USB HID Bootloader host for PIC

Table of Contents

1 Introduction

This document contains notes made during the development of a bootloader host for PICs equipped with Firmware of mikroElektronika. This means the Host's behaviour is compatible with that firmware.

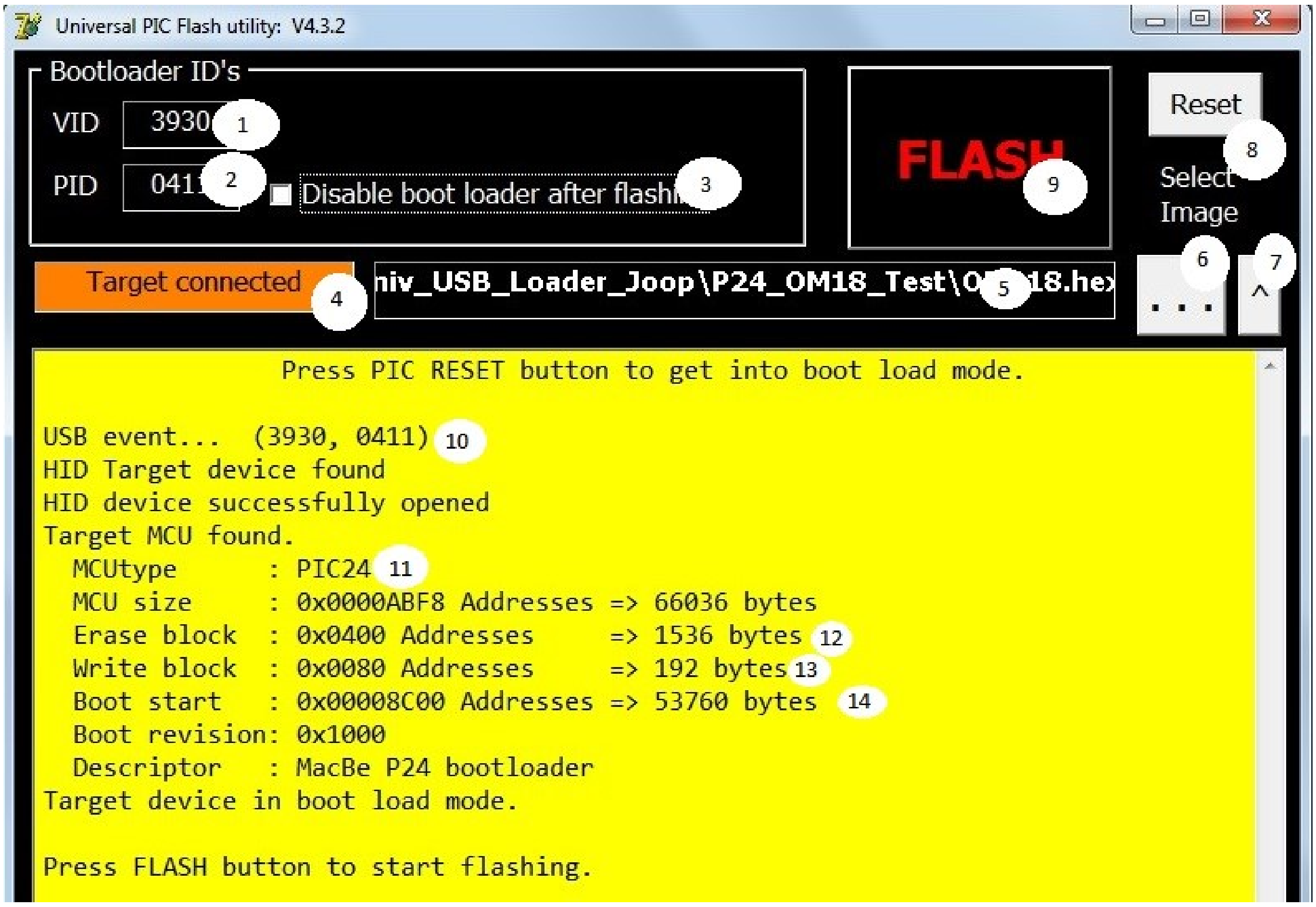
The host software is written in Delphi 7.0.

It contains a few extra features with respect to mE's USB HID (mikrobootloader):

- Progress bars for both erasing and programming
- A more verbatim version with the display of important addresses of the PIC, the Bootloader FW and the program to be loaded.
- The possibility to enter the VID and the PID USB codes of the Firmware.
- Memory for the last selected hex file, plus a file 'history' per kind of Pic.
- A display of the free user flash (between the FW and the program loaded).
- The suggestion for a place of an extra flag (to e.g. disable the bootloader permanently). The bootloader FW has to be extended to make it capable to do this.

2 The User interface

2.1 Before Flashing


Universal PIC Flash utility: V4.3.2
Bootloader ID's
VID 3930
PID 0411
Disable boot loader after flashing
FLASH
Reset
Select Image
Target connected
niv_USB_Loader_Joop\P24_OM18_Test\OM18.hex
...
^
1
2
3
4
5
6
7
8
9
10
11
12
13
14

```
                    Press PIC RESET button to get into boot load mode.

USB event...  (3930, 0411)
HID Target device found
HID device successfully opened
Target MCU found.
  MCUtype       : PIC24
  MCU size      : 0x0000ABF8 Addresses => 66036 bytes
  Erase block   : 0x0400 Addresses     => 1536 bytes
  Write block   : 0x0080 Addresses     => 192 bytes
  Boot start    : 0x00008C00 Addresses => 53760 bytes
  Boot revision: 0x1000
  Descriptor    : MacBe P24 bootloader
Target device in boot load mode.

Press FLASH button to start flashing.
```

Fig. 1

Legend:

(1) USB Vendor ID from Bootloader Firmware

(2) USB Product ID from Bootloader Firmware

(3) Disable bootloader after flashing (see section 7)

(4) Target (Bootloader firmware device) connected

(5) Name of the hexfile to be flashed

(6) Select name of the hexfile to be flashed

(7) History list of hexfiles flashed (select file or press key to leave)

(8) Reset for a new loading and flashing sequence

(9) Flash button

(10) VID and PID of usb device attachment event

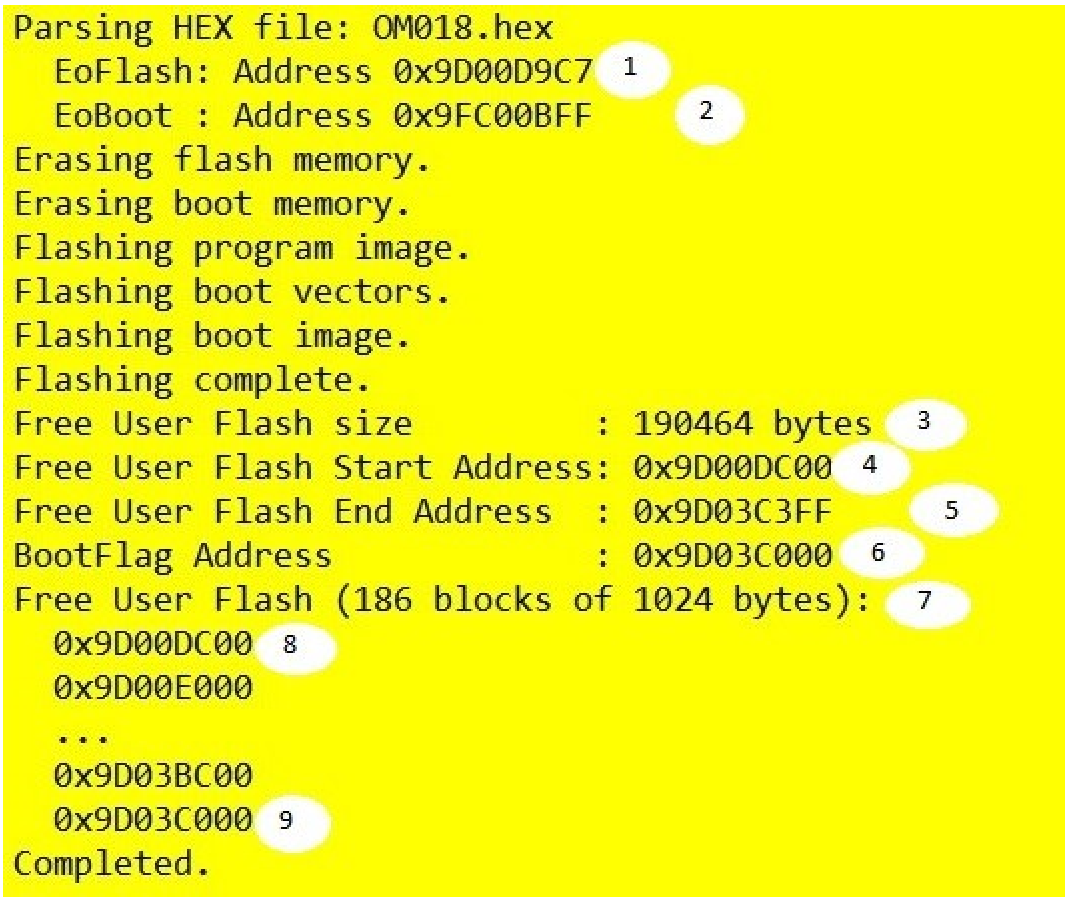
(11) Pic type

(12) Erase block size in bytes

(13) Write block size in bytes

(14) Start Address of the bootloader firmware (actual address in the PIC)

2.2 After Flashing



```
Parsing HEX file: OM018.hex
  EoFlash: Address 0x9D00D9C7
  EoBoot : Address 0x9FC00BFF
Erasing flash memory.
Erasing boot memory.
Flashing program image.
Flashing boot vectors.
Flashing boot image.
Flashing complete.
Free User Flash size          : 190464 bytes
Free User Flash Start Address: 0x9D00DC00
Free User Flash End Address   : 0x9D03C3FF
BootFlag Address              : 0x9D03C000
Free User Flash (186 blocks of 1024 bytes):
  0x9D00DC00
  0x9D00E000
  ...
  0x9D03BC00
  0x9D03C000
Completed.
```

Fig. 2

Legend:

(1) Last byte of the main (flashed) program

(2) Last byte of the Boot memory

(3) Free user flash size in bytes (see section 6)

(4) Free user flash address of the first (lowest address) erase block (see section 6)

(5) Free user flash address of the last (highest address) erase block (see section 6)

(6) Bootflag address (see section 7)

(7) Free User flash overview (the block size is always equal to the erase size)

(8) .. (9) Overview if the free user flash erase blocks

3 Tasks of the Bootloader host.

- Connect to the bootloader firmware in the PIC via USB
- Make the PIC FW go into the bootloader mode
- Get the bootinfo from the PIC firmware (see section3.1)
- Parse the hexfile chosen by the user (see section 3.2)
- Give the commands to the firmware to erase the necessary PIC flash blocks (see section 3.3)
- Give the commands to the firmware to program the parsed data to the PIC flash (see section 3.4)
- Make the firmware leave the bootloader mode

3.1 Get the Bootinfo from the FirmWare

This info contains a few important values that enable range checking when the hex file is parsed and a few ‘sizes’:

- The PIC type
- The bootloader firmware start address (the actual address in the PIC)
- The ‘erase block’ size in #bytes
- The ‘write block’ size in #bytes
- and some other less important ones (not actually used in the process)

Important:

The bootinfo (defined in the firmware as a record) is read via USB as a byte array. This means unfortunately the offset of the record member data in the array depends on the type of PIC (P16/P18, P24 or P32). This is due to the alignment rules for variables larger than one byte in the PIC.

For PIC32 only: the bootloader firmware start address is in the KSEG0 virtual memory area (starting from $9D000000), not in the physical memory range (starting from $1D000000).

3.2 Parse the hexfile

Parsing is reading in the hex file (both with addresses and data) and putting the read in data in a byte array at the correct address provided by the hexfile.

The parsing of the hexfile is straightforward, except for the addresses used in that file for the P16 and the P24 files. Reason: the hex file always assumes 1 byte per address, so the addresses are ‘byte addresses’.

However, the actual addressing for P16 this is 1 address per 2 bytes, and for P24 this is 2 addresses per 4 bytes.

This means that for P16 and P24 addresses in the hex file are a factor 2 higher than the actual addresses where the code will reside in the PIC.

Additionally, in the P24, from the 4 bytes per 2 addresses one byte is a dummy byte, which means that there are 3 bytes per 2 addresses. This means that every fourth byte has to be removed after parsing (in fact this is a postprocessing part of the parsing process). Once this has been done the actual PIC address is 2/3 of the byte address from the hexfile.

During parsing also the highest databyte (except for the config bytes) will be calculated. After parsing a check is done if the parsed data will fit under the bootloader FW, and if not a message is given and the whole process is stopped.

Specific for PIC32:

- The addresses in the hex file are those from the Physical memory area (starting from $1D000000 for the program flash and $1FC00000 for the boot flash).
- Two byte arrays are provided to hold parsed data: one for the program data (addresses below $1FC00000) and one for the bootarea data (addresses >= $1FC00000).

3.3 Erase the PIC Flash blocks

The host will (give commands to the FW to) erase all blocks below the FW bootloader (the bootloader FW start address). The firmware is capable of erasing more than 1 block at a time, but unfortunately there is an undocumented feature here: The FW starts with the highest block and counts down to zero.

If one wishes to e.g. erase 10 blocks fro address zero onwards, the address of the 10th block has to be provided in the command, not address 0.

Since this counting down feature is undocumented, this host gives only erase commands for 1 block to the firmware.

Important:

The actual PIC address has to be provided with the command, not the address provided in the hex file. Keep in mind that the 'erase block' is in bytes, not in addresses (so, divide by 2 for P16, do *2/3 for P24 for the addresses).

For the PIC32: 2 areas have to be erased: the program area (addresses below $1FC00000) and the boot area (addresses >= $1FC00000). The addresses to be used are the Physical area ones.

3.4 Program the data to the PIC flash

The host will (give commands to the FW to) program all data from the parsed hex file. The firmware is capable of programming more than one block (with size 'write block') at a time, it splits the received data in the correct packet size for programming (including address incrementing). To prevent overflow in the FW the data packets sent to the FW should be limited to 'erase size', because that is the size of the receive buffer in the FW.

For the PIC24 the fourth byte (the obsolete one) has already to be stripped from the data sent to the PIC FW.

Important:

The actual PIC address has to be provided with the command, not the address provided in the hex file. Keep in mind that the 'write block' is in bytes, not in addresses (so, divide by 2 for P16, do *2/3 for P24 for the addresses).

For PIC32: The addresses to be used are the Physical area ones.

Additionally:

Before the parsed data is actually programmed in the PIC, the so called 'vectors' have to be placed to make sure that the bootloader takes control after a reset of the PIC, and, if bootloading is not required (FW timeout) or after bootloading, the bootloader can activate the loaded (main) program.

The vectors are inserted into the parsed hex file array before programming:

- For P16, P16 and P24: A jump instruction (or code) to the FW bootloader at address 0 (the MCU reset vector). This code is calculated from the 'bootloader firmware start address', see (a) in the figures below.
- A jump instruction to the loaded (main) program right under the 'bootloader firmware start address' where it is expected by the bootloader FW, see (b) in the figures below.

  For P16, P18 and P24 this jump code is present at address 0 in the parsed hex file and can be copied from there.

Remark: The jump code and its size are dependent of the PIC type at hand.

For P32 specific:

- The jump instruction to the FW bootloader is not placed at address 0, but at address 40 in the Boot flash area, see (a) in the figures below. The address in this instruction has to be calculated. Calculated addresses should be in the KSEG0 memory area.
- The jump instruction to the loaded program always jumps to address 50 in the boot flash area (the original entry for the loaded program). This code is already present in the hex file, see (b) in the figures below.
- Two flash areas have to be programmed: the program area (addresses < $1FC00000) and the boot area (addresses >= $1FC00000).

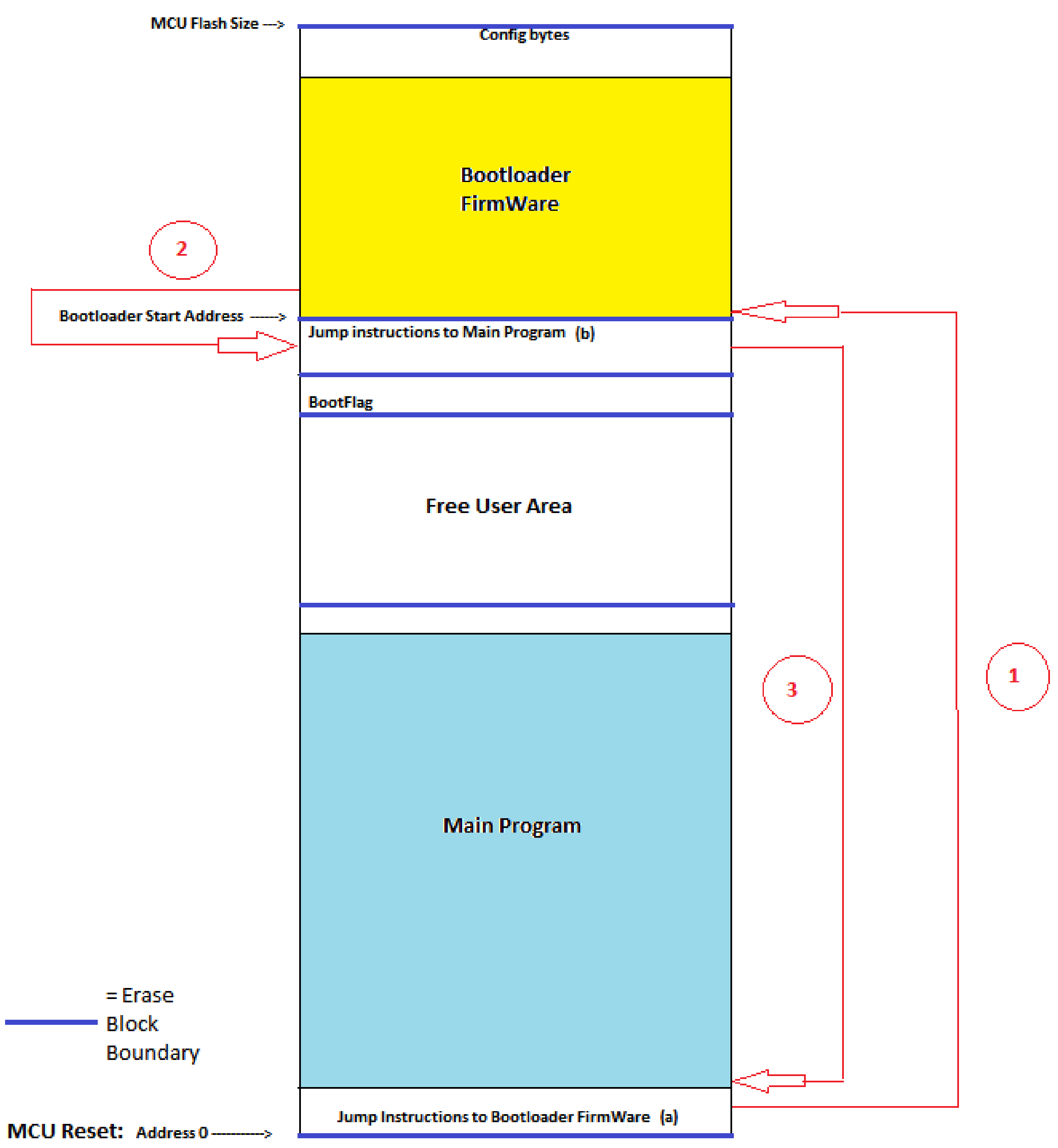
4 Tasks for the Bootloader firmware in the PIC.

- Decide (with a timeout) if a host application makes contact via USB. When Timeout the bootloader FW invokes the main program.
- Go into bootload mode (in case contact is made with a host application and that host issues the 'enter bootloader' command).
- Send the bootinfo when asked for by the host
- Accept and execute erase commands issued by the host
- Accept and execute write commands issued by the host
- Leave bootload mode on command issued by the host
- Optionally: set or reset the bootflag. If the bootflag is set the bootloader always invokes the main program (without timeout), it never gains control itself.

These will not be handled here. Please have a look in some Bootloader FW code.

5 The PIC memory map

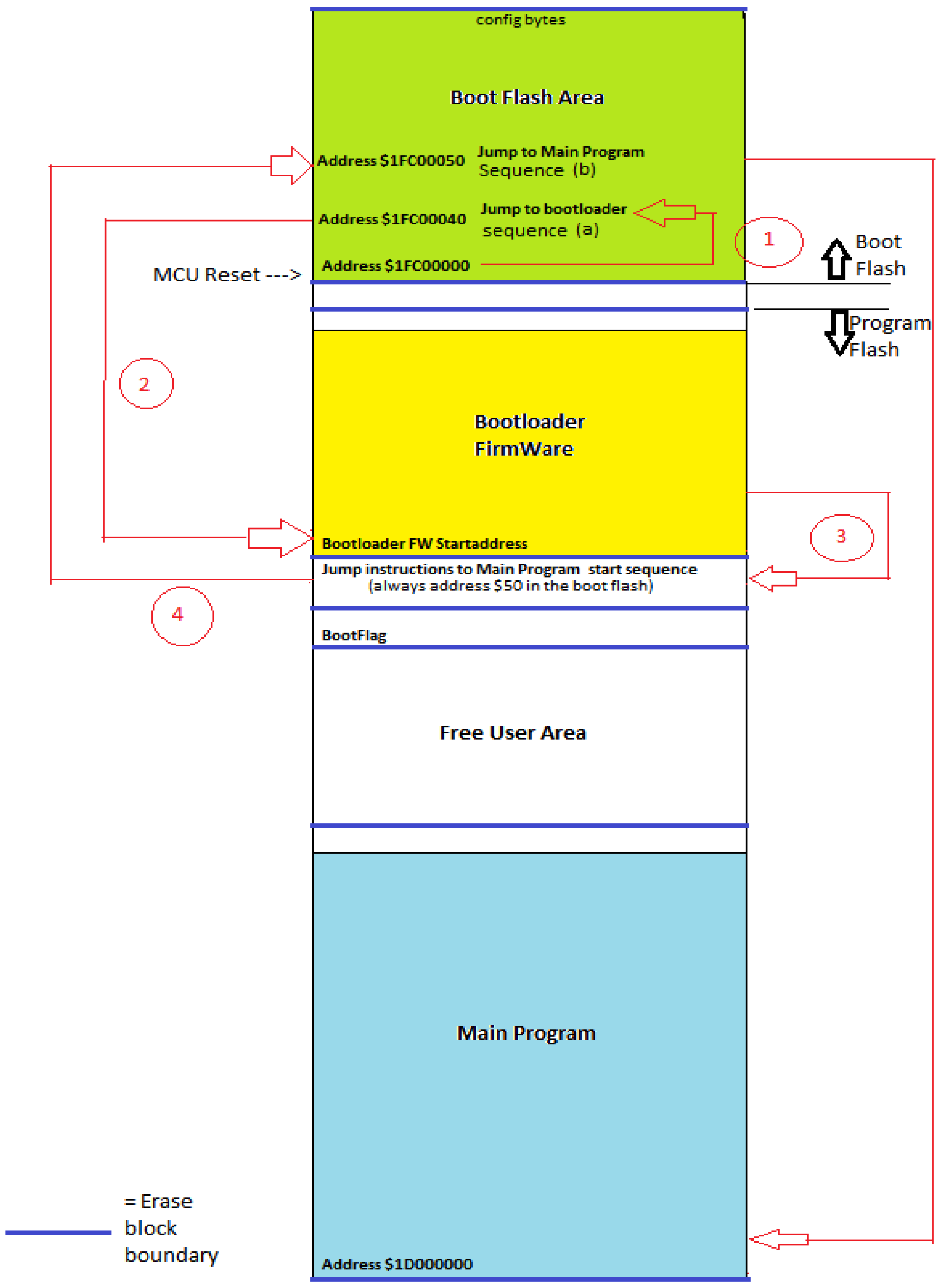
5.1 PIC16, PIC18, PIC24


MCU Flash Size --->
Config bytes
Bootloader FirmWare
2
Bootloader Start Address ------>
Jump instructions to Main Program (b)
BootFlag
Free User Area
3
1
Main Program
= Erase Block Boundary
MCU Reset: Address 0 ---------->
Jump Instructions to Bootloader FirmWare (a)

As can be seen above, the bootloader is located at the top of the program Flash memory. It always starts at an erase block boundary.

For this type of PIC the reset of the PIC starts the execution at address 0. At this address a jump sequence to the bootloader is inserted by the Delphi tool, so the bootloader always gains control, see (1) above. At some time later the bootloader can decide to give control to the main program, this is done by calling a routine containing a jump sequence to the main program, see (2) above. This Jump sequence givers control to the main program, see (3) above. In case the bootloader is ordered to go to the bootloader mode, call (2) never occurs.

5.2 PIC32



The situation here is a little bit more complex that for the non PIC32 types.

After a reset the code execution starts from address $1FC00000 (the start of the boot flash). After a short moment the jump sequence at $1FC00040 is executed, see (1) above, and the bootloader gets control, see (2) above. At some time later the bootloader can decide to give control to the main program, this is done by calling a routine containing a jump sequence to the main program, see (3) above. However, this routine always contains a jump sequence to address $1FC00050, which is the entry in the boot memory for the main program, see (4) above. Finally the main program gets control by jump (5). In case the bootloader is ordered to go to the bootloader mode, call (3) never occurs.

6 The free user Flash area

Free user flash is program flash that is not used by the Bootloader firmware or the main program. The main program can use this area for its own purposes (e.g. storing non volatile data). It starts from the first erase block boundary above the end of the main program, and ends 1 erase block below the bootloader firmware. The block just below the bootloader firmware should not be touched by the main program because the Jump sequence to the main program, used by the bootloader firmware, resides there. Corrupting this sequence makes that the bootloader firmware can not give control to the main program any more.

7 The BootFlag

The flag is used to disable the bootloader. If set, the bootloader FW always invokes the main program without the initial waiting time for connection to a bootloader host.

This flag resides at the beginning of the last erase block of the user flash area. The bootloader host sends the command to the bootloader FW to set its value according the “Disable bootloader after flashing” checkbox. Of course the bootloader FW has to be equipped with the code for setting and testing the flag. This feature is not present in the mE bootloader firmware’s.

[end of document]