*  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                A
'  13                B
'  14                C
'  15                D
```

6.181 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 622

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 Echo_On

Example

```
'-------------------------------------------------------------------------
------------------
'name                     : input.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: INPUT, INPUTHEX
'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 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
```

6.182 ELSE

Action

Executed if the IF-THEN expression is false.

Syntax

ELSE

Remarks

You don't have to use the ELSE statement in an IF THEN .. 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 613 , END IF 613 , SELECT-CASE 719

Example

```
'--------------------------------------------------------------------------------
-----------------
'name                     : if_then.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: IF, THEN, ELSE
'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

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
```

6.183 ENABLE

Action

Enable specified interrupt.

Syntax

ENABLE interrupt

Remarks

Interrupt	Description
INT0	External Interrupt 0
INT1	External Interrupt 1
OVF0,TIMER0, COUNTER0	TIMER0 overflow interrupt
OVF1,TIMER1, COUNTER1	TIMER1 overflow interrupt
CAPTURE1, ICP1	INPUT CAPTURE TIMER1 interrupt
COMPARE1A,OC1A or COMPARE1, OC1	TIMER1 OUTPUT COMPARE A interrupt 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

By default all interrupts are disabled.
To enable the enabling and disabling of interrupts use ENABLE INTERRUPTS.

Other chips might have additional interrupt sources such as INT2, INT3 etc.

See also

DISABLE 543

Partial Example

```
Enable Interrupts        'allow interrupts to be set
Enable Timer1         'enables the TIMER1 interrupt
```

6.184 ENCODER

Action

Reads pulses from a rotary encoder.

Syntax

Var = **ENCODER**(pin1, pin2, LeftLabel, RightLabel , wait)

Remarks

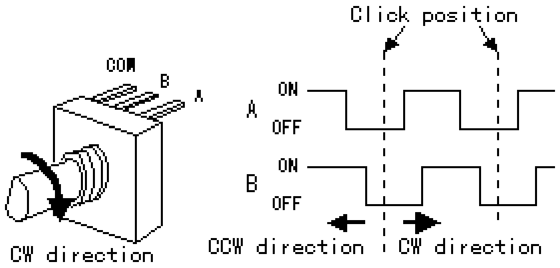
Var	The target variable that is assigned with the result
Pin1 and pin2	These are the names of the PIN registers to which the output of the encoder is connected. Both pins must be on the same PIN register. So Pinb.0 and Pinb.7 is valid while PinB.0 and PinA.0 is not.
LeftLabel	The name of the label that 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.

Example

```
'-----------------------------------------------------------------------
------------------
'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
$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

Print "Encoder test"
Dim B As Byte
'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
```

6.185 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 never-ending 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 775

Example

```
Print "Hello"      'print this
End               'end program execution and disable all interrupts
```

6.186 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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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

6.187 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.

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : exit.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: EXIT
'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 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
```

6.188 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 647 , LOG10 648

Example

```
'--------------------------------------------------------------------------
--------
'copyright                    : (c) 1995-2005, MCS Electronics
'micro                        : Mega88
'suited for demo              : no, but without the DOUBLE, it works for
DEMO too in M48
'commercial addon needed      : no
'purpose                      : demonstrate EXP function
'--------------------------------------------------------------------------
--------

$regfile = "m88def.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 = 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
```

6.189 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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688, FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571, WRITE 814 , INPUT 622

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
```

6.190 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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546, GET 582 , PUT 688, FILELEN 571 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , WRITE 814 , INPUT 622

ASM

Calls	_FileDateS ; with filename	_FileDateS0 ; for current file from DIR ()
Input	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
```

6.191 FILEDATETIME

Action
Returns the file date and time of a file

Syntax
Var = **FILEDATETIME** ()
Var = **FILEDATETIME** (file)

Remarks

Var	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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , GET 582 , PUT 688 , FILELEN 571 , FILEDATE 570 , FILETIME 572 , DIR 542 , WRITE 814 , INPUT 622

ASM

Calls	_FileDateTimeS	_FileDateTimeS0
Input		
Output		

Calls	_FileDateTimeB	_FileDateTimeB0
Input		
Output		

Example
See fs_subfunc_decl_lib.bas in the samples dir.

6.192 FILELEN

Action
Returns the size of a file

Syntax
lSize = **FILELEN** ()
lSize = **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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , WRITE 814 , INPUT 622

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
```

6.193 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.

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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , GET 582 , PUT 688 , FILELEN 571 , FILEDATE 570 , FILEDATETIME 571 , DIR 542 , WRITE 814 , INPUT 622

ASM

Calls	_FileTimeS ; with file param	_FileTimeS0 ; current file
Input	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

6.194 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.
X	The single to get the FIX of.

See Also

INT 625 , ROUND 714 , SGN 740

Example

```
'--------------------------------------------------------------------------
'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 mic
$crystal = 4000000                                   ' used crystal frequen
$baud = 19200                                        ' use baud rate
$hwstack = 32                                        ' default use 32 for th
$swstack = 10                                        ' default use 10 for th
$framesize = 40                                      ' default use 40 for th

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
```

6.195 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 615 , OPEN 669 , CLOSE 370, PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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
```

6.196 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 581

Example

```
'-----------------------------------------------------------------------------------------
```

```
'name                     : format.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : FORMAT
'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 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
```

6.197 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
value	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 piece 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.

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 567

Example

```
'--------------------------------------------------------------------------------
'name                     : for_next.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : 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
```

6.198 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 607 , UPPERLINE 806 , LOWERLINE 652 , THIRDLINE 793, LOCATE 647

Example

```
Thirdline
Lcd "Line 3"
Fourthline
Lcd "Line 4"
Home Third                                                   'goto home
on line three
Home Fourth
Home F                                         'first    letter    also    works
```

6.199 FRAC

Action

Returns the fraction of a single.

Syntax

var = **FRAC**(single)

Remarks

var	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

INT 625

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
$swstack = 40                                                  ' default
use 10 for the SW stack
$framesize = 40                                                ' default
use 40 for the frame space


Dim X As Single

X = 1.123456
Print X
Print Frac(x)
End
```

6.200 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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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
'This way we know that there is no more data we can read
Close #ff
```

6.201 FUSING

Action

FUSING returns a formatted string of a single value.

Syntax

target = **FUSING**(source, "mask")

Remarks

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 575 , STR 775

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : fusing.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : FUSING
'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 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
```

6.202 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.

#channel	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 assigned with the data from the file
Pos	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.
Length	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 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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

ASM

current position	goto new position first
Byte:	
_FileGetRange_1 Input: r24: File number X: Pointer to variable T-Flag cleared	_FileGetRange_1 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Word/Integer:	
_FileGetRange_2 Input: r24: File number X: Pointer to variable T-Flag cleared	_FileGetRange_2 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Long/Single:	
_FileGetRange_4 Input: r24: File number X: Pointer to variable T-Flag cleared	_FileGetRange_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	

_FileGetRange_Bytes Input: r24: File number r20: Count of Bytes X: Pointer to variable T-Flag cleared	_FileGetRange_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	
_FileGetRange Input: r24: File number r20/21: Count of Bytes X: Pointer to variable T-Flag cleared	_FileGetRange 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
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                                         ' get the
position of the next byte
Print Ltemp ; " LOC"                                        ' store the
location of the file pointer
Print Seek(#2) ; " = LOC+1"

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

'now open the file again and write only the single
Open "test.bin" For Binary As #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
```

6.203 GETADC

Action

Retrieves the analog value from the specified channel.

Syntax

var = **GETADC**(channel [,offset])

Remarks

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.
Offset	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.

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 START 769 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

CONFIG ADC 380

Example

```
'--------------------------------------------------------------------------
---------
'name                     : adc.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstration of GETADC() function for 8535
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
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 :
'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.
```

6.204 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 CONFIG KEYBOARD 423 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
B	32	F0,32	`	0E	F0,0E	INSERT	E0,70	E0, F0,70
C	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
E	24	F0,24	\	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
H	33	F0,33	TAB	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
K	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
M	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
O	44	F0,44	R CTRL	E0,14	E0,F0,14	KP -	7B	F0,7B
P	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
T	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
X	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	'	52	F0,52
5	2E	F0,2E	F12	07	F0,07	,	41	F0,41
6	36	F0,36	PRNT SCRN	E0,12 , E0,7C	E0,F0, 7C,E0, F0,12	.	49	F0,49
7	3D	F0,3D	SCROLL	7E	F0,7E	/	4A	F0,4A
8	3E	F0,3E	PAUSE	E1,14 ,77, E1, F0,14 , F0,77	-NONE-			

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 423 , GETATKBDRAW 591

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

'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
  '    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 Kbdinput1      ; yes so try again
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

Keydata:
'normal keys lower case
Data 0 , 0 , 0 , 0 , 0 , 200 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , &H5E , 0
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 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
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
```

6.205 GETATKBDRAW

Action
Reads a key from a PC AT keyboard.

Syntax
var = **GETATKBDRAW**()

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 GETATKBDRAW() function needs 2 input pins and a translation table for the keys. You can read more about this at the CONFIG KEYBOARD 423 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
GETATKBD 587 , CONFIG KEYBOARD 423

Example
See GETATKBD.BAS

6.206 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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, CLOSESOCKET 372 , SOCKETLISTEN 755 , GETDSTPORT 592

Partial Example

```
Dim L as Long
L = GetdstIP(i)  ' store current IP number of socket i
```

6.207 GETDSTPORT

Action

Returns the port number of the peer.

Syntax

Result = **GETDSTPort**(socket)

Remarks

Result	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

CONFIG TCPIP 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, CLOSESOCKET 372 , SOCKETLISTEN 755 , GETDSTIP 591

Partial Example

```
Dim P as Word
P = GetdstPORT(i)' store current port number of socket i
```

6.208 GETKBD

Action

Scans a 4x4 matrix keyboard and return the value of the key pressed.

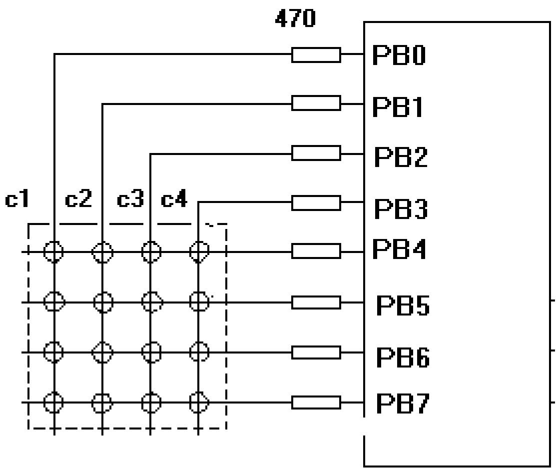
Syntax

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 Lookup() 650 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 421

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : getkbd.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : GETKBD
'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

'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
```

6.209 GETRC

Action

Retrieves the value of a resistor or a capacitor.

Syntax

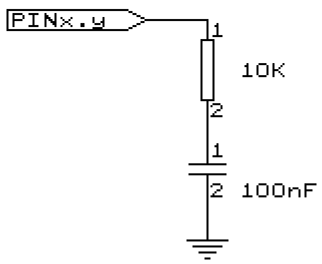
var = **GETRC**(pin , number)

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

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
```

6.210 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.

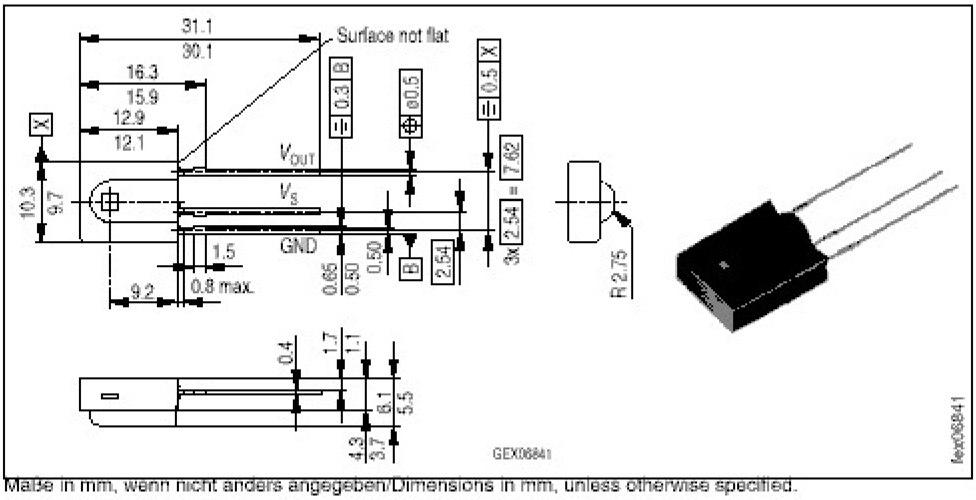
This statement is based on the AVR 410 application note. Since a timer is needed for accurate delays and background processing TIMER0 is used by this statement.

Also the interrupt of TIMER0 is 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.

GETRC5 supports extended RC5 code reception.

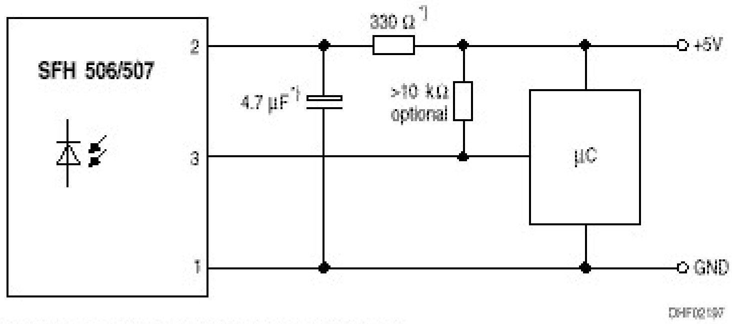
The SFH506-36 is used from Siemens. Other types can be used as well. The TSOP1736 has been tested with success.

IR-Empfänger/Demodulator-Baustein **SFH 506**
IR-Receiver/Demodulator Device



Maße in mm, wenn nicht anders angegeben/Dimensions in mm, unless otherwise specified.

For a good operation use the following values for the filter.



*) only necessary to suppress power supply disturbances

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.

See also

CONFIG RC5 443 , RC5SEND 691, RC6SEND 695

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
$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 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
Loop
End
```

6.211 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 726

ASM

NONE

Example

See SETTCPREGS 726

6.212 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 456, SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755

Partial Example

```
I = Getsocket(0 , Sock_stream , 5000 , 0)' get a new socket
```

6.213 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 426 , LCDAT 632, GLCDDATA 601

Example
NONE

6.214 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
CONFIG GRAPHLCD 426 , LCDAT 632, GLCDCMD 601

Example

NONE

6.215 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 603 , CALL 358 , RETURN 710

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : gosub.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: GOTO, GOSUB and RETURN
'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

Goto Continue
Print "This code will not be executed"

Continue:                                                   'end a label
with a colon
Print "We will start execution here"
Gosub Routine
Print "Back from Routine"
End
```

```
Routine:                                                     'start a
subroutine
  Print "This will be executed"
Return                                                       'return from
subroutine
```

6.216 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 602

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
```

6.217 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 349

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

```
'--------------------------------------------------------------------------
------------------
'name                     : graycode.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : show the Bin2Gray and Gray2Bin functions
'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

Dim B As Byte                                               ' could be
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
```

6.218 HEX

Action

Returns a string representation of a hexadecimal number.

Syntax

var = **HEX**(x)

Remarks

var	A string variable.
X	A numeric variable of data type Byte, Integer, Word, Long, Single or Double.

See also

HEXVAL 605 , VAL 806 , STR 775 , BIN 347 , BINVAL 348

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 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(w)
Print L ; Spc(3) ; Hex(l)
End
```

6.219 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.
X	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 604 , VAL 806 , STR 775 , BIN 347 , BINVAL 348

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 L As Long

Dim S As String * 8
Do
  Input "Hex value " , S
  L = Hexval(s)
  Print L ; Spc(3) ; Hex(l)
Loop
```

6.220 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 651 , HIGHW 607

Example

```
Dim I As Integer , Z As Byte
I = &H1001
Z = High(i)                                                   ' is 10 hex
```

```
or 16 dec
End
```

6.221 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 651 , HIGH 606

Example
```
Dim X As Word , L As Long
L = &H12345678
X = Highw(l)
Print Hex(x)
```

6.222 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
CLS 366 , LOCATE 647

For a complete example see LCD 629

Partial Example

```
Locate 2 , 1                                                    'set cursor
position
Lcd "*"                                                         'display this
Home Upper                                                      'select line
1 and return home
```

6.223 I2CINIT

Action

Initializes the SCL and SDA pins.

Syntax

I2CINIT

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.

ASM

The I2C routines are located in i2c.lib. _i2c_init is called.

See also

I2CSEND 609 , I2CSTART 610 , I2CSTOP 610 , I2CRBYTE 610 , I2CWBYTE 610 , I2C_TWI Library for using TWI 826

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
I2cinit

Dim X As Byte , Slave As Byte
X = 0                                                            'reset
variable
Slave = &H40                                                     'slave
address of a PCF 8574 I/O IC
I2creceive Slave , X                                             'get the
value
Print X                                                          'print it
```

6.224 I2CRECEIVE

Action

Receives data from an I2C serial slave device.

Syntax

I2CRECEIVE slave, var
I2CRECEIVE slave, var , b2W, b2R

Remarks

Slave	A byte, Word/Integer variable or constant with the slave address from the I2C-device.
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!

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.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

I2CSEND 609, I2CSTART 610 , I2CSTOP 610, I2CRBYTE 610 , I2CWBYTE 610

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
Dim X As Byte , Slave As Byte
X = 0                                                    'reset
variable
Slave = &H40                                             'slave
address of a PCF 8574 I/O IC
I2creceive Slave , X                                     'get the
value
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 the
received byte
End
```

6.225 I2CSEND

Action

Send data to an I2C-device.

Syntax

I2CSEND slave, var
I2CSEND slave, var , bytes

Remarks

Slave	The slave address off the I2C-device.
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.

When an error occurs, the internal ERR variable will return 1. Otherwise it will be set to 0.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

I2CRECEIVE 608 , I2CSTART 610, I2CSTOP 610 , I2CRBYTE 610 , I2CWBYTE 610

Example

```
Config Sda = Portb.5
Config Scl = Portb.7
Dim X As Byte , A As Byte , Bytes As Byte
X = 5                                                   'assign
variable to 5
Dim Ax(10)as Byte
Const Slave = &H40                                      'slave
address of a PCF 8574 I/O IC
I2csend Slave , X                                       'send the
value or


For A = 1 To 10
  Ax(a) = A                                             'Fill
dataspace
Next
Bytes = 10
I2csend Slave , Ax(1) , Bytes
End
```

6.226 I2START,I2CSTOP, I2CRBYTE, I2CWBYTE

Action

I2CSTART generates an I2C 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
I2CSTOP
I2CRBYTE var, ack/nack
I2CWBYTE val

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.

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.

ASM

The I2C routines are located in the i2c.lib/i2c.lbx files.

See also

I2CSEND 609 , I2CRECEIVE 608 , I2CSTART 610 , I2CSTOP 610, I2CRBYTE 610 , I2CWBYTE 610

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : i2c.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: I2CSEND and I2CRECEIVE
'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

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 -------
I2csend &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
    I2cwbyte Addressw                                       'slave
address
    I2cwbyte Adres                                          'asdress of
EEPROM
    I2cwbyte 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
   I2cwbyte Addressw                                        'slave
adsress
   I2cwbyte Adres                                           'address of
EEPROM
   I2cstart                                                 'repeated
start
   I2cwbyte 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!

'   I2cstart                                                   'start
condition
'   I2cwbyte &B1010_0000                                       'slave
address
'   I2cwbyte H                                                 'high
address
'   I2cwbyte L                                                 'low address
'   I2cwbyte Value                                             'value to
write
'   I2cstop                                                    'stop
condition
'   Waitms 10
```

6.227 IDLE

Action

Put the processor into the idle mode.

Syntax

IDLE

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.

See also

POWERDOWN 678

Example

IDLE

6.228 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 560

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"
```

6.229 INCR

Action

Increments a variable by one.

Syntax

INCR var

Remarks

Var	Any numeric variable.

See also

DECR 529

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : incr.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: INCR
'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

A = 5                                                       'assign
value to a
Incr A                                                      'inc (by
one)
Print A                                                     'print it
End
```

6.230 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 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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
```

6.231 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.
If fore some reason you would like to initialize it again you can use the INITLCD statement.
For example in 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 as port pin, you can use INITLCD after you have set the WR pin to logic 0.

ASM

The generated ASM code :
Rcall _Init_LCD

See also

LCD 275

Example

NONE

6.232 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.
Channel	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 810 , ISCHARWAITING 626

Example

```
'-------------------------------------------------------------------------------
```

```
'name                     : inkey.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: INKEY , WAITKEY
'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 , S As String * 2
Do
   A = Inkey()                                              'get ascii
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 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
```

6.233 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 Compiler Chip Options 87.

See also

OUT 673 , PEEK 674 , POKE 675

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
$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 ofthe 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
```

6.234 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.
var2	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 669 and CLOSE. 669

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 681

Example

```
Dim A As Byte , C As Integer
Inputbin A , C 'wait for 3 characters
End
```

6.235 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 622 , ECHO 558 , INPUTBIN 620

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : input.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: INPUT, INPUTHEX
'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 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
```

6.236 INPUT

Action

Allows input from the keyboard or file 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.
Ch	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.

See also

INPUTHEX 620 , PRINT 679 , ECHO 558 , WRITE 814 , INPUTBIN 620

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : input.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: INPUT, INPUTHEX
'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 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
```

6.237 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 766

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : instr.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : INSTR function demo
'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 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
```

6.238 INT

Action

Returns the integer part of a single or double.

Syntax

var = **INT**(source)

Remarks

Var	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 579 , FIX 573 , ROUND 714

Example

```
'--------------------------------------------------------------------------
------------------
'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
```

6.239 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 456

6.240 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 810 , INKEY 617

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

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
```

6.241 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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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"
```

6.242 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 797

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

```
$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 * 12 , Z As String * 12
S = "Hello World"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z
End
```

6.243 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 PRINT 679 statement. So SPC 763() 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
$LCD 275 , $LCDRS 280 , CONFIG LCD 426 , SPC 763 , CLS 366 , INITLCD 616 , SHIFTLCD 748 , SHIFTCURSOR 743 , CURSOR 498

Example
```
'------------------------------------------------------------------------
------------------
'name                     : lcd.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
$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


$sim
'REMOVE the above command for the real program !!
'$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

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
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
!rCall _write_lcd                                                  'put it on
LCD
End
```

6.244 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

X	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

CONFIG GRAPHLCD 426 , SETFONT 723, GLCDCMD 601, GLCDDATA 601

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : sed1520.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates the SED1520 based graphical
display support
```

```
'micro                        : Mega48
'suited for demo              : yes
'commercial addon needed      : no
'-------------------------------------------------------------------------
------------------

$regfile = "m48def.dat"                                     ' specify
the used micro
$crystal = 7372800                                          ' used
crystal frequency
$baud = 115200                                              ' 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

'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) , 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"
```

6.245 LCDCONTRAST

Action
Set the contrast of a TEXT LCD.

Syntax
LCDCONTRAST x

Remarks

X	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

Example
NONE

6.246 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 711 , MID 662

Partial Example

```
Dim S As String * 15 , Z As String * 15
S ="ABCDEFG"
Z = Left(s , 5)
Print Z                                                    'ABCDE
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

6.247 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

VAL 806

Partial Example

```
Dim S As String * 15 , Z As String * 15
S ="ABCDEFG"
Print Len(s)
```

6.248 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 635 , CONFIG GRAPHLCD 416 , BOX 353 , BOXFILL 355

Example

```
'-----------------------------------------------------------------------
------------------
'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
'-----------------------------------------------------------------------
------------------

$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            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        RESET          PORTC.4
'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 ,
```

```
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
   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
```

6.249 LINE INPUT

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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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
```

6.250 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 715 , TRIM 796

ASM

NONE

Partial Example

```
Dim S As String * 6
S =" AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

6.251 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

6.252 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
```

6.253 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.

6.254 LOADWORDADR

Action

Loads the Z-register and sets RAMPZ if available.

Syntax

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 641, LOADADR 640

Example

```
LOADWORDADR label
```

6.255 LOC

Action

Returns the position of last read or written Byte of the file

Syntax

lLastReadWritten = **LOC** (#bFileNumber)

Remarks

bFileNumber	(Byte) File number, which identifies an opened file
lLastReadWritten	(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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

ASM

Calls	_FileLoc	
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
```

6.256 LOF

Action

Returns the length of the File in Bytes

Syntax

lFileLength = **LOF** (#bFileNumber)

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
LFileLength	(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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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
```

6.257 LOCAL

Action

Dimensions a variable LOCAL to the function or sub program.

Syntax

LOCAL var As Type

Remarks

Var	The name of the variable
Type	The data type of the variable.

There can be only LOCAL variables of the type BYTE, INTEGER, WORD, 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 $NORAMCLEAR 294 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 539

ASM

NONE

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : declare.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
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
array
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 Test2                                                   'sub without
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
    B = X
    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
    B1(1) = 3                                                      'specify the
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
```

6.258 LOCATE

Action
Moves the LCD cursor to the specified position.

Syntax
LOCATE y , x

Remarks

X	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 426 , LCD 629 , HOME 607 , CLS 366

Partial Example
LCD "Hello"
Locate 1,10
LCD "*"

6.259 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 568 , LOG10 648

Example

Show sample 842

6.260 LOG10

Action

Returns the base 10 logarithm of a single variable.

Syntax

Target = **LOG10**(source)

Remarks

Target	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 568 , LOG 647

Example

Show sample 842

6.261 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 651 , LOOKUP 650

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : lookdown.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : LOOKDOWN
'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 Idx As Integer , Search As Byte , Entries As Byte

'we want to search for the value 3
Search = 3
'there are 5 entries in the table
Entries = 5

'lookup and return the index
Idx = Lookdown(search , Label , Entries)
Print Idx

Search = 1
Idx = Lookdown(search , Label , Entries)
Print Idx


Search = 100
Idx = Lookdown(search , Label , Entries)
Print Idx                                                   ' return -1
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%
```

6.262 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 651

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

Dim B1 As Byte , I As Integer
B1 = Lookup(2 , Dta)
Print B1                                                     ' Prints 3
(zero based)

I = Lookup(0 , Dta2)                                         ' print 1000
Print I
End
```

```
Dta:
Data 1 , 2 , 3 , 4 , 5
Dta2:
Data 1000% , 2000%
```

6.263 LOOKUPSTR

Action

Returns a string from a table.

Syntax

var = **LOOKUPSTR**(value, label)

Remarks

Var	The string returned
Value	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 650 , LOOKDOWN 648

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

Dim S As String * 4 , Idx As Byte
Idx = 0 : S = Lookupstr(idx , Sdata)
Print S                                                    'will print
'This'
End

Sdata:
Data "This" , "is" , "a test"
```

6.264 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 :
B = L , where B is a byte and L is a Long.

See also

HIGH 606 , HIGHW 607

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

Dim I As Integer , Z As Byte
I = &H1001
Z = Low(i)                                                ' is 1
End
```

6.265 LOWERLINE

Action

Reset the LCD cursor to the lower line.

Syntax

LOWERLINE

Remarks

NONE

See also

UPPERLINE 806 , THIRDLINE 793 , FOURTHLINE 578 , HOME 607

Partial Example

```
Lcd "Test"
Lowerline
Lcd "Hello"
End
```

6.266 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 777 , GOSUB 602

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
```

6.267 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 655 , BCD 345 , MAKEINT 654

Example
```
Dim A As Byte
A = 65
Lcd A
Lowerline
Lcd Bcd(a)
A = Makebcd(a)
Lcd " " ; A
End
```

6.268 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 651 , HIGH 606 , MAKEBCD 654 , MAKEDEC 655

Example

```
Dim A As Integer , I As Integer
A = 2
I = Makeint(a , 1)                                    'I = (1 *
256) + 2 = 258
End
```

6.269 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 654 , MAKEBCD 654, MAKEINT 654

Example

```
Dim A As Byte
A = 65
Print A
Print Bcd(a)
A = Makedec(a)
Print Spc(3) ; A
End
```

6.270 MAKEMODBUS

Action

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
varbts	For a function that sends data like function 6 and 16, this must be a 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 679

Example

```
'-------------------------------------------------------------------------
'name                         : rs485-modbus-master.bas
'copyright                    : (c) 1995-2008, MCS Electronics
'purpose                      : demo file for MAKEMODBUS
'micro                        : Mega162
'suited for demo              : yes
'commercial addon needed      : no
'-------------------------------------------------------------------------

$regfile = "m162def.dat"                             ' specify the used micr
$crystal = 8000000
```

```
$baud = 19200                                               ' use baud rate
$hwstack = 42                                               ' default use 42 for th
$swstack = 40                                               ' default use 40 for th
$framesize = 40                                             ' default use 40 for th

$lib "modbus.lbx"                                           ' specify the additiona
Config Print1 = Portb.1 , Mode = Set                        ' specify RS-485 and di


Rs485dir Alias Portb.1                                      'make an alias
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
L = &H12345678                                              'assign a value


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
     ' Print #1 , Makemodbus(2 , 3 , 8 , 2);                ' slave 2, function 3,
     ' Print #1 , Makemodbus(2 , 6 , 8 , W);                ' slave 2, function 6,
       Print #1 , Makemodbus(2 , 16 , 8 , L);               ' slave 2, function 16,
   End If
   If Ischarwaiting(#1) <> 0 Then                           'was something returned
       B = Waitkey(#1)                                      'then get it
       Print Hex(b) ; ",";                                  'print the info
   End If
Loop

End
```

6.271 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 456 , IP2STR 626

Example

```
NONE
```

6.272 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.
M	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

MIN 661

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : minmax.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : show the MIN and MAX functions
'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))
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
```

6.273 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

Example

```
'--------------------------------------------------------------------------
'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
'-------------------------------------------------------------------------
$regfile = "m88def.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 = 16                                              ' default
use 10 for the SW stack
$framesize = 40
```

```
Dim Ars(10) As Byte                                         'source
bytes
Dim Art(10) As Byte                                         'target
bytes
Dim J As Byte                                               'index
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
End
```

6.274 MIN

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
M	Returns the minimum value of the array
Idx	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 658

Example

```
'--------------------------------------------------------------------------------
-------------------
'name                       : minmax.bas
'copyright                  : (c) 1995-2005, MCS Electronics
'purpose                    : show the MIN and MAX functions
'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))
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
```

6.275 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.
l	The number of characters to get/set.

See also

LEFT 634 , RIGHT 711

Example

```
Dim S As String * 15 , Z As String * 15
S ="ABCDEFG"
Z = Left(s , 5)
Print Z                                                          'ABCDE
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

6.276 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 351

Example

```
'--------------------------------------------------------------------------
---------
'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
$swstack = 10                                                 ' default
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
```

6.277 ON INTERRUPT

Action

Execute subroutine when the specified interrupt occurs.

Syntax

ON interrupt label [NOSAVE]

Remarks

Interrupt	INT0, 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

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 667

Partial Example

```
Enable Interrupts
```

```
Enable Int0                                                   'enable the
interrupt
On Int0 Label2 Nosave                                         'jump to
label2 on INT0
Do'endless loop
  nop
Loop
End

Label2:
Dim A AsByte
If A > 1 Then
  Return                                                      'generates a
RET because it is inside a condition
EndIf
Return                                                        'generates a
RETI because it is the first RETURN
Return                                                        'generates a
RET because it is the second RETURN
```

6.278 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

Var	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 664

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
'copyright                     : (c) 1995-2005, MCS Electronics
'purpose                       : demo : ON .. GOSUB/GOTO
'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
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
```

6.279 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 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
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.

UART

The statements that support the device are PRINT 679 , INPUT 622 , INPUTHEX 620 , INKEY 617 and WAITKEY 810

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. And closing the channel is only needed to make the syntax compatible with VB.

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.

See also

CLOSE 370 , CRYSTAL 497, PRINT 679, LINE INPUT 638 , LOC 642 , LOF 643 , EOF 566

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
```

6.280 OUT

Action

Sends a byte to a hardware port or internal or external memory address.

Syntax

OUT address, value

Remarks

Address	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 Compiler Chip Options 87.

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 618 , PEEK 674 , POKE 675

Example

Out &H8000 , 1 'send 1 to the databus(d0-d7) at hex address 8000
End

6.281 PEEK

Action
Returns the content of a register.

Syntax
var = **PEEK**(address)

Remarks

Var	Numeric variable that is assigned with the content of the memory location address
Address	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 675 , CPEEK 487 , INP 618 , OUT 673

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
'--------------------------------------------------------------------------
------------------

$regfile = "m162def.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 ofthe 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
```

6.282 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 674 , CPEEK 487 , INP 618 , OUT 673

Example

Poke 1 , 1 'write 1 to R1
End

6.283 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.

See also

PUSHALL 687

6.284 POWER

Action

Returns the power of a single or double variable and its argument

Syntax

var = **POWER**(source, raise)

Remarks

Var	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 ^ 2 would be returned as -4, but -2 ^ 3 would be returned as -8 which is wring since -2 ^ 3 = -2 x -2 x -2=4 x -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.

The exception handling was also improved.

Base	Raise	Result
0	0	NAN
NAN	x	NAN
x	NAN	NAN
Infinity	x	NAN
x	Infinity	NAN
0	x<0	Infinity
0	x>0	0
x	0	1
x<0	x<>int(x)	NAN

See Also

EXP 568 ,LOG 647, LOG10 648 , SQR 768

Example

Show sample 842

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

Print "POWER() - Test"
Print "=============="

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-3

POWER(-2 , -4) = 62.5E-3

POWER(-2 , -5) = -31.25E-3

POWER(-2 , 3) = -8

POWER(-2 , 4) = 16

POWER(-2 , 5) = -32

6.285 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.

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.

See also

IDLE 613 , POWERSAVE 679

Example

Powerdown

6.286 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.

See also

IDLE 613, POWERDOWN 678

Example

Powersave

6.287 PRINT

Action

Send output to the RS-232 port.
Writes a string to a file.

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.

See also

INPUT 622 ,OPEN 669 , CLOSE 370 , SPC 763

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : print.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: PRINT, HEX
'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 , S As String * 4
A = 1
Print "print variable a " ; A
Print                                                         'new line
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 = -32000
Print 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)
Print A                                                      '9 dec
End
```

6.288 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 669 and CLOSE 370 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 620 , CONFIG PRINTBIN 439

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). Not the whole array will be sent!
End
```

6.289 PSET

Action

Sets or resets a single pixel.

Syntax

PSET X , Y, value

Remarks

X	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 749 , CONFIG GRAPHLCD 416 , LINE 635

Example

```
'--------------------------------------------------------------------------
------------------
'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
'--------------------------------------------------------------------------
------------------

$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            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       RESET          PORTC.4
'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 ,
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
   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
```

6.290 PS2MOUSEXY

Action

Sends mouse movement and button information to the PC.

Syntax

PS2MOUSEXY X , Y, button

Remarks

X	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 728, CONFIG PS2EMU 440

6.291 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 bitwait 350 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 686

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
```

6.292 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 685

Example

```
Dim A As Byte
Config Portb = Output                                   'PORTB all
output pins
Portb = 0                                               'all pins 0
Do
  For A = 0 To 7
    Pulseout Portb , A , 60000                          'generate
pulse
    Waitms 250                                          'wait a bit
  Next
Loop                                                    'loop for
ever
```

6.293 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

See also

POPALL 675

6.294 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.

#channel	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.
Length	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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582, FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

ASM

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
```

6.295 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
```

6.296 RAD2DEG

Action

Converts a value in radians to degrees.

Syntax

var = **RAD2DEG**(Source)

Remarks

Var	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 537

Example

```
'-----------------------------------------------------------------------
--------
'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
```

6.297 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

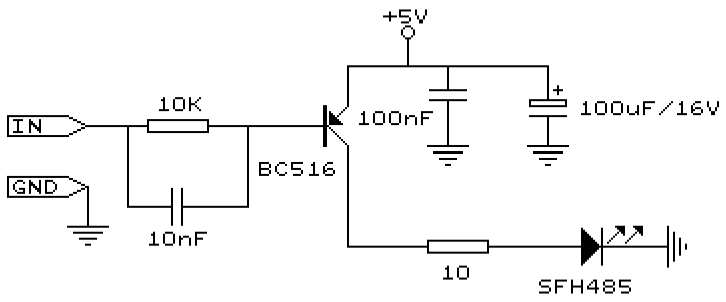
Command	The RC5 command.

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 443 , GETRC5 596 , RC6SEND 695

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
'--------------------------------------------------------------------------
------------------

$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
```

6.298 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.

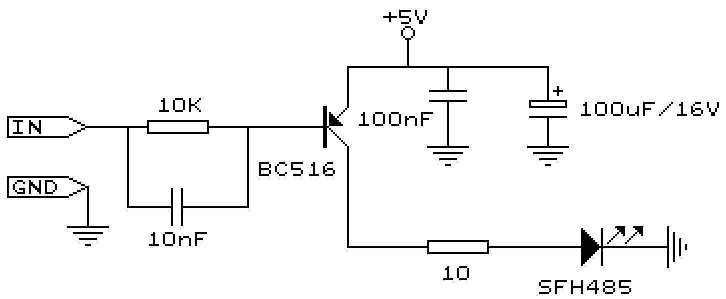
Normal RC5 code uses 2 leading bits with the value '1'. After that the toggle bit follows.
With extended RC5, the second bit is used to select the bank. When you make it 1 (the default and normal RC5) the RC5 code is compatible. When you make it 0, you select bank 0 and thus use extended RC5 code.

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 443 , GETRC5 596 , RC6SEND 695

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
'-----------------------------------------------------------------------
------------------

$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
```

6.299 RC6SEND

Action

Sends RC6 remote code.

Syntax

RC6SEND togglebit, address, command

Uses

TIMER1

Remarks

Togglebit	Make the toggle bit 0 or 1 to set the toggle bit
Address	The RC6 address
Command	The RC6 command.

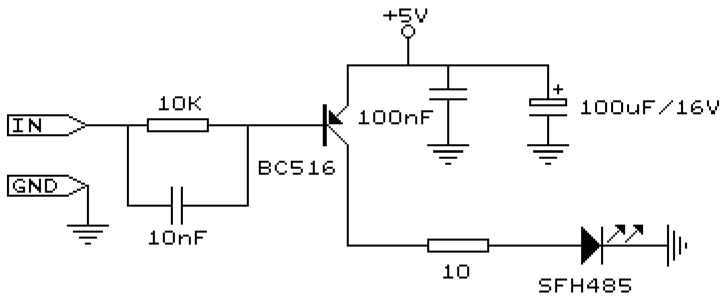
The resistor must be connected to the OC1A pin. In the example a 2313 micro was used. This micro has pin portB.3 connected to OC1A.
Look in a data sheet for the proper pin when used with a different chip.

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 the code.

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 443 , GETRC5 596 , RC5SEND 691

Example

```
'-----------------------------------------------------------------------------
```

```
------------------
'name                     : sendrc6.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
'-----------------------------------------------------------------------
------------------

$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.
' 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
Togbit = 0                                                 ' make it 0
or 32 to set the toggle bit
Address = 0
Do
   Waitms 500
   Rc6send Togbit , Address , Command
Loop
End
```

6.300 READ

Action

Reads those values and assigns them to variables.

Syntax

READ var

Remarks

Var	Variable that is assigned data value.

It is best to place the DATA 501 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 501 , RESTORE 709

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : readdata.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : READ,RESTORE
'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 Dta1                                                 '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
W = Loadlabel(dta1)                                          ' loadlabel
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
```

6.301 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 = 10
V = 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

See also

WRITEEEPROM 815 , $EEPROM 267

ASM

NONE

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : eeprom2.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : shows how to use labels with READEEPROM
'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

'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
'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 , Label1
B = 101
Writeeeprom B

'read it back
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
End
```

6.302 READHITAG

Action

Read HITAG RFID transponder serial number.

Syntax

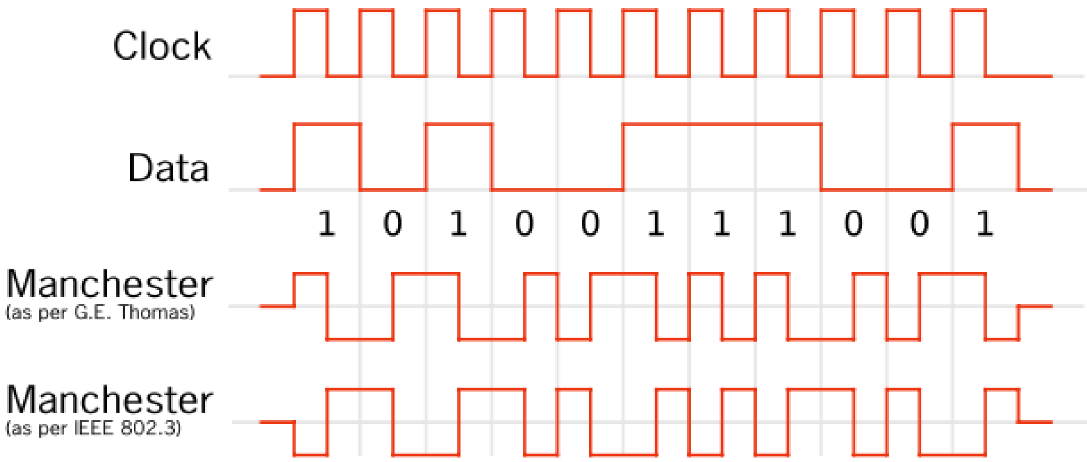
result = **READHITAG**(var)

Remarks

result	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.

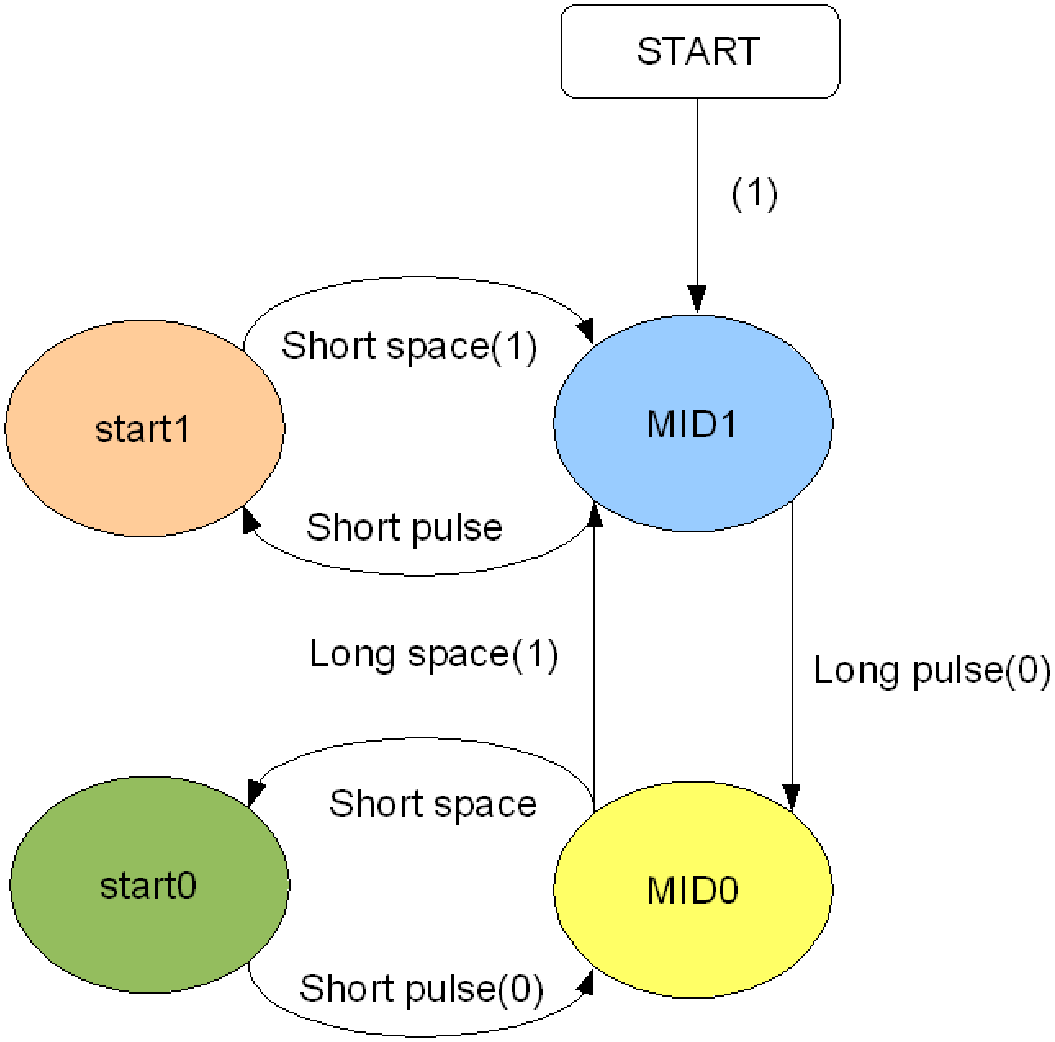
When the transponder is held before the coil of the receiver, the bits stream will be modulated with the bit values. Just like RC5, HITAG is using Manchester encoding. This is a simple and reliable method used in transmission systems.
Manchester encoding is explained very well at the Wiki Manchester 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 704 , CONFIG HITAG 405

Example

See CONFIG HITAG 405 for 2 examples.

6.303 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 returned numbers for 5 bit coding are:

Returned number	ISO characterT
0	0
1	1
2	2
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
'copyright                : (c) 1995-2005, MCS Electronics
'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
'black      - connect to GND
'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
```

```
'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
_mcs Alias 1                                                'CS line
(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
  Readmagcard Ar(1) , B , 5                                 'read the
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
```

6.304 REM

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 "

           ^ - - - This Will Not Be Executed!
```

6.305 RESET

Action

Reset a bit to zero.

Syntax

RESET bit
RESET var.x

Remarks

bit	Can be a SFR such as PORTB.x, or any bit variable where x=0-7.
var	Can be a byte, integer word or long variable.
x	Constant of variable to reset.(0-7) for bytes and (0-15) for Integer/Word. For longs(0-31)

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 constant in the def file.

See also

SET 721 , TOGGLE 796

Example

```
'--------------------------------------------------------------------------
---------
'name                     : boolean.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: AND, OR, XOR, NOT, BIT and MOD
'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
$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 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 c = " ; C                                         ' print
result

C = A Or B1                                                    'also for or
Print "a OR b1 = " ; C

C = A Xor B1                                                   ' and for
xor
Print "a XOR b1 = " ; C

A = 1
C = Not A                                                      'not
Print "c = NOT a " ; C
C = C Mod 10
Print "c MOD 10 = " ; C


If Portb.1 = 1 Then
   Print "Bit set"
Else
   Print "Bit not set"
End If

Aa = 1                                                         'use this or
..
Set Aa                                                         'use the set
statement
If Aa = 1 Then
   Print "Bit set (aa=1)"
Else
   Print "Bit not set(aa=0)"
End If

Aa = 0                                                         'now try 0
Reset Aa                                                       'or use
reset
If Aa = 1 Then
   Print "Bit set (aa=1)"
Else
   Print "Bit not set(aa=0)"
End If


B1 = 255                                                       'assign
variable
Reset B1.0                                                     'reset bit 0
of a byte variable
Print B1                                                       'print it

Set B1.0                                                       'set it
Print B1                                                       'print it
End
```

6.306 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 501 , READ 697 , LOOKUP 650

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : readdata.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : READ,RESTORE
'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 Dta1                                                '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
W = Loadlabel(dta1)                                                 ' loadlabel
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
```

6.307 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 602

Example

```
'--------------------------------------------------------------------------
```

```
------------------
'name                     : gosub.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: GOTO, GOSUB and RETURN
'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

Goto Continue
Print "This code will not be executed"

Continue:                                                  'end a label
with a colon
Print "We will start execution here"
Gosub Routine
Print "Back from Routine"
End

Routine:                                                   'start a
subroutine
  Print "This will be executed"
Return                                                     'return from
subroutine
```

6.308 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 634 , MID 662

Example

```
Dim S As String * 15 , Z As String * 15
S ="ABCDEFG"
Z = Left(s , 5)
Print Z                                                    'ABCDE
Z = Right(s , 3) : Print Z
Z = Mid(s , 2 , 3) : Print Z
End
```

6.309 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

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : rnd.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : RND() function
'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 I As Word                                                   ' dim
variable
Do
  I = Rnd(40)                                                   'get random
number (0-39)
  Print I                                                       'print the
value
  Wait 1                                                        'wait 1
second
Loop                                                            'for ever
End
```

6.310 ROTATE

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 741 , SHIFTIN 743 , SHIFTOUT 747

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
```

6.311 ROUND

Action

Returns a value rounded to the nearest value.

Syntax

var = **ROUND**(x)

Remarks

Var	A single or double variable that is assigned with the ROUND of variable x.
X	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 625 , FIX 573 , SGN 740

Example

```
'--------------------------------------------------------------------------
------------------
'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
```

6.312 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 796 , LTRIM 639

ASM

NONE

Example

```
Dim S As String * 6
S =" AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

6.313 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
TimeStamp	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

Date and Time Routines 852 , SecOfDay 717 , SysSecElapsed 779

Partial Example

```
Lsecofday = Secofday()
_hour = _hour + 1
Lvar1 = Secelapsed(lsecofday)
Print Lvar1
```

6.314 SECOFDAY

Action

Returns the Seconds of a Day.

Syntax

Target = **SECOFDAY**()
Target = **SECOFDAY**(bSecMinHour)
Target = **SECOFDAY**(strTime)
Target = **SECOFDAY**(lSysSec)

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:

1. 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 852 , SysSec 777

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)
Print "Second of Day of " ; Strtime ; " is " ; Lsecofday
```

6.315 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

Syntax

Function: NextReadWrite = **SEEK** (#bFileNumber)
Statement: **SEEk** #bFileNumber, NewPos

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
NextReadWrite	A Long Variable, which is assigned with the Position of the next Byte to be read or written (1-based)
NewPos	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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , FILELEN 571 , WRITE 814 , INPUT 622

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

```
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 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
Put #2 , Sn = 1                                            'specify the
file position
Close #2
```

6.316 SELECT-CASE-END SELECT

Action

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

IF THEN 613

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : case.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
$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 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
```

6.317 SET

Action

Set a bit to the value one.

Syntax

SET bit
SET var.x

Remarks

Bit	Bitvariable.
Var	A byte, integer, word or long variable.
X	Bit of variable (0-7) to set. (0-15 for Integer/Word) and (0-31) for Long

See also

RESET 707 , TOGGLE 796

Example

```
'--------------------------------------------------------------------------
---------
'name                     : boolean.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: AND, OR, XOR, NOT, BIT and MOD
'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
$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 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 c = " ; C                                     ' print
result

C = A Or B1                                                'also for or
Print "a OR b1 = " ; C

C = A Xor B1                                               ' and for
xor
Print "a XOR b1 = " ; C

A = 1
C = Not A                                                  'not
Print "c = NOT a " ; C
C = C Mod 10
Print "c MOD 10 = " ; C


If Portb.1 = 1 Then
   Print "Bit set"
Else
   Print "Bit not set"
End If

Aa = 1                                                     'use this or
..
Set Aa                                                     'use the set
statement
If Aa = 1 Then
   Print "Bit set (aa=1)"
Else
   Print "Bit not set(aa=0)"
End If

Aa = 0                                                     'now try 0
Reset Aa                                                   'or use
reset
If Aa = 1 Then
   Print "Bit set (aa=1)"
Else
   Print "Bit not set(aa=0)"
End If

B1 = 255                                                   'assign
variable
Reset B1.0                                                 'reset bit 0
of a byte variable
Print B1                                                   'print it

Set B1.0                                                   'set it
Print B1                                                   'print it
End
```

6.318 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
CONFIG GRAPHLCD 426 , LCDAT 632, GLCDCMD 601, GLCDDATA 601

Example
```
'--------------------------------------------------------------------------------
'name                     : sed1520.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates the SED1520 based graphical
display support
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------------

$regfile = "m48def.dat"                                    ' specify
the used micro
```

```
$crystal = 7372800                                                  ' used
crystal frequency
$baud = 115200                                                      ' 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

'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) , 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"
```

6.319 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

GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755 , CONFIG TCPIP 456

Example

See the DHCP.BAS example from the BASCOM Sample dir.

6.320 SETTCPREGS

Action

Writes to a W3100A register

Syntax

SETTCPREGS address, var , bytes

Remarks

address	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 599

ASM

NONE

Example

```
'-----------------------------------------------------------------------
------------------
'name                     : regs.bas
'copyright                : (c) 1995-2005, MCS Electronics
```

```
'purpose                    : test custom regs reading writing
'micro                      : Mega88
'suited for demo            : yes
'commercial addon needed    : no
'-----------------------------------------------------------------------
------------------
$regfile = "m88def.dat"                                   ' specify
the used micro

$crystal = 8000000                                        ' used
crystal frequency
$baud = 19200                                             ' use baud
rate
$hwstack = 80                                             ' default
use 32 for the hardware stack
$swstack = 128                                            ' default
use 10 for the SW stack
$framesize = 80                                           ' default
use 40 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
Const Sel_control = 0                                     ' Confirm
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
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
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
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


'we do the usual
Print "Init TCP"                                                    ' 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 = 192.168.0.1 , Localport = 1000 , Tx
= $55 , Rx = $55 , Twi = &H80 , Clock = 400000
Print "Init done"


'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(l)

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
Settcpregs &H91 , L , 4                                             'write


'and check if it worked
L = Gettcpregs(&H91 , 4)
Print Ip2str(l)
'while the address has the right value now the chip needs a reset in
order to use the new settings
L = &B10000001                                                      ' set
sysinit and swrest bits
Settcpregs &H00 , L , 1                                             ' write 1
register

'and with PING you can check again that now it works
End
```

6.321 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 685 , CONFIG PS2EMU 440

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : ps2_emul.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : PS2 Mouse emulator
'micro                    : 90S2313
'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 "l" : 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:
Data 3 , &H08 , &H00 , &H01                                ' mouse up
by 1 unit
```

6.322 SENDSCANKBD

Action

Sends keyboard scan codes to the PC.

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.

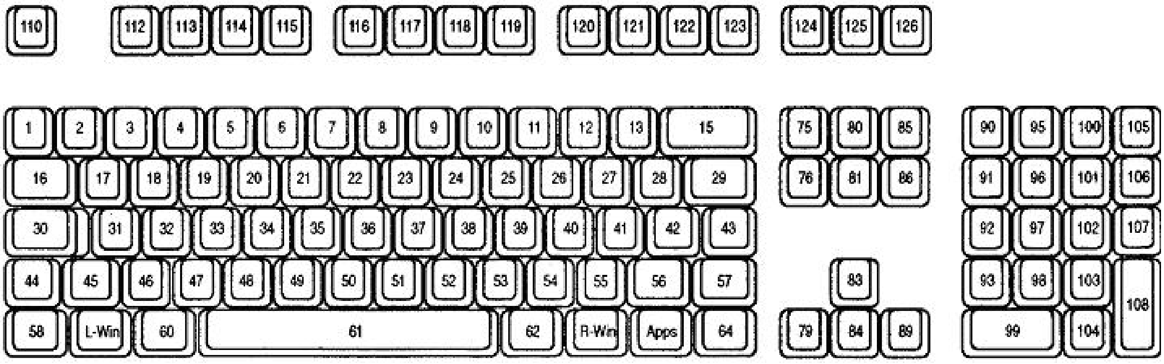
AT KEYBOARD SCANCODES

Table reprinted with permission of Adam Chapweske

http://panda.cs.ndsu.nodak.edu/~achapwes

KEY	MAKE	BREAK		KEY	MAKE	BREAK		KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	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
E	24	F0,24		\	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
H	33	F0,33		TAB	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
K	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
M	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
O	44	F0,44		R CTRL	E0,14	E0, F0,14		KP -	7B	F0,7B
P	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
T	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
X	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		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1, F0,14, F0,77	-NONE-				



ACPI Scan Codes

Key	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

Key	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 382

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : ps2_kbdemul.bas
'copyright                : (c) 1995-2005, MCS Electronics
'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
$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 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)

Mark:                                                           ' send mark
Data 12 , &H3A , &HF0 , &H3A , &H1C , &HF0 , &H1C , &H2D , &HF0 , &H2D ,
 &H42 , &HF0 , &H42
'    ^ send 12 bytes
'           m                    a                   r
 k
```

6.323 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.

Parameter	Description
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 736

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : serin_out.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstration of DYNAMIC software UART
'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

'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
  Serout 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
  '       ^-------------------------------- 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
```

6.324 SEROUT

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.

Parameter	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 734

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : serin_out.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstration of DYNAMIC software UART
'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

'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
  Serout 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
  '       ^-------------------------------- 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
```

6.325 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 726, GETSOCKET 600

ASM

NONE

Example

```
'-----------------------------------------------------------------------
'name                     : PING_TWI.bas          http://www.faqs.org/rfcs/rfc792.
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : Simple PING program
'micro                    : Mega88
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------
$regfile = "m32def.dat"                                     ' specify the used micr

$crystal = 8000000                                          ' used crystal frequenc
$baud = 19200                                               ' use baud rate
$hwstack = 80                                               ' default use 32 for th
$swstack = 128                                              ' default use 10 for th
$framesize = 80                                             ' default use 40 for th

Const Debug = 1

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
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
Const Sock_closed = $00                                     ' Status Of Connection
Const Sock_arp = $01                                        ' Status Of Arp
Const Sock_listen = $02                                     ' Status Of Waiting For
Const Sock_synsent = $03                                    ' Status Of Setting Up
Const Sock_synsent_ack = $04                                ' Status Of Setting Up
Const Sock_synrecv = $05                                    ' Status Of Setting Up
Const Sock_established = $06                                ' Status Of Tcp Connect
Const Sock_close_wait = $07                                 ' Status Of Closing Tcp
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
Const Sock_reset = $0d                                      ' Status Of Closing Tcp
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 initiali
Dta(4) = 0                                                  ' checksum
Dta(5) = 0                                                  ' a signature can be ar
Dta(6) = 1                                                  '   signature
Dta(7) = 0                                                  ' sequence number - any
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 ar

#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
```

6.326 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.
X	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 625 , FIX 573 , ROUND 714

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
```

6.327 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 713 , SHIFTIN 743 , SHIFTOUT 747

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : shift.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : example for SHIFTIN and SHIFTOUT 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

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 - 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
```

6.328 SHIFTCURSOR

Action

Shift the cursor of the LCD display left or right by one position.

Syntax

SHIFTCURSOR LEFT | RIGHT

See also

SHIFTLCD 748

Partial Example

LCD "Hello"
SHIFTCURSOR LEFT
End

6.329 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 high with ext. clock 6 – LSB shifted in first when clock goes low with ext. clock 7 – LSB shifted in first when clock goes high 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 CONFIG SHIFTIN 452 =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 low with ext. clock 5 – MSB shifted in first when clock goes high with ext. clock 6 – LSB shifted in first when clock goes low with ext. clock 7 – LSB shifted in first when clock goes high 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 747 , SHIFT 741

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : shift.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : example for SHIFTIN and SHIFTOUT 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

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
'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
```

6.330 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 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 743 , SHIFT 741

Example

See SHIFTIN 743 sample

6.331 SHIFTLCD

Action

Shift the LCD display left or right by one position.

Syntax

SHIFTLCD LEFT / RIGHT

Remarks

NONE

See also

SHIFTCURSOR 743 , SHIFTCURSOR 743 , INITLCD 616 , CURSOR 498

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

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
```

6.332 SHOWPIC

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 79.

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 682 , $BGF 259 , CONFIG GRAPHLCD 416 , LINE 635 , CIRCLE 362 , SHOWPICE 750

Example

See $BGF 259 example

6.333 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 Tools Graphic Converter 79.

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 682 , $BGF 259 , CONFIG GRAPHLCD 416 , LINE 635 , SHOWPIC 749 , CIRCLE 362

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : showpice.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates showing a picture from EEPROM
'micro                    : AT90S8535
'suited for demo          : yes
'commercial addon needed  : no
'--------------------------------------------------------------------------
------------------

$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 th e 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
```

6.334 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 690 , DEG2RAD 537 , ATN 339 , COS 485

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
```

6.335 SINH

Action

Returns the sinus hyperbole of a float

Syntax

var = **SINH**(source)

Remarks

Var	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 690 , DEG2RAD 537 , ATN 339 , COS 485 , SIN 751 , TANH 792 , COSH 486

Example

Show sample 842

6.336 SOCKETCONNECT

Action

Establishes a connection to a TCP/IP server.

Syntax

Result = **SOCKETCONNECT**(socket, IP, port)

Remarks

Result	A byte that is assigned with 0 when the connection succeeded. It will return 1 when an error occurred.
IP	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 with an IP number. Note that the LSB of the LONG, must contain the MSB of the IP number.
Port	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 456, GETSOCKET 600 , SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788 , TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755

Example

```
'-----------------------------------------------------------------------
------------------
'name                    : servertest.bas
'copyright               : (c) 1995-2005, MCS Electronics
'purpose                 : start the easytcp.exe program after the chip
is programmed
'                          and create 2 connections
'micro                   : Mega161
'suited for demo         : no
'commercial addon needed : yes
'-----------------------------------------------------------------------
------------------

$regfile = "m161def.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

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
Const Sel_control = 0                                         ' Confirm
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
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
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
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



$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 Bclient As Byte                                           ' socket
number
Dim Idx As Byte
```

```
Dim Result As Word                                            ' result
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
               Result = Tcpwrite(idx , "Hello from W3100A{013}{010}")
   ' send welcome
            End If
            Result = Socketstat(idx , Sel_recv)               ' get number
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
```

6.337 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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, TCPREAD 786, CLOSESOCKET 372

Example

See SOCKETCONNECT 752 example

6.338 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 into 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
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 456, GETSOCKET 600 , SOCKETCONNECT 752, TCPWRITE 787, TCPWRITESTR 788, TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755

Partial Example

```
Tempw = Socketstat(i , 0)' get status
Select Case Tempw
  Case Sock_established
  Case Else
End Select
```

6.339 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
>>	38E	0011 1000 1110
<<	D8E	1101 1000 1110
record	78E	0111 1000 1110
pause	98E	1001 1000 1110

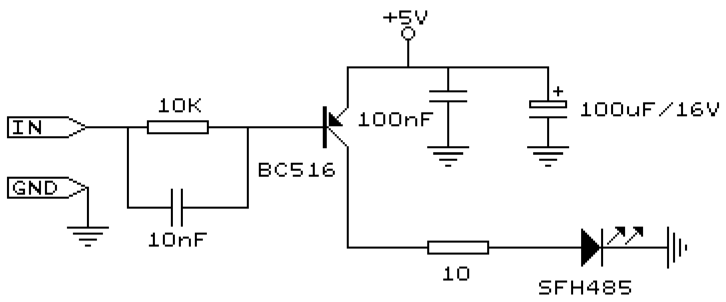
---[SONY TV Infrared Remote Control codes (RM-694)]--------------------------

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 443 , GETRC5 596 , RC5SEND 691 , RC6SEND 695

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : sonysend.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
'--------------------------------------------------------------------------
------------------

$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

Do
   Waitms 500
   Sonysend &HA90
Loop
End
```

6.340 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

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : sound.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo : SOUND
'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 Pulses As Word , Periods As Word
Pulses = 65535 : Periods = 10000                            'set
variables
Speaker Alias Portb.1                                       'define port
pin

Sound Speaker , Pulses , Periods                            'make some
noice
'note that pulses and periods must have a high value for high XTALS
'sound is only intended to make some noise!
```

```
'pulses  range from 1-65535
'periods range from 1-65535
End
```

6.341 SPACE

Action

Returns a string that consists of spaces.

Syntax

var = **SPACE**(x)

Remarks

X	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 776 , SPC 763

Example

```
'--------------------------------------------------------------------------
--------
'copyright                    : (c) 1995-2005, MCS Electronics
'micro                        : Mega48
'suited for demo              : yes
'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
$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 S As String * 15 , Z As String * 15
S = Space(5)
Print " {" ; S ; " }"                                       '{ }

Dim A As Byte
A = 3
S = Space(a)
```

```
End
```

6.342 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 275 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 762

Example

```
'--------------------------------------------------------------------------
--------
'copyright                    : (c) 1995-2005, MCS Electronics
'micro                        : Mega48
'suited for demo              : yes
'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
$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 S As String * 15 , Z As String * 15
Print "{" ; Spc(5) ; "}"                                    '{ }
Lcd "{" ; Spc(5) ; "}"                                      '{ }
```

6.343 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 766, SPIINIT 765, CONFIG SPI 453 , SPIMOVE 765

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
$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
Dim A(10) As Byte

Spiinit
B = 5
Spiout A(1) , B
```

```
Spiin A(1) , B

A(1) = Spimove(a(2))
End
```

6.344 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 764 , SPIOUT 766

ASM
Calls _init_spi

Example
See SPIIN 764

6.345 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 764 , SPIINIT 765 , CONFIG SPI 453

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                                                  'send 5
bytes
Spiout X , 1                                                     'send 1 byte
A(1) = Spimove(5)                                                ' move 5 to
SPI and store result in a(1)
End
```

6.346 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 764 , SPIINIT 765 , CONFIG SPI 453 , SPIMOVE 765

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
```

6.347 SPLIT

Action

Split a string into a number of array elements.

Syntax

count = **SPLIT** (source, array, search)

Remarks

count	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
search	The character to search for. This can be a string or string constant.

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 "1.2.2.2.2.2.2.3.3.3" into an array of 3 elements, you will loose the data.

See also

INSTR 624

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
```

6.348 SQR

Action

Returns the Square root of a variable.

Syntax

var = **SQR**(source)

Remarks

var	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 676

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 = 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
```

6.349 START

Action

Start the specified device.

Syntax

START device

Remarks

Device	TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power), ADC(A/D converter power) or DAC(D/A converter)

You must start a timer/counter in order for an interrupt to occur (when the external gate is disabled).
TIMER0 and COUNTER0 are the same device.

The AC and ADC parameters will switch power to the device and thus enabling it to work.

See also

STOP 775

Example

```
'------------------------------------------------------------------------
---------
'name                       : adc.bas
'copyright                  : (c) 1995-2005, MCS Electronics
'purpose                    : demonstration of GETADC() function for 8535
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
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 :
'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.
```

6.350 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 FR FR FR FR FR FR FR
FR FR FR FR FR FR YY YY SP 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
'copyright                 : (c) 1995-2005, MCS Electronics
'purpose                   : shows how to check for the stack sizes
'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
$framesize = 14                                            ' default
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 r1
  pop r0
#endif
Return
```

6.351 STOP

Action

Stop the specified device. Or stop the program

Syntax

STOP device
STOP

Remarks

Device	TIMER0, TIMER1, COUNTER0 or COUNTER1, WATCHDOG, AC (Analog comparator power) , ADC(A/D converter power) or DAC(D/A converter)

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 769 , END 565

Example

See START 769 example

6.352 STR

Action

Returns a string representation of a number.

Syntax

var = **STR**(x)

Remarks

var	A string variable.

X	A numeric variable.

The string must be big enough to store the result.
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 mcsbyte828.lbx then. This library can only print bytes.
There is also a library for printing integers and words only. This library is named mcsbyteint828.
When you use these libs to print a long you will get an error message.

See also

VAL806 , HEX604 , HEXVAL605 , MCSBYTE828 , BIN347

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
```

6.353 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.
N	The ASCII-code that is assigned to the string.
M	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 762

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 = 40                                                ' default
use 10 for the SW stack
$framesize = 40                                              ' default
use 40 for the frame space


Dim S As String * 15
S = String(5 , 65)
Print S                                                      'AAAAA
End
```

6.354 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 530 , CALL 358

See the DECLARE SUB 532 topic for more details.

6.355 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
BSecMinHour	A Byte, which holds the Sec-value followed by Min(Byte), Hour (Byte), 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	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 852 , SYSSECELAPSED 779, SYSDAY 780

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
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 " ;
Lsyssec
' 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 " ;
Lsyssec
' System Second of System Day 2000 (00:00:00) is 172800000
```

6.356 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
SystemTimeStamp	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

Date and Time Routines 852 , SECELAPSED 716, SYSSEC 777

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"
```

6.357 SYSDAY

Action

Returns a number, which represents the System Day

Syntax

Target = **SysDay**()
Target = **SysDay**(bDayMonthYear)
Target = **SysDay**(strDate)
Target = **SysDay**(lSysSec)

Remarks

Target	A Variable (LONG), that is assigned with the System-Day
bDayMonthDay	A Byte, which holds the Day-value followed by Month(Byte) and Year (Byte)
strDate	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 852 , Config Date 395 , Config Clock 387 , SysSec 777

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"
```

6.358 SWAP

Action

Exchange two variables of the same type.

Syntax

SWAP var1, var2

Remarks

var1	A variable of type bit, byte, integer, word, long or string.
var2	A variable of the same type as var1.

After the swap, var1 will hold the value of var2 and var2 will hold the value of var1.

Example

```
'-----------------------------------------------------------------------
------------------
'name                      : swap.bas
'copyright                 : (c) 1995-2005, MCS Electronics
```

```
'purpose                    : demo: SWAP
'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
Dim Bbit1 As Bit , Bbit2 As Bit
Dim S1 As String * 10 , S2 As String * 10



S1 = "AAA" : S2 = "BBB"
Swap S1 , S2

A = 5 : B1 = 10                                              'assign some
vars
Print A ; "    " ; B1                                        'print them

Swap A , B1                                                  'swap them
Print A ; "    " ; B1                                        'print is
again

Set Bbit1
Swap Bbit1 , Bbit2
Print Bbit1 ; Bbit2
End
```

6.359 TAN

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 690 , DEG2RAD 537 , ATN 339 , COS 485 , SIN 751 , ATN2 340

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
```

6.360 TCPCHECKSUM

Action

Return a TCP/IP checksum, also called Internet Checksum, or IP Checksum.

Syntax

res= **TCPCHECKSUM**(buffer , bytes [,w1] [,w2])

Remarks

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.

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 490 , CRC16 491, CRC32 496 , CHECKSUM 360

ASM

NONE

Example

```
'-----------------------------------------------------------------------
------------------
'name                          : PING_TWI.bas               http://www.faqs.org/
rfcs/rfc792.html
'copyright                     : (c) 1995-2005, MCS Electronics
'purpose                       : Simple PING program
'micro                         : Mega88
'suited for demo               : yes
'commercial addon needed       : no
'-----------------------------------------------------------------------
------------------
$regfile = "m32def.dat"                                     ' specify
the used micro

$crystal = 8000000                                          ' used
crystal frequency
$baud = 19200                                               ' use baud
rate
$hwstack = 80                                               ' default
use 32 for the hardware stack
$swstack = 128                                              ' default
use 10 for the SW stack
$framesize = 80                                             ' default
use 40 for the frame space

Const Debug = 1

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
Const Sel_control = 0                                       ' Confirm
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
```

```
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
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
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


'we do the usual
Print "Init TCP"                                       ' 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 = 192.168.0.1 , Localport = 1000 , Tx
= $55 , Rx = $55 , Twi = &H80 , Clock = 400000
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 initialization
Dta(4) = 0                                             ' checksum
Dta(5) = 0                                             ' a
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)                                          ' calculate
checksum and store in dta(3) and dta(4)

#if Debug
  For J = 1 To 9
    Print Dta(j)
  Next
#endif



Ip = Maketcp(192.168.0.16)                                           'try to
check this server

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 problem
      Print "Network not available"
   End If
   Waitms 1000
Loop
```

6.361 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 **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. 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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, CLOSESOCKET 372 , SOCKETLISTEN 755

Partial Example

```
Result = Socketstat(idx , Sel_recv)            ' get number of bytes
waiting
If Result > 0 Then
   Result = Tcpread(idx , S)
End If
```

6.362 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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITESTR 788, TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755

Example

```
Result = Tcpwrite(idx , "Hello from W3100A{013}{010}")
```

6.363 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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755

Example

```
'-----------------------------------------------------------------------
--------
'                              SMTP.BAS
'                       (c) 2002 MCS Electronics
' sample that show how to send an email with SMTP protocol
'-----------------------------------------------------------------------
--------

$regfile = "m161def.dat"                                    ' used
processor
$crystal = 4000000                                          ' used
crystal
$baud = 19200                                               ' baud rate
$lib "tcpip.lbx"                                            ' specify
the name of the tcp ip lib

'W3100A constants
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
Const Sel_control = 0                                       ' Confirm
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
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
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
```

```
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

Const Debug = -1                                                    ' for
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
   '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
                   Print S                                ' show info
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)            ' get
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
                           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
            Loop
         End If
End If
End                                                            'end program
```

6.364 TANH

Action

Returns the hyperbole of a single

Syntax

var = **TANH**(source)

Remarks

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 690 , DEG2RAD 537 , ATN 339 , COS 485 , SIN 751 , SINH 752 , COSH 486

Example

Show sample 842

6.365 THIRDLINE

Action
Reset LCD cursor to the third line.

Syntax
THIRDLINE

Remarks
NONE

See also
UPPERLINE 806 , LOWERLINE 652 , FOURTHLINE 578

Example
```
Dim A As Byte
A = 255
Cls
Lcd A
Thirdline
Lcd A
Upperline
End
```

6.366 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.

Do 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$ 514 , CONFIG CLOCK 387 , CONFIG DATE 395

Example

See the sample of DATE$ 514

6.367 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**(lSecOfDay)
bSecMinHour = **Time**(lSysSec)
bSecMinHour = **Time**(strTime)

strTime = **Time**(lSecOfDay)
strTime = **Time**(lSysSec)
strTime = **Time**(bSecMinHour)

Remarks

bSecMinHour	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 852 , SECOFDAY 717, SYSSEC 777

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
```

6.368 TOGGLE

Action
Toggles the state of an output pin or bit variable.

Syntax
TOGGLE pin

Remarks

pin	Any port pin like PORTB.0 or bit variable. A port pin must be configured as an output pin before TOGGLE can be used.

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.

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.

See also
CONFIG PORT 436

ASM
NONE

Partial Example
```
Dim Var As Byte
Config Pinb.0 = Output                                        ' portB.0 is
an output now

Do
  Toggle Portb.0                                              'toggle
state
  Waitms 1000                                                 'wait for 1
sec
Loop
```

6.369 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 715 , LTRIM 639

Partial Example

```
Dim S As String * 6
S =" AB "
Print Ltrim(s)
Print Rtrim(s)
Print Trim(s)
End
```

6.370 UCASE

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 628

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

```
$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 * 12 , Z As String * 12
S = "Hello World"
Z = Lcase(s)
Print Z
Z = Ucase(s)
Print Z
End
```

6.371 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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPWRITESTR 788, CLOSESOCKET 372 , SOCKETLISTEN 755 , UDPWRITE 801, UDPWRITESTR 802

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                    : Mega161
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------------
------------------

$regfile = "m161def.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

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
Const Sel_control = 0                                       ' Confirm
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
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
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
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



$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
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
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)                  ' get socket
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 + "."
      Shift Ip , Right , 8
    Next
End Function
End
```

6.372 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.
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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITESTR 788, TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755 , UDPWRITESTR 802 , UDPREAD 798

Example

See UDPwriteStr 802

6.373 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 456, GETSOCKET 600 , SOCKETCONNECT 752, SOCKETSTAT 756 , TCPWRITE 787, TCPREAD 786, CLOSESOCKET 372 , SOCKETLISTEN 755 , UDPWRITE 801, UDPREAD 798

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                    : Mega161
'suited for demo          : no
'commercial addon needed  : yes
'-----------------------------------------------------------------------
------------------

$regfile = "m161def.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

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
Const Sel_control = 0                                        ' Confirm
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
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
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
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



$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
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
Idx = Getsocket(idx , Sock_dgram , 5000 , 0)                ' get socket
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 + "."
      Shift Ip , Right , 8
    Next
End Function
End
```

6.374 UPPERLINE

Action

Reset LCD cursor to the upper line.

Syntax

UPPERLINE

Remarks

Optional you can also use the LOCATE statement.

See also

LOWERLINE 652 , THIRDLINE 793 , FOURTHLINE 578 , LCD 629, CLS 366 , LOCATE 647

Example

```
Dim A As Byte
A = 255
Cls
Lcd A
Thirdline
Lcd A
Upperline
End
```

6.375 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 775 , HEXVAL 605 , HEX 604 , BIN 347 , BINVAL 348

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 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
End
```

6.376 VARPTR

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

NONE

Example

Dim W As Byte
Print Hex(varptr(w)) ' 0060

6.377 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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , WRITE 814 , INPUT 622

The VERSION 809() function is something different. It is intended to include compile time info into the program.

ASM

Calls	_AVRDOSVer
Input	-
Output	R16 loaded with value

Example
Print Ver()

6.378 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.

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 808

Example
Print Version()

6.379 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 538 , WAITMS 811

Example

WAIT 3 'wait for three seconds
Print "*"

6.380 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 617 , ISCHARWAITING 626

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : inkey.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: INKEY , WAITKEY
'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 , S As String * 2
Do
   A = Inkey()                                              'get ascii
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 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
```

6.381 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 538 , WAIT 809 , WAITUS 812

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 "*"
```

6.382 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!

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.

See also

DELAY 538 , WAIT 809 , WAITMS 811

Example

WAITUS 10 'wait for 10 uS
Print "*"

6.383 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 550

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : while_w.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: WHILE, WEND
'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

A = 1                                                    'assign var
While A < 10                                             'test
expression
   Print A                                               'print var
   Incr A                                                'increase by
one
Wend                                                     'continue
loop
End
```

6.384 WRITE

Action

Writes data to a sequential file

Syntax

WRITE #ch , data [,data1]

Remarks

Ch	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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , GET 582 , PUT 688 , FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , WRITE 814 , INPUT 622

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                                    ' write is
also supported
Close #2

Open "write.dmo"for Input As #2
Input #2 , S , W , L                                    ' write is
also supported
Close #2
Print S ; " " ; W ; " " ; L
```

6.385 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 = 10
V = 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.

See also

READEEPROM 699

ASM

NONE

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : eeprom2.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : shows how to use labels with READEEPROM
'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

'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
'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 , Label1
B = 101
Writeeeprom B

'read it back
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
End
```

6.386 X10DETECT

Action

Returns a byte that indicates if a X10 Power line interface is found.

Syntax

Result = **X10DETECT**( )

Remarks

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 480 , X10SEND 819

Example

```
'--------------------------------------------------------------------------
------------------
'name                     : x10.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : example needs a TW-523 X10 interface
'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


'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()
Print X                                                     ' 0 means
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
```

6.387 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 !!!

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.

The following table lists all X10 codes.

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

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 480 , X10DETECT 817 , X10SEND 819

Example

See X10DETECT 817

6.388 #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 constant 483 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 483

Part VII

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 MCS Electronics web.

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.

Part VIII

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 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 150 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.

8.2 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

I2C 610

ASM

NONE

Example

```
'-----------------------------------------------------------------------
--------
'                            (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
  I2cwbyte &B10101010                                        ' write
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
```

8.3 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 78.

See also

mcsbyteint.lib 828

8.4 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 828

8.5 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 http://www.mcselec.com for more info.

The tcpip.lbx is shipped with BASCOM-AVR

The following functions are provided:

CONFIG TCPIP 456	Configures the W3100 chip.
GETSOCKET 600	Creates a socket for TCP/IP communication.
SOCKETCONNECT 752	Establishes a connection to a TCP/IP server.
SOCKETSTAT 756	Returns information of a socket.
TCPWRITE 787	Write data to a socket.
TCPWRITESTR 788	Sends a string to an open socket connection.
TCPREAD 786	Reads data from an open socket connection.
CLOSESOCKET 372	Closes a socket connection.
SOCKETLISTEN 755	Opens a socket in server(listen) mode.
GETDSTIP 591	Returns the IP address of the peer.
GETDSTPORT 592	Returns the port number of the peer.
BASE64DEC 341	Converts Base-64 data into the original data.
BASE64ENC 342	Convert a string into a BASE64 encoded string.
MAKETCP 658	Encodes a constant or 4 byte constant/variables into an IP number
UDPWRITE 801	Write UDP data to a socket.

UDPWRITESTR 802	Sends a string via UDP.
UDPREAD 798	Reads data via UDP protocol.

8.6 LCD

8.6.1 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
'-----------------------------------------------------------------------
'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
```

8.6.2 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 CONFIG LCD 426 statement.

See also the lcd42.lib 831 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.

8.6.3 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 CONFIG LCD 426 statement.

See also the lcd4.lib 831 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.

8.6.4 GLCD

GLCD.LIB (LBX) is a library for Graphic LCD's based on the T6963C chip.

The library contains code for LOCATE 647, CLS 366, PSET 682, LINE 635, CIRCLE 362, SHOWPIC 749 and SHOWPICE 750.

8.6.5 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 632
SETFONT 723
GLCDCMD 601
GLCDDATA 601

See the SED.BAS sample from the sample directory

8.6.6 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.
CS	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

As the color display does not have a built in font, you need to generate the fonts yourself.
You can use the Fonteditor 125 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 879

Example

```
'-------------------------------------------------------------------------------
' 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
$crystal = 8000000                                         '8 MHz

'First we define that we use a graphic LCD
Config Graphlcd = 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
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"
```

8.6.7 LCD-EPSON

This chip is compatible with PCF8533 832.

8.7 AVR-DOS

8.7.1 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 Compact Flash 839) 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

cFileHandles:	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	Type	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
gwLastSearchedCluster	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)

Variable Name	Type	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

Error Codes:

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't create a new file handle with same file number
70	cpFileOpenShareConflict	Tried to open a file in read and write 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

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	●	●	●	dWritePending	Something was written to sector, it must be saved to Card, before loading next sector
4		●		dFATSector	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 615 , OPEN 669 , CLOSE 370, FLUSH 574 , PRINT 679, LINE INPUT 638, LOC 642, LOF 643 , EOF 566 , FREEFILE 580 , FILEATTR 569 , SEEK 718 , BSAVE 356 , BLOAD 352 , KILL 627 , DISKFREE 545 , DISKSIZE 546 , GET 582 , PUT 688 ,FILEDATE 570 , FILETIME 572 , FILEDATETIME 571 , DIR 542 , WRITE 814 , INPUT 622 , FILELEN 571

8.8 CF Card

8.8.1 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:
DriveCheck 551, DriveReset 553 , DriveInit 553 , DriveGetIdentity 552 , DriveWriteSector 555 , DriveReadSector 554

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.

Error Codes:

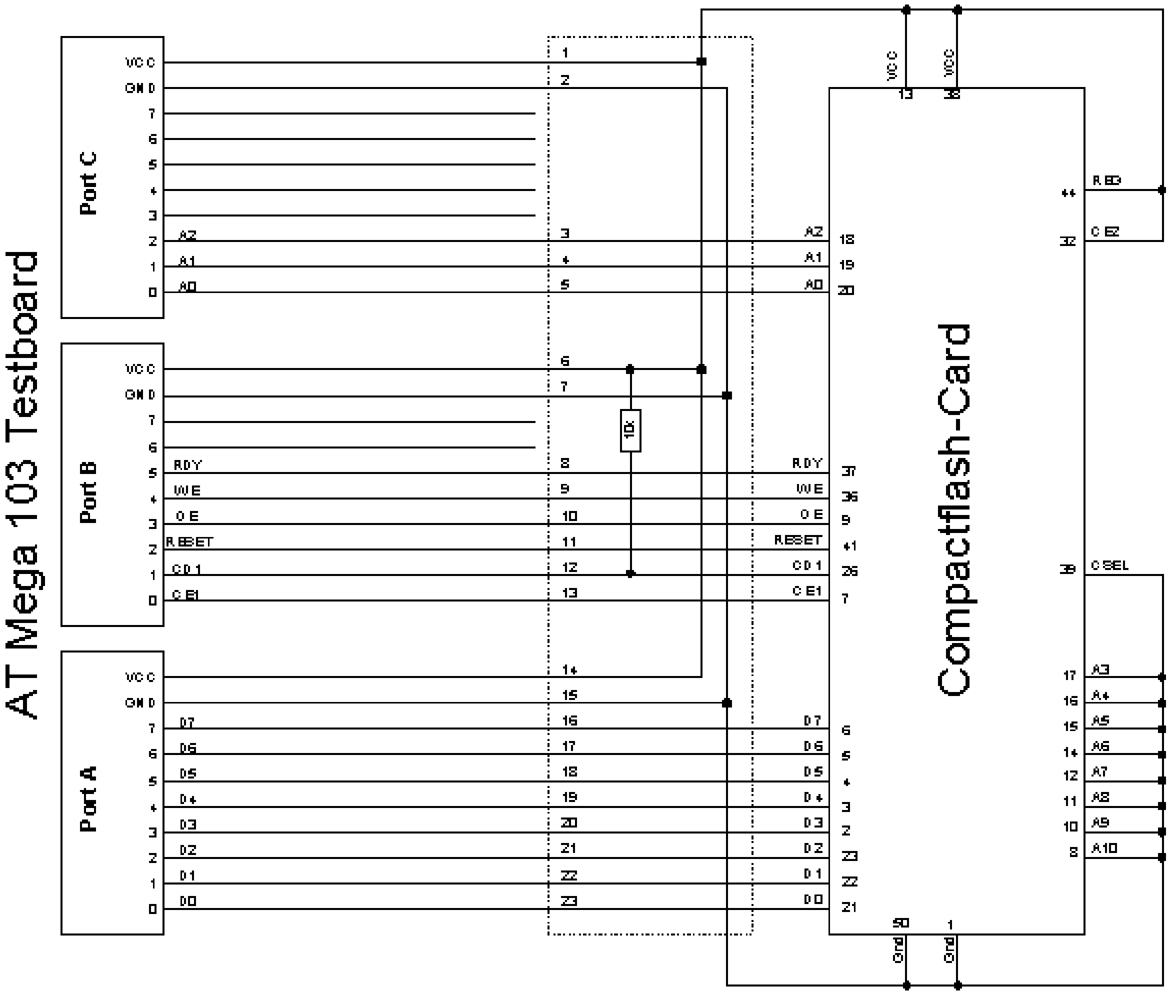
Code	Compiler – Alias	Remark
0	CpErrDriveNoError	No Error
224	cpErrDriveFunctionNotSupported	This driver does not supports this function
225	cpErrDriveNotPresent	No Drive is attached
226	cpErrDriveTimeOut	During Reading or writing a time out occurred
227	cpErrDriveWriteError	Error during writing
228	cpErrDriveReadError	Error during reading

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.



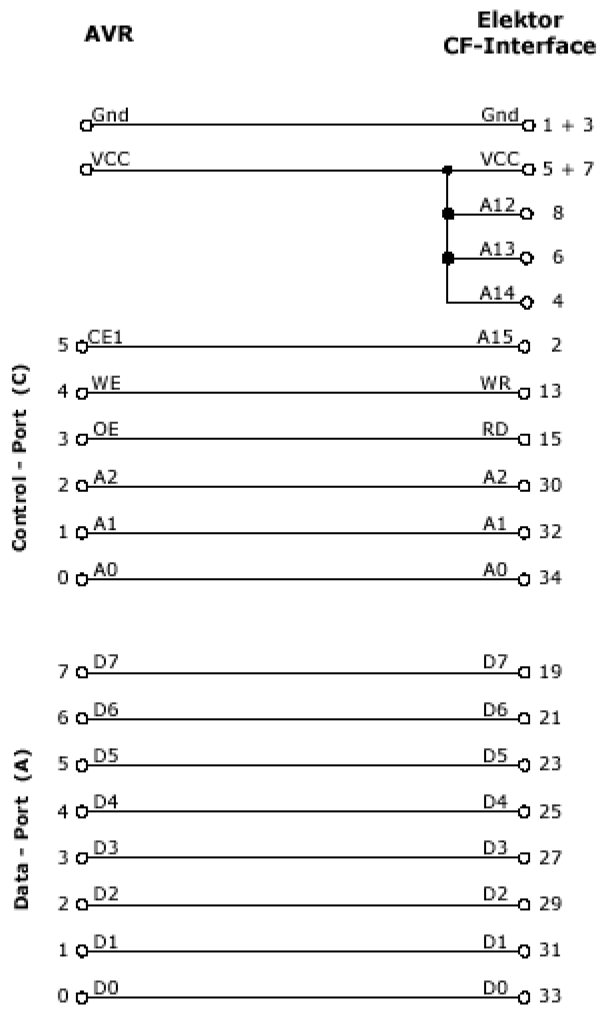
8.8.2 Elektor CF-Interface

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.

8.8.3 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.

8.8.4 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.

8.9 Floating Point

8.9.1 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:

```
'--------------------------------------------------------------------------------
'name                        : test_fptrig2.bas
'copyright                  : (c) 1995-2005, MCS Electronics
'purpose                      : demonstates FP trig library from Josef Franz Vögel
'micro                      : Mega8515
'suited for demo            : no
'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
$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 , Vsin As Single , Vtan As Single , Vatan As Single , S7 As Single
Dim Wi As Single , B1 As Byte
Dim Ms1 As Single



Const Pi = 3.14159265358979

'calculate PI
Ms1 = Atn(1) * 4


Testing_power:
Print "Testing Power X ^ Y"
Print "X              Y            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
```

```
Testing_exp_log:

Print "Testing EXP and LOG"
Print "x        exp(x)              log([exp(x)])        Error-abs           Error-rel"
Print "Error   is   for   calculating   exp   and   back   with   log   together"
For S1 = -88 To 88
   S2 = Exp(s1)
   S3 = Log(s2)
   S4 = S3 - S1
   S5 = S4 \ S1
   Print S1 ; "      " ; S2 ; "      " ; S3 ; "      " ; S4 ; "      " ; S5 ; "  " ;
   Print
Next
Print : Print : Print


Testing_trig:
Print "Testing COS, SIN and TAN"
Print "Angle Degree    Angle Radiant             Cos              Sin              Tan"
For Wi = -48 To 48
   S1 = Wi * 15
   S2 = Deg2rad(s1)
   Vcos = Cos(s2)
   Vsin = 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,        Degree"
S1 = 1 / 1024
Do
   S2 = Atn(s1)
   S3 = Rad2deg(s2)
   Print S1 ; "      " ; S2 ; "      " ; S3
   S1 = S1 * 2
   If S1 > 1000000 Then
      Exit Do
   End If
Loop

Print : Print : Print

Testing_int_fract:
Print "Testing Int und Fract of Single"
Print "Value        Int            Frac"
S2 = Pi \ 10
For S1 = 1 To 8
   S3 = Int(s2)
   S4 = Frac(s2)
   Print S2 ; "     " ; S3 ; "     " ; S4
   S2 = S2 * 10
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)              cosh(x)             tanh(x)"
For S1 = -20 To 20
  S3 = Sinh(s1)
  S2 = Cosh(s1)
  S4 = Tanh(s1)
  Print S1 ; "     " ; S3 ; "     " ; S2 ; "     " ; S4
Next
Print : Print : Print
```

```
Testing_log10:
Print "Testing    LOG10"
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

Print : Print : Print


'test MOD on FP
S1 = 10000
S2 = 3
S3 = S1 Mod S2
Print S3

Print "Testing_SQR-Single"
For S1 = -1 To 4 Step 0.0625
   S2 = Sqr(s1)
   Print S1 ; "  " ; S2
Next
Print
For S1 = 1000000 To 1000100
   S2 = Sqr(s1)
   Print S1 ; "  " ; S2
Next

Testing_atn2:
Print "Testing Sin / Cos / ATN2 / Deg2Rad / Rad2Deg / Round"
Print "X[deg]      X[Rad]       Sin(x)        Cos(x)        Atn2       Deg of Atn2       Rounded"
For S1 = -180 To 180 Step 5
   S2 = Deg2rad(s1)
   S3 = Sin(s2)
   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

8.9.2 DOUBLE

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.

All Trig() functions are supported by the double too!

8.10 I2C SLAVE

8.10.1 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 MCS Electronics

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 CONFIG 411 statement to specify the slave address:

Config I2cslave = &B01000000 ' same as &H40
Optional use : **CONFIG I2CSLAVE = address, INT= int , TIMER = tmr**

Where INT is INT0, 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.

The following table lists the pins for the various chips

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

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)
Return

8.10.2 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
$framesize = 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

'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 with a value
```

```
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
```

8.11 SPI

8.11.1 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
'purpose                   : shows how to implement a SPI SLAVE with
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.
'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 INT0 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 Int0 Isr_sspi Nosave
'we enable the int0 interrupt
Enable Int0
'we configure the INT0 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

_ssspdr = 0                                                         ' we send a
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:

```
'-------------------------------------------------------------------------
------------------
'name                     : spi-slave.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : shows how to create a SPI SLAVE
'micro                    : AT90S8515
'suited for demo          : yes
```

```
'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
Return                                                        ' this will
generate a reti
```

8.12 DATE TIME

8.12.1 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 TIME$ 793 and DATE$ 514.

By default the date format is in MM/DD/YY.

By specifying:
$LIB 283 "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 DATETIME 852 library which offers many more features.

8.12.2 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:

DayOfWeek 504	Returns the day of the week
DayOfYear 513	Returns the day of the year
SecOfDay 717	Returns the second of the day
SecElapsed 716	Returns the elapsed Seconds to a former assigned time-stamp
SysDay 780	Returns a number, which represents the System Day
SysSec 777	Returns a Number, which represents the System Second
SysSecElapsed 779	Returns the elapsed Seconds to a earlier assigned system-time-stamp
Time 794	Returns a time-value (String or 3 Byte for Second, Minute and Hour) depending of the Type of the Target
Date 516	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$!

8.13 PS2-AT Mouse and Keyboard Emulation

8.13.1 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 382
SENDSCANKBD 730

8.13.2 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 440
PS2MOUSEXY 685
SENDSCAN 728

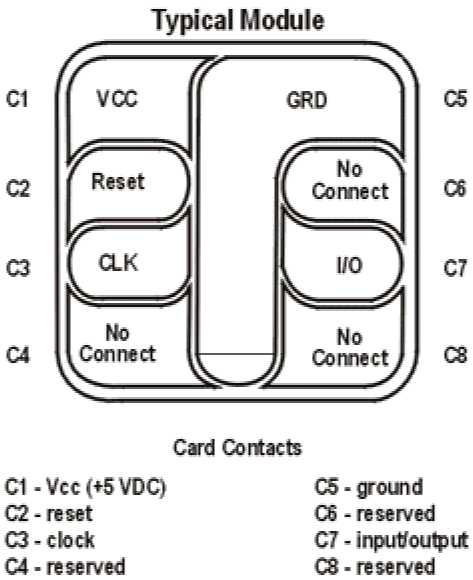
8.14 BCCARD

8.14.1 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.

In the provided example the connections are made as following:

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

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 BCCARD 384 to init the library
BCRESET 861 to reset the card
BCDEF 855 to define your function in the card
BCCALL 855 to call the function in the card

Encryption is not supported by the library yet.

8.14.2 BCDEF

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 384 , BCCALL 855 , BCRESET 861

Partial Example

```
Bcdef Calc(string)
```

8.14.3 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,l as long)
b=b+1
w=w+1
l=l+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 384 , BCDEF 855 , BCRESET 861

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 = D , 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


'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)
S = "1+1+3"                                       ' we want to 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
'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
'          B  a  s  i  c  C  a  r  d     Z  C  1  2  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( l )


'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:
```

```
#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+1
   w=w+1
   l=l+1
end 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$ = Mid$ (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
```

8.14.4 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

' B a s i c C a r d Z C 1 2 3
```

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 [384] , BCDEF [855] , BCCALL [855]

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
' B a s i c C a r d Z C 1 2 3
```

8.15 USB

8.15.1 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 :-)

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 : http://www.atmel.com/dyn/resources/prod_documents/Flip%20Installer%20-%203.3.1.exe
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

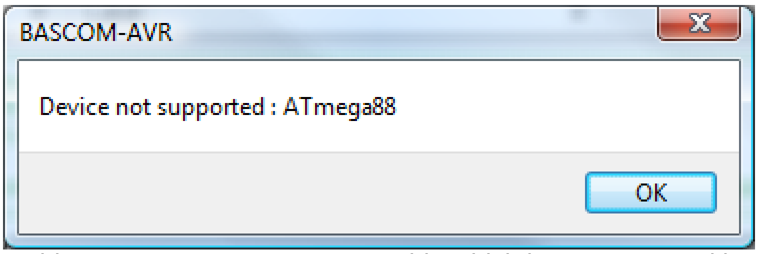
FLIP is a Java application. The BASCOM-IDE can use the FLIP software to program the chip too.
But in order to use the FLIP programmer, you need to install FLIP first.
When FLIP is working, you can select FLIP from Options, Programmer, in order to program quickly without the FLIP executable.
On Vista 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

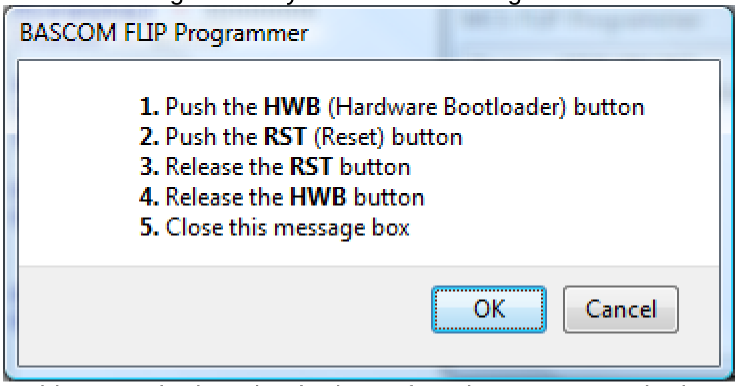
The last line pauses so you can view the result. Notice the . (dot) that will copy the file to the current directory, which is the reason that you need to run this file from the BASCOM application directory.

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:



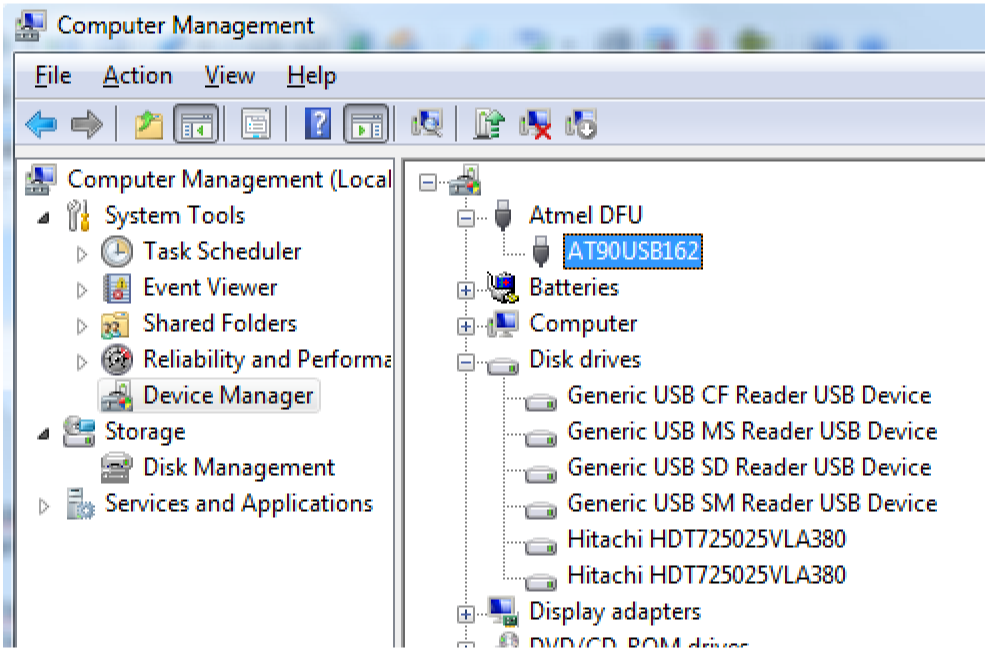
In this case, you try to program a chip which is not supported by FLIP. The Mega88 is not an USB chip so the error makes sense.

The next dialog informs you about a missing DFU device.



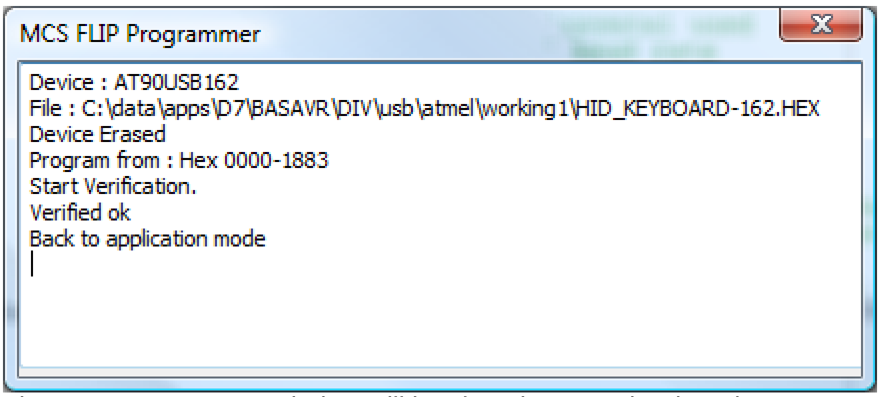
In this case, the boot loader is not found. You can run the boot loader by following the sequence from the dialog box.
In order to make this work, the HWB and RST input both need a small switch to ground.
When HWB is pressed(low) during a reset, the boot loader will be executed.

In the device manager you will find the USB device :



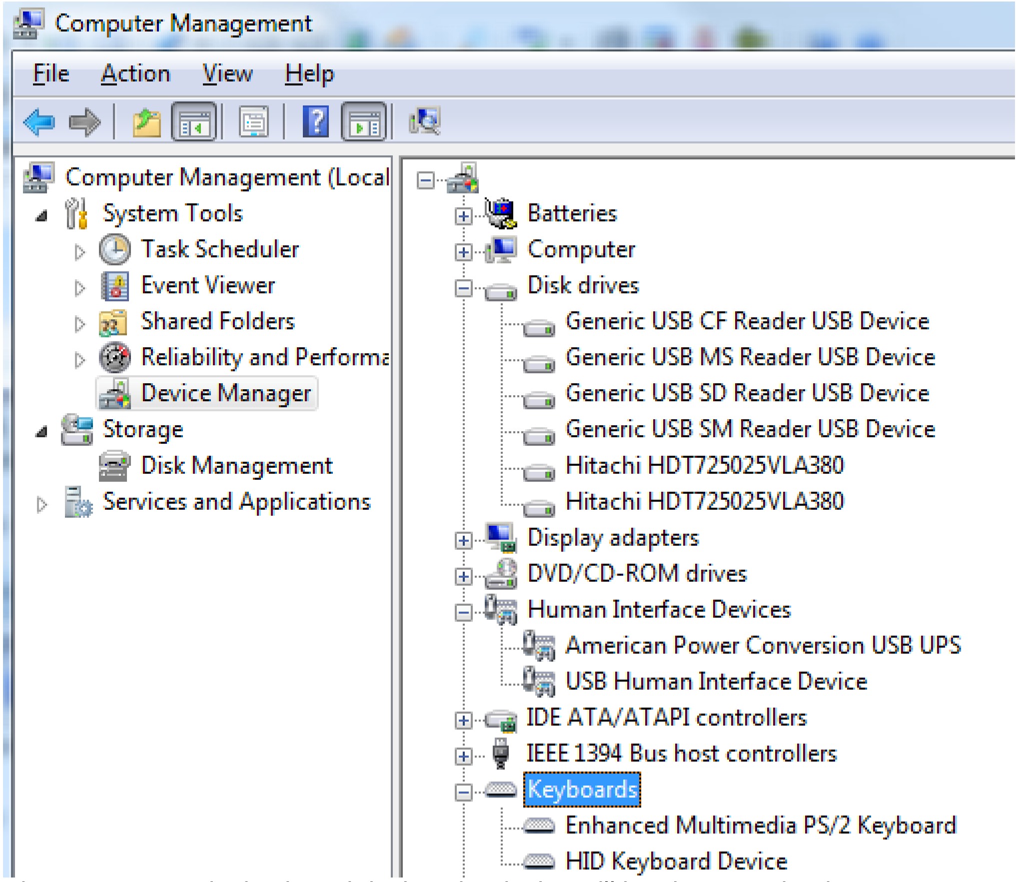
When you have a different chip, a different device will be shown !

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 !



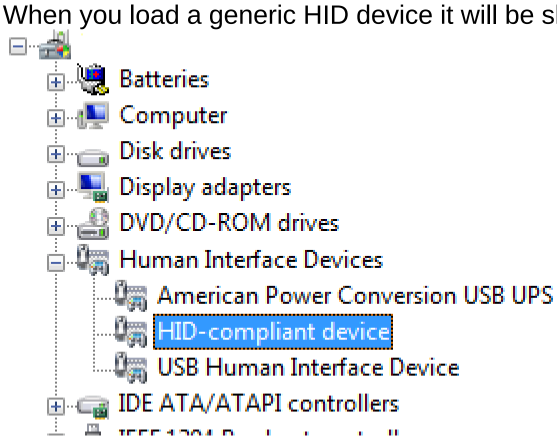
The FLIP programmer window will be closed automatic when the programming succeeds.

The USB device will be shown :



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 ,
Nyet_enabled)
  Call Usb_configure_endpoint(ep_hid_out , Type_interrupt , Direction_out , Size_8 , One_bank ,
Nyet_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
  Usb_task                                     'call this subroutine once in a while
  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                       ' Is_usb_in_ready())
        Uedatx = 1
        Uedatx = 2
        Uedatx = 3
        Uedatx = 4
        Uedatx = 5
        Uedatx = 6
        Uedatx = 7
        Uedatx = 8
        Usb_ack_fifocon                              ' Send data over the USB
      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.

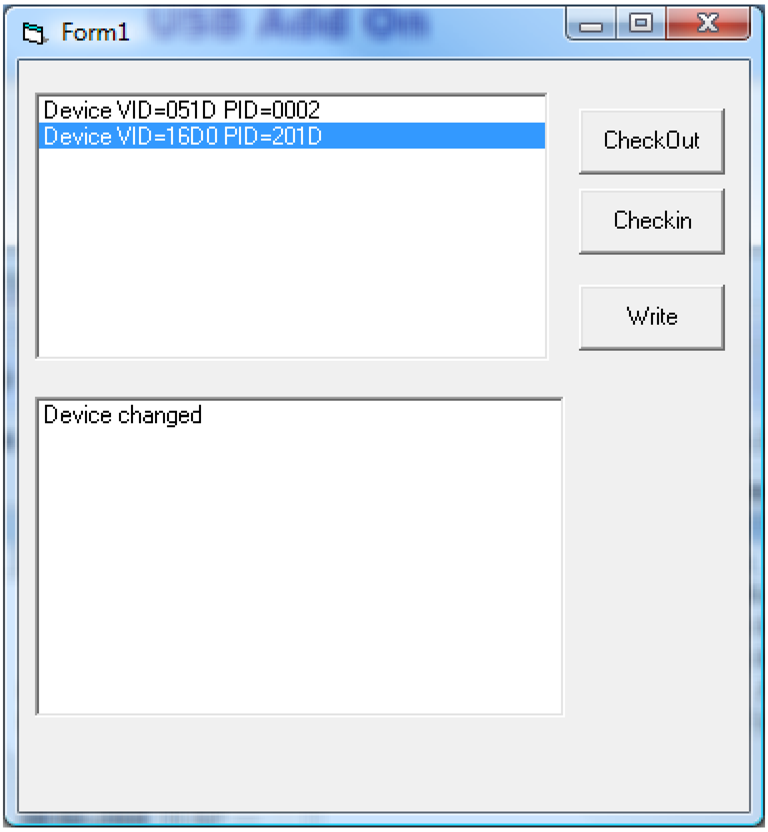
If you want to write some data back, you need to select the end point, and check if you may send data. If that is the case, you assign the data to the UEDATX register and finally, you MUST acknowledge with the USB_ACK_FIFOCON macro.

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

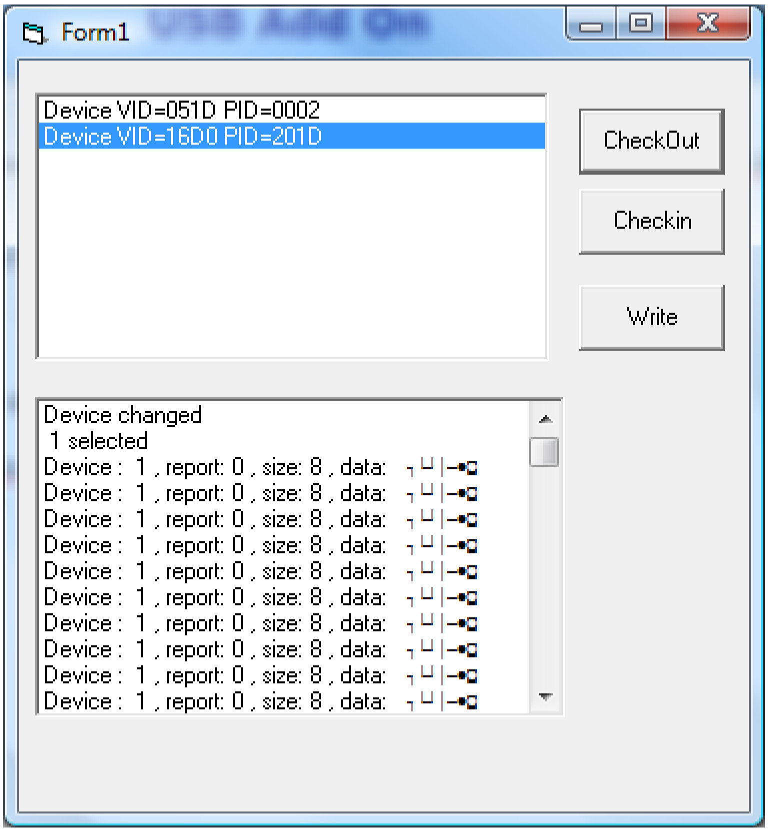
There are plenty of examples on the internet that show how to communicate with HID devices using the windows API.
The HIDX.OCX is an OCX control that can be used for simple communication.
Like all OCX controls, you must register it first with REGSVR32 : regsvr32 hidx.ocx
After it has been registered you can run the VB test application named HIDdemo.exe.

The application will list all HID devices :



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.



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
NumCheckedInDevices	Number of available devices
NumCheckedOutDevices	Number of devices that are checked out and communicating.

NumUnpluggedDevices	
DevThreadSleepTime	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	Parameters • Device : 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

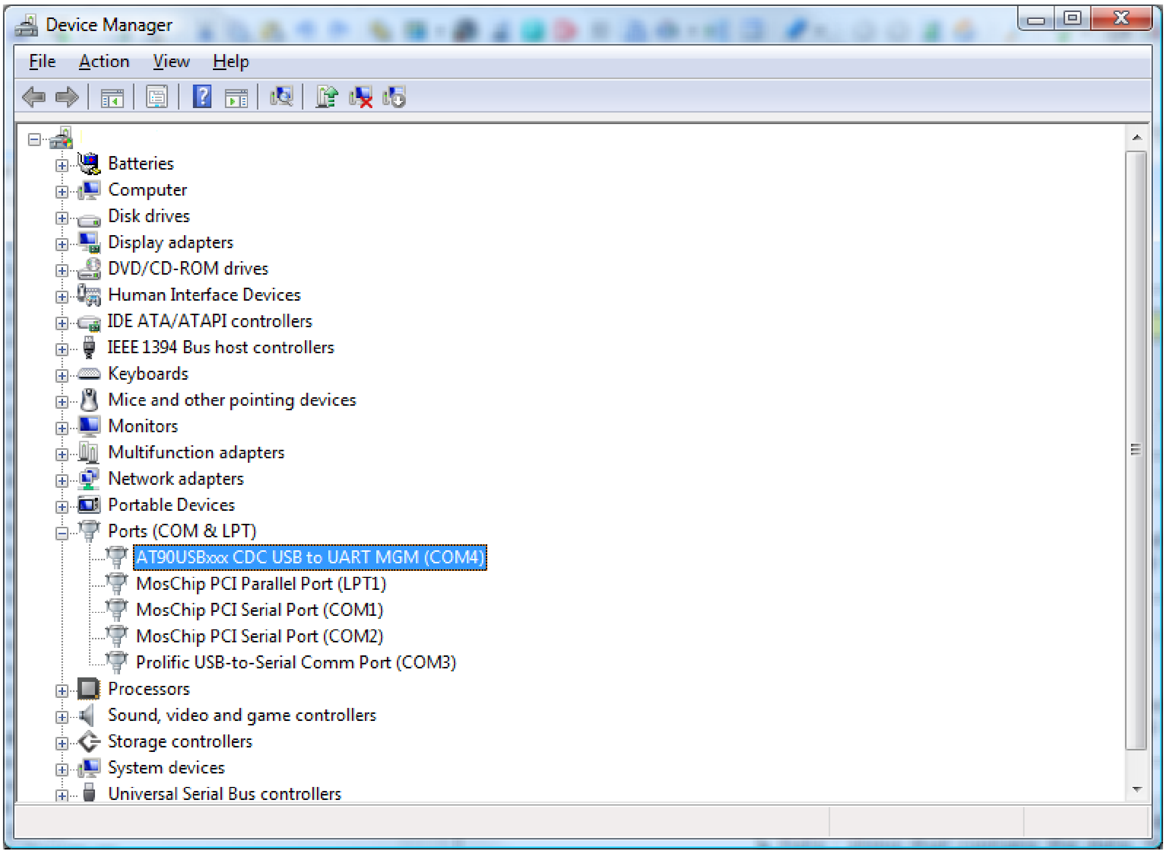
	example : dim ar(8) as Byte For J=1 to Size ar(j) = ASC(Mid(data,J,1)) ' fill the array Next

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.

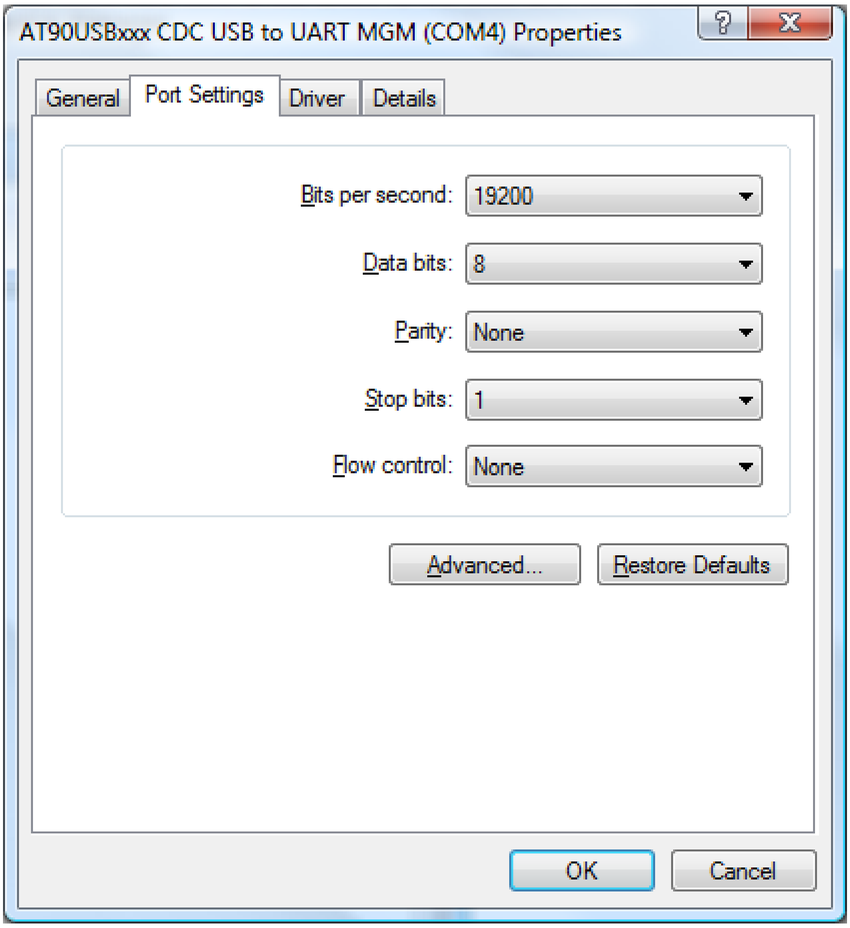
Virtual COM sample

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.



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 :



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.

8.16 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.

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.

Part IX

9 Tools

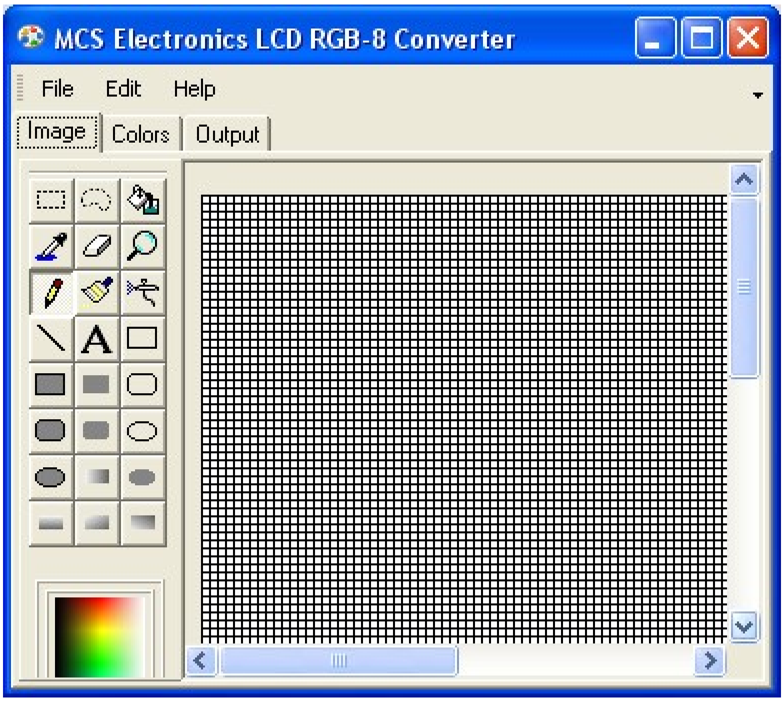
9.1 LCD RGB-8 Converter

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 :



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 :
http://www.mcselec.com/index.php?option=com_docman&task=doc_download&gid=168&Itemid=54

Index

- # -

- $ -

- 1 -

- A -

- B -

- C -

- D -

- I -

- K -

- L -

- M -

- T -

- U -

- V -

- W -

- X -

www.mcselec.com

Making Micro's Easy