A FLASH Bootloader for PIC16 and PIC18 Devices

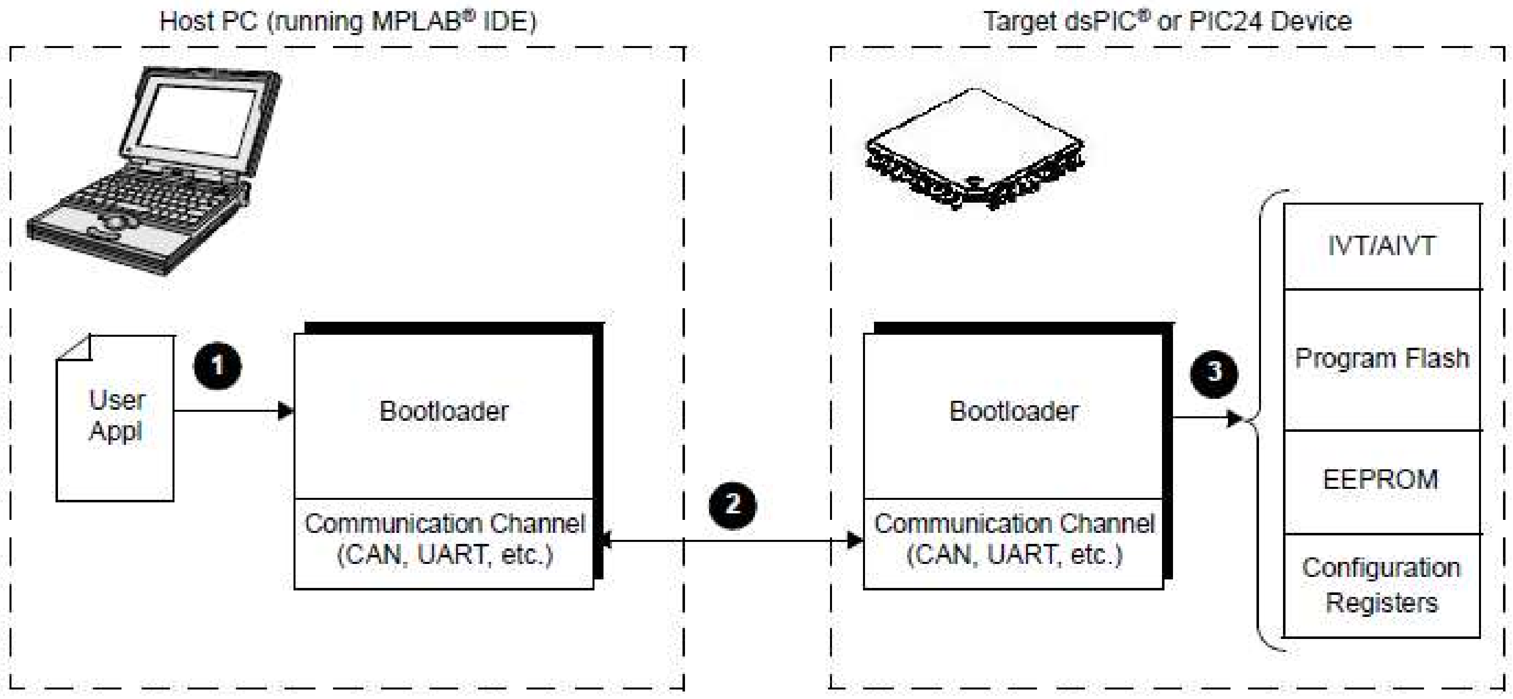
AN851

Introduction

- Among the many features built into Microchip’s Enhanced FLASH Microcontroller devices is the capability of the program memory to self-program. This very useful feature has been deliberately included to give the user the ability to perform bootloading operations.
- Devices like the PIC18F452 are designed with a designated “boot block”, a small section of protectable program memory allocated specifically for bootload firmware.
- This application note demonstrates a very powerful bootloader implementation for the PIC16F87XA and PIC18F families of microcontrollers. The coding for the two device families is slightly different; however, the functionality is essentially the same. The goals of this implementation stress a maximum performance and functionality, while requiring a minimum of code space.

Bootloader Process

1. Bootloader host program reads user application.
2. Bootloader transfers user application to target device (via communication channel).
3. Bootloader target program loads user application into target device memory.


Host PC (running MPLAB® IDE)
Target dsPIC® or PIC24 Device
User Appl
1
Bootloader
Communication Channel (CAN, UART, etc.)
2
Bootloader
Communication Channel (CAN, UART, etc.)
3
IVT/AIVT
Program Flash
EEPROM
Configuration Registers

FIRMWARE
Basic Operation

- Figure 1 summarizes the essential firmware design of the bootloader.
- Data is received through the USART module, configured in Asynchronous mode for compatibility with RS-232 and passed through the transmit/receive engine.
- The engine filters and parses the data, storing the information into a data buffer in RAM.
- The command interpreter evaluates the command information within the buffer to determine what should be done, i.e.
  - Is the data written into a memory unit?
  - Is data read from a memory unit?
  - Does the firmware version need to be read?
- Once the operation is performed, data is passed back to the transmit/receive engine to be transmitted back to the source, closing the software flow control loop.

Bootloader Functional Block Diagram

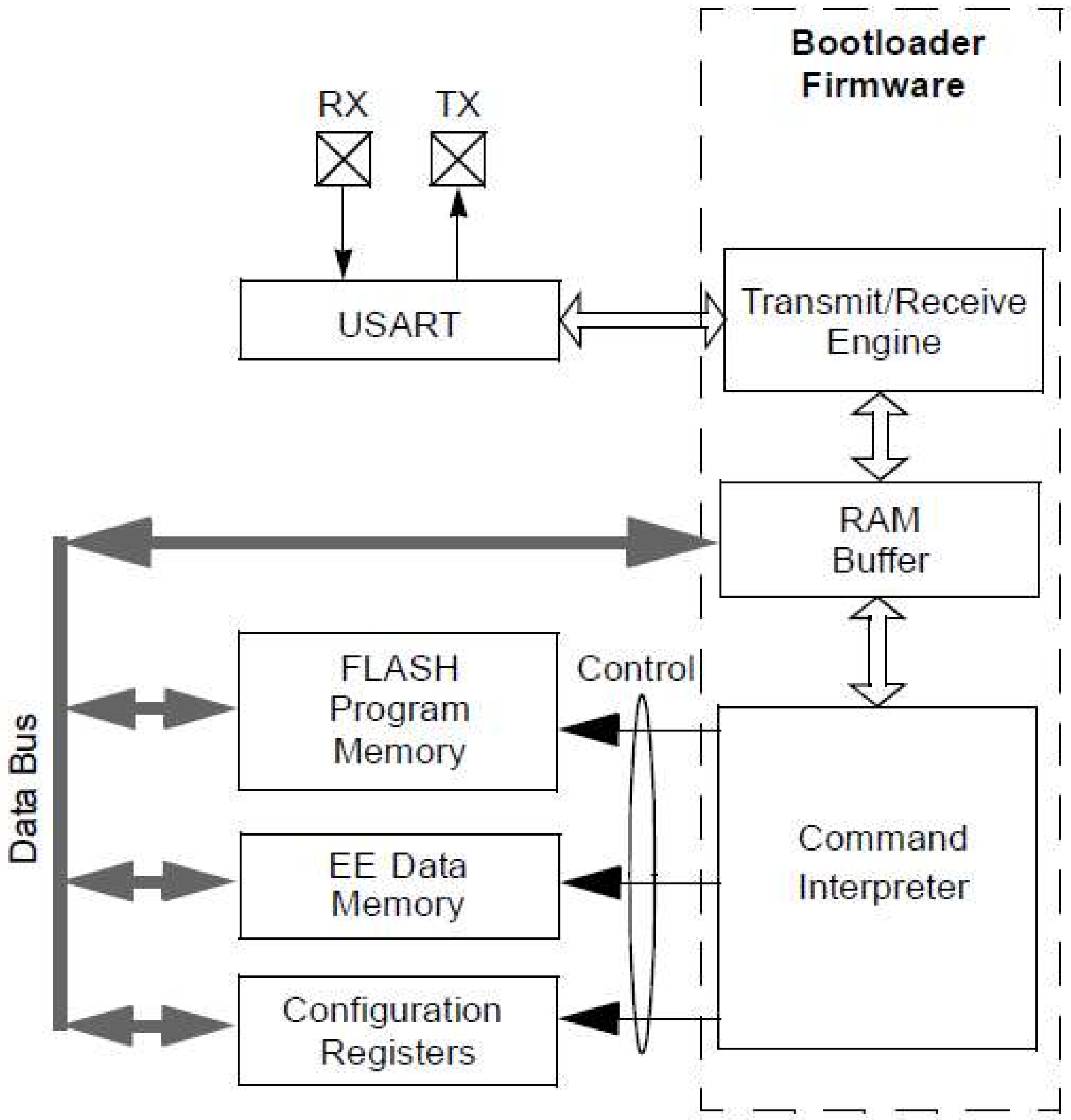


Figure 1: Bootloader Functional Block Diagram

Communications

- The microcontroller’s USART module is used to receive and transmit data; it is configured as a UART to be compatible with RS-232 communications.
  - The device can be set up in an application to bootload from a computer through its standard serial interface.
- The following communications settings are used:
  - 8 data bits
  - No parity
  - 1 STOP bit
- The baud rate setting is variable depending on the application.
  - Baud rate selection is discussed later.

The Receive/Transmit Buffer

- All data is moved through a buffer (referred to as the Receive/Transmit Buffer).
  - The buffer is a maximum of 255 bytes deep.
  - This is the maximum packet length supported by the protocol.
  - However, some devices may not support the largest packet size due to memory limitations.
- Figure 2 shows an example of the mapping of the buffer within the PIC18F452.
- A useful feature of the receive/transmit buffer is that it retains its memory between packets, thus allowing very fast repeat and replication operations.
  - That is, if an empty packet is sent, the data currently in memory will be executed as if it were just received.

Command Interpreter

- The command interpreter decodes and executes ten different commands
  - seven base commands
  - three special commands.
- The base commands allow for *read*, *write*, and *erase* operations on all types of non-volatile memory.
- The other three commands are for special operations, such as
  - repeating the last command,
  - replicating the data, and
  - resetting the device.

Command Interpreter

- Note that the PIC18F devices have greater access to, and control of, memory than PIC16F devices.
  - For example, PIC16F devices do not have access to the configuration memory, thus they do not use the configuration commands.
  - Therefore, not all instructions are available in the PIC16F bootloader.

PIC18F452 - Data Memory Usage

Note: Memory areas not shown to scale

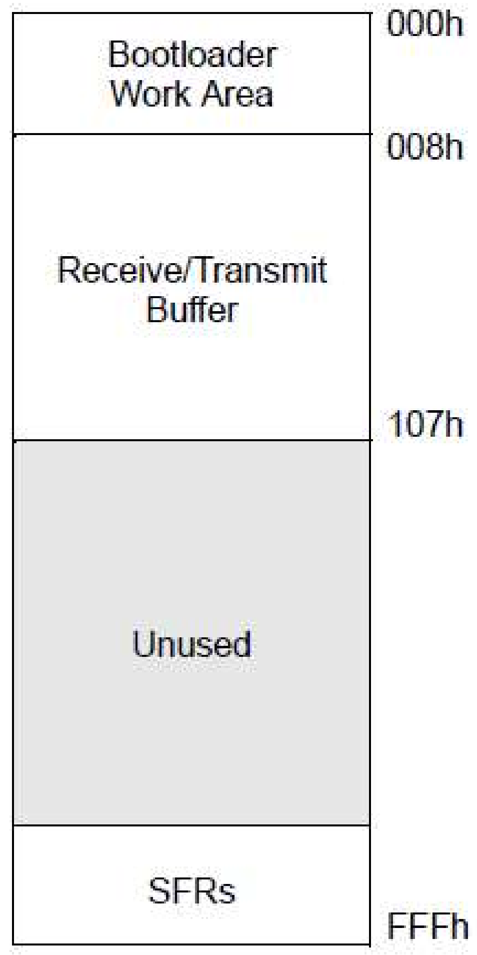


Figure 2: Data Memory Usage On The PIC18F452

Memory Organization - Program Memory Usage

- Currently, PIC18F devices reserve the first 512 bytes of Program Memory as the boot block.
  - Future devices may expand this, depending on application requirements for these devices.
- This bootloader is designed to occupy the current designated boot block of 512 bytes (or 256 words) of memory.
- Figure 3 shows a memory map of the PIC18F452.
  - The boot area can be code protected to prevent accidental overwriting of the boot program.

PIC18F452 - Program Memory Map

Note: Memory areas not shown to scale

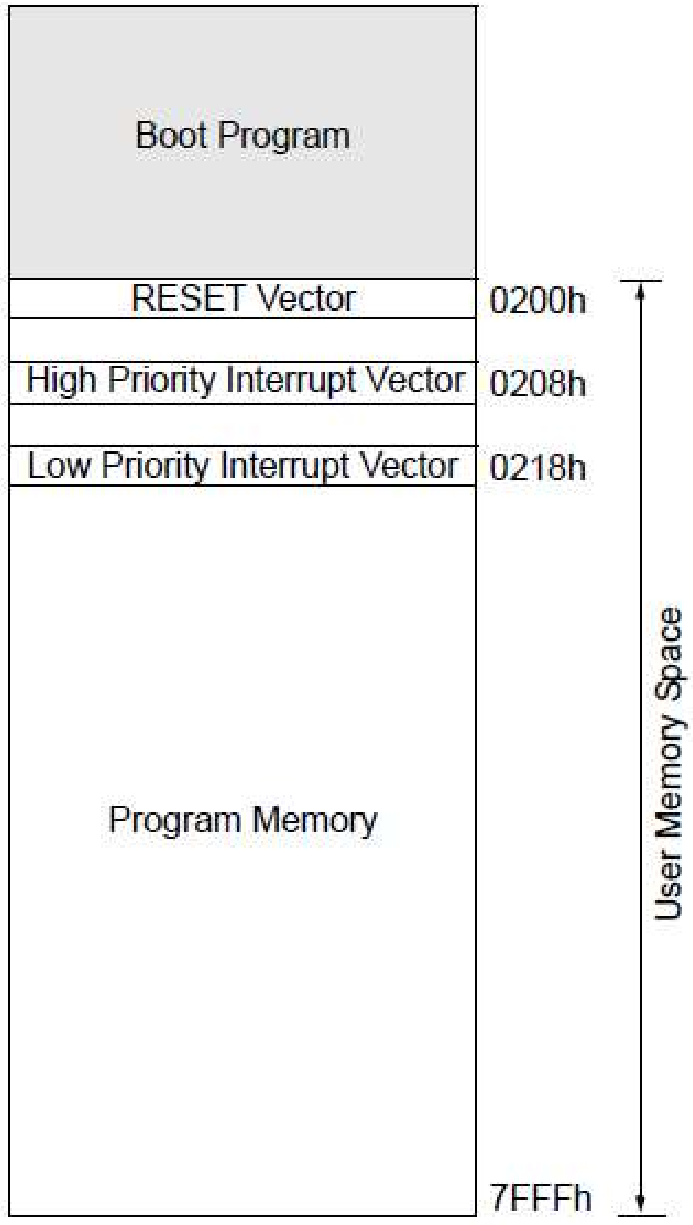


Figure 3: Program Memory Map Of The PIC18F452

PIC16F87XA Bootloader Implementation

- PIC16F87XA enhanced microcontrollers are designed to use the first 256 words of program memory.
- Figure 4 shows the memory map of the PIC16F877A.
- Like the PIC18F452 and other PIC18F devices, the boot area can be write protected to prevent accidental overwriting of the boot program.

PIC16F877Xa – Memories Map

Note: Memory areas not shown to scale

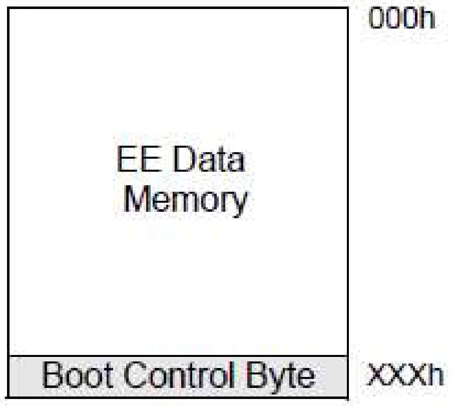


Figure 5: Data Memory Map

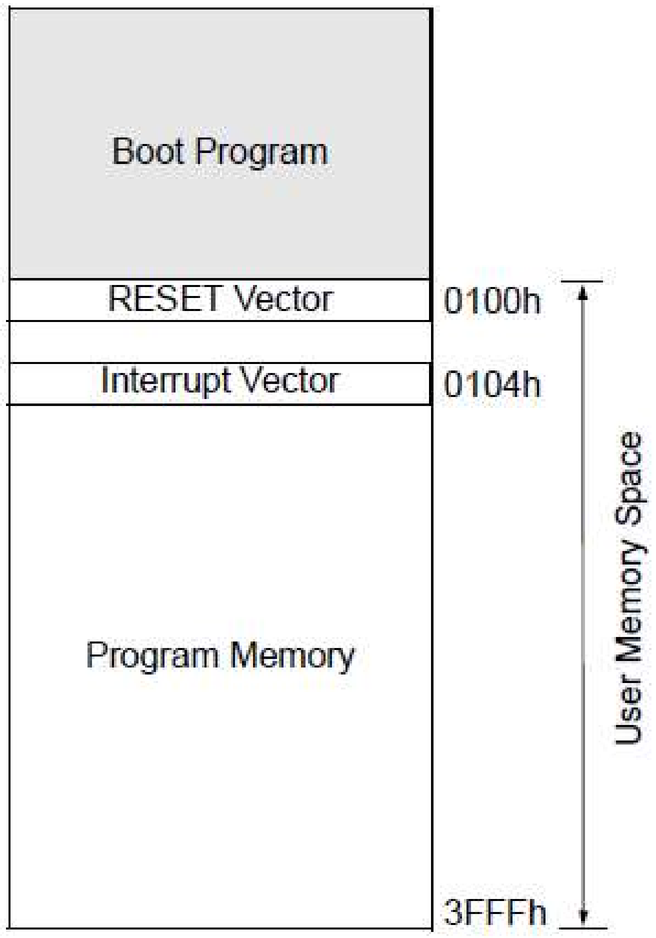


Figure 4: Program Memory Map Of The PIC16F877A

Communication Protocol

- The bootloader employs a basic communication protocol that is
  - robust,
  - simple to use, and
  - easy to implement.

Packet Format

- All data that is transmitted to or from the device follows the basic packet format:

  <STX><STX>[<DATA><DATA>...]<CHKSUM><ETX>

  - where each <...> represents a byte and [...] represents the data field.

  - The start of a packet is indicated by two ‘Start of TeXt’ control characters (<STX>), and is terminated by a single ‘End of TeXt’ control character (<ETX>).
  - The last byte before the <ETX> is always a checksum, which is the two’s complement of the Least Significant Byte of the sum of all data bytes.
- The data field is limited to 255 data bytes.
- If more bytes are received, then the packet is ignored until the next <STX> pair is received.

Control Characters

- There are three control characters that have special meaning. Two of them, <STX> and <ETX>, are introduced above.
- The last character not shown is the ‘Data Link Escape’, <DLE>.
- Table 1 provides a summary of the three control characters.

- **Note:** Control characters are not considered data and are not included in the checksum.

Table 1: Control Characters

Table 1: Control Characters

Control	Value	Description
<STX>	0Fh	Start of TeXt
<ETX>	04h	End of TeXt
<DLE>	05h	Data Link Escape

DLE Usage

- The <DLE> is used to identify a value that could be interpreted in the data field as a control character.
- Within the data field, the bootloader will always accept the byte following a <DLE> as data, and will always send a <DLE> before any of the three control characters.
  - For example, if a byte of value 0Fh is transmitted as part of the data field, rather than as the <STX> control character, the <DLE> character is inserted before the <STX>.
- This is called “byte stuffing”.

Table A-1: Bootloader Commands

Name	Number	Description	Command Device [data field]	Response [data field]	PIC18F	PIC16F
RESET	ANY	Reset the Device	[<COM><0x00>]	none	X	X
RD_VER	00h	Read Bootloader Version Information	[<0x00><0x02>]	[<0x00><0x02><VERL><VERH>]	X	X
RD_FLASH	01h	Read <LEN> bytes from Program Memory	[<0x01><LEN><ADDRL ><ADDRH><ADDRU>]	[<0x01><LEN><ADDRL><ADDRH><ADDRU>... LEN bytes of Data ...]	X	X
WT_FLASH	02h	Write <LEN> blocks to Program Memory	[<0x02><LEN><ADDRL> <ADDRH><ADDRU>... LEN bytes of Data ...]	[<0x02>]	X	X
ER_FLASH	03h	Erase <LEN> rows of Program Memory	[<0x03><LEN><ADDRL><ADDRH><ADDRU>]	[<0x03>]	X	🟥
RD_EEDATA	04h	Read <LEN> bytes from EE Data Memory	[<0x04><LEN><ADDRL><ADDRH><0x00>]	[<0x04><LEN><ADDRL><ADDRH><0x00>... LEN bytes of Data ...]	X	X
WT_EEDATA	05h	Write <LEN> bytes to EE Data Memory	[<0x05><LEN><ADDRL><ADDRH><0x00>...LEN bytes of Data ...]	[<0x05>]	X	X

Table A-1: Bootloader Commands

Name	Number	Description	Command Device [data field]	Response [data field]	PIC18F	PIC16F
RD_CONFIG	06h	Read <LEN> bytes from Configuration Memory	[<0x06><LEN><ADDRL > <0x00> <0x30>]	[<0x06><LEN><ADDRL ><0x00><0x30>... LEN bytes of Data ...]	X	🟥
WT_CONFIG	07h	Write <LEN> bytes to Configuration Memory	[<0x07><LEN><ADDRL><0x00>< 0x30>…LEN bytes of Data ...]	[<0x07>]	X	🟥
REPEAT	COM	Repeat last Command	[Empty data field]	Refer to the appropriate command response for the last command sent	X	X
REPLICATE	COM	Write old Buffer Data to another area	[<COM><LEN><ADDRL><ADDRH> <ADDRU>] where <COM> is any write Command	[<COM>]	X	X

Automatic Baud Rate Detection

- The bootloader is provided with an automatic baud rate detection algorithm that will detect most baud rates for most input clock frequencies (f_{OSC}).
- The algorithm determines
  - the best value for the Baud Rate Generator and then
  - loads the SPBRG register on the PIC with the determined value.
- **SYNCHRONIZING**
  - The first <STX> in the protocol is the synchronization byte.
  - It is used to match the device’s baud rate to the source’s baud rate.
  - Thus, the device is synchronized to the source on every new packet.

Differences between The PIC16 and PIC18F Bootloaders

- There are two main differences between the PIC16F87XA and PIC18F bootloaders.

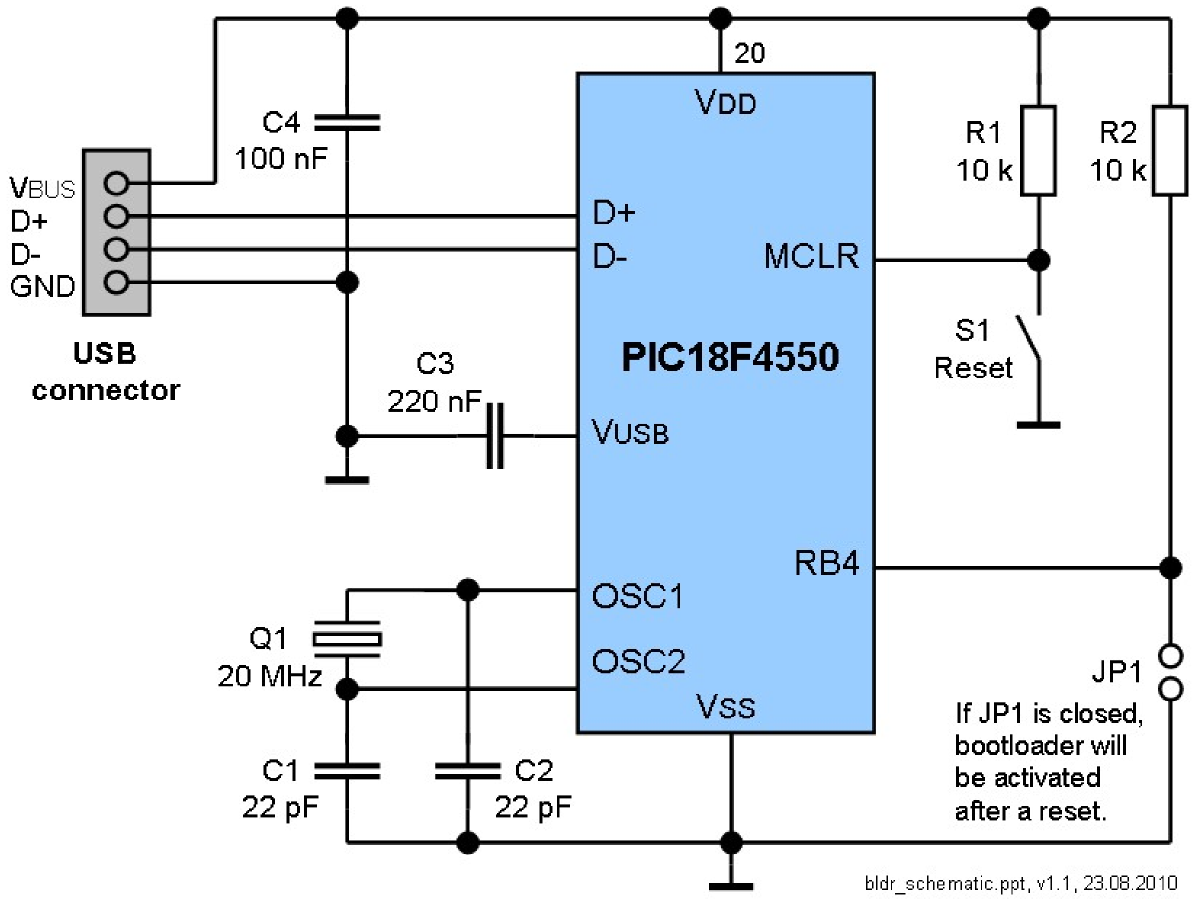
1. The PIC16F87XA bootloader does not support the following commands:
   - Erase FLASH;
   - Read CONFIG
   - Write CONFIG
2. The RESET command is only partially supported. When the microcontroller receives a RESET command, it executes a goto 0x0000. This is not a true RESET of the microcontroller.

- The following registers are not set to their default RESET states on execution of the command:
  - EEADR, EEADRH, EECON1, EEDATA, EEDATH
  - OPTION_REG, STATUS, TRISC
  - RCSTA, TXSTA, SPBRG
  - FSR, PIR1
- This is particularly important when leaving Boot mode via a software RESET.
  - The application software must be prepared to accept non RESET values in the registers listed above.
  - If RESET conditions are necessary, then the listed registers should be initialized in the application code.
- The alternative is to always perform a hardware RESET (MCLR) after completing a bootload operation.

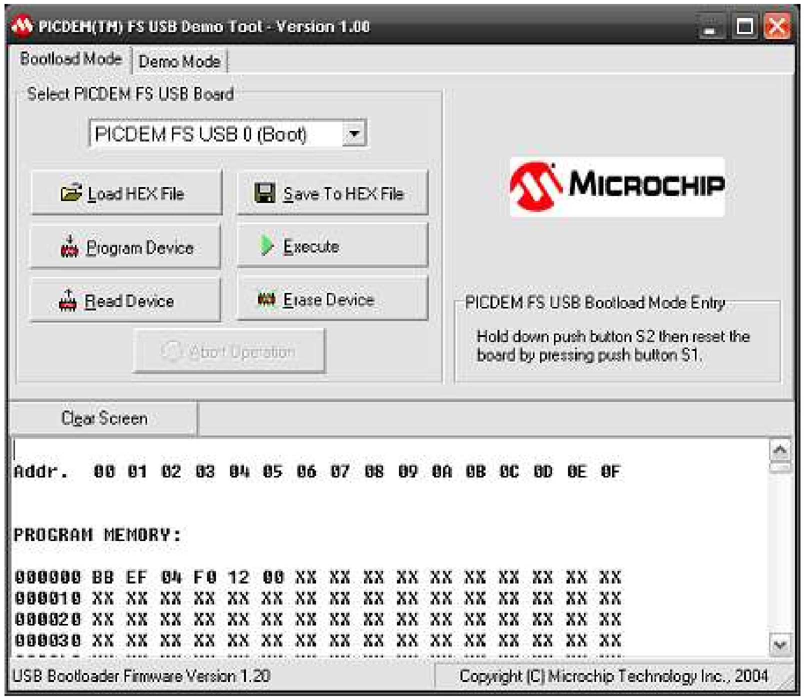
PIC USB Bootloader

- MCHPUSB Bootloader Firmware.
- The bootloader can be used for the all USB PIC devices (PIC18F4550, PIC18F4455, PIC18F2550, PIC18F2455, PIC18F4553, PIC18F4458, PIC18F2553, PIC18F2458).
- This bootloader was designed to be used with the PICDEM FS USB DEMONSTRATION BOARD from Microchip (PIC18F4550).
- The bootloader starts after Power-On or Reset. The bootloader checks if pin RB4 is low or high:
  - RB4 = low (0V): PIC starts in bootloader mode
  - RB4 = high (5V): PIC starts user applicatiob
- The current status of the bootloader is displayed via 4 LEDs which are connected to RD0-RD3.

Modified Schematic



Bootloader PC Software - PDFSUSB.exe



PICDEM FS USB Bootload Mode Entry

Hold down push button S2 then reset the board by pressing push button S1.

```
Addr.   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F


PROGRAM MEMORY:

000000 BB EF 04 F0 12 00 XX XX XX XX XX XX XX XX XX XX
000010 XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
000020 XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
000030 XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
```

USB Bootloader Firmware Version 1.20

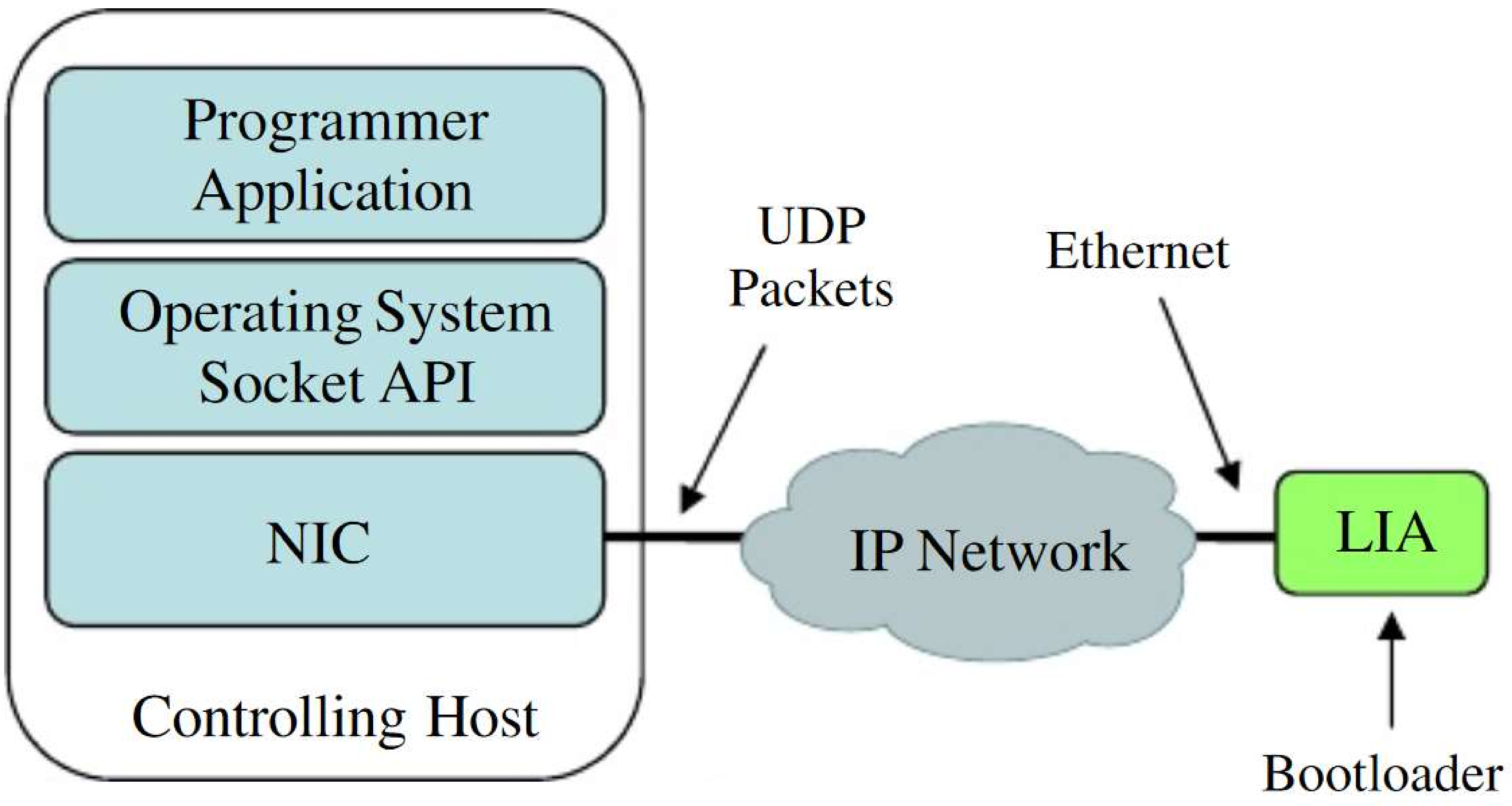
Copyright (C) Microchip Technology Inc., 2004

The Future - Ethernet Bootloaders

- for PIC18F, PIC24/dsPIC33 and PIC32 series microcontrollers http://www.brushelectronics.com/
- The Brush Electronics Ethernet Bootloaders have been developed to deliver a rapiddevelopment environment for an Ethernet enabled Microchip Flash Memory basedPIC Microcontroller and to support remote firmware upgrade for these systemsdeployed in the field. Figure outlines the Ethernet Bootloader’s conceptional systemelements. The LAN Interface Adapter (LIA) is the target device containing thebootloader image.

Ethernet Bootloader

- http://www.scribd.com/doc/78064907/BE-Ethernet-Boot-Loader-Ver-1-1



SBC65EC

- The SBC65EC is an embedded (PIC based) Single Board Computer (SBC)
  - with 10Mbs Ethernet and RS232 interface.
  - It is programmed with a bootloader and the free Modtronix SBC65EC Web Server
- Configuration, control and monitoring can be done via a web based interface, HTTP CGI commands or UDP commands.
- It has 32 general purpose I/O ports,
  - of which 12 are 10 bit ADC (Analog to Digital) inputs,
  - 4 are 10-bit PWM (Pulse Width Modulator) outputs.

SBC65EC

SBC66EC

- The SBC66EC is a Single Board Computer (SBC) with a 10/100 Mbit/sec Ethernet and a micro USB port.
  - It is supplied programmed with a USB Bootloader(for upgrading Firmware) and Webserver(firmware) installed.
  - The Webserver firmware uses the TCP/IP and USB stack from the Microchip Application Library.
- Uses the Microchip 16 MIPs PIC24FJ256GB206 Microcontroller
- 10/100 MBit/sec Ethernet port
- USB port via Micro B USB connector. Can also power board

SBC66EC

- 256KB internal FLASH memory; 96KBs of internal SRAM
- External 32Mbit (4MB) SPI FLASH; External 8KB SPI EEPROM
- External RTC with CR2032 Battery holder, 20+years
- 4 USARTS (RS-232, RS-485, LIN Bus...), 2 SPI Ports, 9 PWM,
  - that can be routed (via Microchip's PPS feature) to any of 18 user I/O Ports
- 2 I2C ports
- 26 Digital 3.3V User I/O ports,
  - of which 8 have can be used with 5V logic
- 11 of the I/O ports can be configured as 10-bit Analog Inputs
- Can be powered via Passive PoE. Like a 24V Passive PoE switch, or a Passive PoE injector

SBC66EC

Recommended Readings

1. E. Schlunder, *High-Speed Serial Bootloader for PIC16 and PIC18 Devices, AN1310*, Microchip.
2. http://www.etc.ugal.ro/cchiculita/software/picbootloader.htm (RO!).
3. Leonard Elevich, Veena Kudva, *Bootloader for dsPIC30F/33F and PIC24F/24H Devices*, AN1094, Microchip.
4. Ganapathi Ramachandra, *PIC32 Bootloader*, AN1388, Microchip.