Course on BASCOM AVR - (1)

Theoretic/Practical course on BASCOM AVR Programming.
Author: DAMINO Salvatore.

Brief Preface.

The purpose of this course is the description of the essential information about programming and use of small cards based on microcontrollers, available on the market. Obviously this description include also hardware technical information.
For this reasons the course defines some basic hardware elements, required to proceed with various experimentations.

INTRODUCTION

The selected programming language is the **BASIC**. This language certainly is really diffused and, for those people that don't know it, it is surely very easy to learn. In details we have chosen the optimum **BASIC** compiler named **BASCOM AVR**. This compiler is tailored to the numerous **CPU** with **AVR core** and it has the advantage to be available in **Demo** version, **Free of Charge**. The demo version is anyway really complete and it can generate up to **4KBytes** of code, maximum.

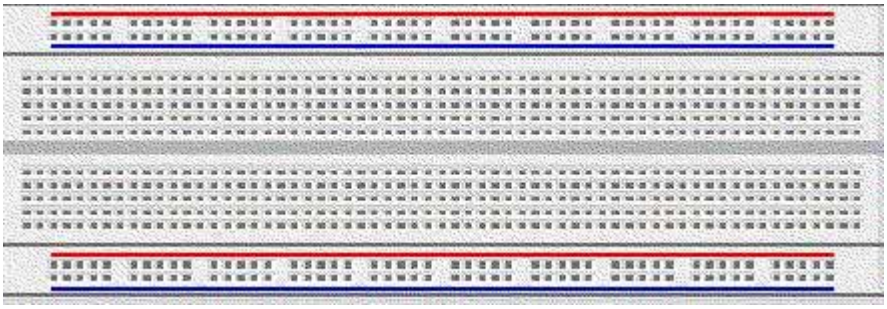
For our course this limit is not a problem in fact, thanks to compiler efficacy, with **4K** of code anyone can develop even medium complexity programs that are certainly sufficient for the didactic purpose. Whenever the user must generates longer, or more complex, programs it is sufficient to acquire the full version of the compiler, capable to generate up to **256KByte** of code.

For the users interested in **BASIC** knowledge enhancement, especially for the features and the syntax, there are some optimum explanation texts. Moreover the **BASCOM AVR** includes a complete documentation that is directly consultable from the Compiler's on line help.

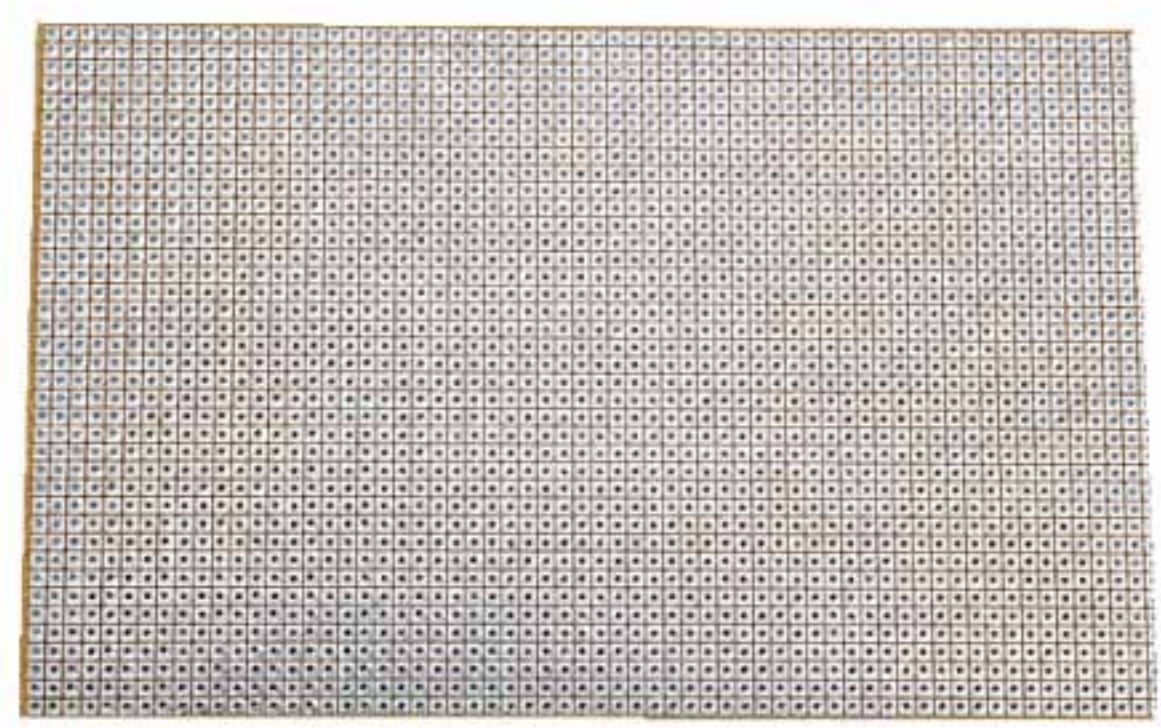
I remind You that on **grifo®** web site it is available a manual in **PDF** format and in English language, that describes each features and instructions. For those that have difficulties with on line help and prefers a complete manual, it could be really useful.

REQUIRED HARDWARE

In order to realize easy to use examples it is necessary to define some unique hardware elements used during the experiments and tests of the course.
This definitions coincide with some different elements as the **GMM AM08** that is a **CPU Mini Module** capable to executes all the projects, and the **GMM TST3** that is the support board where many tests can be performed.



As an alternative, for those users that prefer to directly manage hardware components, it can be used a **Breadboard** where the suggested circuits can be easily developed; in addition, the various projects can be even mounted on a **Prototype board** by using its numerous pods.

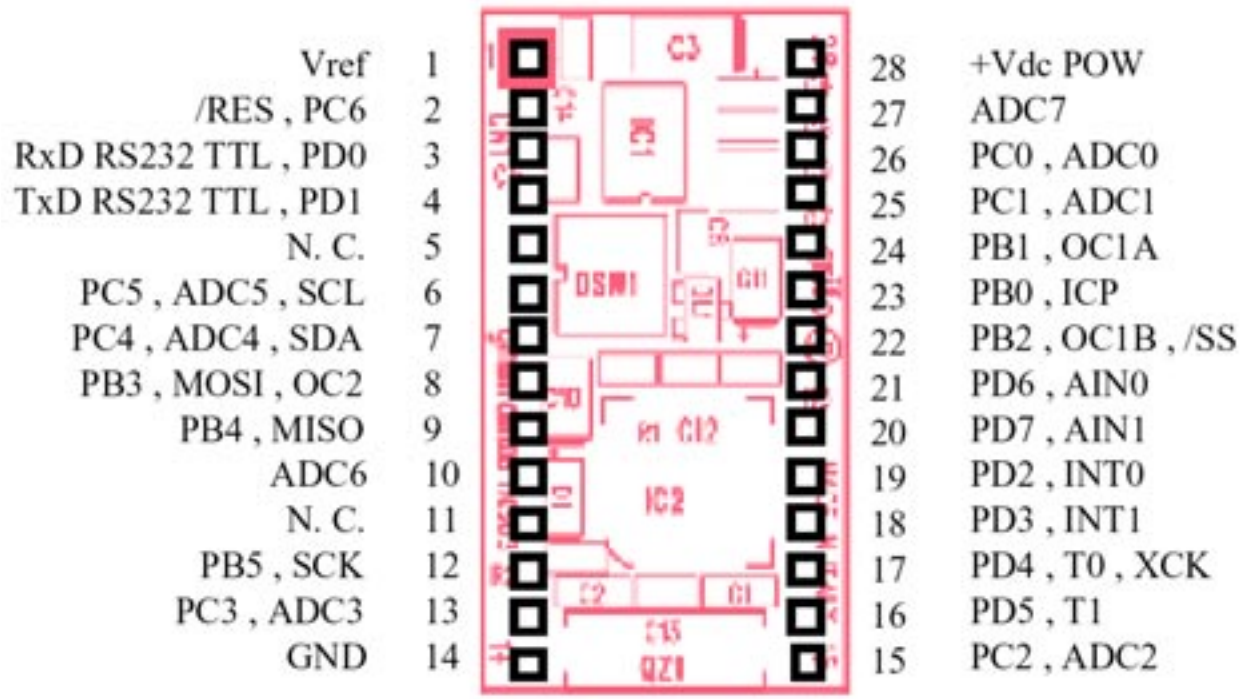


The projects are described with gradual increasing complexity in order to guide the reader during his understanding and his knowledge enlargement.

GMM AM08 Mini Module

This **Mini Module** has **28 pins** and it is based on an **Atmel CPU** provided of **8K internal FLASH memory**; **1KBytes** of **RAM**; **512Bytes** of **EEPROM**; **23 I/O** lines; etc. It has on board all the necessary resources for its correct functionality.

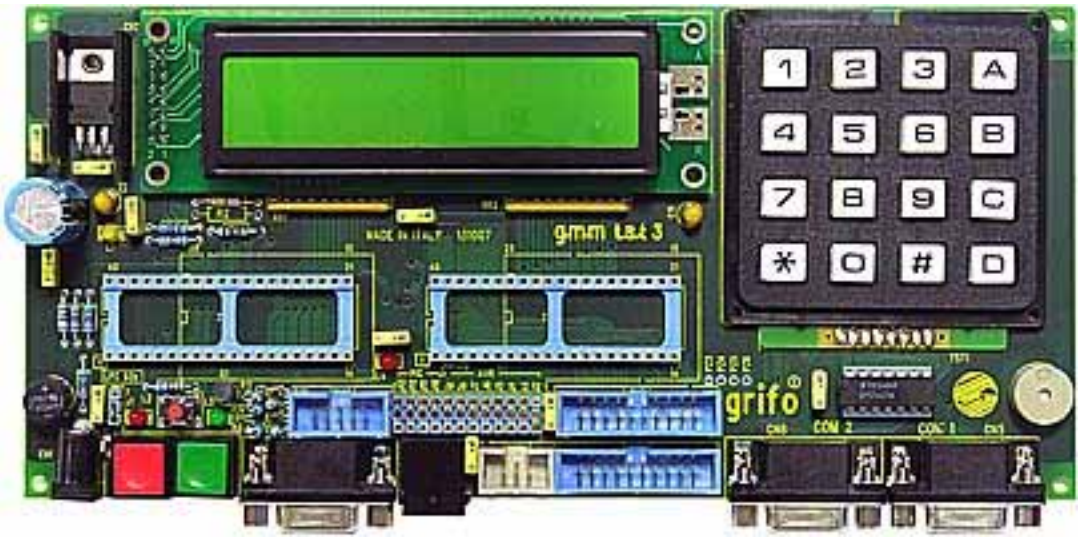
One of the advantages in the use of this **Mini Module** is the possibility to work without any external Programmer. In fact it includes internally a specific program, named **Bootloader** (**2K** length**)**, that perform the erasing and programming operations of the **FLASH** with user code. Everything is comfortably managed by a **Freeware** program, executed on development PC and downloadable from **grifo®**.



This is an interesting commodity that allows to program and re-program the chip for many times, by using only the **PC** serial line.

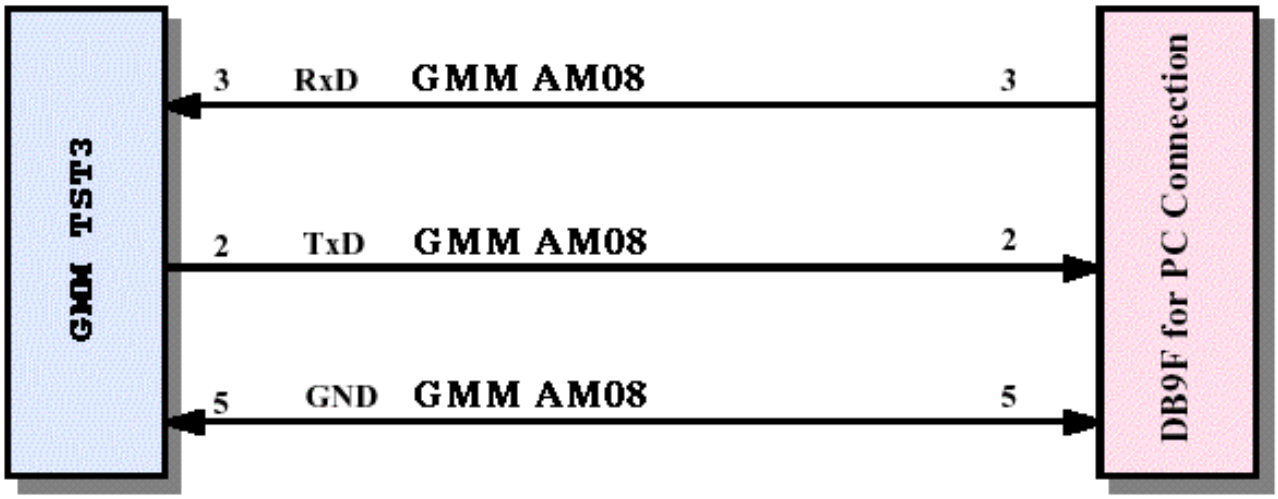
GMM TST3 Experimental Support Board

This card, provided of **2** sockets with **40** pins, has been developed in order to test and evaluate many of the **Mini Modules** produced by **grifo®**, with either **28** or **40** pins.



It is supplied with a series of examples for the available **Mini Modules** and it allows a fast learning tools of the same modules.

The first things to do in order to correctly use this card is the serial connection to a **PC**. The communication cable could be acquired or realized, by using the following indications.

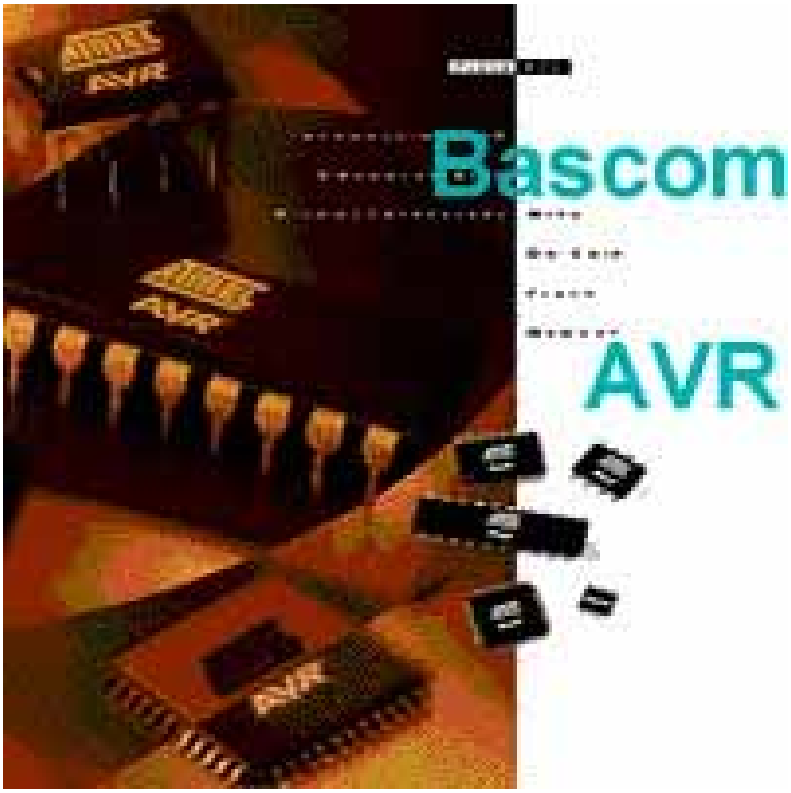


The connection requires only **3** wires as described in the figure.

BASIC Language

The **BASIC** language (the word means ***B****eginner's* ***A****ll purpose* ***S****ymbolic* ***I****nstruction* ***C****ode*) has been developed in **1963** by **Kemeny and Kurtz**. The first **BASIC** program run the 1st May **1964** at **4,00** o' clock.

The **BASIC** program is generated by a suitable **Editor** that allow to type the required **Statements** and **Instructions** with an ordered sequence, from left to right and from top to bottom, as in a normal letter.



The first **BASIC** was **Interpreter** type: this means that each program line was first **interpreted** and then immediately **executed**. This solution type has numerous advantages but it has the intrinsic defect to be really slow in execution. In order to remove this defect and to increase the execution speed, the **BASIC** compilers have been developed.

Thanks to this new technique the execution speed has been increased many times and the compiler can easily manage even very fast events. The **BASCOM AVR** is included in this efficient category of programming languages.

Foundamental Parts of Programs

Normally the parts always available in a **BASIC** program, are:

Definitions = they are all the preliminary information and the necessary associations required by other parts of the program, plus the possible compilers configurations or directives.
With **BASCOM AVR**, among the definitions there is the necessary **.DAT** file that contains all the information about the used microcontroller.

Declarations = they are all the declarations of data structures (**Constants**, **Variables,** etc.), subroutines and functions used in the program.

Statements = they are the real program and they decide the performed operations; this part includes the **BASIC statements**, the operators, the subroutines and functions calls, with possible arguments parameters, the data structures, the remarks, etc.

Moreover it is frequently available an additional component part, defined:

Title block = it is the first part of the program and it coincides with a long remark that shortly lists the program features, useful to recognize, describe and execute it.

BASCOM AVR - General Features

- ***Structured BASIC*** *that supports* ***labels****.*

- *Structured programming through:*
  IF-THEN-ELSE-END IF, DO-LOOP, WHILE-WEND, SELECT- CASE.

- *Fast* ***Machine code*** *in place of* ***Interpreted code***

- *The names of variables and labels can reach the maximum length of* ***32*** *characters.*

- *Variable type:*
  Bit, Byte, Integer, Word, Long, Single*,* ***Double*** *and* ***String****.*

- *Variables can be either* ***global*** *or* ***local****.*

- ***Compiler*** *capable to generate code for each* ***µP*** *of* ***AVR*** *family.*

- ***Statements*** *are higly compatible with* ***Microsoft VB/QB****.*

- *Special Directives, Instructions and Statements tailored to* ***LCD display*** *(alphanumeric and graphic),* ***I2C BUS devices, 1WIRE*** *and* ***SPI*** *management,* ***PC keyboards****, matrix keyboards,* ***infrared*** *transceiver, software* ***serial lines****, memory cards, etc.*

- *Complete support for subroutine and functions* ***arguments****.*

- *Integrated* ***Terminal Emulator*** *with* ***Download*** *capabilities.*

- *Integrated* ***Simulator*** *for debug and test phases.*

- ***Editor*** *with colour syntax features, that simplify the recognition of the program component parts.*

- *Integrated support of many device* ***Programmers*** *both internal and external.*

- *Contextual* ***On line Help****.*

- *Availability of additional libraries that increase language functionalities.*

BASCOM-AVR Supports the Following Statements

1WIRE

They manage the communication with Dallas devices provided of 1wire interface.

1WRESET , 1WREAD , 1WWRITE , 1WSEARCHFIRST , 1WSEARCHNEXT , 1WVERIFY , 1WIRECOUNT

Struttures, Conditional jump, Loops, Flow control

They execute different parts of program according with conditions that can be true or false, or they change and repeat execution flow.

IF-THEN-ELSE-END IF , WHILE-WEND , ELSE , DO-LOOP , UNTIL, EXIT DO , EXIT WHILE , SELECT CASE-END SELECT , FOR-NEXT, TO, DOWNTO , STEP , EXIT FOR, ON...GOTO/GOSUB , CALL , GOSUB , GOTO , EXIT SUB , EXIT FUNCTION , END SUB , RETURN , STOP , END.

Configuration

The configuration statements initializes many hardware devices into a required state.

CONFIG , CONFIG ACI , CONFIG ADC , CONFIG BCCARD , CONFIG CLOCK , CONFIG COM1 , CONFIG COM2 , CONFIG DATE , CONFIG DMXSLAVE , CONFIG PS2EMU , CONFIG ATEMU , CONFIG I2CSLAVE , CONFIG INPUT , CONFIG GRAPHLCD , CONFIG KEYBOARD , CONFIG TIMER0 , CONFIG TIMER1 , CONFIG LCDBUS , CONFIG LCDMODE , CONFIG 1WIRE , CONFIG LCD , CONFIG SERIALOUT , CONFIG SERIALIN , CONFIG SPI , CONFIG LCDPIN , CONFIG SDA , CONFIG SCL , CONFIG DEBOUNCE , CONFIG WATCHDOG , CONFIG PORT , COUNTER0 , COUNTER1 , CONFIG TCPIP , CONFIG TWISLAVE , CONFIG SINGLE , CONFIG X10 , CONFIG XRAM , CONFIG USB

Conversion

They converts a data from one format to a different one.

BCD , GRAY2BIN , BIN2GRAY , BIN , MAKEBCD , MAKEDEC , MAKEINT , FORMAT , FUSING , BINVAL , CRC8 , CRC16 , CRC16UNI , CRC32 , HIGH , HIGHW , LOW

Time and Date

They manages the current time and date plus the elapsed time.

DATE , TIME , DATE$, TIME$, DAYOFWEEK , DAYOFYEAR , SECOFDAY , SECELAPSED , SYSDAY , SYSSEC , SYSSECELAPSED

Delay

These statements delay the program execution for a specified amount of time.

WAIT , WAITMS , WAITUS , DELAY

Compiler Directives

These are special instructions for the compiler that defines many features of the program and they can replace IDE settings.

$ASM , $BAUD , $BAUD1 , $BGF , $BOOT , $CRYSTAL , $DATA , $DBG , $DEFAULT , $EEPLEAVE , $EEPROM , $EEPROMHEX , $EXTERNAL , $HWSTACK , $INC , $INCLUDE , $INITMICRO , $LCD , $LCDRS , $LCDPUTCTRL , $LCDPUTDATA , $LCDVFO , $LIB , $LOADER , $LOADERSIZE , $MAP , $NOCOMPILE , $NOINIT , $NORAMCLEAR , $PROJECTTIME , $PROG , $PROGRAMMER , $REGFILE , $RESOURCE , $ROMSTART $SERIALINPUT , $SERIALINPUT1 , $SERIALINPUT2LCD , $SERIALOUTPUT , $SERIALOUTPUT1 , $SIM , $SWSTACK , $TIMEOUT , $TINY , $WAITSTATE , $XRAMSIZE , $XRAMSTART , $XA

Files

The files statements can be used with AVR-DOS library: an operating system with disks support.

BSAVE , BLOAD , GET , VER , DISKFREE , DIR , DriveReset , DriveInit , LINE INPUT , INITFILESYSTEM , EOF , WRITE , FLUSH , FREEFILE , FILEATTR , FILEDATE , FILETIME , FILEDATETIME , FILELEN , SEEK , KILL , DriveGetIdentity , DriveWriteSector , DriveReadSector , LOC , LOF , PUT , OPEN , CLOSE

Alphanumeric LCD Display

These statements work with text on standard alphanumeric displays.

HOME , CURSOR , UPPERLINE , THIRDLINE , INITLCD , LOWERLINE , LCD , LCDAT , FOURTHLINE , DISPLAY , LCDCONTRAST , LOCATE , SHIFTCURSOR , DEFLCDCHAR , SHIFTLCD , CLS

Graphic LCD Display

These statements extend the functionalities of alphanumeric displays by adding graphic visulaizations.

GLCDCMD , GLCDDATA , SETFONT , LINE , PSET , SHOWPIC , SHOWPICE , CIRCLE, BOX

I2C BUS

These statements allows the communication with devices provided of I2C BUS interface either through TWI hardware controller or generic I/O lines software emulated control.

I2CINIT , I2CRECEIVE , I2CSEND , I2CSTART, I2CSTOP, I2CRBYTE, I2CWBYTE

Input, Output

Statements dedicated to management of microcontroller I/O pins and Port.

ALIAS , BITS , BITWAIT , TOGGLE , RESET , SET , SHIFTIN , SHIFTOUT , DEBOUNCE , PULSEIN , PULSEOUT

Microcontroler

Statements provided for management of specific microcontroller features.

IDLE , POWERDOWN , POWERSAVE , ON INTERRUPT , ENABLE , DISABLE , START , VERSION , CLOCKDIVISION , CRYSTAL

Memories

These statement allow to read and write the SRAM , EEPROM and FLASH memories.

ADR , ADR2 , WRITEEEPROM , CPEEK , CPEEKH , PEEK , POKE , OUT , READEEPROM , DATA , INP , READ , RESTORE , LOOKDOWN , LOOKUP , LOOKUPSTR , CPEEKH , LOAD , LOADADR , LOADLABEL , LOADWORDADR , MEMCOPY

Infrared Remote Control

Statements that receive/send infrared commands from/to a remote controller.

RC5SEND , RC6SEND , GETRC5 , SONYSEND

RS 232 or Console

These statements manage serial communication through hardware UART or software emulated ones. The serial lines can be used as serial console.

BAUD , BAUD1 , BUFSPACE , CLEAR , ECHO , WAITKEY , ISCHARWAITING , INKEY , INPUTBIN , INPUTHEX , INPUT , PRINT , PRINTBIN , SERIN , SEROUT , SPC , OPEN , CLOSE , MAKEMODBUS

SPI

These statements allows the communication with devices provided of SPI interface either through hardware controller or generic I/O lines software emulated control.

SPIIN , SPIINIT , SPIMOVE , SPIOUT

Strings

Statements dedicated to strings management.

ASC , UCASE , LCASE , TRIM , SPLIT , LTRIM , INSTR , SPACE , STRING , RTRIM , LEFT , LEN , MID , RIGHT , VAL , STR , CHR , CHECKSUM , HEX , HEXVAL , QUOTE

TCP/IP

The TCP/IP statements support network protocols by using W3100/IIM7000/IIM7010 modules.

BASE64DEC , BASE64ENC , IP2STR , UDPREAD , UDPWRITE , UDPWRITESTR , TCPWRITE , TCPWRITESTR , TCPREAD , GETDSTIP , GETDSTPORT , SOCKETSTAT , SOCKETCONNECT , SOCKETLISTEN , GETSOCKET , CLOSESOCKET , SETTCP , GETTCPREGS , SETTCPREGS , SETIPPROTOCOL , TCPCHECKSUM

Mathematic, Logic, Trig

These are the functions and operators for the main operations on variables and numeric values.

*+ , * , - , / , \ , ^ , < , > , >= , <= , = , <> , ABS , ACOS , AND , ASIN , ATN , ATN2 , EXP , RAD2DEG , FRAC , TAN , TANH , COS , COSH , LOG , LOG10 , ROUND , INT , MAX , MIN , NOT , SQR , SGN , OR, POWER , SIN , SINH , FIX , INCR , DECR , DEG2RAD , SHIFT , ROTATE , RND , XOR*

Data structures

These statements declare and manage the data structures.

DIM , BIT, BYTE , INTEGER , WORD , LONG , SINGLE , DOUBLE , STRING , CONST , LOCAL , DEFBIT , DEFBYTE , DEFINT , DEFWORD , DEFLNG , DEFSNG , DEFBBL , BYVAL ,

Various statement collection

These statement are not included in any of the previous groups.

DBG , DECLARE FUNCTION , DEBUG , DECLARE SUB , DTMFOUT , EXIT , ENCODER , GETADC , GETKBD , GETATKBD , GETRC , POPALL , PS2MOUSEXY , PUSHALL , SENDSCAN , SENDSCANKBD , SOUND , STCHECK , SUB , SWAP , X10DETECT , X10SEND , READMAGCARD , REM , #IF , #ELSE , #ENDIF , READHITAG

BASCOM AVR Compiler Structure

The **BASCOM AVR** compiler is a complete development tools that includes all the necessary items capable to generate, in a really efficient way, the management **Firmware** of any **Hardware**, based on **AVR Core CPU**.
The main modules of this tool are:

EDITOR

It is a specialized program that allow to write and save the management firmware source. This program has a list of commodities that simplify the Editing operations. Among these commodities there are, for examples, the Coloured Syntax that speed up the recognition of source parts, the automatic indentation, the remarks alignement at the end of lines, the remark/un-remark of program blocks, Bookmarks management that simplify movements inside long programs, etc.

COMPILER

This section is charged to convert the program source, written with **Editor**, in executable code for **CPU**.
Once compiled the program, it must be checked for possible errors signalized in proper window and then proceed with following steps.

SIMULATOR

It allows to check the right functionality of a compiled program, or a part of it, by following the execution on the **PC** that simulates the **AVR CPU**. In this way the user can perform preliminary tests without saving the compiled code on the **FLASH** memory of the used card.

TERMINAL Emulator

It allows the communication, between the **PC** and the target card, by using the on board **RS 232** serial line. All the features of communication protocol, as for example the **Baud Rate**, are selectable in a proper window of the compiler, dedicated to **Terminal Emulator**.

OPTIONS and UTILITY

Among **BASCOM AVR** components there are some utilities concerning displays managements, formats conversions, libraries use, etc. Moreover a rich list of options defined by user allow to define many functionalities of the same **BASCOM 8051** and of possible external programs launched by the compiler. For example the options define colours and shown information, printing attributes, use of programmers, simulation modalities, adds libraries and functions, open documentation and diagrams on microcontroller, etc.

DAT File

In order to correctly operate with **BASCOM AVR** one of the first operations to perform is to search, or generate, a file that describes all the resources of the used **CPU**. Many of these **.DAT** files are supplied with the compilers, as it is for **GMM AM08** module, that is named **M8DEF.DAT**. Anyway You can find this file, ready to use, on our web site.

Configuration File

In order to simplify the **BASCOM AVR** use, it generates a configuration file where are saved all its settings. These settings coincides with the configurations of the numerous I**DE** windows and they can be directly performed by the user, as described in the program title block. In details the configuration file has the same name of source file and the **.CFG** extension; if the file exist the **BASCOM AVR** load all the configurations during the open phase of the program and vice versa. Moreover when the source program is saved, the configuration file is automatically generated or updated.

Additional Program AVR Bootloader grifo®

This program is not directly integrated in compiler architecture but it is anyway an essential elements for all **AVR Mini Modules**. I am talking about the program that writes the **FLASH** and **EEPROM** memories of **CPU** without any additional external programmer.

This program is supplied by **grifo®** company and it writes an executable code on internal **FLASH** or data on internal **EEPROM** of **CPU**. The code generated by **Compiler** is sent to **GMM AM08** trough the **AVR Bootloader grifo®** program, that stores it permanently on **Mini Module**. All the instructions about **AVR Bootloader grifo®** use are available directly on card technical manual or inside next course chapter.

Example.001. LED and Push Button Management

Definitions:
$regfile ; $romstart ; $crystal ; $hwstack ; $swstack ; $framesize ; $map ; Alias

Declarations:
Dim …. As Bit

Instructions:
Do ; Loop ; ' ; End

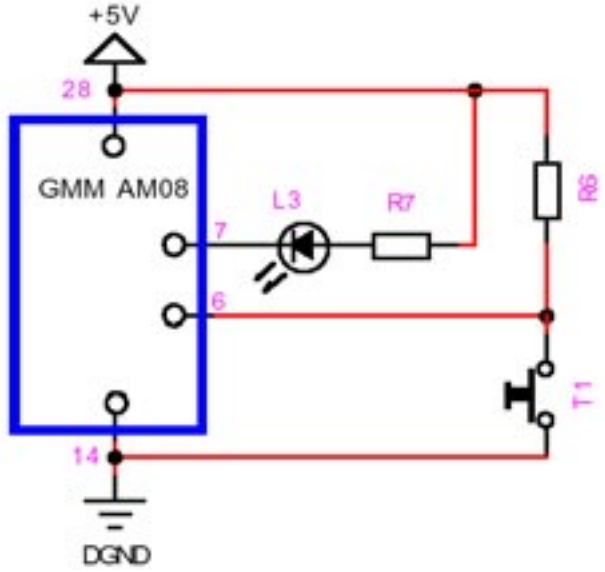
Operators:
=

This program executes simple operations on digital **I/O**, by using one **push button** and one **LED** available on **GMM TST3**.

The referenced electric diagrams are those on pages **43-45** (from B-1 to B-3) of **GMM TST3** manual.

- The used **I/O** lines are:
  - pin **13** of **GMM TST3** Z1 socket (= pin **7** of **GMM AM08**) connected to green LED L3 through R7 and to green push button T2;
  - pin **12** of **GMM TST3** Z1 socket (= pin **6** of **GMM AM08**) connected to red LED L2 through R6 and to red push button T1.
- The program manages the pin **6** as an input lines and the pin **7** as an output lines.
- At power on both **LED** are turned off.
- By pushing the **T1** push button, the **LED L2** is turned on by hardware, as it is electrically connected, and either **LED L3** is turned on by software.
- By releasing the **T1** push button both **L2** and **L3** will turn off.

Below You can examine the simplified electric diagram that must be realized, on a **Breadboard** or a **Prototype board**, in order to use the described **Example.001**.



Electric Diagram Used by Program

Program Development

I take advantage from this example to describe, in a really concise manner, the main elements that realize a normal management program. It is important underline that, even when the program is very short, as in this chance, it is a good habits to provide all the following elements.

Especially when You periodically spend time on programming activity, I suggest You to add large remarks and comments regarding all the steps and the adopted decisions. In fact, what today is completely clear, after a time interval (short too) it could become less clear and sometimes even incomprehensible.

Following this **Mini Practical Course**, anytime new arguments are introduced (as a new **Statement** or something else), I suggest You to consult the on line help of **BASCOM AVR**. By reading the reported information You will have a solid knowledge of the language, in a short time.
Below are listed the most important elements normally inserted in a program.

Title Block

It is placed at the beginning of the source and it always start with the "*Apostrophe*" character. The header lists all the information useful to identify the program and it is commonly rounded by a frame. It is composed by a title with the program name, the compiler version, the hardware used to execute the program, name of the company and of the author that developed the same program.

Program Description

Also this element is always preceded by an Apostrophe and it shortly describe the program functionalities. Moreover it indicates the used hardware items involved in the program and how they are interconnected.

IDE Configurations

This element lists the necessary configurations that must be performed in the Integrated Development Environment (**IDE**) of **BASCOM AVR** in order to correctly use the program.

Compiler Directives

Here are listed all the directives required by compiler. Among them, usually there is the **.DAT** file of the used microcontroller, the Code start address on **FLASH**; the Data start address on internal **RAM**; etc.

Definitions

The definitions normally specify with details the resources used by the program. For each resource (as for example a LED or a serial interface, a display, a key, etc.) it is defined the name and the signal connected or associated to.
In the **Example.001** are defined the microcontroller's pins connected to T1 Push button and L3 LED plus the register's bit used to acquire and set the same signals.

Constants Declaration

This element declares all the **Constants** used by management program, with their values.
In this example there aren't constants.

Variables Declaration

This element declares all the **Variables** used by management program, with their type.
In this example there is the **Boolean variable T1stat** that save the status of **T1** Push button.

Subroutines Declaration

In this element are declared all the **Subroutine** and **Functions** realized in management program, with the scope to facilitate reading and reduce code size.
In this example there aren't subroutines.

Main Program

The program start with **Main** and finish with **End**. In the middle there are all the statements and many of the other elements (I.e. Variables, Constants, Operators, Subroutines, etc.) that establish the real functionality of the program.
The example, in the first instructions, **initializes** the signal connected to **T1** Push button as digital input and the signal connected to **L3 LED** as digital output, at high level.

After these operations there are two new **Statements** named **Do** and **Loop**.
The program executes the instructions included between **Do** and **Loop** in an endless cycle, and in details:

- assign the status of **T1** Push button to **T1stat** variable;
- set **L3 LED** with the status of **T1** Button. This mean that when **T1** is released the **L3** will be turned off; when **T1** is pressed then **L3** will be turned on and it is lighted.

The just described program can be opened, examined and compiled directly with **BASCOM AVR** IDE, by executing the following simple steps:

- Check the availability of **M8DEF.DAT** definition file into the directory where the **BASCOM AVR** is installed

  \MCS Electronics\BASCOM-AVR

  and copy it if not present..

- Execute the **BASCOM AVR** and, once its **IDE** is started, load the source file (with .BAS extension), trough the *File* | *Open* menu. The file to load is those involved in this example, that has the name

  uk_BASAVR_001.BAS.

- Open the configuration window of **BASCOM 8051** compiler, through the *Option* | *Compiler* | *Chip* menu, then check the settings described in the opened program (see IDE Configurations) and perform them if they are not already done, and at the end confirm with *Ok* button.
  All these configuration are already done if it has been downloaded from internet also the proper configuration file ***uk_BASAVR_001.CFG***, as described in **BASCOM AVR** documentation.

- Compile the example program source with a simple pressure of rapid key ***F7***, or select the command ***Program | Compile***, and check that no errors are found. In this way You must obtain the **.HEX**.
  The compilation belong a variable time that depend on used **PC**; in any case You must wait the completion of both passes, properly signalized by a status window displayed during the compilation. By remembering that second pass is very fast and it cold be not readable, at the end You must check that no errors are reported in the low area of **IDE**.

- The **HEX** file generated by compiler coincides with the executable code for the microcontroller, coded with an abounding format, that ensure its validity. In the course this format is not examined closely, but the interested readers can find a detailed documentation on internet.