



16-bit PIC serial bootloader

User's guide

Version 1.0
03-11-2008

ingenia

© 2008, INGENIA-CAT S.L.

16-bit PIC serial bootloader user's guide

Copyright and trademarks

Copyright © 2008 INGENIA-CAT, S.L.

Microchip, MPLAB and PIC are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries

Scope

This document applies to Ingenia 16-bit PIC serial bootloader.

Disclaimers and limitations of liability

Except in cases specifically indicated in other agreements and **INGENIA-CAT**, this product and its documentation are provided "as is", with no warranties or conditions of any type, whether express or implied, including, but not limited to the implied warranties or conditions of merchantability, fitness for a particular purpose or non-infringement.

INGENIA-CAT rejects all liability for errors or omissions in the information or the product or in other documents mentioned in this document.

INGENIA-CAT shall in no event be liable for any incidental, accidental, indirect or consequential damages (including but not limited to those resulting from: (1) dependency of equipment presented, (2) costs or substituting goods, (3) impossibility of use, loss of profit or data, (4) delays or interruptions to business operations (5) and any other theoretical liability that may arise as a consequence of the use or performance of information, irrespective of whether **INGENIA-CAT** has been notified that said damage may occur.

Some countries do not allow the limitation or exclusion of liability for accidental or consequential damages, meaning that the limits or exclusions stated above may not be valid in some cases.

This document may contain technical or other types of inaccuracies. This information changes periodically.

Contents

Overview.....	5
What is a bootloader?	5
Ingenia 16-bit PIC serial bootloader.....	6
Introduction.....	6
Requisites	6
Ingenia 16-bit PIC serial bootloader firmware.....	6
How firmware works?	7
How the process works?.....	8
Communication protocol	10
Synchronization	10
Commands Format	10
Version command.....	11
Read command	12
Write command.....	13
Erase command.....	14
Checksum command	15
User Program command.....	16
Unknown command	17
Firmware modifications.....	18
How to use it with another PIC	18
Memory organization	19
Graphical User Interface.....	20
Introduction.....	20
RS232 interface setup	20
Loading and writing.....	21
Checksum operation.....	22
The XML PIC list file	22

Icons

Icons that the reader may encounter in this manual are shown below, together with their meanings.



Additional information

Provides the user with tips, tricks and other useful data.



Warning

Provides the user with important information. Ignoring this warning may cause the device not to work properly.



Critical warning

Provides the user with critical information. Ignoring this critical warning may cause damage to the device.

This chapter introduces the bootloader strategy for MCUs and DSCs.

What is a bootloader?

A **bootloader** is a firmware (software embedded in a hardware device) located into the non-volatile memory of a microcontroller unit (MCU) that allows in-circuit reprogramming of the device using its standard communication ports.

Usually, the process to program a MCU or DSC implies the need of an expensive hardware device. Such devices, also called *programmers*, use the special purpose pins of the MCUs/DSCs to access to the internal memory. Modifying the voltage applied to these pins, a read or write cycle of the memory could be performed.

Moreover, the programmers also should incorporate a communications interface in order to allow the communications with the sender device (normally a Personal Computer (PC)). Together with the hardware programmer, comes software that helps the final user to send his own firmware through the serial port of the PC to the MCU/DSC.

In the other hand, a **bootloader** is just a piece of code that works with the communication ports of an MCU/DSC and takes advantage of the capacity to write into his own non-volatile memory.

This means that hardware programmer must be used at least once to load it into the MCU/DSC. Then, the user can reprogram the MCU/DSC as many times as required without the need of the hardware programmer.

One of the main advantages of using a **bootloader** in a hardware device that contains a firmware, is that adds to it the capacity to be easily upgradeable (the user just needs a PC to update the firmware version). This procedure will save the cost of disassemble and send the device back to the factory.

Description of the Ingenia bootloader (firmware and GUI).

Introduction

Ingenia has developed a serial bootloader package specially focused on the 16-bit PIC families of Microchip (dsPIC30F, dsPIC33F, PIC24F and PIC24H).

Mainly the bootloader package is divided into two parts:

- A firmware code and,
- a Windows based Graphical User Interface (**GUI**).

Requisites

The system requirements to use Ingenia bootloader package are shown in the Figure 1. This system is composed by the following elements:

- A personal computer (Intel Pentium II 366 MHz or higher, 64Mb of RAM and Windows 2000/XP/Vista) with Ingenia 16-bit PIC serial bootloader Graphical User Interface installed and with a serial port.
- A **dsPIC30F/dsPIC33F/PIC24F/PIC24H board** with the bootloader firmware programmed into it and with a serial transceiver.
- The appropriate serial communication cable

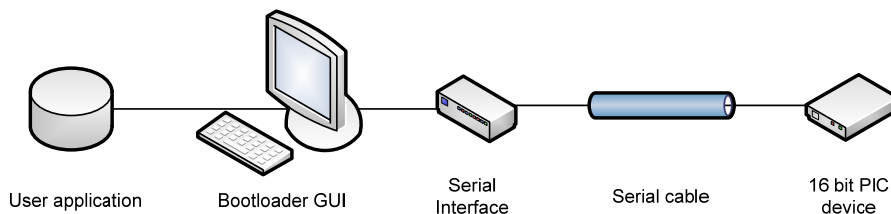


Figure 1: Typical system

Ingenia 16-bit PIC serial bootloader firmware

As explained above, the firmware must be loaded into the MCU/DSC using a hardware programmer. The main features of Ingenia's bootloader firmware are:

- **Auto-Baud rate detection** – The bootloader has the ability to adjust its own baud rate to the one used by the sender by mean of a synchronization protocol.
- **Perform read and write program (flash) memory operations** – The bootloader is able to access to the whole non-volatile memory dedicated to program code.
- **Perform read and write EEPROM memory operations** - The bootloader is able to access to the whole non-volatile memory dedicated to data (only in dsPIC30F devices).
- **Perform read and write configuration registers operations** - The bootloader is able to access to the configuration registers zone.

- **Checksum operation** – The bootloader can perform a checksum operation in order to verify the correct transmission/writing process of the program memory.

How firmware works?

Conceptually the firmware can be seen as a flow of states (see Figure 2). Below there is a short description about each of them and the conditions necessary to move from one to another.

- **Reset** – When a power-up or a reset occurs the MCU/DSC enters in this state and jumps directly to the *Baud Rate Detection* state.
- **Baud rate detection** – In this state, synchronization with the sender is performed in order to compute the used baud rate. After a time if no synchronization is established a timeout occurs and the *User program* is executed. If the baud rate is detected correctly the execution continues in the *Wait commands* state.
- **Wait commands** – During this state the MCU listens continuously the UART interface. If a known command is received, the program will jump to the corresponding state (version, read, erase or write). Otherwise it will stays in this state indefinitely.
- **Version** – The version of the firmware is sent through the UART and the execution returns to *Wait commands* state.
- **Read** – A read memory operation is performed, the result is sent through the UART and the execution returns to *Wait commands* state.
- **Write** – A write memory operation is performed, the result is sent through the UART and the execution returns to *Wait commands* state.
- **Erase** – An erase memory operation is performed, the result is sent through the UART and the execution returns to *Wait commands* state.
- **Checksum** – The result of a memory checksum operation is sent through the UART and the execution returns to *Wait commands* state.

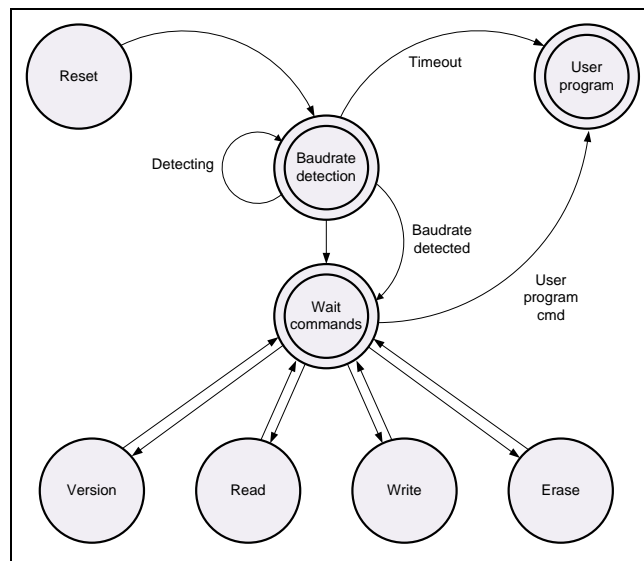


Figure 2: Flow state diagram of the firmware

How the process works?

The following figure shows the workflow of the bootloader process and the interaction between the PIC and the GUI.

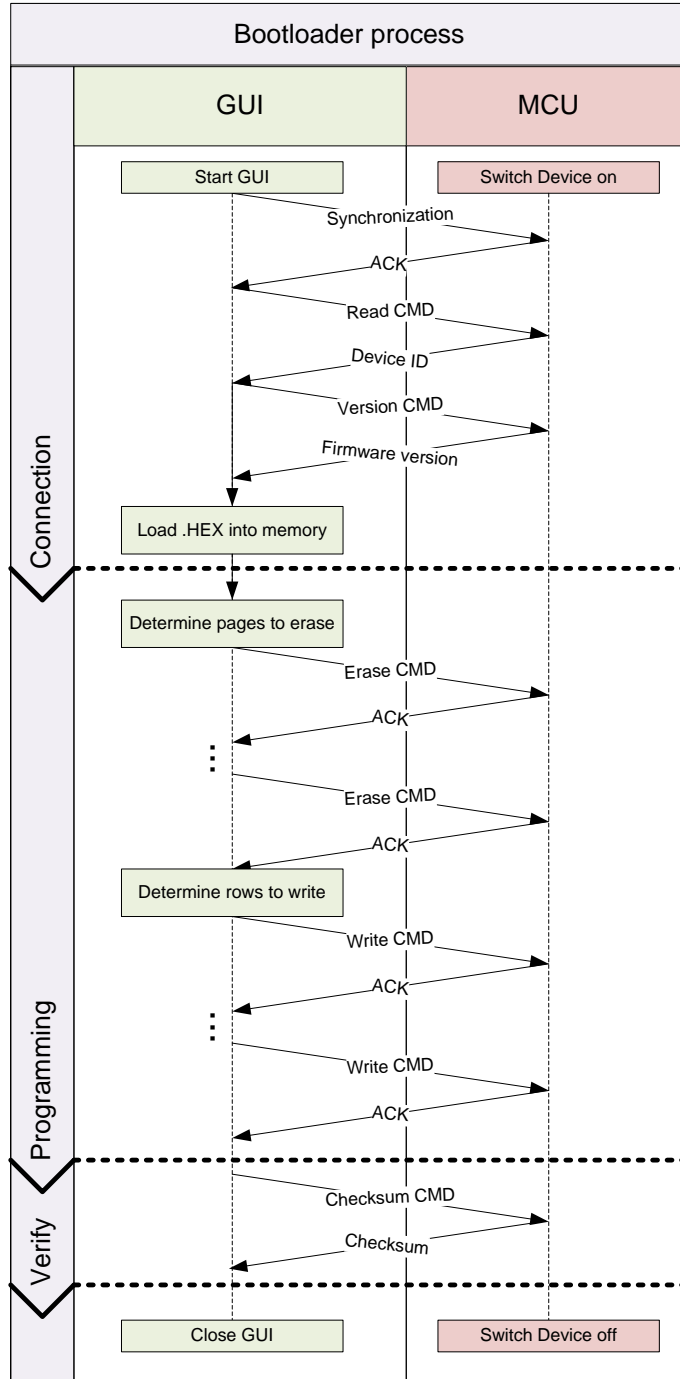


Figure 3: Process workflow

1. Once the MCU/DSC is released from reset, it will wait for synchronization (reception of 0x55 character) during a while.
2. The GUI must send this character within this time in order to establish the communication with the bootloader.

3. The MCU/DSC will answer with an ACK command (0x55).
4. The GUI will send a Read command message in order to read the device ID of the MCU/DSC.
5. Once the device is identified the GUI is able to load a user compiled .hex file.
6. With the information of the device and the user compiled file, the GUI is able to determine which memory pages must be erased and rewritten and therefore it will send the corresponding command to the MCU/DSC.
7. Finally, the GUI computes the theoretical checksum and compares with the one computed in the device.

The row and page size of each memory region (FLASH, EEPROM, CFG) depends on the kind of device family.

The following table shows the differences between families.

MCU/DSC family	Rows per page	Instructions per row		
		Config Register	EEPROM	Program Memory
dsPIC30F	1	1	16	32
dsPIC33F	8	1	-	64
PIC24H	8	1	-	64
PIC24F	8	64(*)	-	64

(*) Configuration registers are written in the last position of program memory.

Description of the communication protocol used by Ingenia 16bit PIC serial bootloader.

Synchronization

In order to achieve a correct synchronization, the host should send continuously the ASCII character 'U' (0x55) to the MCU/DSC.

The representation of this character in binary is 01010101b giving the maximum frequency of transitions in a fixed baud rate (See Figure 1).

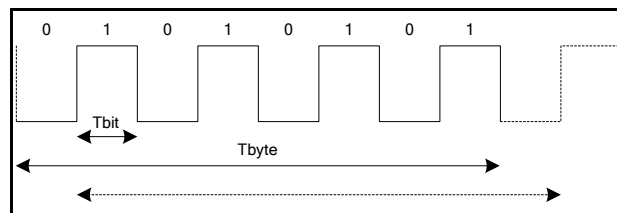


Figure 1: Representation of the character 0x55

In dsPIC30F family, when the bootloader detects the first rising edge, starts a timer and looks for the next four rising edges. Once is detected the last edge one, the timer is stopped and the baud rate computed by means of a simple division.

The dsPIC33F, PIC24F and PIC24H families incorporate a new hardware feature that automatically computes the baud-rate of the communication by using a Sync field (character 55h). This feature simplifies the code for the auto-baud and avoids the use of a Timer and the Input capture module.

Commands Format

The communication protocol used in the Ingenia 16bit PIC serial bootloader is based on RS232.

The frame of all the commands used by the bootloader starts with an *Identification Byte*. The answer frame is always ended with *acknowledge* (ACK = 0x55) or *non-acknowledge* (NACK = 0xFF) but the Reset command.

The following table shows the list of all available commands, and its corresponding identifier byte:

Command	Action	Value
VERSION	Check the firmware version	0x03
READ	Read a word from memory	0x01
WRITE	Write a row into memory	0x02
ERASE	Erase a page of memory	0x04
CHECKSUM	Performs a memory checksum operation	0x05
USR_PROGRAM	Executes user program	0x0F

Version command

Check the major and minor version of the firmware.

Request from Host

8 bits
VERSION (0x03)

Answer from Bootloader

8 bits	8 bits	8 bits
Major version	Minor version	ACK (0x55)

Read command

This command is used to read the content of a position of the memory, which could be FLASH, EEPROM or Configure registers addressed by a 24bits word.

The answer is also a 24bits data word.

Request from Host

8 bits	8 bits	8 bits	8 bits
READ (0x01)	Address 23:16	Address 15:8	Address 7:0

Answer from bootloader

8 bits	8 bits	8 bits	8 bits
Data 23:16	Data 15:8	Data 7:0	ACK (0x55)

Write command

Carries out a write memory operation, which could be FLASH, EEPROM or Configure registers.

The writing operation is done in row mode access, thus you should specify the initial address, the length of the row and the whole row content.



The row content must be previously erased.

The *Num. of Bytes* in this version is 16-bits long.

The frame ends with a CRC that is computed as the 256 module of all the data value addition.



The host must ensure that the initial memory address matches with the beginning of the memory row, otherwise the command will fail.

Request from Host

8 bits	24 bits	8 bits	8 bits	8 bits	...	8 bits	8 bits
WRITE (0x02)	Address (23:0)	Num. bytes (N) 15:8	Num. bytes (N) 7:0	Data 0		Data N-1	CRC

Answer from bootloader

8 bits		8 bits
ACK (0x55)	or	NACK (0xFF)

Erase command

This command is used to erase a whole memory page. The length of the page depends on the memory region and of the kind of device family.



The host must ensure that the initial memory address matches with the beginning of the memory page, otherwise the command will fail.

Request from Host

8 bits	8 bits	8 bits	8 bits
ERASE (0x04)	Address 23:16	Address 15:8	Address 7:0

Answer from bootloader

8 bits		8 bits
ACK (0x55)	or	NACK (0xFF)

Checksum command

This command is used to verify the content of the whole memory.

The answer is a 24bits data word with the computed CRC.

Request from Host

8 bits	8 bits	8 bits	8 bits
CHECKSUM (0x05)	End Address 23:16	End Address 15:8	End Address 7:0

Answer from bootloader

8 bits	8 bits	8 bits	8 bits
Data 23:16	Data 15:8	Data 7:0	ACK (0x55)

User Program command

Force the user program execution, which should be located at address 0x100.

Request from Host

8 bits

USR_PROGRAM
(0x0F)

Answer from Bootloader

None.

Unknown command

When an unknown command is received the bootloader sends a non-acknowledge.

Answer from bootloader

8 bits
NACK (0xFF)

This chapter explains how to change the bootloader location and how to use another device.

How to use it with another PIC

The evaluation bootloader package comes with both, an MPLAB project and a compiled version of the code valid for the dsPIC30F6015.

However, if you want to use the firmware with another PIC device you will need to change the project and recompile it.

These are the steps to follow:

1. Start MPLAB and open the 16bitPICserialboot.mcp project.
2. In the option Select Device of the menu Configure, select the device of your system (see Figure 5).

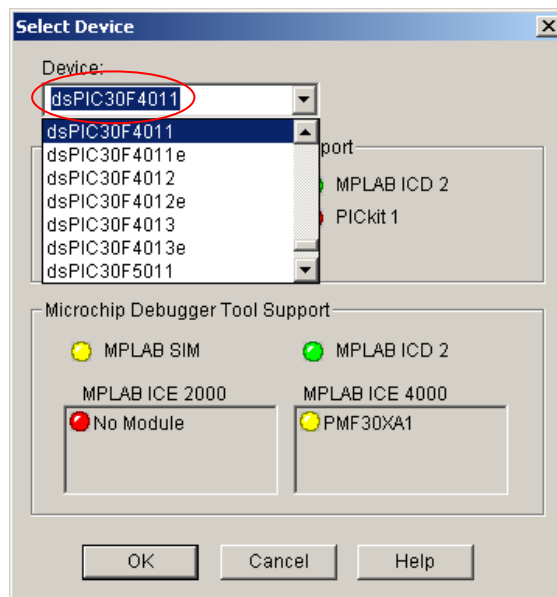


Figure 2: Selection of the device

3. In the project window remove the default linker script (p30f6015.gld) and add the corresponding to your device (See Figure 3).

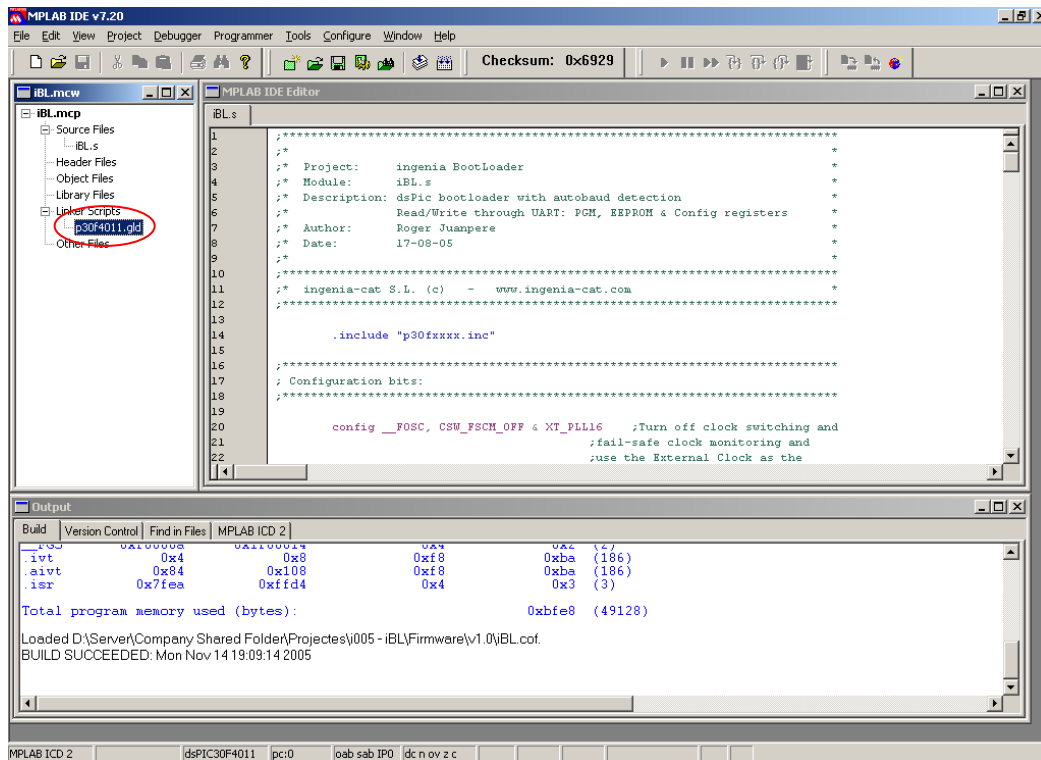


Figure 3: Modifying the linker script

4. Modify configuration registers definition in file *16bitPICbootloader.s* if needed
5. After that you should be able to recompile the project normally.

Memory organization

The memory organization of dsPIC30F differs from the dsPIC33F, PIC24F & PIC24H. The first User Program Memory for dsPIC30F is 0x100. In the other families the position has moved to 0x200.

The position of the bootloader is specified in the linker script file.

The memory region called iBLreset points to the desired starting position of the bootloader. If you use the bootloader in a different device you must add this section to your linker script.

Moreover, you should replace the Reset instruction and add iBLreset and const regions of Program memory. You could take the provided p30f6015.gld as example.

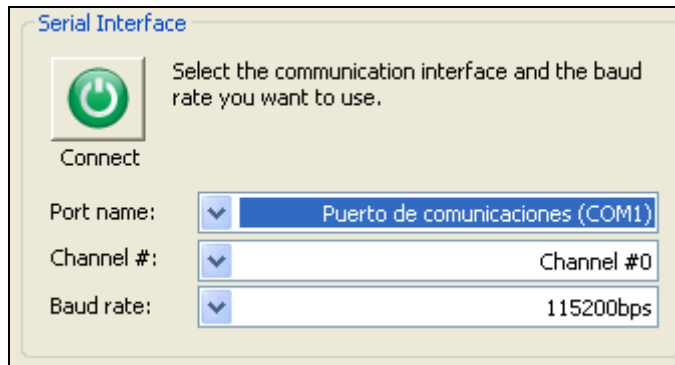


Figure 5: RS232 interface setup

Once you have configured the communication interface, you can start the bootloader operation by clicking on the *Connect* button.

A progress bar will start. At this time, the GUI will start sending the ASCII character 'U' (0x55) requesting to the listening device to enter in *bootload mode*. Reboot your PIC hardware (with bootloader firmware programmed into it) to enter in this mode.

Loading and writing

Once you are connected to a device, you can load as many programs as you want into its memory.

To load a file, click on the *Load HEX* button and browse for it.

Supported file formats are **Intel 16-bit and 32-bit hexadecimal object file format**.



Intel's Hex-record format allows program or data files to be encoded in a printable (ASCII) format. This allows viewing of the object file with standard tools and easy file transfer from one computer to another, or between a host and target.

16bit PIC serial bootloader GUI will detect the written memory regions in the file and will enable the appropriate write check boxes (see Figure 6).



Figure 6: Memory regions

Writeable memory zones of a PIC can be divided into:

- program flash
- write data EEPROM
- and configure registers

Ingenia 16bit PIC serial bootloader shows you the three zones and its associated range addresses.



Ingenia 16bit PIC serial bootloader may detect possible overwrite conflicts when you load an HEX file.

Once you have loaded the file you can start the write process by clicking on the *Write* button.

After programming the target device, you can run it by manually restarting your platform (hardware disconnection/reconnection).

Checksum operation

After a complete programming, Ingenia 16bit PIC serial bootloader will perform a checksum operation in order to verify the correct transmission/writing process of the program memory.

The checksum operation will be performed in the MCU/DSC side and in the GUI side. The results will be showed and compared as seen in Figure 7 and Figure 8.

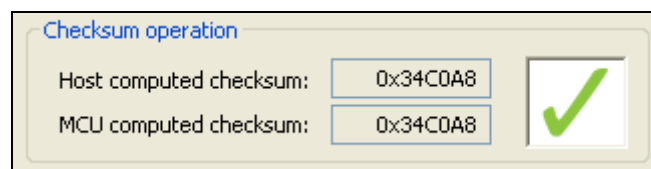


Figure 7: Successful checksum operation

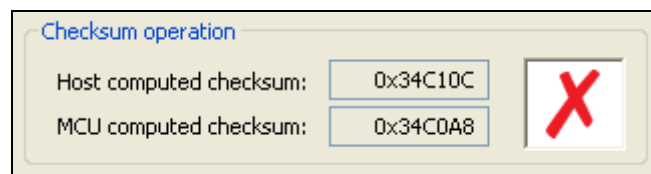


Figure 8: Wrong checksum operation

The XML PIC list file

Ingenia 16bit PIC serial bootloader can work with all 16bit MCU/DSC families from Microchip (dsPIC30F/dsPIC33F/PIC24F/PIC24H). The detection process of processor uses **16biPICList.xml** file to identify the controller and its features. This file is located in the installation folder and consists on a list of supported devices.

If the MCU/DSC that uses your platform doesn't appear in the list, you can add it by following the XML syntax used in the file (see Adding a new device). A DTD enclosed with the XML file will help you to check the XML syntax.

For further information on writing XML files refer to <http://www.w3.org/XML/>.



A DTD ("Document Type Definition") is a set of declarations that conform to a particular markup syntax and that describe a class, or "type", of SGML or XML documents, in terms of constraints on the structure of those documents.

Adding a new device

Each MCU/DSC is named as a device in the XML PIC list file. A device is a description of a MCU/DSC. It is characterized by an identifier, a name and a number of instruction rows per page.

The identifier (*id* tag) is the Microchip device ID (DEVID).

The name (*name* tag) is the Microchip device name.

The number of rows per page (*rowsxpage* tag) is related to erase and program process of PIC.

Within tags `<device>`/`</device>` you have to define three memory zones:

- Code or programming,
- data,
- and configuration.

Code zone is represented with `<memcode>` tag.

Data zone is represented with `<memdata>` tag.

And configuration zone is represented with `<memconfig>` tag.

In each zone you need to define the start address, the end address and the number of instructions per row as attributes of the tag.

Also within `memcode` zone you have to specify bootloader region by using `<bootloader>` tag.

The bootloader region defines the zone where bootloader is located. This zone will be protected against overwrites, so be sure to define its start and end address properly (you will not be able to write code in this region).

The following example shows a complete definition of a device.

```
<device id="0x0101" name="dsPIC30F4011" rowsxpage="1">
  <memcode startaddress="0x000000" endaddress="0x007FFE" instructionxrow="32">
    <bootloader startaddress="0x007E00" endaddress="0x007FFE" />
  </memcode>
  <memdata startaddress="0x7FFC00" endaddress="0x7FFFFE" instructionxrow="16" />
  <memconfig startaddress="0xF80000" endaddress="0xF8000B" instructionxrow="1">
  </memconfig>
</device>
```