1. Features

- Bootloader of the universal processor UNIV 3 CPU.
- Bootloader version: 3.4
- Three working modes:
  - communication only via HAPCAN bus
  - communication via HAPCAN bus and serial port UART
  - programming mode via HAPCAN bus or/and serial port UART
- Responds to 7 UART messages
- Responds to 14 CAN messages
- Possibility to programme FLASH and EEPROM memory using UART or/and CAN ports without hardware programmer
- Possibility to write own functional firmware

2. Overview

The bootloader is the program that is executed immediately when processor is powered up. The main task of this program is to enable communication with the processor, eg. from PC, without the use of special hardware programmer. Communication with the processor can be established through the serial port UART (RS232) or CAN bus. Bootloader also allows uploading and configuring functional firmware which makes processor working as the specific device type, eg. button or dimmer, etc. With the bootloader, communication with the processor is possible even if uploaded firmware is incorrect or is not uploaded at all.
This document is a supplement to the original documentation of Microchip PIC18F26K80 processor available on the microchip.com website.

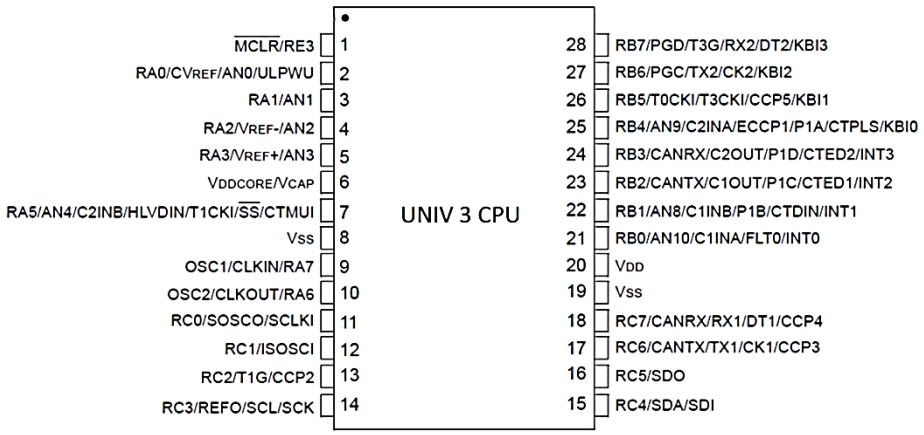


Figure 1. UNIV 3 CPU pin diagram

3. Table of contents

4. Bootloader working modes

Bootloader can operate in one of three modes:

1. Normal CAN & UART mode (32MHz)
2. Normal CAN mode (8MHz)
3. Programming mode (CAN or CAN & UART)

4.1. CAN and UART mode (32 MHz)

This mode enables communication with the processor via the HAPCAN bus or serial port (UART - universal asynchronous receiver / transmitter). The processor in this mode is mainly used to build interfaces between the PC and HAPCAN bus.
This mode is enabled hardware way by connecting pin 3 of UNIV 3 CPU processor with the positive supply (+5 V) and rebooting the processor (Figure 2). In addition, the quartz frequency is multiply by 4, so the CPU is clocked at 32 MHz.
The bootloader mode is signalled on pin 27 of UNIV 3 CPU after processor rebooting (Figure 5).

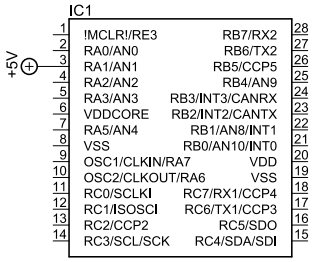


Figure 2. Connecting UNIV 3 CPU for CAN and UART mode

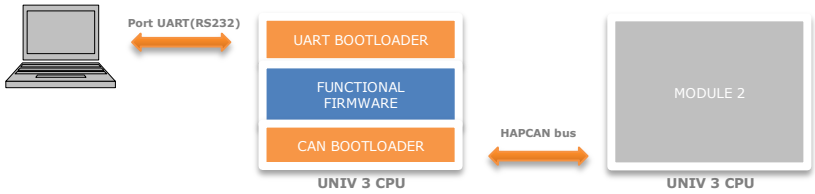


Figure 3. Communicating to UNIV 3 CPU processor in bootloader CAN and UART mode

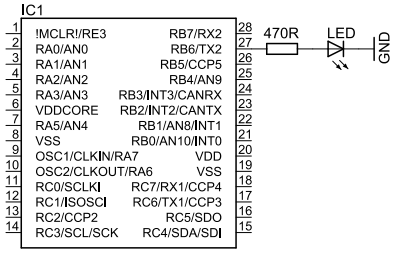


Figure 4. Bootloader CAN mode signalling pin

Bootloader in CAN and UART mode (32 MHz)
– communication with the processor is possible via the HAPCAN bus or via the UART serial port (RS232). Processor clock frequency is 32 MHz.

Figure 5. Bootloader CAN and UART mode signalling

4.2. CAN mode (8 MHz)

This mode enables communication with the processor only via the HAPCAN bus. To communicate to the HAPCAN bus, a PC interface is needed. The PC interface passes messages from PC to the HAPCAN bus.
This mode is enabled hardware way by connecting pin 3 of UNIV 3 CPU processor to the negative supply (GND) and rebooting the processor (Figure 6). The CPU is clocked at 8 MHz.
The bootloader mode is signalled on pin 27 of UNIV 3 CPU after processor rebooting (Figure 9).

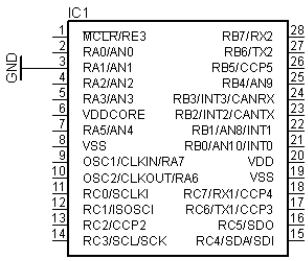


Figure 6. Connecting UNIV 3 CPU for CAN mode

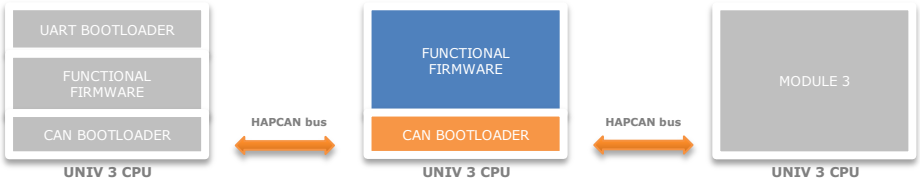


Figure 7. Communicating to UNIV 3 CPU processor in bootloader CAN mode

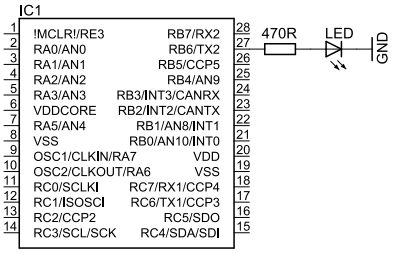
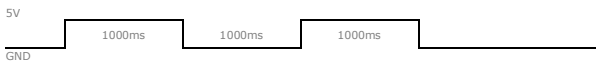


Figure 8. Bootloader CAN mode signalling pin

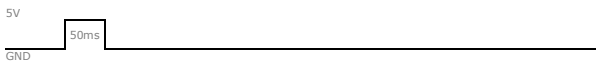


Bootloader in CAN mode (8 MHz)
– communication with the processor is possible via the HAPCAN bus only. Processor clock frequency is 8 MHz.

Figure 9. Bootloader CAN mode signalling waveform

4.3. Programming mode

In this mode, you can change the processor program and data memory. It is used to upload firmware and module configuration.
It is a software-switched mode by sending a message to the processor. If the processor hardware (by shorting pin 3 to +5 V) is defined to work with UART and CAN bootloader, then the programming mode allows you to change the processor memory both through the UART serial interface (RS232) and the HAPCAN bus. Defining the hardware (by shorting pin 3 to GND) to operate as a CAN bootloader will allow the CPU programming only via HAPCAN bus.
The programming mode is indicated on pin27 of UNIV 3 CPU after rebooting the (Figure 10).

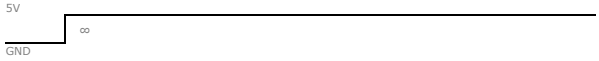


Bootloader in the programming mode

Figure 10. Bootloader programming mode signalling

4.4. Error mode

The bootloader initialization procedure starts from self-testing. If the result of this test is positive, bootloader goes to normal operation: CAN or UART & CAN.
The error is signalled on pin pin27 of UNIV 3 CPU as continuous high state (Figure 11).



Bootloader in error mode

Figure 11. Bootloader error mode signalling

5. Bootloader communication

5.1. HAPCAN message construction

HAPCAN messages are messages which are visible on PC side. It means they are slightly modified (from CAN messages). They are modified in HAPCAN<->PC interface firmware. The HAPCAN frame is made of 12 bytes. 4, the first is the ID CAN frame, and the remaining 8 bytes are data bytes. The difference between the CAN frame and the HAPCAN frame is shown in (Table 1). In addition, the HAPCAN frame has got the start, stop and checksum bytes.

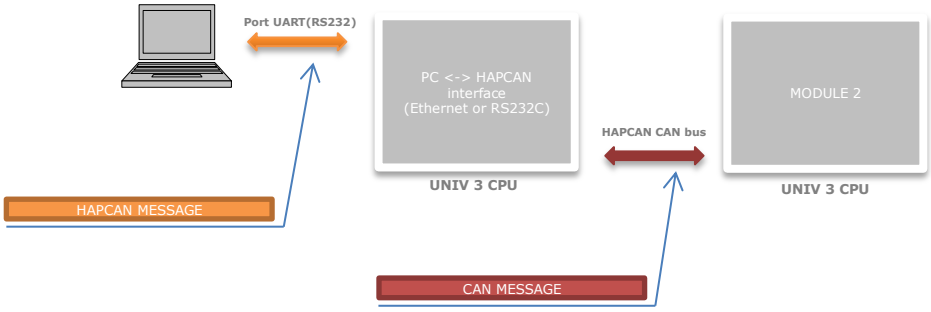
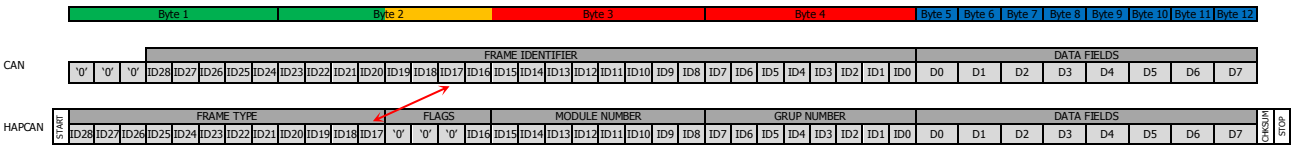


Figure 12. HAPCAN and CAN messages



START - Byte that signalizes frame beginning (always 0xAA)
FRAME TYPE - Frame type defines meaning of bytes in DATA FIELDS
FLAG ID16 - RE - Response Flag. The flag is equal "1" if message is sent in response to received request. Otherwise the flag is equal "0".
Other flags are not used.
MODULE NUMBER - Number of transmitting module
GROUP NUMBER - Group number of transmitting module
DATA FIELDS - 8 data bytes. Meaning of these bytes depends on FRAME TYPE
CHKSUM - Frame checksum. It is arithmetical sum of bytes from 1 to 12
STOP - Byte that signalizes frame end (always 0xA5)

Table 1. The difference between CAN and HAPCAN frame

5.2. Types of HAPCAN system messages

There are two types of HAPCAN messages:
- System message used to control and program system from the computer;
- Normal messages used for communication between the modules themselves.

The message type is recognized by FRAME TYPE - the first 12 bits of the message. Currently used message types are summarized in the table below. Only shadowed messages are handled by bootloader. Others can be handled in functional firmware.

SYSTEM MESSAGES	
0x010 – exit all from bootloader programming mode	Messages handled by the **bootloader in programming mode**
0x020 - exit one node from bootloader programming mode	
0x030 - address frame	
0x040 - data frame	
0x100 – enter into programming mode request to node	Messages handled by the **bootloader in normal mode**
0x101 - reboot request to group	
0x102 - reboot request to node	
0x103 - hardware type request to group	
0x104 - hardware type request to node	
0x105 - firmware type request to group	
0x106 - firmware type request to node	
0x107 - set default node and group numbers request to node	
0x108 - status request to group	Messages that can be handled by the **functional firmware** when bootloader is in normal mode
0x109 - status request to node	
0x10A - control message	
0x10B – supply voltage request to group	Messages handled by the **bootloader in normal mode**
0x10C - supply voltage request to node	
0x10D - description request to group	
0x10E - description request to node	
0x10F - DEV ID request to group	
0x111 - DEV ID request to node	
0x112 - up time request to group	Messages that can be handled by the **functional firmware** when bootloader is in normal mode
0x113 - up time request to node	
0x114 - health check request to group	
0x115 - health check request to node	

NORMAL MESSAGES	
0x301 - button node message	Messages that can be handled by the **functional firmware** when bootloader is in normal mode
0x302 - relay message	
0x303 - infrared receiver message	
0x304 - temperature sensor message	
0x305 - infrared transmitter message	
0x306 - dimmer message	
0x307 - blind controller message	
0x308 - LED controller message	

Table 2. List of HAPCAN messages

5.3. UART system messages

These messages are used to communicate with the bootloader via processor serial port UART (RS232). Communication with the processor requires a PC with RS232C port and voltage level transforming device connected between the processor and the PC - for example MAX232 chip.
Only part of the system messages is supported by the bootloader (Table 3). Others may be implemented in functional firmware. The following explains how to communicate with the bootloader via processor UART serial port (RS232).

0x100 – enter into programming mode request to node	Messages handled by the **bootloader in normal mode** (not programming)
0x102 - reboot request to node	
0x104 - hardware type request to node	
0x106 - firmware type request to node	
0x109 - status request to node	Messages that can be handled by the **functional firmware** when bootloader is in normal mode
0x10A - control message	
0x10C - supply voltage request to node	Messages handled by the **bootloader in normal mode** (not programming)
0x10E - description request to node	
0x111 - DEV ID request to node	
0x113 - up time request to node	Messages that can be handled by the **functional firmware** when bootloader is in normal mode
0x115 - health check request to node	

Table 3. List of HAPCAN system messages used to communicate via processor UART serial port

START	FRAME	FLAGS	CHKSUM	STOP

START - start byte 0xAA
FRAME - frame type
FLAGS - special bits
bit <0> RE -response flag. The RE = 1 if message is sent in response to received request. Otherwise RE = 0.
bit <3:1> not used, always "0"
CHKSUM - frame checksum (arithmetical sum of byte values between start and checksum byte)
STOP - stop byte 0xA5

Table 4. Construction of message to communicate with processor via serial port

0x100 – *enter into programming mode request to node*
It is a request for entering the programming mode, in which it is possible to make changes in the program and data memory of the processor. In response, the processor sends the frame confirming the command. Exiting the programming mode is described in section *5.5. UART Programming mode messages*.

Sent to UART ⇨	0xAA	0x100	0x0	0x10	0xA5								
Response ⇦	0xAA	0x104	0x1	0xFF	0xFF	BVER1	BVER2	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

BVER1 | BREV2 – bootloader version BVER1.BREV2

0x102 – *reboot request to node*
The message will restart the processor. The processor does not send any response.

Sent to UART ⇨	0xAA	0x102	0x0	0x30	0xA5
Response ⇦	NONE				

0x104 – *hardware type request to node*
It is a request for the hardware type. In response, the processor sends the device type, version and serial number.

Sent to UART ⇨	0xAA	0x104	0x0	0x50	0xA5								
Response ⇦	0xAA	0x104	0x1	HARD1	HARD2	HVER	0xFF	ID0	ID1	ID2	ID3	CHKSUM	0xA5

HARD1 | HARD2 – 0x3000 – universal processor or module UNIV
HVER – 0x03 – processor or module version
IDx – serial number

0x106 – *firmware type request to node*
This is a request for the firmware type. In response, the processor sends uploaded firmware data. If there is no firmware or it is incorrect, the error frame will be sent.

Sent to UART ⇨	0xAA	0x106	0x0	0x70	0xA5								
Response ⇦	0xAA	0x106	0x1	HARD1	HARD2	HVER	ATYPE	AVERS	FVERS	BVER1	BREV2	CHKSUM	0xA5

HARD1 | HARD2 – 0x3000 – firmware for universal processor or module UNIV
HVER – 0x03 – processor version
ATYPE – application (hardware) type
0x01 – button
0x02 – relay
0x03 – infrared receiver
0x04 – temperature sensor
0x05 – infrared transmitter
0x06 – dimmer
0x07 – blind controller
0x08 – LED controller
AVERS – application (hardware) version
FVERS – firmware version
BVER1 | BREV2 – bootloader version BVER1.BREV2

0x1F1 – *incorrect firmware*
If the uploaded firmware is incorrect, the processor sends the following frame.

Response ⇦	0xAA	0x1F1	0x1	FIRMFLAGS	FSUM2	FSUM1	FSUM0	0xFF	0xFF	BVER1	BREV2	CHKSUM	0xA5

FIRMFLAGS – error code
bit <0> - firmware error
bit <1> - bit is not used
bit <2> - incorrect firmware checksum
bit <3> - firmware is not written for UNIV 3 CPU processor (hardware mismatch)
bit <4:7> - bits are not used
FSUM2 | FSUM1 | FSUM0 – expected by bootloader firmware checksum
BVER1 | BREV2 – bootloader version BVER1.BREV2

0x10C – *supply voltage request to node*
It is a request for the supply voltage. In response, the processor sends a HAPCAN bus and processor supply voltage.

Sent to UART ⇨	0xAA	0x10C	0x0	0xD0	0xA5								
Response ⇦	0xAA	0x10C	0x1	VOLBUS1	VOLBUS2	VOLCPU1	VOLCPU2	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

VOLBUS1 | VOLBUS2 – bus voltage U_{BUS}=(VOLCPU1*256+VOLCPU2)*30,5/65472
VOLCPU1 | VOLCPU2 – processor core voltage U_{CPU}=(VOLCPU1*256+VOLCPU2)*5/65472

0x10E – *description request to node*
It is a request for user-defined 16-character description of the processor.

Sent to UART ⇨	0xAA	0x10E	0x0	0xF0	0xA5								
Response ⇦	0xAA	0x10E	0x1	abc0	abc1	abc2	abc3	abc4	abc5	abc6	abc7	CHKSUM	0xA5
	0xAA	0x10E	0x1	abc8	abc9	abc10	abc11	abc12	abc13	abc14	abc15	CHKSUM	0xA5

abcx – 16 character processor description

0x111 – *DEV ID request to node*
It is a request for the processor identification number written originally by Microchip. In response, the processor sends the data, which includes: CPU type and revision number. More information about DEV ID is in Microchip PIC18F26K80 processor documentation.

Sent to UART ⇨	0xAA	0x111	0x0	0x21	0xA5								
Response ⇦	0xAA	0x111	0x1	DEV ID1	DEV ID2	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

DEV ID1 | DEV ID2 - identification number written originally by Microchip

5.4. CAN system messages

These messages are used to communicate with the bootloader via HAPCAN bus. Communication with the bus requires a PC and HAPCAN RS232C or Ethernet interface.
Only part of the system messages is supported by the bootloader (Table 5). Others may be implemented in functional firmware. The following explains how to communicate with the bootloader via the HAPCAN bus.

0x100 - enter into programming mode request to node	Messages handled by the **bootloader in normal mode** (not programming)
0x101 - reboot request to group	
0x102 - reboot request to node	
0x103 - hardware type request to group	
0x104 - hardware type request to node	
0x105 - firmware type request to group	
0x106 - firmware type request to node	
0x107 - set default node and group numbers request to node	
0x108 - status request to group	Messages that can be handled by the **functional firmware** when bootloader is in normal mode
0x109 - status request to node	
0x10A - control message	
0x10B - supply voltage request to group	Messages handled by the **bootloader in normal mode** (not programming)
0x10C - supply voltage request to node	
0x10D - description request to group	
0x10E - description request to node	
0x10F - DEV ID request to group	
0x111 - DEV ID request to node	
0x112 - up time request to group	Messages that can be handled by the **functional firmware** when bootloader is in normal mode
0x113 - up time request to node	
0x114 - health check request to group	
0x115 - health check request to node	

Table 5. List of HAPCAN system messages used to communicate via HAPCAN bus

START	FRAME	FLAGS	MODUL No	GROUP No	D0	D1	D2	D3	D4	D5	D6	D7	CHKSUM	STOP

START - start byte 0xAA
FRAME - frame type
FLAGS - special bits
bit <0> RE -response flag. The RE = 1 if message is sent in response to received request. Otherwise RE = 0.
bit <3:1> not used, always "0"
MODUL No - sender module number
GROUP No - sender group number
Dx - data byte
CHKSUM - frame checksum (sum of byte values between start and checksum byte)
STOP - stop byte 0xA5

Table 6. Construction of message to communicate with processor via HAPCAN bus

0x100 – *enter into programming mode request to node*
It is a request for entering the programming mode, in which it is possible to make changes in the program and data memory of the processor. In response, the processor sends the frame confirming the command. Exiting the programming mode is described in section *5.6. CAN programming messages*.

Sent to CAN ⇨	0xAA	0x100	0x0	MODUL No	GROUP No	0xXX	0xXX	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x100	0x1	MODULE	GROUP	0xFF	0xFF	BVER1	BVER2	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

MODUL No - sender module number
MODULE - responder module number
GROUP No - sender group number
GROUP - responder group number
0xXX - irrelevant data can be any value
MODULE - module number of being asked node
GROUP - group number of being asked node
BVER1 | BVER2 - bootloader version BVER1.BREV2

0x101 – *reboot request to group*

The message will restart all processor in one or all groups. Processors don't send any response.

one group restart

Sent to CAN ⇨	0xAA	0x101	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	NONE														

all groups restart

Sent to CAN ⇨	0xAA	0x101	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	0x00	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	NONE														

- 0x00 - means all processors in group
- GROUP - group number of nodes to be restarted
- 0x00 - means all group

0x102 – *reboot request to node*

The message will restart one processor. The processor doesn't send any response.

Sent to CAN ⇨	0xAA	0x102	0x0	MODUL No	GROUP No	0xXX	0xXX	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	BRAK														

- MODUŁ - module number of node to be restarted
- GRUPA - group number of node to be restarted

0x103 – *hardware type request to group*

It is a request for the hardware type to all processor in one or all groups. In response, processors send device types, versions and serial numbers.

request to all nodes in one group

Sent to CAN ⇨	0xAA	0x103	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x103	0x1	MODULE	GROUP	HARD1	HARD2	HVER	0xFF	ID0	ID1	ID2	ID3	CHKSUM	0xA5

request to all groups

Sent to CAN ⇨	0xAA	0x103	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	0x00	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x103	0x1	MODULE	GROUP	HARD1	HARD2	HVER	0xFF	ID0	ID1	ID2	ID3	CHKSUM	0xA5

- MODUL No - sender module number
- MODULE - responder module number
- GROUP No - sender group number
- GROUP - responder group number
- 0xXX - irrelevant data can be any value
- HARD1 HARD2 - 0x3000 – universal processor or module UNIV
- 0x00 - means all processors in group
- HVER - 0x03 – processor or module version
- GROUP - group number of being asked node
- 0x00 - means all group
- IDx - serial number

0x104 – *hardware type request to node*

It is a request for the hardware type. In response, the processor sends the device type, version and serial number.

Sent to CAN ⇨	0xAA	0x104	0x0	MODUL No	GROUP No	0xXX	0xXX	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x104	0x1	MODULE	GROUP	HARD1	HARD2	HVER	0xFF	ID0	ID1	ID2	ID3	CHKSUM	0xA5

- MODUL No - sender module number
- MODULE - responder module number
- GROUP No - sender group number
- GROUP - responder group number
- 0xXX - irrelevant data can be any value
- HARD1 HARD2 - 0x3000 – universal processor or module UNIV
- MODULE - module number of being asked node
- HVER - 0x03 – processor or module version
- GROUP - group number of being asked node
- IDx - serial number

0x105 – *firmware type request to group*

This is a request for the firmware type. In response, processors send uploaded firmware data. If there is no firmware or it is incorrect, then error frames are sent.

request to all nodes in one group

Sent to CAN ⇨	0xAA	0x105	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x105	0x1	MODULE	GROUP	HARD1	HARD2	HVER	ATYPE	AVERS	FVERS	BVER1	BREV2	CHKSUM	0xA5

request to all groups

Sent to CAN ⇨	0xAA	0x105	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	0x00	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x105	0x1	MODULE	GROUP	HARD1	HARD2	HVER	ATYPE	AVERS	FVERS	BVER1	BREV2	CHKSUM	0xA5

MODUL No - sender module number
MODULE - responder module number
GROUP No - sender group number
GROUP - responder group number
0xXX - irrelevant data can be any value
HARD1 HARD2 - 0x3000 – firmware for universal processor or module UNIV
0x00 - means all processors in group
HVER - 0x03 – firmware for processor or module version
GROUP - group number of being asked node
0x00 - means all group
ATYPE - application (hardware) type
0x01 – button
0x02 – relay
0x03 – infrared receiver
0x04 – temperature sensor
0x05 – infrared transmitter
0x06 – dimmer
0x07 – blind controller
0x08 – LED controller
AVERS - application (hardware) version
FVERS - firmware version
BVER1 BREV2 - bootloader version
BVER1.BREV2

0x106 – *firmware type request to node*

This is a request for the firmware type. In response, the processor sends uploaded firmware data. If there is no firmware or it is incorrect, the error frame is sent.

Sent to CAN ⇨	0xAA	0x106	0x0	MODUL No	GROUP No	0xXX	0xXX	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x106	0x1	MODULE	GROUP	HARD1	HARD2	HVER	ATYPE	AVERS	FVERS	BVER1	BREV2	CHKSUM	0xA5

MODUL No - sender module number
MODULE - responder module number
GROUP No - sender group number
GROUP - responder group number
0xXX - irrelevant data can be of any value
HARD1 HARD2 - 0x3000 – firmware for universal processor or module UNIV
MODULE - module number of being asked node
HVER - 0x03 – firmware for processor or module version
GROUP - group number of being asked node
ATYPE - application (hardware) type
0x01 – button
0x02 – relay
0x03 – infrared receiver
0x04 – temperature sensor
0x05 – infrared transmitter
0x06 – dimmer
0x07 – blind controller
0x08 – LED controller
AVERS - application (hardware) version
FVERS - firmware version
BVER1 BREV2 - bootloader version
BVER1.BREV2

0x1F1 – *incorrect firmware*

If the uploaded firmware is incorrect, the processor sends the following frame.

Response ⇦	0xAA	0x1F1	0x1	MODULE	GROUP	FIRFLAGS	FSUM2	FSUM1	FSUM0	0xFF	0xFF	BVER1	BVER2	CHKSUM	0xA5

MODULE - responder module number
GROUP - responder group number
FIRFLAGS – error code
bit <0> - firmware error
bit <1> - bit is not used
bit <2> - incorrect firmware checksum
bit <3> - firmware is not written for UNIV 3 CPU processor (hardware mismatch)
bit <4:7> - bits are not used
FSUM2 FSUM1 FSUM0 - expected by bootloader firmware checksum
BVER1 BREV2 - bootloader version
BVER1.BREV2

0x107 – *set default node and group numbers request to node*

It is a request for changing module and group number of the node (module ID on the bus). Module and group numbers will be converted to the default (respectively to the ID2 byte and ID3 byte of the serial number). This feature is useful when two or more modules on the network are given the same ID. This feature allows distinguishing them. In response, the processor sends a frame with the current number identifying the module on the network.

Sent to CAN ⇨	0xAA	0x107	0x0	MODUL No	GROUP No	0xXX	0xXX	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x107	0x1	ID2	ID3	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

MODUL No - sender module number
ID2 - responder current module number (after the change)
GROUP No - sender group number(after the change)
ID3 - responder current group number (after the change)
0xXX - irrelevant data can be any value
MODULE - module number (before the change) of being asked node
GROUP - group number (before the change) of being asked node

0x10B – *supply voltage request to group*

It is a request for the power supply voltage values. In response, processors send voltages read on the bus and on the processor itself.

request to all nodes in one group

Sent to CAN ⇨	0xAA	0x10B	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x10B	0x1	MODULE	GROUP	VOLBUS1	VOLBUS2	VOLCPU1	VOLCPU2	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

request to all groups

Sent to CAN ⇨	0xAA	0x10B	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	0x00	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x10B	0x1	MODULE	GROUP	VOLBUS1	VOLBUS2	VOLCPU1	VOLCPU2	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

- MODUL No - sender module number
- MODULE - responder module number
- GROUP No - sender group number
- GROUP - responder group number
- 0xXX - irrelevant data can be any value
- VOLBUS1 VOLBUS2 - HAPACN bus voltage U_{BUS}=(VOLCPU1*256+VOLCPU2)*30,5/65472
- 0x00 - means all processors in group
- VOLCPU1 VOLCPU2 - processor core voltage U_{CPU}=(VOLCPU1*256+VOLCPU2)*5/65472
- GROUP - group number of being asked node
- 0x00 - means all group

0x10C – *supply voltage request to node*

It is a request for the supply voltage. In response, the processor sends a HAPCAN bus and processor supply voltage.

Sent to CAN ⇨	0xAA	0x10C	0x0	MODUL No	GROUP No	0xXX	0xXX	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x10C	0x1	MODULE	GROUP	VOLBUS1	VOLBUS2	VOLCPU1	VOLCPU2	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

- MODUL No - sender module number
- MODULE - responder module number
- GROUP No - sender group number
- GROUP - responder group number
- 0xXX - irrelevant data can be any value
- VOLBUS1 VOLBUS2 - HAPACN bus voltage U_{BUS}=(VOLCPU1*256+VOLCPU2)*30,5/65472
- MODULE - module number of being asked node
- VOLCPU1 VOLCPU2 - processor core voltage UCPU=(VOLCPU1*256+VOLCPU2)*5/65472
- GROUP - group number of being asked node

0x10D – *description request to group*

It is a request for user-defined 16-character description of the processor. In response, processors send two frames, in each 8 characters of description.

request to all nodes in one group

Sent to CAN ⇨	0xAA	0x10D	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x10D	0x1	MODULE	GROUP	abc0	abc1	abc2	abc3	abc4	abc5	abc6	abc7	CHKSUM	0xA5
	0xAA	0x10D	0x1	MODULE	GROUP	abc8	abc9	abc10	abc11	abc12	abc13	abc14	abc15	CHKSUM	0xA5

request to all groups

Sent to CAN ⇨	0xAA	0x10D	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	0x00	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x10D	0x1	MODULE	GROUP	abc0	abc1	abc2	abc3	abc4	abc5	abc6	abc7	CHKSUM	0xA5
	0xAA	0x10D	0x1	MODULE	GROUP	abc8	abc9	abc10	abc11	abc12	abc13	abc14	abc15	CHKSUM	0xA5

- MODUL No - sender module number
- MODULE - responder module number
- GROUP No - sender group number
- GROUP - responder group number
- 0xXX - irrelevant data can be any value
- abcx - 16 character processor description
- 0x00 - means all processors in group
- GROUP - group number of being asked node
- 0x00 - means all group

0x10E – *description request to node*

It is a request for user-defined 16-character description of the processor. In response, the processor will send two frames, in each 8 characters of description.

Sent to CAN ⇨	0xAA	0x10E	0x0	NR MOD	NR GRUP	0xXX	0xXX	MODUŁ	GRUPA	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x10E	0x1	MODUŁ	GRUPA	abc0	abc1	abc2	abc3	abc4	abc5	abc6	abc7	CHKSUM	0xA5
	0xAA	0x10E	0x1	MODUŁ	GRUPA	abc8	abc9	abc10	abc11	abc12	abc13	abc14	abc15	CHKSUM	0xA5

- MODUL No - sender module number
- MODULE - responder module number
- GROUP No - sender group number
- GROUP - responder group number
- 0xXX - irrelevant data can be any value
- abcx - 16 character processor description
- MODULE - module number of being asked node
- GROUP - group number of being asked node

0x10F – *DEV ID request to group*

It is a request for the processors identification numbers written originally by Microchip. In response, processors send the data, which includes: CPU types and revision number. More information about DEV ID's documentation Microchip PIC18F26K80 processor.

request to all nodes in one group

Sent to CAN ⇨	0xAA	0x10F	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x10F	0x1	MODULE	GROUP	DEVID1	DEVID2	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

request to all groups

Sent to CAN ⇨	0xAA	0x10F	0x0	MODUL No	GROUP No	0xXX	0xXX	0x00	0x00	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x10F	0x1	MODULE	GROUP	DEVID1	DEVID2	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

- MODUL No - sender module number
- MODULE - responder module number
- GROUP No - sender group number
- GROUP - responder group number
- 0xXX - irrelevant data can be any value
- DEV ID1, DEV ID2 - identification number written originally by Microchip
- 0x00 - means all processors in group
- GROUP - group number of being asked node
- 0x00 - means all group

0x111 – *DEV ID request to node*

It is a request for the processor identification number written originally by Microchip. In response, the processor sends the data, which includes: CPU type and revision number. More information about DEV ID's documentation Microchip PIC18F26K80 processor.

Sent to CAN ⇨	0xAA	0x111	0x0	MODUL No	GROUP No	0xXX	0xXX	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x111	0x1	MODULE	GROUP	DEVID1	DEVID2	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

- MODUL No - sender module number
- MODULE - responder module number
- GROUP No - sender group number
- GROUP - responder group number
- 0xXX - irrelevant data can be any value
- DEV ID1, DEV ID2 - identification number written originally by Microchip
- MODULE - module number of being asked node
- GROUP - group number of being asked node

5.5. UART Programming mode messages

These messages allow making changes in program memory (flash memory) and data memory (FLASH and EEPROM) via the processor UART serial port (RS232). Therefore they allow uploading firmware and configuration to the processor via UART.

0x020 - exit one node from bootloader programming mode	Messages handled by the **bootloader in programming mode**
0x030 - address frame	
0x040 - data frame	

Table 7. List of HAPCAN system messages used to communicate via UART with bootloader in programming mode

0x020 – *exit one node from bootloader programming mode*

It is a request for exiting from the programming mode and returning to the normal operation. After exiting the programming mode, the processor restarts. The processor does not send any response.

Sent to UART ⇨	0xAA	0x020	0x0	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	NONE												

- 0xXX - irrelevant data can be of any value

0x030 – *address frame*

This message contains the memory address to be read or modified. The address must be a multiple of 8 (for FLASH write command), or a multiple of 64 (for erasing FLASH memory). Read and write command reads or writes 8 consecutive memory locations starting at the address specified in bytes ADRU:ADRH:ADRL. When writing to FLASH memory, it must be erased first. The erase command deletes the 64 consecutive memory cells starting from the address specified in bytes ADRU:ADRH:ADRL.

Sent to UART ⇨	0xAA	0x030	0x0	ADRU	ADRH	ADRL	0xXX	0xXX	CMD	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x030	0x1	echo	echo	echo	echo	echo	echo	echo	echo	CHKSUM	0xA5

- ADRU, ADRH, ADRL - memory address must be a multiple of 8 for writing and 64 for erasing FLASH memory
- echo - byte identical to the transmitted
- CMD - command read/write/erase
  - 0x01 – read EEPROM or FLASH memory
  - 0x02 – write EEPROM or FLASH memory
  - 0x03 – erase FLASH memory

0x040 – *data frame*
This message contains data that is to be saved to the memory addressed in the address frame. If command in the address frame was read or erase, sending data frame will make reading or erasing memory.

if CMD=0x01 in address frame (read EEPROM or FLASH memory)

Sent to UART ⇨	0xAA	0x040	0x0	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x040	0x1	DATA R0	DATA R1	DATA R2	DATA R3	DATA R4	DATA R5	DATA R6	DATA R7	CHKSUM	0xA5

0xXX	- irrelevant data can be any value
DATA Rx	- received read bytes

if CMD=0x02 in address frame (write EEPROM or FLASH memory)

Sent to UART ⇨	0xAA	0x040	0x0	DATA W0	DATA W1	DATA W2	DATA W3	DATA W4	DATA W5	DATA W6	DATA W7	CHKSUM	0xA5
Response ⇦	0xAA	0x040	0x1	DATA R0	DATA R1	DATA R2	DATA R3	DATA R4	DATA R5	DATA R6	DATA R7	CHKSUM	0xA5

DATA Wx	- bytes to be written into memory
DATA Rx	- received bytes confirming correctness of writing (should be: DATA Rx = DATA Wx)

if CMD=0x03 in address frame (erase FLASH memory)

Sent to UART ⇨	0xAA	0x040	0x0	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x040	0x1	DATA R0	DATA R1	DATA R2	DATA R3	DATA R4	DATA R5	DATA R6	DATA R7	CHKSUM	0xA5

0xXX	- irrelevant data can be any value
DATA Rx	- 8 first bytes of erased 64 byte block (should be equal 0xFF)

0x0F0 – *error frame*
Error frame is sent by the bootloader if address or command in address frame was incorrect.

Response ⇦	0xAA	0x0F0	0x1	0xFF	0xFF	BVER1	BVER2	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5

BVER1	BVER2	- bootloader version BVER1.BREV2

5.6. CAN programming messages

These messages allow making changes in program memory (flash memory) and data memory (FLASH and EEPROM) via the HAPCAN bus. Therefore they allow uploading firmware and configuration to the processor via HAPCAN bus.

0x010 – exit all from bootloader programming mode	Messages handled by the **bootloader in programming mode**
0x020 - exit one node from bootloader programming mode	
0x030 - address frame	
0x040 - data frame	

Table 8. List of system messages used to communicate via HAPCAN bus with bootloader in programming mode.

0x010 – *exit all from bootloader programming mode*
It is a request for exiting from the programming mode and returning to the normal operation for all processors on the bus. After exiting the programming mode, processors will restart. Processors do not send any response.

Sent to CAN ⇨	0xAA	0x010	0x0	0x00	0x00	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	NONE														

0x00	0x00	- means request directed to all processors
	0xXX	- irrelevant data can be any value

0x020 – *exit one node from bootloader programming mode*
It is a request for exiting from the programming mode and returning to the normal operation. After exiting the programming mode, the processor restarts. The processor does not send any response.

Sent to CAN ⇨	0xAA	0x020	0x0	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	NONE														

MODULE		- module number of being programmed node
	GROUP	- group number of being programmed node
	0xXX	- irrelevant data can be any value

0x030 – *address frame*
This message contains the memory address to be read or modified. The address must be a multiple of 8 (for FLASH write command), or a multiple of 64 (for erasing FLASH memory). Read and write command reads or writes 8 consecutive memory locations starting at the address specified in bytes ADRU:ADRH:ADRL. When writing to FLASH memory, it must be erased first. The erase command deletes the 64 consecutive memory cells starting from the address specified in bytes ADRU:ADRH:ADRL.

Sent to CAN ⇨	0xAA	0x030	0x0	MODULE	GROUP	ADRU	ADRH	ADRL	0xXX	0xXX	CMD	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x030	0x1	MODULE	GROUP	echo	echo	echo	echo	echo	echo	echo	echo	CHKSUM	0xA5

MODULE - module number of being programmed node
GROUP - group number of being programmed node
ADRU ADRH ADRL - memory address must be a multiple of 8 for writing and 64 for erasing FLASH memory
echo - byte identical to the transmitted
CMD - command read/write/erase
0x01 – read EEPROM or FLASH memory
0x02 – write EEPROM or FLASH memory
0x03 – erase FLASH memory

0x040 – *data frame*

This message contains data that is to be saved to the memory addressed in the address frame. If command in the address frame was read or erase, sending data frame will make reading or erasing memory.

if CMD=0x01 in address frame (read EEPROM or FLASH memory)

Sent to CAN ⇨	0xAA	0x040	0x0	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x040	0x1	MODULE	GROUP	DATA R0	DATA R1	DATA R2	DATA R3	DATA R4	DATA R5	DATA R6	DATA R7	CHKSUM	0xA5

MODULE - module number of being programmed node
GROUP - group number of being programmed node
0xXX - irrelevant data can be any value
DANE Rx - received read bytes

if CMD=0x02 in address frame (write EEPROM or FLASH memory)

Sent to CAN ⇨	0xAA	0x040	0x0	MODULE	GROUP	DATA T0	DATA T1	DATA T2	DATA T3	DATA T4	DATA T5	DATA T6	DATA T7	CHKSUM	0xA5
Response ⇦	0xAA	0x040	0x1	MODULE	GROUP	DATA R0	DATA R1	DATA R2	DATA R3	DATA R4	DATA R5	DATA R6	DATA R7	CHKSUM	0xA5

MODULE - module number of being programmed node
GROUP - group number of being programmed node
DATA Tx - bytes to be written into memory
DATA Rx - received bytes confirming correctness of writing (should be: DATA Rx = DATA Tx)

if CMD=0x03 in address frame (erase FLASH memory)

Sent to CAN ⇨	0xAA	0x040	0x0	MODULE	GROUP	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	CHKSUM	0xA5
Response ⇦	0xAA	0x040	0x1	MODULE	GROUP	DATA R0	DATA R1	DATA R2	DATA R3	DATA R4	DATA R5	DATA R6	DATA R7	CHKSUM	0xA5

MODULE - module number of being programmed node
GROUP - group number of being programmed node
0xXX - irrelevant data can be any value
DATA Rx - 8 first bytes of erased 64 byte block (should be equal 0xFF)

0x0F0 – *error frame*

Error frame is sent by the bootloader if address or command in address frame was incorrect.

Response ⇦	0xAA	0x0F0	0x1	MODULE	GROUP	0xFF	0xFF	BVER1	BVER2	0xFF	0xFF	0xFF	0xFF	CHKSUM	0xA5
	NONE														

MODULE - module number of being programmed node
GROUP - group number of being programmed node
BVER1 BVER2 - bootloader version BVER1.BREV2

5.7. Programming algorithm

Read, write, and erase of EEPROM and FLASH memory in the programming mode, requires the use of a sequence of commands (Figure 13). The whole process of read/write/erase of EEPROM and FLASH uses two types of frames: ADDRESS FRAME (0x030) and DATA FRAME (0x040). The address frame must contain correct address of the memory cell and command: read, write or erase. Data frame activates the read or erase memory or should contain the data to be stored in memory when read command is chosen.

Beginning of programming process

To start a reading, writing or erasing memory, the processor must be in programming mode. Bootloader enters the programming mode when receives ENTER PROGRAMMING MODE frame (0x100).

Programming program and data FLASH memory

To program FLASH memory, the address between 0x001000 - 0x00FFF8 must be chosen. The address must be a valid value that is to be a multiple of 8 (for read and write commands) or a multiple of 64 (for the erase command). The address frame must also contain command 0x01 (to read memory), 0x02 (to write memory), or 0x03 (to erase memory). Writing to flash memory should be preceded by erasing block of memory.

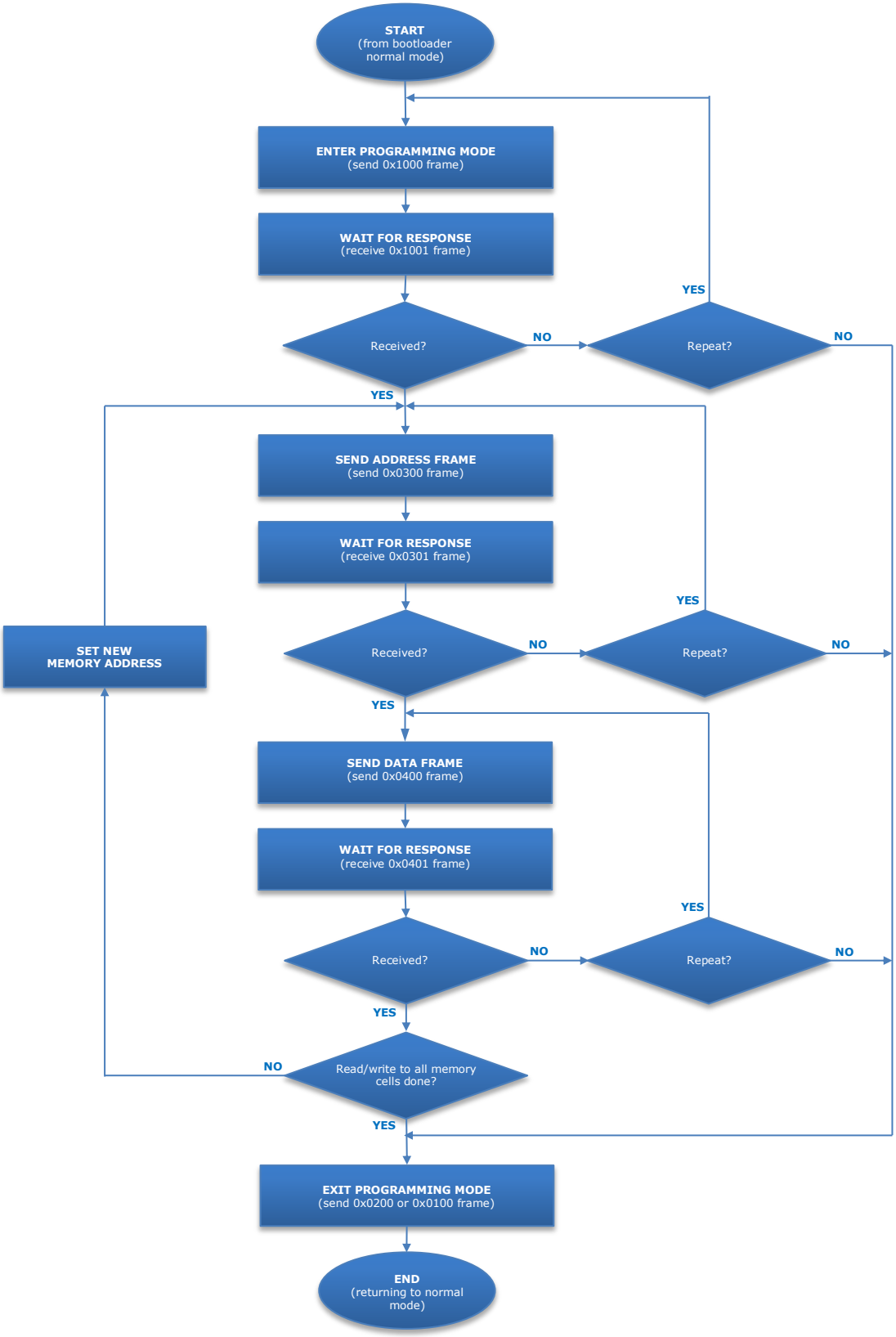


Figure 13. Programming flow chart

Programming data EEPROM memory
To program EEPROM memory, the address between 0xF00000 - 0xF003F8 must be chosen. The address frame must also contain command 0x01 (to read memory), 0x02 (to write memory). EEPROM programming does not support erase command. To erase EEPROM, the write command should be used with data values equal "0xFF".

Completion of programming process
Bootloader exits programming mode and goes to normal operation after receiving 0x010 or 0x020 frames.

6. Processor memory

The processor has a 64kB FLASH program memory, 1kB EEPROM data memory and 3.6 kB of RAM. Bootloader reserves part of the memory to work properly. Memory reserved for the bootloader must not be used by the functional firmware, except as described below.

6.1. EEPROM memory

Bootloader uses several EEPROM data memory cells. Flag BOOTFL = 0xFF enables entering bootloader programming mode after rebooting the processor. The remaining bytes are module configuration. These are the module and group numbers and 16-character description. These memory cells can be modified by the functional firmware.

0 1 2 3 4 5 6 7 8 9 A B C D E F		
0xF00000 0xF000F0	256 EEPROM data memory bytes	0xF0000F 0xF000FF
0xF00100 0xF001F0	256 EEPROM data memory bytes	0xF0010F 0xF001FF
0xF00200 0xF002F0	256 EEPROM data memory bytes	0xF0020F 0xF002FF
0xF00300 0xF003F0	256 EEPROM data memory bytes	0xF0030F 0xF003FF

Table 9. EEPROM memory of UNIV 3 CPU processor

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0xF00000	BOOTFL																0xF0000F
0xF00010																	0xF0001F
0xF00020							MODULE	GROUP									0xF0002F
0xF00030	DESCR 0	DESCR 1	DESCR 2	DESCR 3	DESCR 4	DESCR 5	DESCR 6	DESCR 7	DESCR 8	DESCR 9	DESCR 10	DESCR 11	DESCR 12	DESCR 13	DESCR 14	DESCR 15	0xF0003F
0xF00040																	0xF0004F
0xF00050																	0xF0005F
0xF00060																	0xF0006F
0xF00070																	0xF0007F
0xF00080																	0xF0008F
0xF00090																	0xF0009F
0xF000A0																	0xF000AF
0xF000B0																	0xF000BF
0xF000C0																	0xF000CF
0xF000D0																	0xF000DF
0xF000E0																	0xF000EF
0xF000F0																	0xF000FF

BOOTFL - Bootloader flag. If BOOTFL = 0x00 after rebooting bootloader will be in normal mode, if BOOTFL = 0xFF bootloader will enter programming mode.

MODULE - Module number of node

GROUP - Group number of node

DESCR x - 16 character processor description

Table 10. EEPROM memory cells used by bootloader

6.2. FLASH memory

Bootloader code is placed at the beginning of FLASH program memory. It occupies an area of 4kB. In addition, the bootloader uses the processor configuration bytes. Access to these areas is disabled. The area marked as "28 kB of functional firmware program memory" (Table 12) is used to calculate the firmware checksum and therefore must not contain any data that is changed when the firmware is working, but only the code of the program. The variable data can be placed in the area marked "32 kB of functional firmware program and data memory".

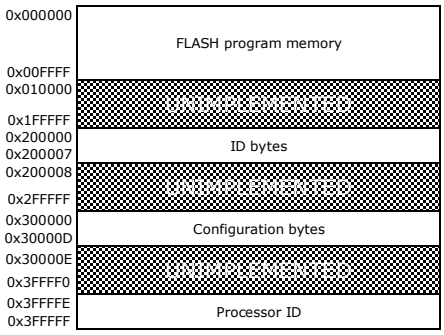


Table 11. FLASH memory of UNIV 3 CPU processor

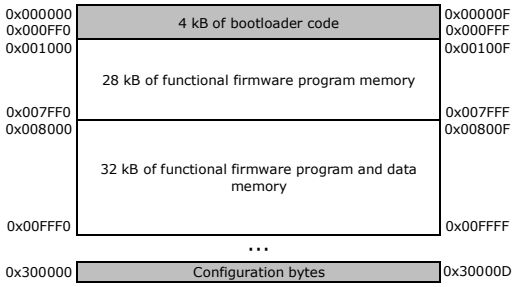


Table 12. FLASH memory cells used by bootloader

6.3. RAM memory

Bootloader uses most of the bank 1 of UNIV 3 CPU processor RAM memory. Bytes used by bootloader (Table 14) can be read and possibly cleared by functional firmware. Other areas of bootloader RAM memory must not be changed, as it can cause dysfunctionality, and even damage to the module.

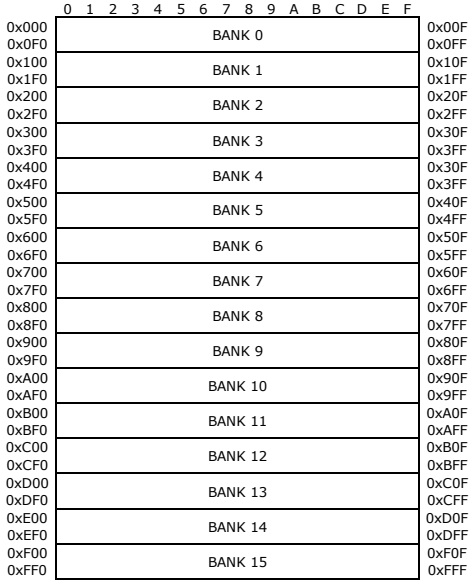


Table 13. RAM memory of UNIV 3 CPU processor

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0x100	RxBCON	CANFRAME1	CANFRAME2	CANNODE	CANGROUP	CANDLC	CAND0	CAND1	CAND2	CAND3	CAND4	CAND5	CAND6	CAND7	CANFULL	FIRMFLAG	0x10F
0x110	UART0	UART1	UART2	UART3	UART4	UART5	UART6	UART7	UART8	UART9	UART10	UART11	UART12	UART13	UART14	UART15	0x11F
0x120	UART16	UART17	UART18	UART19	UART20	UART21	UART22	UART23	UART24	UART25	UART26	UART27	UART28	UART29	UARTOVF	UARTCNT	0x12F
0x130																	0x13F
0x140																	0x14F
0x150																	0x15F
0x160																UARTON	0x16F
0x170	STATUS_H	WREG_H	BSR_H	PIR1_H	PIR4_H	PIR5_H	CANSTAT_H	CANCON_H	FSR0L_H	FSR0H_H	FSR1L_H	FSR1H_H	TABLAT_H	TBLPTRL_H	TBLPTRH_H	TBLPTRU_H	0x17F
0x180	INTCON_H	EECON1_H	EEDATA_H	EEADRH_H	EEADR_H	TRISA_H	ADCON2_H	ADCON1_H	ADCON0_H	ANCON0_H	ADRESL_H	ADRESH_H	RCSTA1_H	TXSTA1_H	PMD1_H	COMSTAT_H	0x18F
0x190	STATUS_L	WREG_L	BSR_L	PIR1_L	PIR4_L	PIR5_L	CANSTAT_L	CANCON_L	FSR0L_L	FSR0H_L	FSR1L_L	FSR1H_L	TABLAT_L	TBLPTRL_L	TBLPTRH_L	TBLPTRU_L	0x19F
0x1A0	INTCON_L	EECON1_L	EEDATA_L	EEADRH_L	EEADR_L	TRISA_L	ADCON2_L	ADCON1_L	ADCON0_L	ANCON0_L	ADRESL_L	ADRESH_L	RCSTA1_L	TXSTA1_L	PMD1_L	COMSTAT_L	0x1AF
0x1B0																	0x1BF
0x1C0																	0x1CF
0x1D0																	0x1DF
0x1E0																	0x1EF
0x1F0																	0x1FF

RxBCON - CAND7 - CAN receive buffer

CANFULL - Value 0xFF indicates that new message received from CAN bus is saved in CAN receive buffer

FIRMFLAG - Flag is cleared by bootloader at rebooting. Set in the functional firmware, signals the end of initialization and willingness to receive CAN or UART messages

UART0 - UART29 - UART receive buffer

UARTOVF - Value other than 0x00 indicates UART buffer overflow

UARTCNT - Gives number of bytes received by UART serial port

UARTON - Value 0xFF indicates that UART bootloader is active

xxxx_H - Shadow registers saved when high priority interrupt occurs

xxxx_L - Shadow registers saved when low priority interrupt occurs

Table 14. RAM memory cells used by bootloader

6.4. Configuration bytes

Configuration bytes are essential ration of the processor. They are in the area of 0x300000 - 0x30000D of FLASH memory. Changing their values is not possible. Detailed description of these bytes is in the documentation of PIC18F26K80 processor.

	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value	
0x300000	CONFIG1L	-	XINST	-	SOSCSEL1	SOSCSEL0	INTOSCSEL	-	$\overline{\text{RETEN}}$	15h	-0-1 01-1
0x300001	CONFIG1H	IESO	FCMEN	-	PLLCFG	FOSC3	FOSC2	FOSC1	FOSC0	03h	00-0 0011
0x300002	CONFIG2L	-	BORPWR1	BORWPR0	BORV1	BORV0	BOREN1	BOREN0	$\overline{\text{PWRTEN}}$	2Ah	-010 1010
0x300003	CONFIG2H	-	WDTPS4	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN1	WDTEN0	2Eh	-010 1110
0x300005	CONFIG3H	MCLRE	-	-	-	MSSPMSK	-	-	CANMX	89h	1--- 1--1
0x300006	CONFIG4L	$\overline{\text{DEBUG}}$	-	-	BBSIZ0	-	-	-	STVREN	91h	1--1 ---1
0x300008	CONFIG5L	-	-	-	-	CP3	CP2	CP1	CP0	00h	---- 0000
0x300009	CONFIG5H	CPD	CPB	-	-	-	-	-	-	00h	00-- ----
0x30000A	CONFIG6L	-	-	-	-	WRT3	WRT2	WRT1	WRT0	0Fh	---- 1111
0x30000B	CONFIG6H	WRTD	WRTB	WRTC	-	-	-	-	-	80h	100- ----
0x30000C	CONFIG7L	-	-	-	-	EBTR3	EBTR2	EBTR1	EBTR0	0Fh	---- 1111
0x30000D	CONFIG7H	-	EBTRB	-	-	-	-	-	-	00h	-0-- ----

Table 15. Configuration bytes values of UNIV 3 CPU processor

7. Functional firmware

Device built on the UNIV 3 CPU chip requires firmware which implements the functionality of the device. Functional firmwares for the UNIV 3 CPU processor are available at HAPCAN Project website hapcan.com.

7.1. Own functional firmware

You can adjust the functionality of the firmware to suit your needs by modifying the finished code or create your own firmware from scratch.
To create or modify a code, you need:

1. Microchip MPLAB software to write and compile code. MPLAB program is available free of charge at microchip.com.
2. Detailed information about PIC18F26K80 processor, which was used to create UNIV 3 CPU. These are also available on the website microchip.com.
3. Converter of created in MPLAB .hex file to .haf file - hapcan automation firmware file. The converter is available at hapcan.com.

4. HAPCAN Programmer – Windows software to upload the new firmware to the processor. HAPCAN Programmer is available at hapcan.com.

7.2. Functional firmware program memory

The functional firmware can cover an area of FLASH memory at address 0x001000 to 0x00FFFF (Table 16). Between addresses 0x001000 - 0x007FFF the firmware must not store any variables, as the area is analysed to calculate the firmware checksum.

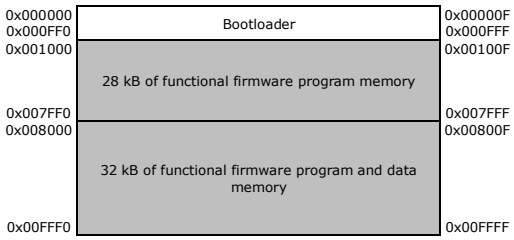
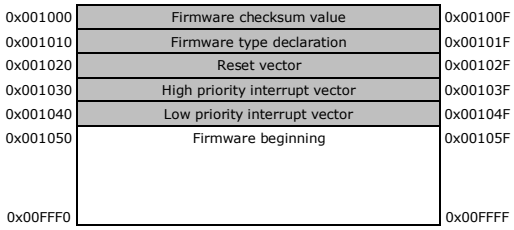


Table 16. Functional firmware program memory field

In order to ensure bootloader can properly communicate with the functional firmware, in the firmware code must be declared a certain values (checksum, the type of firmware) and used special areas (reset and interrupt vectors) (Table 17).



Firmware checksum value	– This value is calculated and staved automatically in the firmware during code conversion from MPLAB compiler .hex file to HAPCAN automation firmware .haf file.
Firmware type declaration	– type, version, revision. Detail description regarding these values can be found in section *7.6. Functional firmware template*.
Reset vector	– the position from which the program will be executed after a reset of the processor.
High priority interrupt vector	– the position from which the program will be executed when an high priority interrupt occurs.
Low priority interrupt vector	– the position from which the program will be executed when an low priority interrupt occurs.
Firmware beginning	– the position from which the functional firmware should begin

Table 17. Special areas in functional firmware

7.3. Functional firmware data memory

Firmware data can be placed in RAM, EEPROM and FLASH memory, but outside areas used by the bootloader. In addition, in the area of 0x001000 - 0x007FFF, the firmware must not store any variables, as the area is analysed to calculate the checksum firmware.

7.4. Receiving UART messages

Bootloader supports transmission via UART1 serial port using pin 17 (TX1) and pin 18 (RX1) of UNIV 3 CPU processor. Receiving messages from UART is handled by a high priority interrupt. By default, the serial port is set to support transmissions with the following parameters: baud rate 115200 bps, 8 data bits, 1 stop bit.

Port starts to receive when the first byte arrives. The byte is stored in the buffer and the port waits for the next byte. Maximum waiting time for the next byte is equal to one byte receiving time increased by 20%. For the transmission speed of 115,200 bps it is about 90us. Subsequent bytes are written to the buffer. The maximum buffer capacity is 30 bytes. Serial port ends receiving when break time after last byte is greater than 90us (for 11520bps speed). The received data is stored in the UART receive buffer (bytes from UART0 to UART29 – address 0x110-0x12D of RAM) (Table 14). Number of bytes received and stored in the buffer is indicated by the value of

register UARTCNT (address 0x12F) (Table 14). If the serial port receives more than 30 bytes, the 31st one is saved back to the beginning of the receive buffer (into byte UART0) and any such overflow is indicated by an incrementing the value of UARTOVF register (address 0x12E) (Table 14).

After receiving the message, bootloader checks if the received message is a system message (*5.3. UART system messages).* If so, it sends the response and then passes a high priority interrupt to the functional firmware (program memory address 0x001030, see Table 17). If the received message is not a system message, a high interrupt is passed immediately to the address 0x001030.

Functional firmware should check UARTCNT register and if it is different from zero, the appropriate amount of data should be read from the receive buffer (UART0-UART29). Procedure of reading data from a buffer must ends with clearing UARTCNT register.

7.5. Receiving CAN messages

Bootloader supports transmission via CAN port using pin 23 (CANTX) and pin 24 (CANRX) of UNIV 3 CPU processor. Receiving messages from the CAN port is handled by a low priority interrupt. The default port parameters are: 125kbps bit rate, format CAN 2.0B (only frames with extended 29-bit identifier), 8 data bytes - fixed number.

Properly received CAN message is written to the CAN receive buffer (registers from CANFRAME1 to CAND7 – address from 0x101 to 0x10D of RAM) (Table 14). Receiving the messages is indicated by a 0xFF value in CANFULL register (address 0x10E) (Table 14).

After receiving the bootloader checks if the received message is a system message (*5.4. CAN system messages*). If so, it sends the response and then passes a low priority interrupt to the functional firmware (program memory address 0x001040, see Table 17). If the received message is not a system, an low priority interrupt is passed immediately to the address 0x001040.

Functional firmware should check CANFULL register and if it is equal to 0xFF, can start reading the CAN receive buffer. Procedure of reading data from a buffer must ends with clearing CANFULL register.

7.6. Functional firmware template

FUNCTIONAL FIRMWARE CODE TEMPLATE

```
;==========================================================================
;   HAPCAN - Home Automation Project Firmware (http://hapcan.com)
;   Copyright (C) 2013 hapcan.com
;
;   This program is free software: you can redistribute it and/or modify
;   it under the terms of the GNU General Public License as published by
;   the Free Software Foundation, either version 3 of the License, or
;   (at your option) any later version.
;
;   This program is distributed in the hope that it will be useful,
;   but WITHOUT ANY WARRANTY; without even the implied warranty of
;   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;   GNU General Public License for more details.
;
;   You should have received a copy of the GNU General Public License
;   along with this program.  If not, see <http://www.gnu.org/licenses/>.
;==========================================================================
;   Filename:               univ3-template-rev2.asm
;   Associated diagram:     none
;   Author:                 Jacek Siwilo
;   Note:                   This is a template file for UNIV 3 CPU processor
;==========================================================================
;   Revision History
;   Rev:  Date:     Details:
;   0     01.2013   Original version
;   1     03.2013   Minor changes
;   2     09.2013   Minor changes
;==========================================================================
;===  FIRMWARE DEFINITIONS  ===============================================    ;Firmware type declaration
;==========================================================================
    #define    ATYPE    .255                    ;application type [0-255]    ;Hardware type - value 255 for user created application
    #define    AVERS    .0                     ;application version [0-255]    ;Hardware version - possible values between 0 - 255
    #define    FVERS    .0                        ;firmware version [0-255]    ;Firmware version for this hardware - possible values between 0 - 255

    #define    FREV     .0                     ;firmware revision [0-65536]    ;Firmware revision - possible values between 0 - 65536
;==========================================================================
;===  NEEDED FILES  =======================================================    ;Files included in project
;==========================================================================
    LIST P=18F26K80                          ;directive to define processor
    #include <P18F26K80.INC>          ;processor specific variable definitions
    #include "univ3-template-rev2.inc"                    ;project variables
INCLUDEDFILES    code
    #include "univ3-fake_bootloader-rev2.inc"         ;fake bootloader only for
;                                                              debugging
;==========================================================================
;===  FIRMWARE CHECKSUM  ==================================================    ;Firmware checksum is saved in code here
;==========================================================================    ;This value will be automatically counted and saved here during .hex to .haf
FIRMCHKSM   code    0x001000                                                   ;file conversion (nothing must be changed here)
    DB      0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
;==========================================================================
;===  FIRMWARE ID  ========================================================    ;Firmware declaration is saved in code here
;==========================================================================
FIRMID      code    0x001010
    DB      0x30, 0x00, 0x03,ATYPE,AVERS,FVERS,FREV>>8,FREV
;            |     |     |     |     |     |     |_____|_____ firmware revision
;            |     |     |     |     |     |_________________ firmware version
;            |     |     |     |     |_____________________ application version
;            |     |     |     |___________________________ application type
;            |     |     |_________________________________ hardware version '3'
;            |_____|_____|_______________________________ hardware type 'UNIV'
```

```
;===============================================================================
;===  MOVED VECTORS  ===========================================================    ;Vectors
;===============================================================================
;PROGRAM RESET VECTOR                                                              ;Place where program counter comes after processor restart
FIRMRESET   code    0x1020
        goto    Main
;PROGRAM HIGH PRIORITY INTERRUPT VECTOR                                            ;Place where program counter comes when high priority interrupt occurs
FIRMHIGHINT code    0x1030
        call    HighInterrupt
        retfie
;PROGRAM LOW PRIORITY INTERRUPT VECTOR                                             ;Place where program counter comes when low priority interrupt occurs
FIRMLOWINT  code    0x1040
        call    LowInterrupt
        retfie

;===============================================================================
;===  FIRMWARE STARTS  =========================================================    ;Firmware beginning
;===============================================================================
FIRMSTART   code    0x001050
;-------------------------------------------------------------------------------
;---  LOW PRIORITY INTERRUPT  --------------------------------------------------    ;Low priority interrupt procedure
;-------------------------------------------------------------------------------
LowInterrupt
        movff   STATUS,STATUS_LOW       ;save STATUS register                      ;Saving registers used in low priority interrupt
        movff   WREG,WREG_LOW           ;save working register
        movff   BSR,BSR_LOW             ;save BSR register
        movff   FSR0L,FSR0L_LOW         ;save other registers used in high int
        movff   FSR0H,FSR0H_LOW
        movff   FSR1L,FSR1L_LOW
        movff   FSR1H,FSR1H_LOW

    ;main firmware ready flag                                                      ;Checking if functional firmware has finished its initialization
        banksel FIRMREADY
        btfss   FIRMREADY,0
        bra     ExitLowInterrupt        ;main firmware is not ready yet
    ;CAN buffer                                                                    ;Checking if CAN message has been received
        banksel CANFULL
        btfsc   CANFULL,0
        call    CANInterrupt            ;proceed with CAN interrupt

ExitLowInterrupt
        movff   BSR_LOW,BSR             ;restore BSR register                      ;Restoring registers used in low priority interrupt
        movff   WREG_LOW,WREG           ;restore working register
        movff   STATUS_LOW,STATUS       ;restore STATUS register
        movff   FSR0L_LOW,FSR0L         ;restore other registers used in high int
        movff   FSR0H_LOW,FSR0H
        movff   FSR1L_LOW,FSR1L
        movff   FSR1H_LOW,FSR1H
    return

;-------------------------------------------------------------------------------
;---  HIGH PRIORITY INTERRUPT  -------------------------------------------------    ;High priority interrupt procedure
;-------------------------------------------------------------------------------
HighInterrupt
        movff   STATUS,STATUS_HIGH      ;save STATUS register                      ;Saving registers used in high priority interrupt
        movff   WREG,WREG_HIGH          ;save working register
        movff   BSR,BSR_HIGH            ;save BSR register
        movff   FSR0L,FSR0L_HIGH        ;save other registers used in high int
        movff   FSR0H,FSR0H_HIGH
        movff   FSR1L,FSR1L_HIGH
        movff   FSR1H,FSR1H_HIGH

    ;main firmware ready flag                                                      ;Checking if functional firmware has finished its initialization
        banksel FIRMREADY
        btfss   FIRMREADY,0
        bra     ExitHighInterrupt       ;main firmware is not ready yet
    ;uart                                                                         ;Checking if UART message has been received
        banksel UARTCNT
        tstfsz  UARTCNT                 ;check if UART received anything
        call    UartInterrupt           ;proceed with UART interrupt

ExitHighInterrupt
        movff   BSR_HIGH,BSR            ;restore BSR register                      ;Restoring registers used in low priority interrupt
        movff   WREG_HIGH,WREG          ;restore working register
        movff   STATUS_HIGH,STATUS      ;restore STATUS register
        movff   FSR0L_HIGH,FSR0L        ;restore other registers used in high int
        movff   FSR0H_HIGH,FSR0H
        movff   FSR1L_HIGH,FSR1L
        movff   FSR1H_HIGH,FSR1H
    return

;===============================================================================
;===  MAIN PROGRAM  ============================================================    ;Main program starts here
;===============================================================================
Main:
    ;disable interrupt for startup                                                ;Make sure interrupts will not interfere with initialization process
        bcf     INTCON,GIEH             ;disable high interrupt
        bcf     INTCON,GIEL             ;disable low interrupt
    ;firmware initialization                                                      ;Put initialization procedures here
        ;
    ;firmware started
        banksel FIRMREADY                                                        ;Indicate that initialization has finished
        bsf     FIRMREADY,0             ;set flag "firmware started and ready for interrupts"
    ;enable global interrupts
        bsf     INTCON,GIEH             ;enable high interrupt                   ;Turn on interrupts
        bsf     INTCON,GIEL             ;enable low interrupt
Loop:                                                                            ;Firmware main loop
        clrwdt                                                                   ;Clear Watchdog timer
        bra      Loop

;===============================================================================
;===  FIRMWARE ROUTINES  =======================================================
;===============================================================================
CANInterrupt                                                                     ;Procedure saves received CAN message to FIFO buffer
        ;put your code here
    return
UartInterrupt                                                                    ;Procedure saves received UART message to FIFO buffer
        ;put your code here
    return

;===============================================================================
;===  END  OF  MAIN PROGRAM  ===================================================
;===============================================================================
    END
```

7.7. Incorrect firmware

Firmware modifications may cause improper operating of the processor. For example, when functional firmware interferes with memory area restricted for the bootloader, it can make access to the bootloader impossible. In this case, disconnect the power supply of UNIV 3 CPU processor. After reconnecting the power bootloader start correctly (to allow communication with it), and after 3 seconds, proceeds to run the functional firmware. Within 3 seconds from powering processor, the bootloader can be switched to programming mode (when receives 0x100 message) and then incorrect firmware can be erased from FLASH memory.

8. Changes in bootloader versions

Bootloader version	Changes	Full compatibility with previous version
3.0	Original version.	-
3.1	Removed problem with UART receiving for other baud rates than 115200bps	-
3.2	Improved power management	√
3.3	Implemented bootloader error mode	√
3.4	CAN interrupt changes	√

9. Document version

File	Note	Date
boot_3-4a.pdf	Original version	March 2013
boot_3-4b.pdf	Added a few figures, updated firmware template, corrected description of 0x105 and 0x106 frames	September 2013