USB BOOTLOADER

The principal function this system is fulfilling is to bootload your code into a PIC® using the USB port. The bootloader uses the native USB functions of specific PIC® processors as opposed to other USB Bootloaders which use a USB to serial converter and appear on an attached PC as a com port.

The code is written in Proton Basic and should be easy to understand and modify. The other half of the system is the PC end and for this JohnB has written an application that will take your hex code and send it, along with the appropriate commands, to the PIC® where it is loaded into the flash memory.

THE BASICS OF WRITING A BOOTLOADER IN PROTON BASIC

Before delving into the details of the bootloader code it is worth discussing the basic operations of writing code to flash memory and some of the design choices available.

Writing code to the Flash ram of a PIC® is done in 2 stages:

- Clear a block of flash memory to enable it to accept new data
- Write that data to PIC flash memory

Clearing - PIC® memory is cleared in banks, generally 64 bytes in length; in Proton it can be done using the command CErase.[1]

`CErase Address` - *Where address is the start of the bank you wish to erase*.

Thus the first step is to run through the memory erasing all the banks with CErase

Writing - To write data you use the command Cwrite. Once again data is written in blocks, this time 16 bytes.

`CWrite Address,[Data...Data]` - *where address is the start address in memory of the 1st data byte*.

NOTE you have to start from the start of the 16 byte block on the boundary and you have to write the full no of bytes before the process will finish.

DESIGNING A BOOTLOADER

There are a number of points to consider when designing a USB bootloader:

- Where the bootloader should reside in memory
- Interrupt handling
- What USB protocol to use
- How the bootloader will be accessed i.e. How to know when to run the bootloader
- What happens if the writing is interrupted?

Location - There can only be 2 locations a bootloader can reside; the Start of Memory or the End. Most serial bootloaders reside at the top of memory. Typically they work as follows:

At the PC, the loader modifies the incoming Hex and inserts a jump to the bootloader code at the top of memory. On start-up it can look for incoming data. If, after a set period, there is no data on the USART it

[1] The Proton manual says nothing about the **CErase** or **CWrite** commands but after a little testing and looking at the code it is pretty simple to use.

returns control back to the main program. The main advantage of this configuration is that the user code requires no modification to run (apart from the couple of instructions inserted automatically by the PC app).

This is fine for a Bootloader that uses the USART as the code required to read a USART is minimal. A USB driver, on the other hand, requires a significant amount of code and while it would be possible to place a copy of this at the top of memory, it is an unnesesary waste of code space. Thus USB bootloader code is best placed at the bottom of memory and the main code adjusted to suit. This can be achieved simply by inserting the following line in Proton.

```
PROTON_START_ADDRESS = Address
```

Where address is the starting address of your code.

Interrupts - A further issue is interrupts, since a hardware interrupt always jumps to either $0008 or $0018 depending on the priority level. However this can be resolved with a few of lines in the bootloader to redirect the calls to the right place in the main code.

Choice of USB Protocol - There are 2 basic types of USB Proton protocols:

The first is the HID which stands for **H**uman **I**nterface **D**evice. This is a driverless system that requires no user intervention apart from waiting a few seconds while the PC does some initializations the first time you use it.

The second is CDC which is **C**omposite **S**erial **D**evice. This will create a virtual serial port on the PC and any PC apps talk through it as if it was a serial port. There are a couple of disadvantages: a special driver is required which must be installed before using the connection. Also, you cannot be sure which serial port number will be assumed for the USB connection so it has to be manually selected by the user.

For simplicity and all round ease of use the HID interface is the preferred protocol.

Initiating the bootloader - Knowing when to run the Bootloader is a very fundamental issue. With a serial port you can stream data at the port even if it's not connected. In the PIC®, during the first 500mS or so following startup a serial Bootloader can decide if it is needed simply by monitoring the serial port. USB on the other hand can take a long time to connect, especially the first time it is used on a PC, making this approach impractical.

The solution is ultimately down to the designer of the device and how it fits their application. There is more on this later when we talk about the hardware, but as an early tease, you can tell if the PC is connected to a PC using just 1 resistor.

Resilience - When you load your new program onto the PIC® it can take a few seconds, especially if it's a big program. Plenty of time for something to go wrong, like pulling the connection plug. There has to be a method where you can monitor the code was written successfully.

The solution is pretty simple. Set an EEPROM flag when you clear the memory and only clear it when it the program load was deemed successful by the PC loader app. On start up check this flag, if it is set then something went wrong and the probability is that application code is incomplete.

THE PROTON CODE

The full code can be downloaded below but for now we will pick out some of the main points. You might find it useful to have the bootloader code open in PDS while you read the following:

Some basics

- This PIC® code is written for is the PIC18F14K50. You can change it yourself and this is covered later. The PIC18F14K50 was chosen as it is a great little device and very easy to use.
- It runs on a 12Mhz Xtal externally and is PPL to 48Mhz then divided internally to run at 24mhz. 12Mhz is need for the USB side to work internally and the rest is sorted using the fuse settings.
- The detection of the USB connection is achieved by monitoring one of two lines on the USB interface. This is discussed more fully in the Hardware section.

USB Descriptor - Every USB device has to have a descriptor file which, among many other things, carries the VID (Vendor ID) and PID (Product ID) number. In the bootloader this file is an include file and is declared as:

```
USB_Descriptor "JGBBootLDescMC.inc"
```

This file uses the standard MicroChip USB bootloader VID and PID which you can use for personal use. However, if you plan to sell a product then you need to sort out your own VID and PID.

The section to modify in the descriptor is this

```
        dt      0xD8, 0x04    ; idVendor low byte, high byte
        dt      0x3C, 0x00    ; idProduct low byte, high byte
```

Note how the Hex bytes are reversed so the VID is $04D8 and the PID is $003C

There are other sections you may wish to modify as well, like the Serial number. Look through the code to see what needs to be changed.

Alternative PIC®s - These are the main defines you will need to alter to make the code work with another PIC®.

```
Symbol MainCodeStart = $0B80 ; Main code start address / Keep on a 64 byte boundry
```

This address is where your code will start, it must be a multiple of the FlashEraseSeg value.
It will be used to create the declaration `PROTON_START_ADDRESS = Address`

```
Symbol EndOfCodeSpace = $4000; End of code space
```

This is the address of the end of the memory in the PIC® so it knows where to stop erasing.

```
Symbol FlashEraseSeg = 64    ; The size of memory bank erased in one go
```

Read the PIC® data sheet for this information or look in the PDS PPI file.

```
Symbol FlashWriteBlock = 16  ; The memory size Flash write needs to do in one block
```

Again read the manual or look at the PPI file.

```
IOCA = %00000011 ; Set enable interrupts on port A to get the read on porta.0/1 to
work
```

The Data sheet on the PIC18F14K50 is missing some sections notably how to detect the input on the D+ line. To do this you have to enable interrupts on the port. In my set up if the line is high then you are not connected, when the USB line is connected the line is pulled low. It works 99.9% of the time but it's always better to have a backup, like a button.

USB BOOTLOADER COMMANDS

There are 4 fundamental commands required to communicate with the bootloader. Each command comprises a 64 byte packet sent from the PC via USB. Its format comprises a 1 Byte command followed optionally by Address and Data as required.

EraseProgram - This will erase all the flashram between MainCodeStart and EndOfCodeSpace. It also sets a flag at the top end of EEPORM to say flash was erased.

FlashWrite - Data is sent with the following packet info Address as a DWord followed by 16 bytes of data.

The data is written to memory and then read back to compare it against what was sent. If the data is OK it responds with the default acknowledge of $55. if there is a discrepancy then the Error message of $AA is sent.

FinishWrite - All this really does is clear the EEPROM set when you did an erase.

RunMainCode - Jumps to the beginning of the user code.

NOTE - Every command sends back an acknowledge of $55 to say message was actioned. (the exception being a failure to write to flash properly were it will send back $AA)

EEPROM Programming - The Bootloader can also write to EEPROM if there is any EData in the hex file. EEPROM data is recognised by its address and then uses the **Ewrite** command to load the EEPROM data.

MODIFYING THE CODE FOR YOUR OWN USE

There are many areas that can be modified; here is a list some of them and how to do it.

Changing the VID, PID Serial no etc. - It is pretty easy to change the VID and PID in the descriptor; however the PC loader application only works with the Microchip™ VID and PID so for practical purposes it can only be used for development or personal use.

If you require a loader with your own VID and PID which can generate standalone .exe's that can be passed on to customers to update their firmware then please contact us. We have products which offer this and a lot more such as customization and encryption etc.

Adding your own commands - The USB interface does not have to be restricted to bootloading new firmware. It could be used to act as a conduit to send data back to a PC or to allow the PIC® application to be controlled by the PC. This method is used in my product the TL1000 www.milelogik.com It's a simple matter to add commands to the command Select...Case list which will call routines in your application code. However, you will have to write your own PC application to handle these additional commands.

Calling a Bootloader Function - This may sound odd but can be useful. E.g. There may be a routine in the Bootloader section you want access to from the main code to save you replicating that code in your main application.

Place a GoTo in the Bootloader code at a fixed place that jumps to the routine you require. Then from your main code make a Call to that address.

E.g. in your bootloader code

```
Org (MainCodeStart - $12) ; space for redirection gotos

GoTo YourCodeSub
```

In your main code

```
Symbol YourCodeSub = MainStartAddress - $12
....
....
@ Call YourCodeSub  ; Call your code in the Bootloader section
```

It should be noted that if the routine is using variables they will affect the main routines variables as well. In this case you can allocate the variables in a location not used normally by either set of code. For example

```
Dim Commonvar1 as Word At __USBIn_Buffer ; Allocate space for common variables

Dim Commonvar1 as Byte At Commonvar1 + 2
```

If the same set of variables are used in both routines you can call a routine and access the data produced on its return.

Lastly any modifications will require more room for the code, adjust it by modifying the declare

```
Symbol MainCodeStart = $0B80 ; Main code start address / Keep on a 64 byte boundary
```

Note - Start high then work back once you are happy with the routine. Remember to alter the value in your main code.

WRITING YOUR APPLICATION CODE

In general you write your application code in exactly the same was as you would for any other PIC® application with a couple of minor provisos. Since the bootloader is located at the start of the code space your main code will need to sit above it.

There is only one line of code needed to achieve this:-

```
PROTON_START_ADDRESS = Address
```

Where address is the starting address of your code.

The only other thing you need to consider is that interrupts will take one instruction longer to reach your interrupt handler so finely tuned timer interrupts will need adjusting slightly.

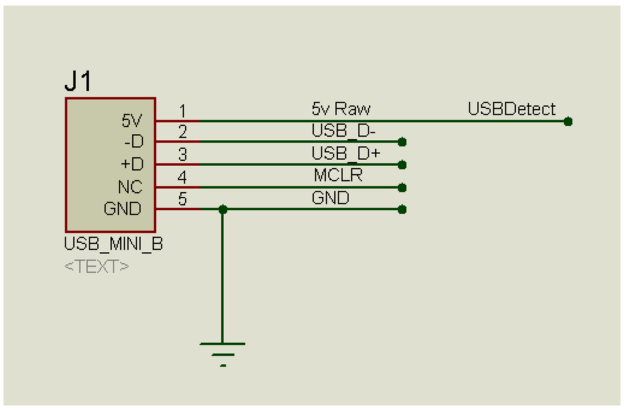
And really that is all!

HARDWARE

As mentioned above there will be instances where you need to know if the device is connected to a PC. This turns out to be pretty simple as there a several signals you can pick up on.

There are 2 methods available depending on how the device will be powered:

Device powered independently

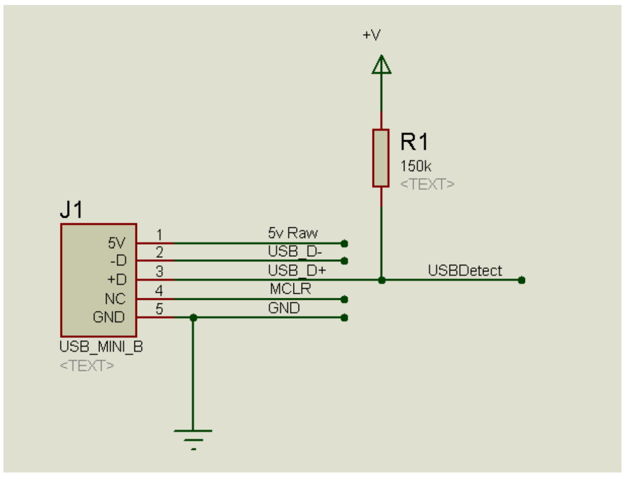


Detect the USB connection via the USB power line

When using a separate power supply to power your device connect the USB power pin to a spare pin on your device.

On power up use check for a high on this pin to determine the presence of the USB connection.

Device powered from the USB connector



Detect a USB connection using D+

If you are using the USB connector as the power source for your device the above option is unavailable to you.

However, when connected to a USB port on a PC the D+ line is pulled low. By putting a weak pullup on the D+ line and connecting it to a pin you can test the line on start up. High not connected. Low connected.

NOTE: This system works 99% of the time but it’s always better that there is a backup like a button.

THE **USB BASIC PIC® BOOTLOADER**

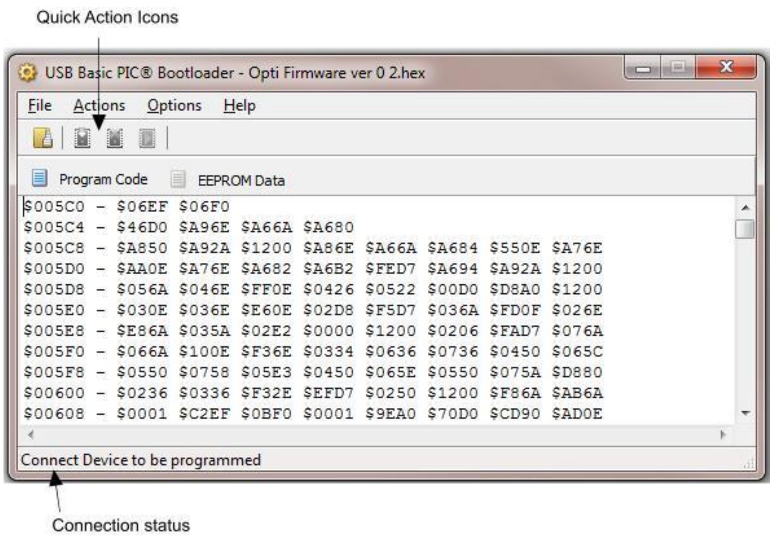
Obviously the other half of the equation is the PC end application and we have that covered.

Note - *This is a development tool and not intended for commercial use. If you require an application capable of producing a standalone .exe complete with encryption and version control (as used to update the firmware on the TL1000 www.milelogik.com) then contact either Johnb or Tim Box for more info.*

This section covers the operation and installation of the Bootloader.

USING THE USB BASIC PIC® BOOTLOADER

The PC Boot loader should be familiar to anyone who has used the Loader supplied with PDS. The controls are similar but without the Read and Identify functions of the PDS loader.



```
$005C0 – $06EF $06F0
$005C4 – $46D0 $A96E $A66A $A680
$005C8 – $A850 $A92A $1200 $A86E $A66A $A684 $550E $A76E
$005D0 – $AA0E $A76E $A682 $A6B2 $FED7 $A694 $A92A $1200
$005D8 – $056A $046E $FF0E $0426 $0522 $00D0 $D8A0 $1200
$005E0 – $030E $036E $E60E $02D8 $F5D7 $036A $FD0F $026E
$005E8 – $E86A $035A $02E2 $0000 $1200 $0206 $FAD7 $076A
$005F0 – $066A $100E $F36E $0334 $0636 $0736 $0450 $065C
$005F8 – $0550 $0758 $05E3 $0450 $065E $0550 $075A $D880
$00600 – $0236 $0336 $F32E $EFD7 $0250 $1200 $F86A $AB6A
$00608 – $0001 $C2EF $0BF0 $0001 $9EA0 $70D0 $CD90 $AD0E
```

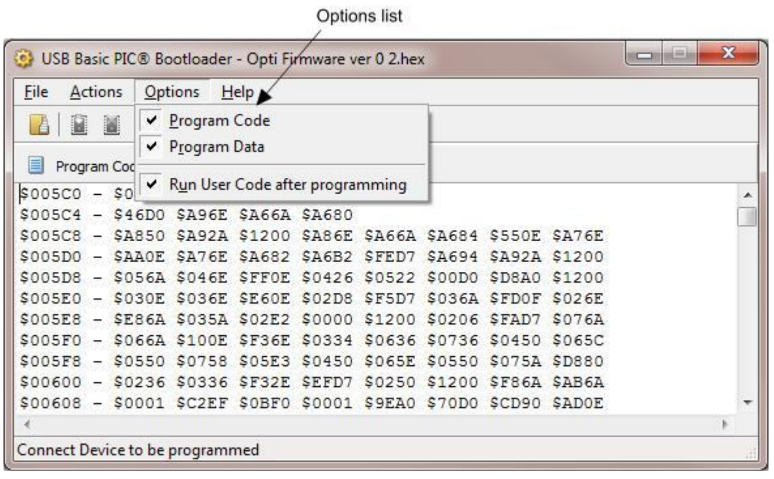
Connect Device to be programmed

The USB Basic PIC® Bootloader installs in PDS as an IDE plug-in.

You first need to program the USB bootloader code into the device using a standard programmer.

Thereafter just connect the board to the PC via the USB connection and the "USB Basic PIC® Bootloader" will detect the connection.

When started from PDS the loader will automatically open the Hex file for the current page.


Options list
USB Basic PIC® Bootloader - Opti Firmware ver 0 2.hex
File
Actions
Options
Help
Program Code
✓ Program Code
✓ Program Data
✓ Run User Code after programming

```
$005C0 – $0
$005C4 – $46D0 $A96E $A66A $A680
$005C8 – $A850 $A92A $1200 $A86E $A66A $A684 $550E $A76E
$005D0 – $AA0E $A76E $A682 $A6B2 $FED7 $A694 $A92A $1200
$005D8 – $056A $046E $FF0E $0426 $0522 $00D0 $D8A0 $1200
$005E0 – $030E $036E $E60E $02D8 $F5D7 $036A $FD0F $026E
$005E8 – $E86A $035A $02E2 $0000 $1200 $0206 $FAD7 $076A
$005F0 – $066A $100E $F36E $0334 $0636 $0736 $0450 $065C
$005F8 – $0550 $0758 $05E3 $0450 $065E $0550 $075A $D880
$00600 – $0236 $0336 $F32E $EFD7 $0250 $1200 $F86A $AB6A
$00608 – $0001 $C2EF $0BF0 $0001 $9EA0 $70D0 $CD90 $AD0E
```

Connect Device to be programmed

You can choose an alternative hex file from **File**, **Open** menu.

You can view the **Code** and **EEPROM** data for your application by clicking on the appropriate tabs.

To program your device, click on the Program Icon or choose **Program** from the **Action** menu.

You can choose what actions are taken when you program the device from the **Options** menu.

USB Basic Pic® Bootloader Command list

File		**Actions**		**Options**	**Help**	
Open	Ctl+ O	Program	Ctrl+ P	Program Code	About	F1
Recently used		Erase	Ctrl+ E	Program Data		
Exit	Ctl+X	Run	Ctrl+ R	Run User Code after Programming		

INSTALLING THE USB BASIC PIC® BOOTLOADER

To install the USB Basic PIC® Bootloader run "**USB Basic Pic® Bootloader Setup.exe**". This will install the bootloader as an IDE plug-in under JGB Tools in the plug-in menu.

You can add this application as a Programmer by doing the following:

- In PDS select **Compile and Program Options** from the **View** menu.
- Select the **Programmer** Tab
- Click **Edit** and enter the name of the bootloader "**PICBootloaderPlusHID.exe**"
- Click **Next**
  - Either let the PDS **Find Automatically** (this can take some time) or click **Find Manually** and navigate to the path in which ProtonIDE has been installed plus "\Plugin\JGBTools" - Typically on an XP system this would be:

    c:\Program Files\ProtonIDE\Plugin\JGBTools.

    On a Win7 system this would be:

    c:\Program Files(x86)\ProtonIDE\Plugin\JGBTools
- Click **Next** and in parameters enter **$hex-filename$** then click **Finished**
- Click **OK** to close the form.

To make the USB Basic PIC® Bootloader your default programmer:

- Click on the dropdown arrow in the **Program toolbar button**
- Select the **USB Basic PIC® Bootloader** entry.

Now, when you hit Program or Compile and Program this will bring up the USB Basic PIC® Bootloader with your new hex file loaded ready for programming.