



16bit PIC CAN bootloader

Product manual

Version 1.2

14-01-2010

ingenia

© 2010, INGENIA-CAT S.L.

16bitPIC CAN bootloader product manual

Copyright and trademarks

Copyright © 2010 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 16bitPIC CAN bootloader.

16bitPIC CAN bootloader disclaimer

This software is provided "as is". Ingenia expressly disclaims any warranty of any kind, whether express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

In no event shall Ingenia be liable for any incidental, special, indirect or consequential damages, lost profits or lost data, harm to your equipment, cost of procurement of substitute goods, technology or services, any claims by third parties (including but not limited to any defense thereof), any claims for indemnity or contribution, or other similar costs.

Contents

Overview.....	5
What is a bootloader?.....	5
Ingenia 16bitPIC CAN bootloader.....	6
Introduction.....	6
Requisites.....	6
Bootloader firmware.....	7
Introduction.....	7
How the bootloader firmware works?.....	7
How the process works?.....	9
Communication protocol.....	11
Format.....	11
Commands.....	12
Enter in bootloader mode command.....	12
Read a word command.....	13
Erase a page command.....	14
Write a data row command.....	15
Compute CRC command.....	17
Run user code command.....	18
Bootloader firmware modifications.....	19
How to use the bootloader firmware with different PIC devices.....	19
Other changes.....	21
Bootloader software interface.....	22
Introduction.....	22
Init file.....	23
Graphical User Interface (GUI) operation.....	25
CAN interface setup.....	25
Loading and writing.....	26
Checksum operation.....	27
Customizing GUI layout.....	28
Command line Interface (CLI) operation.....	29
The XML PIC list file.....	31
Adding a new device.....	31
Bootloader software tools.....	33
Encryption tool.....	33

Icons

Following is shown the meaning of the icons that reader can find on this document.



Information

Provides the user with tips and other useful data.



Warning

Provides the user with important information. Omitting this information may cause the device do not work properly.



Critical warning

Provides the user with critical information. Omitting this information may destroy the device.

This chapter introduces the bootloader strategy for MCU/DSC devices on CAN bus systems.

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.

Getting started with Ingenia 16bitPIC CAN bootloader.

Introduction

Ingenia has developed a CAN bootloader package specially focused on the 16bitPIC families of Microchip (dsPIC30F, dsPIC33F and PIC24H).

Mainly the bootloader package is divided into two parts:

- A firmware code (one for each device manufacturer) and,
- A PC software interface for Windows OS that acts as the master in the process of bootloading (both graphical user interface and command line versions are available).

The package also includes an encryption tool to encrypt HEX files.

Requisites

The system requirements to use the 16bitPIC CAN bootloader package are shown in the Figure 1.

The 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 16bitPIC CAN bootloader software installed (GUI or command line version).
- A CAN interface (Kvaser, IXXAT, PeakSystem, Vector or ESD are supported).
- A dsPIC30F/dsPIC33F/PIC24H board with the bootloader firmware programmed into it and with a CAN transceiver.
- The appropriate CAN communication cable.

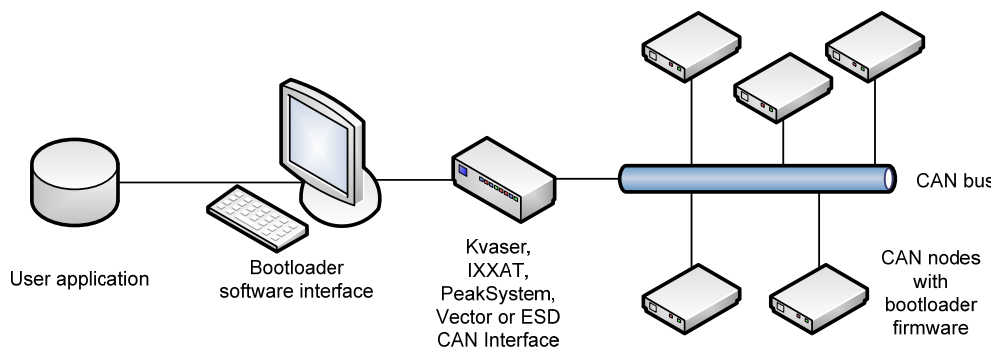


Figure 1: Typical system

Description of 16bitPIC CAN bootloader firmware.

Introduction

As explained in previous chapters, the bootloader firmware must be loaded into the MCU/DSC using a hardware programmer.

The main features of 16bitPIC CAN bootloader firmware are:

- **Possibility of read and write program (Flash) memory** – The bootloader is able to access to the whole non-volatile memory dedicated to program code.
- **Possibility of read and write EEPROM memory** - The bootloader is able to access to the whole non-volatile memory dedicated to data (only in dsPIC30F devices).
- **Possibility of read and write configuration registers** - 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.
- **Ready to use MPLAB project** – The Developer version of CAN bootloader is delivered with a ready to use MPLAB project.

How the bootloader firmware works?

Conceptually the bootloader firmware can be seen as a flow of states (see Figure 2). Below there is a description 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 wait bootloader command.
- **Wait bootloader command** – In this state, the MCU/DSC is listen during 400ms the CAN interface. If within this time a message with a specific ID is received, then the MCU/DSC goes to wait command state. Otherwise, it will execute the user program.
- **Wait Commands** – During this state the MCU/DSC listens continuously the CAN interface. If a known command is received, the program will jump to the corresponding state (read, erase or write). Otherwise it will stay in this state indefinitely.
- **Read** – A read memory operation is realized, the result is sent through the CAN and the execution returns to Wait Commands state.
- **Write** – A write memory operation is realized, the result is sent through the CAN and the execution returns to Wait Commands state.

- **Erase** – An erase memory operation is realized, the result is sent through the CAN and the execution returns to Wait Commands state.
- **Checksum** – In this state, the MCU/DSC computes a CRC of all written data in order to verify the correct transmission/writing process.

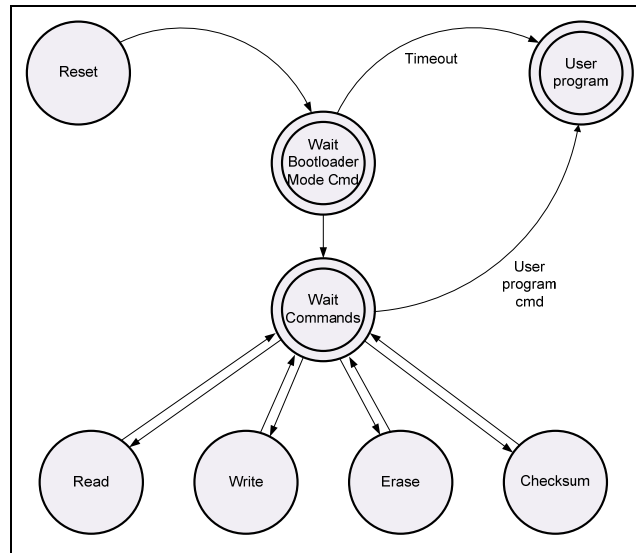


Figure 2: Flow state diagram of the firmware

How the process works?

Figure 2 shows the workflow of the bootloader process and the interaction between the bootloader firmware and the bootloader software.

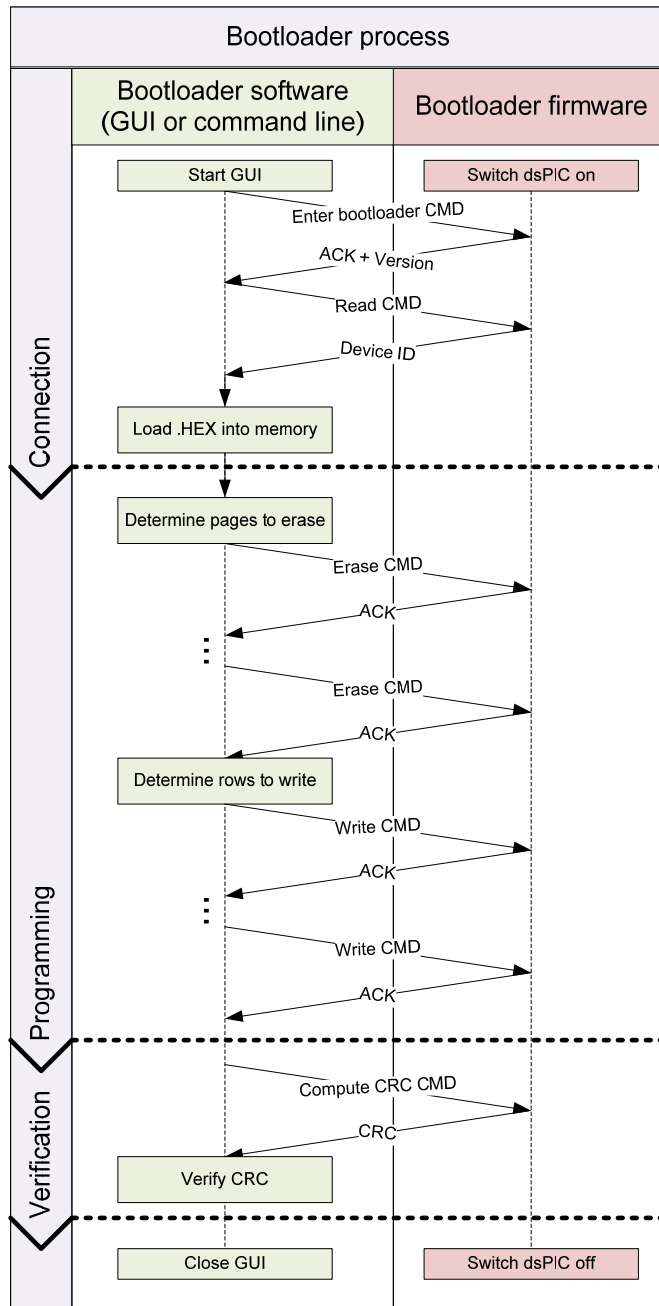


Figure 3: Process workflow

1. Once the MCU/DSC is released from reset, it will wait for the reception of a CAN message during 400ms. This message is called *Enter in bootloader mode command*

2. The master software (GUI or equivalent) must send this message within this time in order to establish the communication with the bootloader.
3. The MCU/DSC will answer with the same message ID and with the version of the firmware and node ID as a data (see message Format in chapter 4).
4. The master software will send a Read command message in order to read the device ID of the MCU/DSC.
5. Once the MCU/DSC is identified, the master software will be able to load a user compiled INTEL HEX (.hex) file.
6. With the information of the device and the user compiled file, the master software 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 master software will check the CRC with the computed by the MCU/DSC in order to verify the correct writing.

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

The following table shows the differences between families.

MCU family	Instruction size	Page/Sector size	Rows per page/sector	Instructions per row		
				Config Register	EEPROM	Program Memory
dsPIC30F	24bits	32 instr.	1	1	16	32
dsPIC33F	24bits	512 instr.	8	1	-	64
PIC24H	24bits	512 instr.	8	1	-	64

The MCU/DSC devices write the content in row mode¹.

¹ In Microchip devices the content of the non-volatile memory cannot be written directly from RAM. First, the target content must be moved from RAM to holding latches and when the programming instruction is executed the content is transferred to NVM. This process could be done at instruction or row level. Row mode is used by 16bit PIC CAN bootloader because it reduces the needed programming time.

Description of the communication protocol used by 16bitPIC CAN bootloader.

Format

The communication protocol used by the 16bitPIC CAN bootloader is based on CAN 2.0A format. The messages have the following fields:

Identifier field	Control field	Data field
11 bits	1 byte	8 bytes max

- **Identifier field (11bits):** It is used to identify the requested command.
- **Control field (1byte):** It is used to assign the length of the transmitted data in the command.
- **Data field (up to 8bytes):** It contains the transmitted data.

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

Identifier field	Action	Value
CAN_BOOTLOADER_MODE	Forces a node to enter in bootloader mode	0x308
CAN_READ_WORD	Read a word from memory	0x309
CAN_ERASE_PAGE	Erase a page of memory	0x30A
CAN_WRITE_ROW	Write a row into memory	0x30B
CAN_COMPUTE_CRC	Computes the CRC of program memory	0x30C
CAN_RUN_USR_CODE	Run user code	0x30D

Commands

Enter in bootloader mode command

This command is used to force a CAN node to enter in bootloader mode. It must be send just after a MCU/DSC boot in order to avoid the user code to be executed.

The target node is selectable through a data parameter.

The MCU/DSC will answer with the bootloader firmware version and Node ID.

Request from master

Identifier	Length	data[0]
CAN_BOOTLOADER_MODE	1 byte	Node ID

Answer from bootloader firmware

Identifier	Length	data[0]	data[1]	data[2]
CAN_BOOTLOADER_MODE	3 bytes	Major Version	Minor Version	Node ID

Example

	Identifier	Length	Data
MASTER	CAN_BOOTLOADER_MODE	01	FF
FW BOOTLOADER	CAN_BOOTLOADER_MODE	03	01 01 00

Read a word command

This command is used to read a word of memory (24bits).

Request from master

Identifier	Length	data[0]	data[1]	data[2]
CAN_READ_WORD	3 bytes	Address 7:0	Address 15:8	Address 23:16

Answer from bootloader firmware

Identifier	Length	data[0]	data[1]	data[2]
CAN_READ_WORD	3 bytes	Data 7:0	Data 15:8	Data 23:16

Example

The following example shows how to read the memory position that contains the device ID (0xFF0000).

The bootloader firmware will answer with an Identifier, i.e. dsPIC30F4011 (0x000101).

	Identifier	Length	Data
MASTER	CAN_READ_WORD	03	00 00 FF
FW BOOTLOADER	CAN_READ_WORD	03	01 01 00

Erase a page command

This command is used to erase a whole memory page. The length of the page depends on the device. Master software must ensure that the initial memory address matches with the beginning of memory page, otherwise the command will fail.

Request from master

Identifier	Length	data[0]	data[1]	data[2]
CAN_ERASE_PAGE	3 bytes	Start Address 7:0	Start Address 15:8	Start Address 23:16

Answer from bootloader firmware

Identifier	Length	data[0]	Description
CAN_ERASE_PAGE	1 byte	0x00	Command OK
		0x01	Command Not OK

Example

The following example shows how to erase the first memory page of a device.

	Identifier	Length	Data
MASTER	CAN_ERASE_PAGE	03	00 00 00
FW BOOTLOADER	CAN_ERASE_PAGE	01	00

Write a data row command

This command is used to write a row into the memory. The length of the row (expressed in bytes) depends on the device. Master software must ensure that the initial memory address matches with the beginning of the memory row, otherwise the command will fail.

Request from master

Identifier	Length	data[0]	data[1]	data[2]	data[3]	data[4]
CAN_WRITE_ROW	5 bytes	Start Address 7:0	Start Address 15:8	Start Address 23:16	Length 7:0	Length 15:8

Answer from bootloader firmware

Identifier	Length	data[0]	Description
CAN_WRITE_ROW	1 byte	0x00	Command OK
		0x01	Command Not OK

Request from master

Identifier	Length	data[0]	...	data[7]
CAN_WRITE_ROW	Up to 8	x		x

Answer from bootloader firmware

Identifier	Length	data[0]	Description
CAN_WRITE_ROW	1 byte	0x00	Command OK
		0x01	Command Not OK
		0x02	Wait for new command

Example

The following example writes the second row of memory in a dsPIC30F (32 instructions per row x 4bytes per instruction = 128bytes = 0x80).

	Identifier	Length	Data
MASTER	CAN_WRITE_ROW	05	80 00 00 80 00
FW BOOTLOADER	CAN_WRITE_ROW	01	00
MASTER	CAN_WRITE_ROW	08	01 02 03 04 05 06 07 08
FW BOOTLOADER	CAN_WRITE_ROW	01	02
MASTER	CAN_WRITE_ROW	08	09 0A 0B 0C 0D 0E 0F 10
FW BOOTLOADER	CAN_WRITE_ROW	01	02

MASTER	CAN_WRITE_ROW	08	11 12 13 14 15 16 17 18
FW BOOTLOADER	CAN_WRITE_ROW	01	02
...	
MASTER	CAN_WRITE_ROW	08	70 7A 7B 7C 7D 7E 7F 80
FW BOOTLOADER	CAN_WRITE_ROW	01	00

Compute CRC command

This command is used to compute the CRC of program memory. This computation is implemented as an 'exclusive or' of the program memory content.

The initial address of the operation is always zero but the end address is configurable in order to allow the usage in all the MCU/DSC devices. The master software must ensure that end address match the address before of bootloader region.

Request from master

Identifier	Length	data[0]	data[1]	data[2]
CAN_COMPUTE_CRC	3 bytes	End address 7:0	End address 15:8	End address 23:16

Answer from bootloader firmware

Identifier	Length	data[0]	data[1]	data[2]
CAN_COMPUTE_CRC	3 bytes	CRC 7:0	CRC 15:8	CRC 23:16

Example

The following example shows how to compute the CRC of the program memory.

	Identifier	Length	Data
MASTER	CAN_COMPUTE_CRC	03	FE 7B 01
FW BOOTLOADER	CAN_COMPUTE_CRC	03	A8 C0 34

Run user code command

Force the user program execution, which should be located at the address defined in bootloader linker script (the linker script uses the `__CODE_BASE` definition to assign the user code starting address and normally points to 0x100 in dsPIC30F devices).

Request from master

Identifier	Length
CAN_RUN_USR_CODE	0

Answer from bootloader firmware

None.

Example

The following example shows how to run a user code once in bootloader mode.

	Identifier	Length	Data
MASTER	CAN_RUN_USR_CODE	00	N/A
FW BOOTLOADER	N/A		

This chapter explains how to configure the bootloader firmware for different PIC devices.

How to use the bootloader firmware with different PIC devices

The **developer package** of 16bitPIC CAN bootloader comes with both, an MPLAB project and a compiled version of the code valid for the selected device.

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

These are the steps to follow:

1. Start MPLAB and open the CANbootloader.mcp project.
2. In the option Select Device of the menu Configure, select the device of your system (see Figure 4).

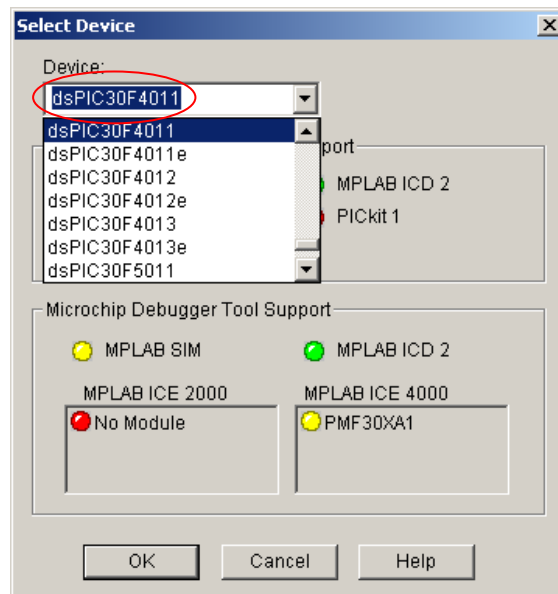


Figure 4: Selection of the device

3. In the project window remove the default linker script (e.g. p30f5011.gld).
4. Copy the linker script of your target device from Microchip folder into the project folder "/gld".
5. Add this linker script to the project (See Figure 5).

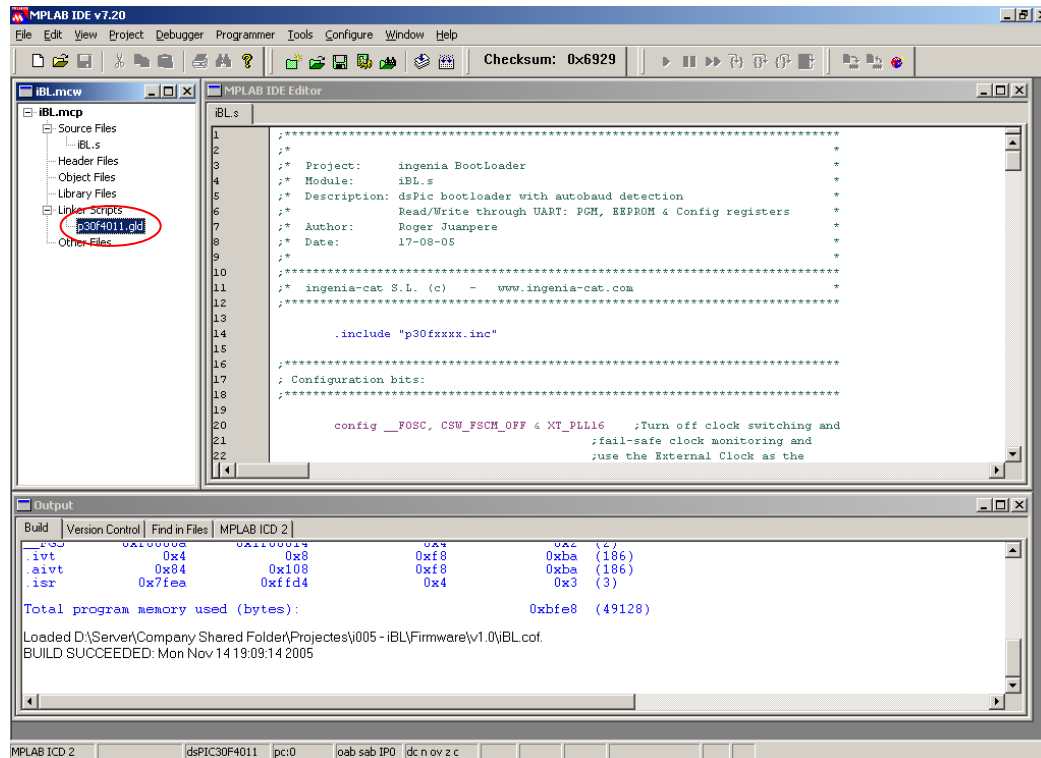


Figure 5: Modifying the linker script

- Open the linker script and add a memory region called `iBLreset` in the position you want to place the bootloader code (normally at the end). The length of the bootloader zone must be `0x400`. You should also decrease the length of the user program memory.

Example:

```
program (xr)          :ORIGIN = 0x200,          LENGTH = (0x2AA00 - 0x100 - 0x200)
iBLreset              :ORIGIN = 0x2A500,        LENGTH = 0x400
```

- If the target user code will not be placed at the start of the program memory (i.e. `0x100` for `dsPIC30F` devices), the `__CODE_BASE` definition of the linker script must be also modified.

Example:

```
__CODE_BASE = 0x100; /* Handles, User Code, Library Code */
```

- Modify the reset instruction as follows:

```
.reset :
{
    SHORT(ABSOLUTE(__reset_1));
    SHORT(0x04);
    SHORT((ABSOLUTE(__reset_1) >> 16) & 0x7F);
    SHORT(0);
} >reset
```

- Add `.iBLreset` and `.iBLdata` sections just after `.text` section.

Example:

```
.iBLreset :
{
```

```

        *(iBLreset);
        *(iBL);
    } >iBLreset

.iBLdata :
{
    *(iBLdata);
} >iBLreset

```

10. Modify configuration registers definition in file *config.c* if needed
11. If Fcy has been modified (due to a changes of PLL, Fosc, etc) modify the FXT and PLL definitions in file *config.h*.
12. Depending on the target Fcy could be necessary to rewrite the CAN bit segments definition in the file *canall.h*. This table has a row for each possible baudrate (1Mbps, 500Kbps, 250Kbps, 125Kbps, 100Kbps, 62.5Kbps and 50Kbps) in where the (BRP-1) register, propagation time segment, phase segment 1 and phase segment 2 of the CAN interface are defined.
13. Recompile the project normally.

Other changes

The file *config.h* contains a set of definitions that user could change if it is needed.

- **CAN_BAUDRATE:** Used to fix the desired CAN transmission speed. The possible options are BAUDRATE_1M, BAUDRATE_500k, BAUDRATE_250k, BAUDRATE_125k, BAUDRATE_100k, BAUDRATE_62k, BAUDRATE_50k.
- **CAN_TIMEOUT_S:** Used to set the desired timeout in *wait bootloader mode command*. The range of allowed values depends directly of the internal frequency of the MCU/DSC (Fcy).
- **USE_CAN_MODULE:** Used to select the CAN hardware module used (only valid is devices with more than one CAN interface).
- **NODE_ID:** In devices without internal EEPROM the Node ID is hard-code using this constant.



Please pay special attention if you modify these parameters to avoid undesired behaviors.

Description of Ingenia 16bitPIC CAN bootloader software interface (GUI and command line application).

Introduction

Ingenia 16bitPIC CAN bootloader package comes with two different software interfaces that can act as masters during the bootloader process:

- A Graphical User Interface (GUI) and,
- A Command Line Interface (CLI).

Figure 6 shows the window layout of the 16bitPIC CAN bootloader GUI.

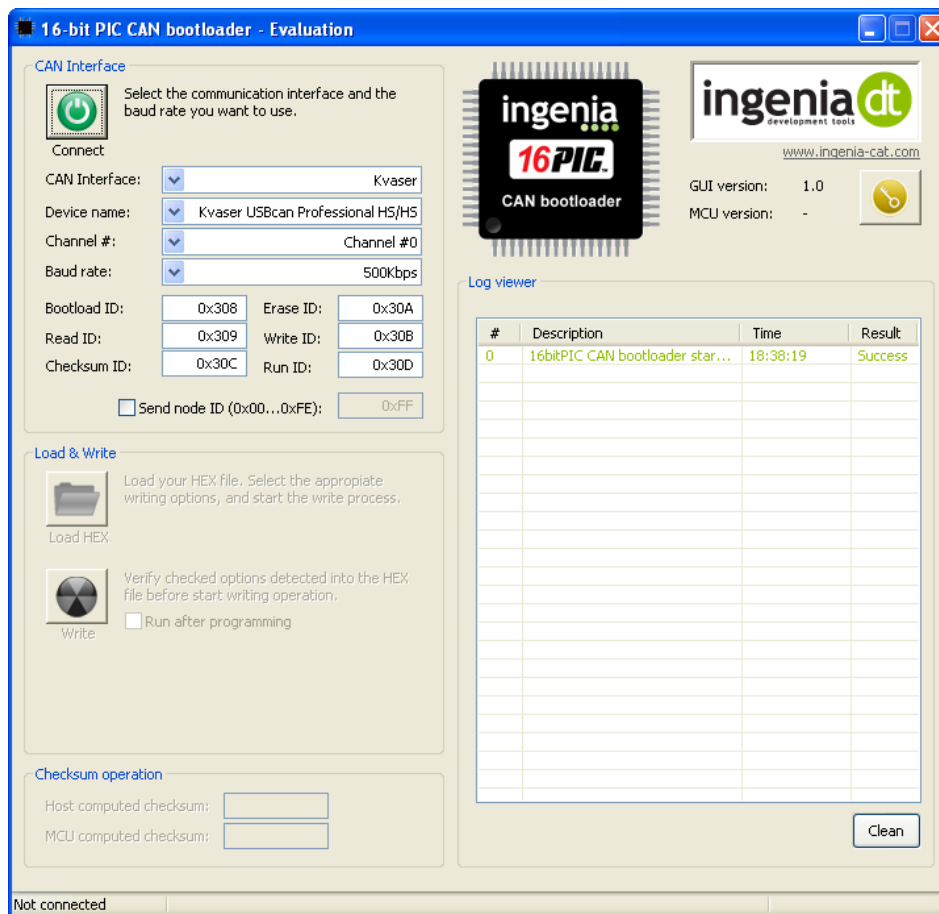


Figure 6: 16bitPIC CAN bootloader GUI

Both software interfaces (GUI and CLI) are equivalent and have the same functionality. Next subchapters describe the options available for both of them.

The selection of the options for the CLI application has to be performed via an initialization file. The same file can be used to set the default options for the GUI application when starting up.

Init file

16bitPIC CAN bootloader CLI is a text-only software interface. It works with an initialization file, or init file for short, to set the desired bootloader options. The same file can be used by the GUI to set defaults when starting up.

The init file has to be specified as an argument of the command input for the CLI:

```
> 16bitPICCANbootloaderCLI.exe [-options] InitFileName.txt
```

Valid values for *-options* field are:

-h to display help

-a=activation_code to introduce an activation code

(e.g.: `16bitPICCANbootloaderCLI.exe -a=16f785236864`)

To use the init file to set the default parameters of the GUI, the file should be named **CANbootloader.txt**, and needs to be placed at the executable working directory.

The syntax of the init file is as follows:

Parameter name:	[space]	value	[Carriage return]
-----------------	---------	-------	-------------------

The valid parameter names and values are specified in the Table 1.

Parameter	Description	Valid values
CAN interface	Manufacturer of CAN interface	Kvaser, ESD, IXXAT, PCAN-Light or Vector
Device name	Name of CAN device	PCAN-USB, CAN_USB2, ...
Channel #	Channel number to use	0...# channels
Baud rate	Desired baud rate: 0 ... 1Mbps 1 ... 500Kbps 2 ... 250Kbps 3 ... 125Kbps 4 ... 100Kbps 5 ... 62.5Kbps 6 ... 50Kbps	0 – 6
Bootload ID	ID for bootload message	0x000000 - 0x3FFFFFF
Erase ID	ID for erase message	0x000000 - 0x3FFFFFF
Read ID	ID for read message	0x000000 - 0x3FFFFFF
Write ID	ID for write message	0x000000 - 0x3FFFFFF
Checksum ID	ID for checksum message	0x000000 - 0x3FFFFFF
Run ID	ID for run message	0x000000 - 0x3FFFFFF
Node ID	ID of node to bootload	0x001 - 0xFF

Send node ID	Specify whether to perform specific node bootloading or not	Yes - No
Default HEX	Specify default HEX location	Valid URI
Update flash	Update flash memory of MCU with flash data from HEX file	Yes - No
Update EEPROM	Update EEPROM memory of MCU with EEPROM data from HEX file	Yes - No
Update configs	Update configs of MCU with configs from HEX file	Yes - No
Run after update	Specify whether to execute the code after programming or not	Yes - No
Company logo	Specify logo file	BITMAP file (*.bmp)
Company link	Specify logo link	Valid HTTP address

Table 1: Parameters and Values for Init file

Following is an example of an Init file definition:

```

CAN interface: ESD
Device name: CAN_USB2
Channel #: 0
Baud rate: 4
Bootload ID: 0x308
Erase ID: 0x30A
Read ID: 0x309
Write ID: 0x30B
Checksum ID: 0x30C
Run ID: 0x30D
Node ID: 0xFF
Send node ID: Yes
Default HEX: D:\myPIC.hex
Update flash: Yes
Update EEPROM: yes
Update configs: yes
Run after update: Yes
Company logo: Ingenia-logo.bmp
Company link: www.ingenia-cat.com

```



HEX files can be loaded from HTTP or FTP sites. To specify a file on an FTP or HTTP server use a standard FTP or HTTP URI.

Ex: ftp://username:password@mysite.com/myfile.hex

Graphical User Interface (GUI) operation

Next subchapters describe how to perform a bootloader operation when working with the 16bitPIC CAN bootloader GUI software interface.

CAN interface setup

To start the bootloader process it has to be selected the CAN interface, the device name, the channel number and the baud rate to use for the communication with the PIC platform (see Figure 7).

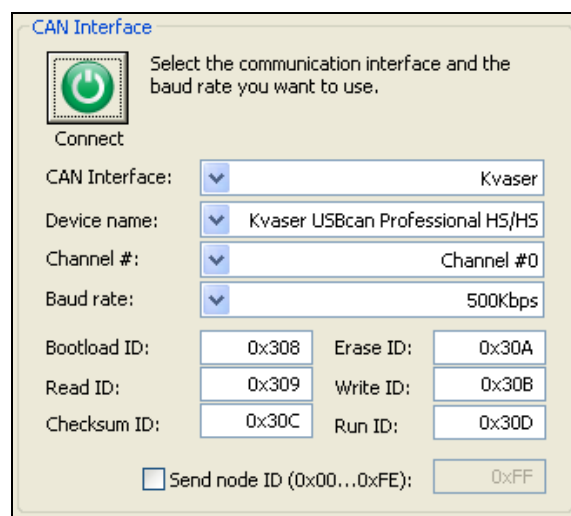


Figure 7: CAN interface setup



Virtual channels CAN NOT be used to communicate with any MCU/DSC PIC hardware.

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 CAN messages requesting to the listening device to enter in *bootloader mode*. Reboot your MCU/DSC hardware (with bootloader firmware programmed into it) to enter in this mode.



Use the bootload ID message field to specify which message is used to enter in bootloader mode. You can also specify the CAN ID for Erase, Read, Write Checksum and Run messages.



*To use **specific node bootloading** feature, check the box “Send node ID” and specify the desired node to bootstrap.*

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 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.



HEX files can be loaded from HTTP or FTP sites. To specify a file on an FTP or HTTP server use a standard FTP or HTTP URI.

Ex: ftp://username:password@mysite.com/myfile.hex

Click on the check boxes to select the memory zones to be written (see Figure 8).



The screenshot shows a rectangular box containing three rows of text. Each row has a checkbox on the left, followed by the name of the memory region, and then its address range in parentheses. The first row has a checked checkbox, the second has an unchecked checkbox, and the third has an unchecked checkbox.

<input checked="" type="checkbox"/>	Program flash	(0x0000 - 0x17FFE)
<input type="checkbox"/>	Write data EEPROM	(0x7FF000 - 0x7FFFFE)
<input type="checkbox"/>	Configure registers	(0xF80000 - 0xF8000B)

Figure 8: Memory regions

Writeable memory zones of a device can be divided into:

- program flash
- write data EEPROM
- and configure registers

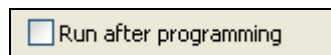
16bitPIC CAN bootloader shows you the 3 zones and its associated range of addresses.



Ingenia 16bitPIC CAN 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.

If you want to run the uploaded code after the programming process check the box *Run after programming* (see Figure 9).



The screenshot shows a single checkbox with the text "Run after programming" to its right.

<input type="checkbox"/>	Run after programming
--------------------------	-----------------------

Figure 9: Run after programming checkbox



Run after programming operation will be performed ONLY if programming operation succeeds.

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

Checksum operation

After a complete programming, the 16bitPIC CAN 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 10 and Figure 11.

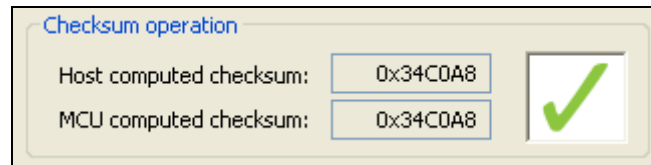


Figure 10: Successful checksum operation

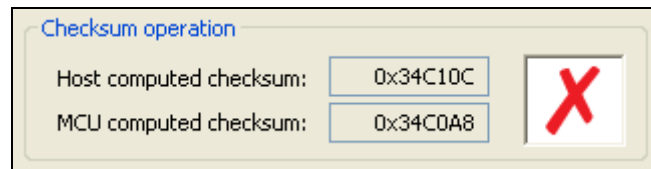


Figure 11: Wrong checksum operation

Customizing GUI layout

The layout of the GUI can be easily customized through the init file. The available customization options are:

- Logo replacement.
- Hyperlink modification.

To change the logo on the GUI layout (see Figure 11) you need to specify a valid image file for the *Company logo* field in the init file. Logo file must be BITMAP (*.bmp) formatted.

To change the hyperlink displayed on the GUI layout (see Figure 11) you need to modify the *Company link* field in the init file.



Figure 12: GUI customization

Command line Interface (CLI) operation

16bitPIC CAN bootloader CLI is a text-only software interface which operation is equivalent to the 16bitPIC CAN bootloader GUI.

To execute the application write:

```
> 16bitPICCANbootloaderCLI.exe InitFileName.txt
```

See the Init file chapter for further information on init file syntax.



To display the usage help of 16bitPIC CAN bootloader CLI syntax use `-h` option.

Once the application is executed, the CLI will load and configure the CAN interface specified in the init file and start sending CAN messages requesting to the listening device to enter in *bootloader mode* (see Figure 12). Reboot your MCU/DSC hardware (with bootloader firmware programmed into it) to enter in this mode.

```
C:\WINDOWS\system32\cmd.exe
CAN bootloader Command Line Interface v1.0
Copyright (c) 2008-2011 Ingenia-CAT S.L.
License# :
Kvaser CAN library version 1.0 loaded.
Vector CAN library version 1.0 loaded.
ESD CAN library version 1.0 loaded.
>>>>>> Bootloader process STARTED.
>> Parse configuration file.....Ok
>> Open CAN interface .....Ok
>> Connecting.....
.....
```

Figure 13: CLI connecting process

If the connection process succeeds, the CLI will start loading the HEX file and uploading it into your MCU/DSC hardware platform. At the end of this process the checksum operation will be performed and the results displayed.

```

C:\WINDOWS\system32\cmd.exe
CAN bootloader Command Line Interface v1.0
Copyright (c) 2008-2011 Ingenia-CAT S.L.
License# :
Kvaser CAN library version 1.0 loaded.
Vector CAN library version 1.0 loaded.
ESD CAN library version 1.0 loaded.
>>>>>> Bootloader process STARTED.
>> Parse configuration file.....Ok
>> Open CAN interface .....Ok
>> Connecting.....Ok
.....Ok
>> Downloading HEX file.....Ok
>> Uploading code.....Ok
.....Ok
>> Performing checksum.....Ok
>> Launching program.....Ok
>> Close CAN interface.....Ok
>>>>>> Bootloader process FINISHED.

```

If any of the above processes fails an error code will be returned. Following is a table indicating the meaning of each error code:

Error Code	Meaning
257	Bad arguments
513	Failed loading CAN interfaces
769	Failed opening init file
770	Syntax error on init file
1025	Failed while opening CAN interface
1281	Non-specified connection error
1282	Invalid manufacturer
1283	MCU not found on the XML file list
1537	Non-specified error loading HEX file
1793	Failed during firmware upload
2049	Failed while computing host checksum
2050	Failed while computing MCU checksum
2305	Failed while trying to launch application

The XML PIC list file

16bitPICCAN bootloader can work with some 16 bit MCU/DSC families from Microchip (dsPIC30F/dsPIC33F/PIC24H). The detection process of processor uses **CANbootloaderMCUList.xml** file to identify the connected device and define its memory map. This file is located in the installation folder and consists on a list of supported MCU devices.

If your hardware device doesn't appear in the list, you can add it manually (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 list file. A device is a description of a MCU/DSC. It is characterized by a manufacturer, an identifier and a name.

The manufacturer (*manufacturer* tag) is the name of the manufacturer. Only one value is allowed: "Microchip".

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

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

Within tags `<device></device>` you have to define the memory zones of the MCU/DSC. These can be:

- Code or programming (FLASH),
- Data (EEPROM),
- and Configuration (CONFIGS).

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

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

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

For each zone you need to define its attributes: *start address*, *end address*, *instructions per row*, *rows per sector (or page)*, *addresses per instruction* and *bytes per instruction*.

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).

16bitPIC CAN bootloader assumes that MCU/DSC internal memory is arranged into rows and pages/sectors as in Table 1.

For Microchip devices each instruction is 4 bytes length (24bits data+8 phantom bits) for FLASH and EEPROM memory and 2 bytes length for CONFIGS memory.

The number of addresses used for each instructions are 2 for Microchip devices.

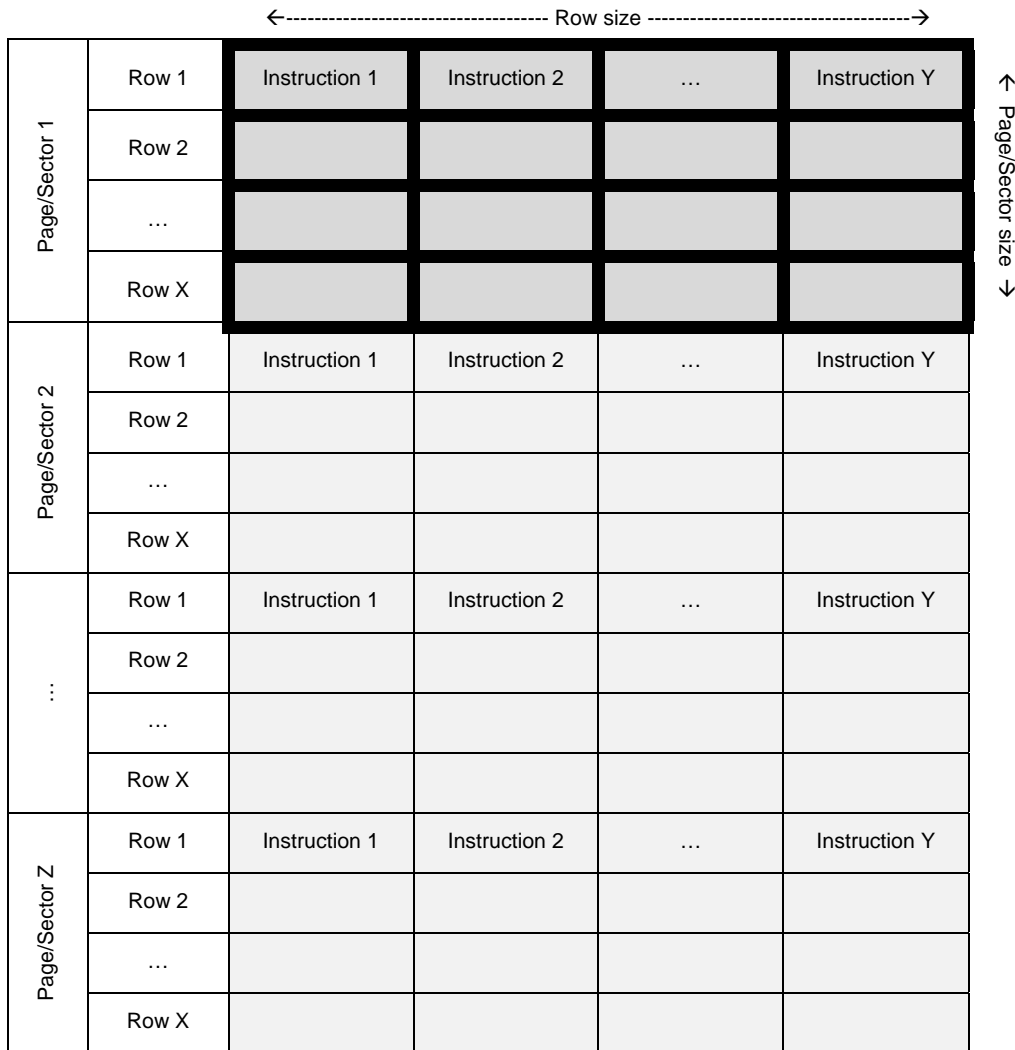


Table 2: Memory organization

The following example shows a complete definition of a device.

```

<device manufacturer="Microchip" id="0x0101" name="dsPIC30F4011">
<memcode startaddress="0x000000" endaddress="0x007FFE" instructionxrow="32"
rowsxsector="1" addressesxinstruction="2" bytesxinstruction="4">
  <bootloader startaddress="0x007E00" endaddress="0x007FFE" />
</memcode>
<memdata startaddress="0x7FFC00" endaddress="0x7FFFE" instructionxrow="16"
rowsxsector="1" addressesxinstruction="2" bytesxinstruction="4"/>
<memconfig startaddress="0xF80000" endaddress="0xF8000B" instructionxrow="1"
rowsxsector="1" addressesxinstruction="2" bytesxinstruction="2">
</memconfig>
</device>

```

18bitPIC CAN bootloader software tools

Encryption tool

Included in the 16bitPIC CAN bootloader package is delivered an HEX file encryption tool suitable for use together with 16bitPIC CAN bootloader GUI or CLI. Both software interfaces have the ability to load files encrypted with this utility (*.hexenc).

The encryption tool is a very easy to use application. User just need to select the desired HEX file to encrypt and click on the encrypt button (see Figure 14). An encrypted file with the same path and name will be generated. The extension will be changed to .hexenc.

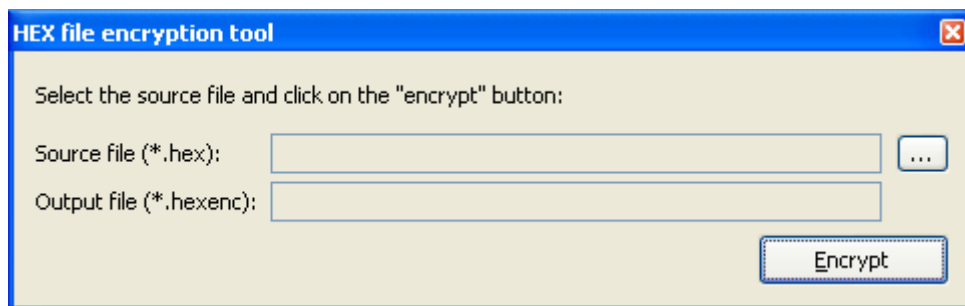


Figure 14: Encryption tool layout



Encrypting HEX files allow product designers to deliver protected HEX files to its customers (cannot be copied or modified).
