VLCB Versatille Layout Control Bus

PIC Bootloader Service Specification
Service# 10

Version 0.99B

Compatible with CBUS ® 4.0 Rev 8j

Table of Contents

Document History

Date	Changed by	Summary of changes	Service version
23rd December 2022	Ian Hogg M.5144	Initial document	1
14 April 2023	Ian Hogg M.5144	Changed name to VLCB	1

1 VLCB Services

This document describes the VLCB Bootloader service. This service is an optional addition to the VLCB Minimum Node Specification.

This service is

- BOOT for the CBUS compatible bootloader to allow VLCB firmware to be downloaded to existing CBUS modules.

It is anticipated that another Service will be available

- BOOT2 for new, transport independent protocol.

Application firmware must be able to be loaded using either protocol/service.

Modules wishing to use the existing CBUS PIC bootloader shall conform to this specification.

2 Bootloading

FCU (and other bootloading programs) and the bootloader firmware communicate using the bootloader protocol.

The end users and module applications have little control over the bootloader firmware – it is unchanged when loading module firmware using the bootloader. The current bootloader firmware will already be present on CBUS modules.

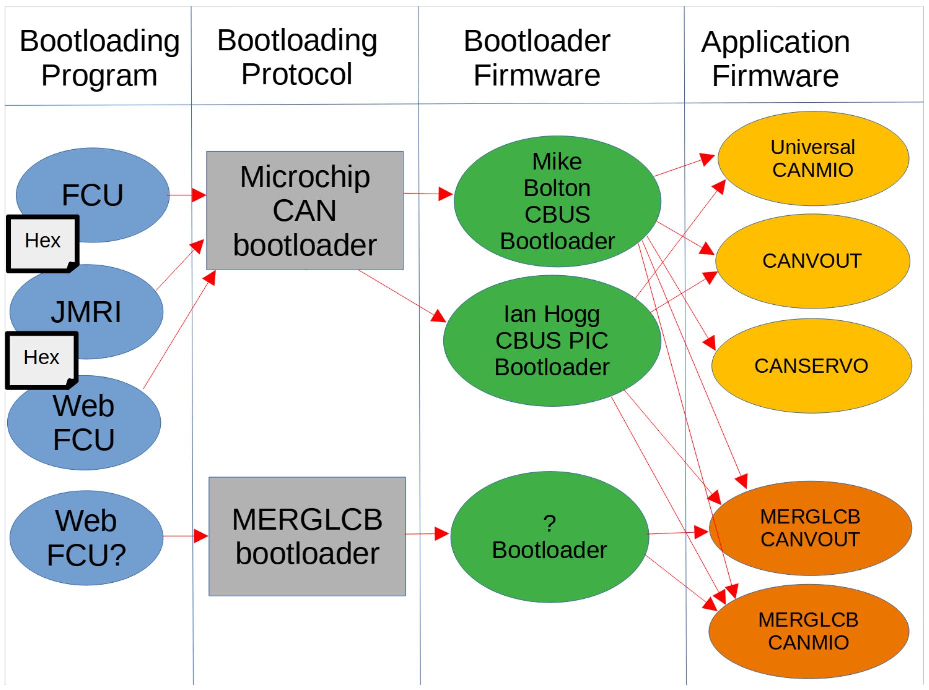
The bootloader protocol is little more than transferring an address and a set of bytes.

The PIC bootloader firmware uses the upper byte of the address to distinguish between Config, EEPROM and Flash but this is not enforced nor required by the bootloader protocol. The Bootloader code resides in the first 2K of flash. At power on the bootloader runs first and if a flag in the top byte of EEPROM is clear control is transferred to the application at its loadAddress.

If the flag is not clear then the bootloader awaits and processes bootloader messages. It will write application flash (0x800 upwards) directly along with config and EEPROM.

The bootloader itself is never overwritten with this process.

There are a number of different components at work with bootloading and each needs to work with other components. The diagram below shows the components and their interworking.



The CBUS PIC bootloader protocol is described in the MERG wiki
https://merg.org.uk/merg_wiki/doku.php?id=cbus:bootloader

2.1 Dependencies on other services

The BOOT service depends upon the mandatory Minimum Module Service.

2.2 Problems with the Bootloading process.

- Having the parameter block at the fixed address 0x820~0x838 means that it is not compatible with some processor types e.g. PIC24
- The bootloading process uses CAN extended frames outside of the CBUS spec.
- The protocol is explicitly tied to CAN.
- The protocol is limited to 24bit addressing.
- Getting the correct CONFIG bits (fuses) can be difficult if the application and bootloader use different configurations

3 Requirements for compatibility with the Microchip CAN bootloader firmware

The bootloader firmware is based on the Microchip protocol. There are a few implementations are available such as:

- C18 assembler by Mike Bolton
- XC8 C by Ian Hogg

In order for module application firmware to be compatible with the bootloader firmware it must comply with the following requirements:

- 0x0000~0x07FF reserved for bootloader

- Top byte of EEPROM for boot flag
- Code assembled for 0x800 offset so that reset vector is at 0x800, High priority interrupt vector at 0x0008 and Low priority interrupt vector at 0x0018.
- Application reset vector stored at the loadAddress at 0x082A
- Entry with the PLL disabled
- BOOTM opcode to set top byte of EEPROM to 1 and cause CPU reset
- CONFIG settings compatible with those of the bootloader

4 Additional requirements for compatibility with FCU

In order to be compatible with the FCU Bootloading program the application must comply with the following requirements:

- Be presented as a single file in Intel hex format. See https://en.wikipedia.org/wiki/Intel_HEX
- Hex file to contain the parameter block between 0x820~0x83F.

FCU attempts to check compatibility between the module and the hex file. It tries to check:

- The correct processor family
- The module to have the required FLASH memory or more1) To allow a VLCB app to be loaded using the FCU it must conform to the FCU bootloader requirements:
- Have the bootable flag set in bit 3 of parameter 8 returned by RQNPN
- Support the BOOTM opcode which sets the bootflag and then resets the processor to enter the bootloader

In order to allow FCU check for compatibility it is also recommended to have

- The CPU manufacturer set in parameter 19 and at address 0x832 of the hex file
- Have the processor set in parameter 9 and at address 0x828 of the hex file
- Return the actual CPU type using parameters 15~182) To be compatible with the existing CBUS bootloader code the VLCB application must:
- Reserve the top byte of EEPROM for the boot flag
- Have a reset vector at 0x800, high interrupt vector at 0x808, low interrupt vector at 0x818. Note the loadAddress seems to be ignored by the current bootloader but I believe it should be set to 0x800 for future bootloaders.
- The application's config bits (fuses) must be compatible with the bootloader's config bits.
- The clock frequency expected by the application must be compatible with the bootloader/hardware.

5 Loading the bootloader firmware

The bootloader firmware does not allow itself to be overwritten by the bootloading process therefore to bootstrap the bootloader firmware it must be loaded another way such as using the ICSP connector.

It is required that the single hex file for the application is bundled with a bootloader so that subsequent loading of the application firmware can be done via the bootloading process.

6 Summary of Opcodes

Refer to the VLCB Opcode Specification document for details of the opcodes.

Request to Module	**Module's Response**	**Use/meaning**
BOOTM		Enter bootloader

7 Service Specific Modes

No additional operating modes are specified by the Bootloader service.

The BOOTM opcode does transfer control away from the module's application and therefore the module can be considered to be operating in "bootloader mode".

8 Service Specific Status Codes

No additional GRSP status codes are specified by the Bootloader service.

9 Service Specific Diagnostic Data

No additional RDGN diagnostic data numbers are specified by the Bootloader service.

10 Service Specific Automatic Power-up Tests

No service specific power-up tests are specified by the Bootloader service.

11 Documentation

The module's documentation must state if it can be downloaded by the existing CBUS bootloader firmware.

The module's documentation must state if it is compiled with a BOOT compatible bootloader firmware.

12 Service data

12.1 Parameters

The following parameters are associated with the Bootloader service.

The following Parameters are required to be compatible with the bootloader:

Address	Param#	Name	Usage	VLCB should set these values
0x82A~ 0x82D	11~15	Load Address	The start address of the application code.	0x0800[1]

The following parameters are required to be compatible with FCU bootloading program:

Address	Param#	Name	Usage	VLCB should set these values
0x822	3	ModuleId	Module type identifier. See cbusdefs.h	maybe a single ID for all VLCB e.g. 0xFC.
0x828	8.3	Bootable	Indicates if FCU can use the CBUS PIC bootloading protocol	Set to 1 if the requirements of the FCU bootloading process are met.
0x828	9	Processor Id		Type of CPU, 15=PIC18F2K80. See cbusdefs
0x82E~ 0x831	15	Processor code	PICs support the ability to determine the processor type (DEVID)	Hex file contains 0x0. RQNPN to read processor code dynamically from hardware.
0x832	19	Manufacturer code		1 for microchip, 2 for Atmel, 3 for Arm
0x83A~ 0x83D		Address of module name	The module name is stored as a left justified, padded with spaces 7 character ASCII string within the	Address of module name

[1] Load Address: According to the bootloader protocol the application firmware can specify any start address however the current CBUS PIC based bootloader firmware assumes an address of 0x800.

			module address space. This is the address of the name and can be used within the hex file.	

12.2 ESD data bytes

The BOOT service does not provide any data within the ESD message, all data bytes are returned set to 0.

Appendix A - PIC Bootloader protocol

A.1 Licence

The FCU bootloader for PIC processors is based upon that of Microchip AN247. Implementations are distributed under the Microchip license which requires that it is only used on Microchip hardware.

See page 15 of http://ww1.microchip.com/downloads/en/AppNotes/00247a.pdf for details of the Microchip licensing. This bootloader is distributed with the CBUS software as a "system library" as defined in section 1 of the GNU Public license and is not licensed under GNU or Creative Commons. You must conform to Microchip license terms in respect of the bootloader.

A.2 FCU BootLoader Process

The VLCB/CBUS implementation is as follows:

1. Upon selecting a node to be reloaded the FCU will prompt for a hex file to be loaded.

2. The target processor of the hex file (Address 0x828) is compared with the module's processor type (RQNPN param 15) . A warning is generated if they are incompatible.

3. The FCU will send that node a BOOTM message.

4. The module will write 0xFF into the top most byte of EEPROM (the boot flag) and then cause a processor reset.

5. The bootloader code resides at address 0x0000 and will be executed at reset.

6. The bootloader checks the value of the boot flag and if it is 0x00 it will pass control to the module application code.

7. Otherwise the bootloader awaits receipt of extended CAN messages containing the address and data to be written to FLASH program memory plus EEPROM and CONFIG memory if requested. The HEX files will contain any EEPROM and CONFIG data by default but the user can opt to not program these in the FCU settings.

8. Upon receiving the final bootload message FCU will send a RESET command and the module will verify the data, clear the boot flag and perform a reset itself.

Note that FCU will automatically fill any unused space up to the highest specified address with 0xFF. Therefore uninitialised data must occupy space **above** the program space.

The FCU will ignore any sections of the hex file below 0x800 as this is reserved for the bootloader, which shouldn't be overwritten by itself.

Important. The PIC18F2*K80 allows the PLL to be enabled using CONFIG bits. This should **NOT** be done. The standard bootloader clears CLKSEL in CIOCON which means the CAN clock is driven by the PLL output. The CAN bit rate is calculated assuming the PLL is disabled. If the CONFIG enabled the PLL then the Bootloader's CBUS timing will be disrupted and loose communications with the FCU towards the end of the bootloading process.

A.3 Bootloader CAN Message Format

Commands: Put commands received from source (Master –> Slave) The count (DLC) can vary.

PIC18 CAN registers:

```
               |----ID (header) 29bits-----------|  d[0]  d[1]  d[2]  d[3]  d[4]  d[5]  d[6]  d[7]
               -----------  DLC  ----------------- ----- ----- ----- ----- ----- ----- ----- -----
Control frame: XXXXXXXXXXX 0 0 8 XXXXXXXX XXXXXX00 ADDRL ADDRH ADDRU RESVD CTLBT SPCMD CPDTL CPDTH
Put Data frame:XXXXXXXXXXX 0 0 8 XXXXXXXX XXXXXX01 DATA0 DATA1 DATA2 DATA3 DATA4 DATA5 DATA6 DATA7
```

Where:

ADDRL - Bits 0 to 7 of the memory pointer.

ADDRH - Bits 8 - 15 of the memory pointer.

ADDRU - Bits 16 - 23 of the memory pointer.

RESVD - Reserved for future use.

CTLBT - Control bits. See table below for definitions

SPCMD - Special command. See table below for defintions

CPDTL - Bits 0 - 7 of 2s complement checksum

CPDTH - Bits 8 - 15 of 2s complement checksum

DATA* - General data.

Header bit 0 - CD bit: Control = 0, Data = 1

Header bit 1 - PG bit: Put = 0, Get = 1

Control bits:

Bit name	Bit position	Meaning
MODE_WRT_UNLCK	0x01	Set this to allow write and erase operations to memory.
MODE_ERASE_ONLY	0x02	Set this to only erase Program Memory on a put command. Must be on 64-byte boundary.
MODE_AUTO_ERASE	0x04	Set this to automatically erase Program Memory while writing data.
MODE_AUTO_INC	0x08	Set this to automatically increment the pointer after writing.
MODE_ACK	0x10	Set this to generate an acknowledge after a 'put' (PG Mode only)

Special Commands:

Command	Value	Usage
CMD_NOP	0x00	Do nothing
CMD_RESET	0x01	Issue a soft reset after setting last EEPROM data to 0x00
CMD_RST_CHKSM	0x02	Reset the checksum and status flags
CMD_CHK_RUN	0x03	Add the checksum to special data, verify that total is zero, respond with nok/ok
CMD_BOOT_TEST	0x04	Just responds with a message frame back to verify boot mode

Responses have a single data byte:

NOK	0x00
OK	0x01
BOOT	0x02

A.4 Bootloader use of Extended Frames

An extended frame has the same number of data bytes compared to a standard frame, but it has a larger id field of 29 bits instead of 11 bits. This is implemented in a PIC by providing 2 extra header bytes, called EIDH and EIDL. An extended frame is identified by having bit 3 of the standard frame header byte SIDL set to a 1. All standard frames have this set to zero.

The bootloader code sets up a hardware filter so that it will only accept extended frames, and then only where all bits of EIDH are 0 and bits 3-7 of EIDL are 0.

It then uses EIDL bit 0 to flag whether a packet is control or data, and bit 1 to flag if it is a get or put operation (the boot loader can also be used to read data from a module, but I don't think we have implemented that anywhere for CBUS).

I can't see it referencing EIDL bit 2 anywhere, presumably reserved for future use.

So this means that extended frames can also be used for other things without conflicting with the bootloader by using extended frames where any bits in EIDH and any of bits 3-7 of EIDL are set to 1, so the bootloader code will ignore such frames.

Normal CBUS messages are carried in standard CAN frames so the code that receives these messages can safely ignore all extended frames, which, depending on processor capabilities, can either be done with a hardware filter setting or by the code checking for the extended frame bit in the CAN header.

A.5 Typical sequence

Dir	Message
->	VLCB RQNPN
->	VLCB BOOTM
->	Control, 0x000000 Boot test (AUTO_INC, AUTO_ERASE, UNLOCK)
<-	OK
->	Control, 0x000800 Reset checksum (AUTO_INC, AUTO_ERASE, UNLOCK)
->	Put Data, data bytes
->	Put Data, data bytes
->	Control, 0x300000 (AUTO_INC, AUTO_ERASE, UNLOCK)
->	Put Data, data bytes
->	Control, 0xF00000 (AUTO_INC, AUTO_ERASE, UNLOCK)
->	Put Data, data bytes
->	Control, 0x000000 check run (AUTO_INC, AUTO_ERASE, UNLOCK) checksum

A.6 Bootloader Module requirements

Since the bootloader itself is not overwritten during a load from FCU there must be compatibility between the bootloader execution and the module application. The PIC18F2XKXX series of PICs supports PLL which may be enabled either within configuration bits or under software control. To permit compatibility and interoperability when operating with a 16MHz oscillator the PLL should be DISABLED in configuration bits so that the bootloader executes at a clock speed of 16MHz. The application can enable the PLL if required so that the application executes at 64MHz.

The user program must have the following vectors.

● User code reset vector	0x0800
● User code HPINT vector	0x0808
● User code LPINT vector	0x0818

This would normally be achieved using XC8 option --codeoffset=0x800