



AMICUS18®

www.protonbasic.co.uk



Powered by Proton Development Suite® Compiler of Crownhill Associates Limited©

A UNIVERSAL PDS BOOTLOADER

USING THE TEMPLATES: PART1

PIC®, MPLAB®, PICkit3® and ICD3® are registered trademarks of Microchip Technology Inc®.
Proton Development Suite® or PDS® are a registered trademark of Crownhill Associates Limited®.

The project has been developed and written by Alberto Freixanet.
The document has been edited by John Drew.

Manual version 1.1: Typos & minors corrections.

INTRODUCTION:

The "AGV Bootloader" allows loading firmware via serial port (UART1, UART2 or more) without the need for dedicated host software using the "Proton Development Suite®". Modern microcontrollers have an interesting ability to write to their own memory and this feature opens up an entirely different programming method. A special program running on the microcontroller could communicate with an external device and receive data containing the desired program that it could then write to itself.

History:

After a bad experience with the fuses with an external bootloader, I decided to try to learn how these systems work. It has been a long journey for me to understand how to write to a PIC® and the different protocols used. The documentation in the web is not always clear or precise, sometimes erroneous. But I was puzzled because the protocol of specialized programs that send the hex file are always missing. I could not waste much time for decoding firmware in the "C" language.

The Challenge:

To get rid of the specialized programs running in a computer, I decided that all the decoding software of the HEX file would be done in the PIC®, using only the PDS® compiler and the PIC® datasheet.

What to choose?:

Reading the experience of others, I realized that the best and most reliable solution would be to create a unique and customized firmware FOR EACH USER PROJECT. In this way you could get several firmwares for a given PIC®. What is the problem?

General Features:

1. The bootloader has been written with the Proton Basic Compiler Version 3.6.0.3. All routines are included in one BASIC file only.
2. It does not require a special download management PC application.
3. The Host computer program would be simple RS232 communicator software.
4. It requires a handshaking software using XON/XOFF.
5. That is the most automatic possible. The user does not waste time writing the code.
6. Customized firmware for each user project.
7. Self-protects so that large user applications do not over-write the bootloader.
8. The bootloader code does not use the interrupt and so avoids some issues.
9. Successful downloads start user application automatically.
10. Only uses three wires for communication: TX, RX, ground and the pin Reset for starting.
11. All resources are released after download and available to the user application.
12. Highly adaptable/maintainable using the PDS® compiler.
13. Can be compiled for many combinations of crystal speeds and PIC® family members.
(PIC18® ROM \geq 128KB)
14. Many controls have been introduced in the code making a very secure bootloader.
15. Several additional routines are available to debug the downloading file.
16. Several additional OPTIONS are available for the PDS® user.
17. Before downloading a Password access system is available for the user.

Specific Features:

- To Write programs to devices with up to 128KB ROM, minus the size of the bootloader.
- To Write discontinued or plain user code.
- To Write to Program FLASH, ID Locations and Data EEPROM.
- Download user code by using the UART1 or UART2, at 115200, 57600, 38400 Baud adjusted automatically to maximum speed according to the FOSC used.
- Wide range of usable FOSC: 64, 48, 40, 32, 25, 24, 20, 16, 14 (14.32MHz), 12, 10.
- FOSC = 80 Mhz for the PIC18FxxK20 series.
- The internal oscillator of the PIC® is available for downloading. (maximum speed 57600 Baud)

- The firmware uses a double UART buffer to increase speed and reliability.
- Self write the boot "Goto PIC#Loader#Boot" in position 0 of ROM. (HSM only)
- Can be compiled with all options of the Watch-dog Timer.
- To control the PIC® device ID before write.
- Writing the Config Fuses is not supported for security reasons.

Precautions:

According to the version of the compiler there may be an error. I have had the experience of finding a compilation failure or a substantial difference in the length of the asm file. Also I have noted an error affecting the bootloader in the (HSM) High Side of ROM. Do not use the Compiler Version PDS 3.5.9.9.

To avoid these problems, I have made a large number of compilations and obtained a wide variety of firmware examples to ensure the loader is as standard as possible. In special cases the user will need to configure his example and compile it.

WARNING .def & .PPI files:

I recommend a user should verify the Picxxxx.def and Picxxxx.PPI files by reading these files with NotePad and to compare with the datasheet before building a new firmware for a new PIC®. You will definitely find some differences. As I have bad luck, in some PIC®'s I had tested some .def and .PPI files that were erroneous about the Write block, the number of Banks or other information. I enclose all copies of the new modified files. Before doing anything make a copy of all .def & .PPI files of the PDS® compiler.

If these files do not contain errors, modifying them will no longer be necessary to introduce the "bankAend" parameter (which is forgotten) because this information is written manually in the description of the PIC®.

To be able to check parameters used in the file.def and file.ppi, when compiling the code of the bootloader generates important parameters for the bootloader. You would have to verify the correct values reading the asm file.

```
CTL_WRITE_FLASH_BLOCKSIZE
CTL_ERASE_FLASH_BLOCKSIZE
CTL_CODE_FLASH_SIZE
CTL_RAM_SIZE
CTL_RAM_BANKS
CTL_EEPROM_SIZE
CTL_BANKAEND_SIZE
CTL_NUMBER_UART
CTL_EEP_ERASE_LOW_ADDR      \ Configured Low Address to erase the eeprom.
CTL_EEP_ERASE_HIGH_ADDR     \ Configured High Address to erase the eeprom.
```

Compiled Firmware:

A - Low Side of Memory Bootloader: (LSM)

The LSM bootloader is the simplest to write. There is no need to delete the ROM where the jumps to the bootloader or the user code are already implemented. These different jumps to user code and for interrupts are already well defined and written to the ROM forever.

The firmware is always positioned at the **BOTTOM** of the ROM in every compilation, like a user program. Its code is automatically written in the boot section of the PIC® starting at the address 0. The firmware does not need an external program to write or/and compile the boot. This Bootloader does not need any modification of the **Asm file** executed manually or by the MPLAB®.

The bootloader written in the bottom of the memory (Low Side of Memory) has a single advantage: it can be protected by the PIC® against spurious writing or reading by the Config Fuses. But this system has two drawbacks:

- Redirect the interrupt vectors. (2 goto's for every vector, only one "goto" for High Side Memory bootloader)
- It is necessary to move the start address of the user program to a pre-determined address, in this case the origin is 2048, after the end of the boot sector of the PIC®.

Bootloader Code:

Verify the ASM file:

The Result must be like this:

```
proton#code#start
Org 0
Nop
Nop
GoTo proton#main#start
Org 8
GoTo ISR_High
Org 24
GoTo ISR_Low
proton#main#start
...
```

Important information to know:

In this bootloader all the code to decode the HEX lines is included. To know that the file is valid, the first line of the HEX file is read and decoded. If this is valid, then the ROM of the user code begins to be deleted and the procedure is followed. In almost all cases of subsequent errors the old user code is lost.

Fixed some errors:

Case 1:

When the bootloader starts to burn and if a fatal error occurs, sometimes the program is in a limbo with the memory ROM deleted and where it does not know where to go next. Recovery is difficult in all these cases of mistakes. The incorrect use of the terminal with the USB Port may also cause a malfunction. Most errors occur when the terminal is misconfigured.

In this case the bytes of the control address are deleted to inform the bootloader code that there is no correct written code and thus an error is shown in the terminal. In the case of error the user memory has been deleted.

Case 2:

In this bootloader model (LSM) the user program must be moved from 2048 bytes. If the programmer forgets to place the: [**Declare PROTON_START_ADDRESS = 2048**] at the beginning of his program, a new detection code will send a general error and the bootloader will not record anything. The user memory has been deleted.

Configuration of the LSM Bootloader:

All the configuration of the bootloader has been reorganized and grouped so that the user can choose the parameters very easily.

The configuration is divided into several consecutive parts headed by a label:

CONFIG01, CONFIG02, CONFIG03, CONFIG04, CONFIG09.

CONFIG01:

Device / Declare Xtal / PLL On-OFF / Internal FOSC On-OFF / Clock Out function / Define UART number

You have to choose the block that corresponds to your device. (uncomment one block)

```
' ' The USER must copy this "Declare" in the user main project.
Device = 18F25K22
Declare Xtal = 64

' ' DEFINE THE PLL: On / OFF
$define PLL_ConfigFuses On

' ' Enable the to use the internal Oscillator.
' ' Uncomment the next line to enable the configuration.
'$define _InternalOSC_

' ' Enable the Line to use the CLOCK OUT FUNCTION:
' ' If CLKOUT function is enabled, CLKOUT on RA6 & Port function On RA7
otherwise Port function On RA6 & RA7.
'$Define _CLKOUT_Function_

' ' DEFINE the UART name used for the bootloader. (UART1 OR UART2)
$define UART_USED_FOR_BOOTLOADER UART1
```

Uncomment the line or write your options.

Device = Already defined.

Declare Xtal = Write the desired FOSC value according to the predefined list.

FOSC: 80, 64, 48, 40, 32, 25, 24, 20, 16, 14 (14.32MHz), 12, 10

\$Define PLL_ConfigFuses On or OFF Write the desired option that will automatically configure the Config Fuses.

\$Define _InternalOSC_ Uncomment the line to enable the function. The code for the

internal oscillator is written. Nothing for the user to write.

If this option is enabled then copy the internal oscillator code [**CONFIG17**] in the top of your project.

\$Define _CLKOUT_Function_ Uncomment the line to enable the function. A Clock output could be defined if the PIC® will use the internal oscillator or the EC external Xtal oscillator, otherwise the pin is a digital output.

\$Define UART_USED_FOR_BOOTLOADER Write **UART1** or **UART2**.
Uart used by the Bootloader.

When the PIC® contains a single UART then this line is not necessary and is not written here.

CONFIG02:

Declare the Bootloader Options. Uncomment the line to choose an option. Four options are chosen by default.

`_EnableMessages_, _EnableWriteIDLOCS_, _EnablePrintDeviceID_,
EnableProtectBootBlock`

With the [**EnableProtectBootBlock**] option enabled, the Bootloader is full protected againsts reading and writing.

In some cases, when the PIC® has a lot of RAM and ROM, all the options can not be chosen because the total code does not fit in 2048 bytes of the bootblock. In this case, choose the number of options that allow compiling.

CONFIG03:

RANGE to ERASE THE EEPROM MEMORY.

Define the range of the eeprom memory to erase when the **\$define** `_EnableEraseEeprom_` option is chosen.

If the **PasswordMode0** option has been chosen, the high limit will be modified automatically to not delete the Password and other parameters in the eeprom.

CONFIG04:

Declare the Keys for Passwords if a Password option has been chosen. Write your new values for passwords respecting the hex format.

Symbol `KEYNumber` = **\$1DF4E178**
Symbol `KEYASCII` = **\$65**
Symbol `PW_CONTROL` = **\$35**

CONFIG09:

Config Fuses.

Some parameters of the Config Fuses have already been written and will be chosen according to some previous options. Do not modify the parameters located between

preprocessor lines code. The other lines could be modified by the user with great care, you should know what you are doing. The corresponding PIC® Config Fuses must be copied into the user's program. It will be useful to make a copy of the original Config Fuses.

Compile your new firmware:

At this stage you can compile and obtain the desired firmware.

I recommend you write your hex file in the following way, in case of using UART1.

PIC18F25K22_16MhzPLL_LSM_V50.hex

Or this one in case of using UART2.

PIC18F25K22_16MhzPLL_LSM_U2_V50.hex

B - High Side of Memory Bootloader: (HSM)

The HSM bootloader is the most complicated to write because every time the block zero is deleted from the ROM there are the different jumps to the bootloader and the user program. It is complicated to restore.

The length of the code and the position in the high memory is always dependent on the compiler version, the model, the PIC® family and the options available to the user. Several possibilities exist that make it completely unavoidable to get an error in the compilation when the code is too long. In the opposite direction the bootloader could be placed too low in the memory leaving some unused erase blocks and removing memory for the user's application.

In this new version (5.0), an adjustment system is added allowing modifying the code length data according to each compilation. In this way you can always place the Bootloader code in its correct position at the top of the ROM, without complications.

The firmware is always positioned dynamically at the **TOP** of the ROM in every compilation. Its size and position (*PIC#Loader#Boot = label*) is automatically calculated according to the choice of the number of user options. The firmware does not need an external program to write or/and compile the boot. This Bootloader does not need any modification of the **Asm file** executed manually or by the MPLAB®. The vector "*Goto PIC#Loader#Boot*" is written by the firmware at the first Reset of the PIC® (use the OPCode debug to see it). Then it is necessary to make only a **Reset** after loading the Bootloader firmware by a programmer.

After compile the bootloader.bas file or the user.bas file, the ASM files must be like this.

Bootloader file after compiling:

Code:

```
proton#code#start
  Org 0
  Nop
  Nop
  GoTo proton#main#start
  Org 8
...
```

```
proton#main#start
...
PIC#Loader#Boot
...
```

Bootloader code in the PIC® ROM after a first reset:

Code:

```
proton#code#start
  Org 0
  Goto PIC#Loader#Boot    ' Address 0 of ROM
  GoTo proton#main#start  ' Address 4 of ROM
  Org 8
...
proton#main#start
...
PIC#Loader#Boot
...
```

The user file with "Declare Watchdog = On":

Code:

```
proton#code#start
  Org 0
  clrdt
  clrdt
  GoTo proton#main#start
  Org 8
...
proton#main#start
...
```

Important information to know:

In this bootloader all the code to decode the HEX lines is included. To know that the file is valid, the first line of the HEX file is read and decoded. If this is valid, then the ROM of the user code begins to be deleted and the procedure is followed. In almost all cases of subsequent errors the old user code is lost.

Fixed some error:

When the bootloader starts to burn and if a fatal error occurs, sometimes the program is in a limbo with the memory ROM deleted and where it does not know where to go. Recovery is difficult in all these cases. The use of the terminal with the USB Port is also important in potentially causing a malfunction. Most errors occur when the terminal is misconfigured.

In this case the bytes of the control address are deleted to inform the bootloader code that there is no correct written code and thus must show an error to the terminal. In case of error, the user memory has been deleted.

Configuration of the HSM Bootloader:

All the configuration of the bootloader has been reorganized and grouped so that the user can choose the parameters very easily.

The configuration is divided into several consecutive parts headed by a label:

CONFIG01, CONFIG02, CONFIG03, CONFIG04, CONFIG09.

CONFIG01:

Device / Declare Xtal / PLL On-OFF / Internal FOSC On-OFF / Clock Out function / Define UART number

You have to choose the block that corresponds to your device. (uncomment one block)

```
' ' The USER must copy this "Declare" in the user main project.
Device = 18F25K22
Declare Xtal = 64

' ' DEFINE THE PLL: On / OFF
$define PLL_ConfigFuses On

' ' Enable the to use the internal Oscillator.
' ' Uncomment the next line to enable the configuration.
'$define _InternalOSC_

' ' Enable the Line to use the CLOCK OUT FUNCTION:
' ' If CLKOUT function is enabled, CLKOUT on RA6 & Port function On RA7
otherwise Port function On RA6 & RA7.
'$Define _CLKOUT_Function_

' ' DEFINE the UART name used for the bootloader. (UART1 OR UART2)
$define UART_USED_FOR_BOOTLOADER UART1
```

Uncomment the line or write your options.

Device = Already defined.

Declare Xtal = Write the desired FOSC value according to the predefined list.

FOSC: 80, 64, 48, 40, 32, 25, 24, 20, 16, 14 (14.32MHz), 12, 10

\$Define PLL_ConfigFuses (On or OFF) Write the desired option that will automatically configure the Config Fuses.

\$Define _InternalOSC_ Uncomment the line to enable the function. The code for the internal oscillator is written. Nothing to write by the user.

If this option is enabled then copy the internal oscillator code [CONFIG17] in the top of your project.

\$Define _CLKOUT_Function_ Uncomment the line to enable the function. A Clock output could be defined if the PIC® will use the internal oscillator or

the EC external Xtal oscillator, otherwise the pin is a digital output.

\$Define UART_USED_FOR_BOOTLOADER Write **UART1** or **UART2**.
Uart used by the Bootloader.

When the PIC® contains a single UART then this line is not necessary and is not written here.

CONFIG02:

Declare the Bootloader Options. Uncomment the line to choose an option. Three options are chosen by default.

`_EnableMessages_, _EnableWriteIDLOCS_, _EnablePrintDeviceID_.`

In some cases, depending on the number of options chosen and the size of the RAM and the ROM the position of the hex firmware in the ROM needs to be checked see the chapter [\[Adjust the hex file\]](#).

CONFIG03:

RANGE to ERASE THE EEPROM MEMORY.

Define the range of the eeprom memory to erase when the **\$define** `_EnableEraseEeprom_` option is chosen.

If the **PasswordMode0** option has been chosen, the high limit will be modified automatically to not delete the Password and other parameters in the eeprom.

CONFIG04:

Declare the Keys for Passwords if a Password option has been chosen. Write your new values for passwords respecting the hex format.

```
Symbol KEYNumber = $1DF4E178
Symbol KEYASCII = $65
Symbol PW_CONTROL = $35
```

CONFIG09:

Config Fuses.

Some parameters of the Config Fuses have already been written and will be chosen according to some previous options. Do not modify the lines located between Preprocessor lines code. The other lines could be modified by the user with great care, you should know what you are doing. The corresponding PIC® Config Fuses must be copied into the user's program. It will be usefull to make a copy of the original Config Fuses.

Compile your new firmware:

At this stage you can compile and obtain the desired firmware.

I recommend you write your hex file in the following way, in case of using UART1.

PIC18F25K22_16MhzPLL_HSM_V50.hex

Or this one in case of using UART2.

PIC18F25K22_16MhzPLL_HSM_U2_V50.hex

Adjust the hex file:

Case 1: See a correct firmware:

MEMORY USAGE MAP ('X' = USED, '-' = UNUSED)

```
0000 : XXXXXXXXXXXXXXXX-- -----
79C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7A00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7A40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7A80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7AC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7B00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7B40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7B80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7BC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7C00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7C40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7C80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7CC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7D00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7D40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7D80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7DC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7E00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7E40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7E80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7EC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7F00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7F40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7F80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
```

Last 64 Bytes erase block

```
7FC0 : XXXXXXXXXXXXXXXX - - - - -
```

The last erase block has been written at least one part: correct result

```
0000 : XXXXXXXX-----
0000 : -XX-XX-XXXXX-- - - - - -
```

ALL OTHER MEMORY BLOCKS UNUSED.
PROGRAM MEMORY BYTES USED: 1585
PROGRAM MEMORY BYTES FREE: 31183

Case 2: No byte of the last erase block has been written. The firmware is written too low in the ROM and is not efficient.

```
7EC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7F00 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
7F40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
```

The last erase block -1 has been written at least one part:

```
7F80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXX - - - - -
```

Last 64 Bytes erase block

```
7FC0 : - - - - -
```

The last erase block is not written: bad result, it is not so efficient.

In this case, a value [**BLCodeLenght**] must be modified in the [**CONFIG07**] corresponding to the chosen PIC®.

Original values:

```
$if _device = _18F25K22
' Read the Datasheet or MPLAB .dev files to know the Device ID value.
$define __DEVICE_ID 682
' Read the Datasheet to know the erase block value.
$ifndef _erase
$define _erase 64
$endif
' Read the Datasheet to know the _bankAend value.
$ifdef _bankAend
$undef _bankAend
$endif
$define _bankAend 95
' Declare the Address of the MyDummyByte variable in RAM after compiling.
$define MyDummyByteAddress 79
' Declare the lenght of minimum code.
$define PDSCodeVariation 0
$define BLCodeLenght 1272
' Correction is only valid for one choosen configuration.
' $define BLCodeLenght 1272 - ERASE_FLASH_BLOCKSIZE
$endif
```

Normally the difference is 1 erase block size because of the calculation formula. Then you must change the [**\$define BLCodeLenght**] like this.

```
$if _device = _18F25K22
' Read the Datasheet or MPLAB .dev files to know the Device ID value.
$define __DEVICE_ID 682
' Read the Datasheet to know the erase block value.
$ifndef _erase
$define _erase 64
$endif
' Read the Datasheet to know the _bankAend value.
$ifdef _bankAend
$undef _bankAend
$endif
$define _bankAend 95
' Declare the Address of the MyDummyByte variable in RAM after compiling.
$define MyDummyByteAddress 79
' Declare the lenght of minimum code.
$define PDSCodeVariation 0
' $define BLCodeLenght 1272 <<<< Disable this line
' Correction is only valid for one choosen configuration.
$define BLCodeLenght 1272 - ERASE_FLASH_BLOCKSIZE <<<< Enable this line
$endif
```

Now you must compile and check the **firmware.list** file again. Some part of the last erase block must be written. Please use this technique for one configuration only.

Case 3: In this case the compiler gives an error. There is too much code for the ROM of the PIC®.

The calculation of the bootloader code length has failed. Then you have to artificially add more bytes to the firmware.

```

$if _device = _18F25K22
' Read the Datasheet or MPLAB .dev files to know the Device ID value.
$define __DEVICE_ID 682
' Read the Datasheet to know the erase block value.
$ifndef _erase
$define _erase 64
$endif
' Read the Datasheet to know the _bankAend value.
$ifdef _bankAend
$undef _bankAend
$endif
$define _bankAend 95
' Declare the Address of the MyDummyByte variable in RAM after compiling.
$define MyDummyByteAddress 79
' Declare the lenght of minimum code.
$define PDSCodeVariation 8 <<<< Add some bytes here
$define BLCodeLenght 1272
$endif

```

Add some bytes, always an even number, to [**\$define PDSCodeVariation**] parameter, and compile again. If there is a new compilation error add some bytes again until the last erase block you checked is written in the file.lst.

You should not forget that the erase block could be 64, 128, 256 or 512 bytes.

After getting your firmware, do not forget to put the parameters back to their original value.
[\$define PDSCodeVariation 0]

The CoolTerm Terminal:

The external communication software:

Simple RS232 communicator software is needed to send the .HEX file to the PIC® with software Handshake protocol XON/XOFF capability. A push button connected to the reset pin of the PIC® is needed to start the bootloader.

I am using a good and simple terminal: The CoolTerm **Version 1.4.6** built 322 for WIN7. (It is the best for me)

The new **version 1.4.7** does not allow a user to visualize the HEX files. But it is possible to modify the extension from **.hex** to **.asc** and the terminal program can work.

CoolTerm is a simple serial port terminal application (not a terminal emulation). It is quite fast sending the ASCII bytes of the .HEX file. All terminals do not work with same speed when using the XON/XOFF protocol (?).

Setting the CoolTerm Terminal communication:

Press the OPTIONS icon.

Serial Port:

set the COM Port: COMx

Baudrate: 115200 (for example for FOSC = 64Mhz) (see the Baud Rate table)

Data Bits: **8**

Parity: none

Stop Bits: **2**

CTS, DTR: nothing

Flow Control: XON

Initial Line States when Port open: DTR = Off, RTS = Off.

Terminal:

Standard settings = OK

Terminal Mode: Raw Mode

Special character handling: Set the "Convert Non printable character".

Receive:

Choose: Ignore receive signal errors

Change: Receive Buffer Size: 100000 (for debugging)

Wait for terminate string: Termination string OD, OA

Others: NO change

Transmit:

Transmit: (for password option)

Send String Options: Termination String (Hex) = 0D 0A

Miscellaneous:

Automatically disconnect on close.

Press OK and connect. If a reception error occurred then reconnect.

Use the Connection/Flush Serial Port if it is necessary.

How to download a HEX file:

In the tool bar, press "Connection".

Press "Send Text File"...

Browse the .HEX file.

Press the Reset button of the PIC® board. (2 seconds Time Out) (Option with 5 seconds)

Press the "OPEN" icon. => sending hex file.

How to download a HEX file with Password:

In the tool bar, press "Connection".

Press "Send String"...

In the new window, set ASCII option. Paste your Password in the new Window (including the ":" first character) eg. (:12A4B67F)

Press the Reset button of the PIC® board. (2 seconds Time Out)

Press "Send" in the Send String window (Password) (Now a new 10 seconds Time Out is starting)

In the tool bar, press "Connection".

Press "Send text File"...

In the new window browse the .HEX file.

Press the "OPEN" icon. => sending hex file.

NOTE:

Every firmware could have a unique Password or an incremental Password.

In Password Mode 1 the Password (KEYnumber) is written in the Flash Memory.

In Password Mode 0 the next Password (KEYnumber) to use is written in the Eeprom Memory.

It is a good idea to change these parameters for every PIC design.

Password Mode 1:

The password is unique and defined by the user in STEP04.

```
' Definitions of the KEYS. (Different KEYS For every project)
Symbol KEYNumber = $1DF4E178
```

The unique Password is defined by the **KEYnumber** parameter.

Password Mode 0:

The Password is incremental. Every time the bootloader is used the password will change.

The KEY for encoding the password is defined by the user in STEP04.

```
' Definition of the KEYS. (Different KEYS For every project)
Symbol KEYNumber = $1DF4E178
Symbol KEYASCII = $E9
Symbol PW_CONTROL = $5E
```

The KEY to encode all passwords is defined by the **KEYnumber** parameter.

The **KEYASCII** parameter encodes the Password written in the end of Eeprom. (Mode 0 only)

The PW_CONTROL parameter allows a user to know if the password in the Eeprom has been erased. (Mode 0 only)

A special routine is used to encode 1000 KEYS to send them to the terminal.

CONCLUSION

This new version allows a better configuration of the bootloader only applicable to the families of PICs described in the templates.

In the future manuals I will discuss how to use the test codes, prepare the user code and modify the parameters of a template to adapt it to a new PIC® family.

PDS Bootloader version 5.0. 24 December 2017

Alberto Freixanet

