MICROCHIP

AN732

Implementing a Bootloader for the PIC16F87X

Author: Mike Garbutt
Microchip Technology Inc.

INTRODUCTION

The PIC16F87X family of microcontrollers has the ability to write to their own program memory. This feature allows a small bootloader program to receive and write new firmware into memory. This application note explains how this can be implemented and discusses the features that may be desirable.

In its most simple form, the bootloader starts the user code running, unless it finds that new firmware should be downloaded. If there is new firmware to be downloaded, it gets the data and writes it into program memory. There are many variations and additional features that can be added to improve reliability and simplify the use of the bootloader, some of which are discussed in this application note.

The general operation of a bootloader is discussed in the OPERATION section. Appendix A contains assembly code for a bootloader developed for the PIC16F877 and key aspects of this bootloader are described in the IMPLEMENTATION section.

For the purpose of this application note, the term “boot code” refers to the bootloader code that remains permanently in the microcontroller and the term “user code” refers to the user’s firmware written into FLASH memory by the boot code.

FEATURES

The more common features a bootloader may have are listed below:

- Code at the Reset location.
- Code elsewhere in a small area of memory.
- Checks to see if the user wants new user code to be loaded.
- Starts execution of the user code if no new user code is to be loaded.
- Receives new user code via a communication channel if code is to be loaded.
- Programs the new user code into memory.

OPERATION

The boot code begins by checking to see if there is new user code to be downloaded. If not, it starts running the existing user code. If there is new user code to be downloaded, the boot code receives and writes the data into program memory. There are many ways that this can be done, as well as many ways to ensure reliability and ease of use.

Integrating User Code and Boot Code

The boot code almost always uses the Reset location and some additional program memory. It is a simple piece of code that does not need to use interrupts; therefore, the user code can use the normal interrupt vector at `0x0004`. The boot code must avoid using the interrupt vector, so it should have a program branch in the address range `0x0000` to `0x0003`.

The boot code must be programmed into memory using conventional programming techniques, and the configuration bits must be programmed at this time. The boot code is unable to access the configuration bits, since they are not mapped into the program memory space. Setting the configuration bits is discussed in the next section.

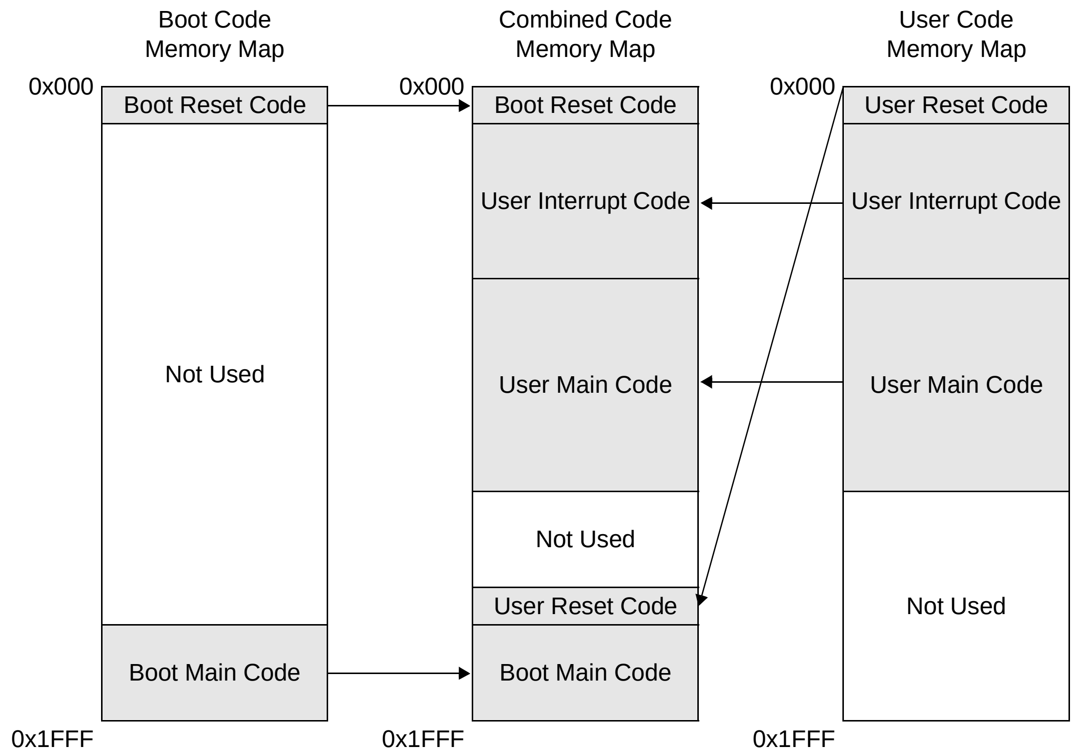
In order for the boot code to begin executing the user code, it must know where the code starts. Since the boot code starts at the Reset vector, the user code cannot start at this location. There are two methods for placing the starting point of the user code.

One method is to use an `ORG` directive to force the user code to start at a known location, other than the Reset vector. To start executing the user code, the boot code must branch to this fixed location, and the user code must always use this same location as its start address.

An alternative method is to start the user code at the normal Reset vector and require that the user code has a `goto` instruction in the first four instructions to avoid the interrupt vector. These four instructions can then be relocated by the boot code and programmed into the area of program memory used by the boot code. This simplifies the development of code for use with the bootloader, since the user code will run when programmed directly into the chip without the boot code present. The boot code must take care of paging and banking so the normal Reset conditions apply before executing the relocated code.

© 2000 Microchip Technology Inc.

FIGURE 1: INTEGRATING USER CODE WITH BOOT CODE



Configuration Bits

The configuration bits cannot be changed by the boot code since they are not mapped into the program memory space. This means that the following configuration options must be set at the time that the boot code is programmed into the device and cannot be changed:

CPx	Program Memory Code Protection Enable
DEBUG	In-Circuit Debugger Mode Enable
WRT	Program Memory Write Enable
CPD	Data EEPROM Code Protection Enable
LVP	Low Voltage In-Circuit Programming Enable
BODEN	Brown-out Reset Enable
PWRTE	Power-up Timer Enable
WDTE	Watchdog Timer Enable
FOSCx	Oscillator Selection

Most of these configuration options are hardware or design-dependent, and being unable to change them when the user code changes is of no consequence.

The various PIC16F87X devices have different code protection implementations. Please consult the appropriate data sheet for details.

Some devices (such as the PIC16F877), can code protect part of the program memory and prevent internal writes to this protected section of memory. This can be used to protect the boot code from being overwritten, but also prevents the user code from being code protected, however.

On some devices, code protecting all the program memory still allows internal program memory write cycles. This provides security against the user code being read out of the chip, but does not allow the boot code to be protected from being overwritten.

Data EEPROM Code Protection Enable would normally not need to be set, unless data is programmed into the data EEPROM when the boot code is originally programmed and this data needs to be protected from being overwritten by the user code.

Program Memory Write Enable must be enabled for the boot code to work, since it writes to program memory. Low Voltage In-Circuit Serial Programming (ICSP™) enable only needs to be set if the user wishes to program the PICmicro MCU in-circuit, using logic level signals on the RB3, RB6 and RB7 pins. Since the purpose of the boot code is to program user code into the PICmicro MCU, in most cases, it would be redundant to have facilities for low voltage ICSP.

If the Watchdog Timer is enabled, then the boot code must be written to support the Watchdog Timer and all user code will have to support the Watchdog Timer.

 © 2000 Microchip Technology Inc.

Determining Whether to Load New Code or to Execute User Code

After a Reset, the boot code must determine whether to download new user code. If no download is required, the bootcode must start execution of existing user code, if available.

There are many ways to indicate whether or not new user code should be downloaded. For example, by testing a jumper or switch on a port pin, polling the serial port for a particular character sequence, or reading an address on the I^{2}C™ bus. The particular method chosen depends on the way that user code is transferred into the microcontroller. For example, if the new user code is stored on an I^{2}C EEPROM that is placed in a socket on the board, then an address in the EEPROM could be read to determine whether a new EEPROM is present.

If an error occurred while downloading new user code, or the bootloader is being used for the first time, there might not be valid user code programmed into the microcontroller. The boot code should not allow faulty user code to start executing, because unpredictable results could occur.

Receiving New User Code to Load into Program Memory

There are many ways that the microcontroller can receive the new firmware to be written into program memory. A few examples are from a PC over a serial port, from a serial EEPROM over an I^{2}C or SPI™ bus, or from another microcontroller through the parallel slave port.

The boot code must be able to control the reception of data, since it cannot process any data sent to it while it is writing to its own program memory. In the case of data being received via RS-232, there must be some form of flow control to avoid data loss.

The data received by the boot code will usually contain more than just program memory data. It will normally contain the address to which the data is to be written and perhaps a checksum to detect errors. The boot code must decode, verify and store the data, before writing it into program memory. The available RAM (GPR registers) of the device limits the amount of data that can be received before writing it to program memory.

Programming the FLASH Program Memory

The PIC16F87X devices have special function registers that are used to write data to program memory. There is a specific sequence of writes to these registers that must be followed to reduce the chances of an unintended program memory write cycle occurring. Because code cannot be executed from the FLASH program memory while it is being written, program execution halts for the duration of the write cycle. Program memory is written one word at a time.

Error Handling

There are several things that can go wrong during execution of the boot code or user code. The bootloader should handle the following error conditions:

- No valid user code written into the chip.
- Error in incoming data.
- Received user code does not have any code at its Reset vector.
- Received user code overlaps boot code.
- User code causes execution into the boot code area.

If the bootloader is being used for the first time, or if the user code is partially programmed because of a previous error, there might not be valid user code programmed into the microcontroller. The boot code should not allow potentially faulty user code to start executing.

The transfer of data can be interrupted, which will cause the boot code to stop receiving data. There are several ways to handle this depending on how the data is being received. For example, the boot code may be able to time-out and request the data to be sent again. The simplest method is to wait, trying to receive more data with no time-out, until the user intervenes and resets the device. Since the boot code needs to leave the most possible program memory space for the user code and also be reliable, the smallest, simplest implementation is often the best.

Incoming data may be corrupted by noise or some other temporary interruption, and this should be detected, otherwise, incorrect data could be programmed. A checksum or other error detection method can be used.

Incorrect use of flow control can result in data being sent to the PICmicro MCU while it is not ready to receive data. This can cause overrun errors that should be handled by the boot code. Once an overrun has occurred, the data is lost and this is essentially the same as a data transfer interruption, discussed above.

In some cases, data could be sent to the microcontroller before the boot code is running, causing part of the data to be lost. If this type of error is possible, then it should be detected. This error may manifest itself as user code that does not seem to have any code at the Reset location and can be detected by checking the addresses being programmed. An alternative is to generate a checksum on all the code that is written into program memory and transmit this to the user for verification, after programming has been completed.

© 2000 Microchip Technology Inc.

The code developer should take care that the user code does not use the same program memory space that the boot code uses. The exception is the user code at the Reset location that can be relocated, as explained earlier. If the user code does try to use program memory that contains boot code, the boot code should detect the conflicting address and not overwrite itself. In some devices, part of the program memory can be code protected to prevent internal writes to the part of the memory that contains the main boot code. Note that this does not apply to all PIC16F87X devices.

Faulty user code, or a brown-out condition that corrupts the program counter, can cause execution to jump to an unprogrammed memory location and possibly run into the start of the boot code. If the user code at the Reset location is being relocated, as explained earlier, then execution can enter the boot code area if a program branch does not occur in these four relocated instructions. The boot code should trap the program execution to avoid these errors from causing any unintended operation.

When an error is detected, it is useful to indicate this in some way. This can be as simple as turning on an LED, or sending a byte out the serial port. If the system includes a display and the display drivers are incorporated into the boot code, then more sophisticated error messages can be used.

 © 2000 Microchip Technology Inc.

FIGURE 2: FLOWCHART FOR BOOTLOADER

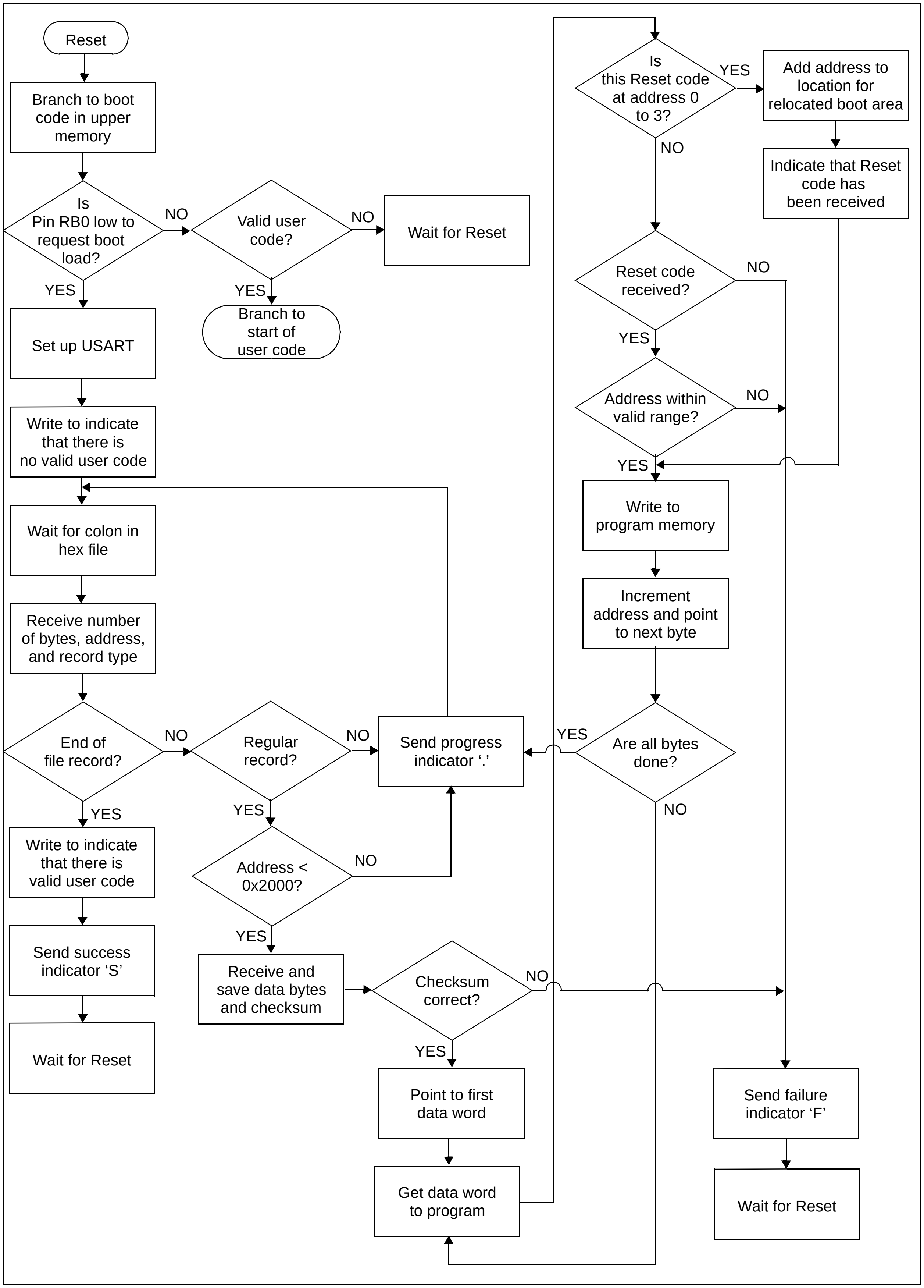
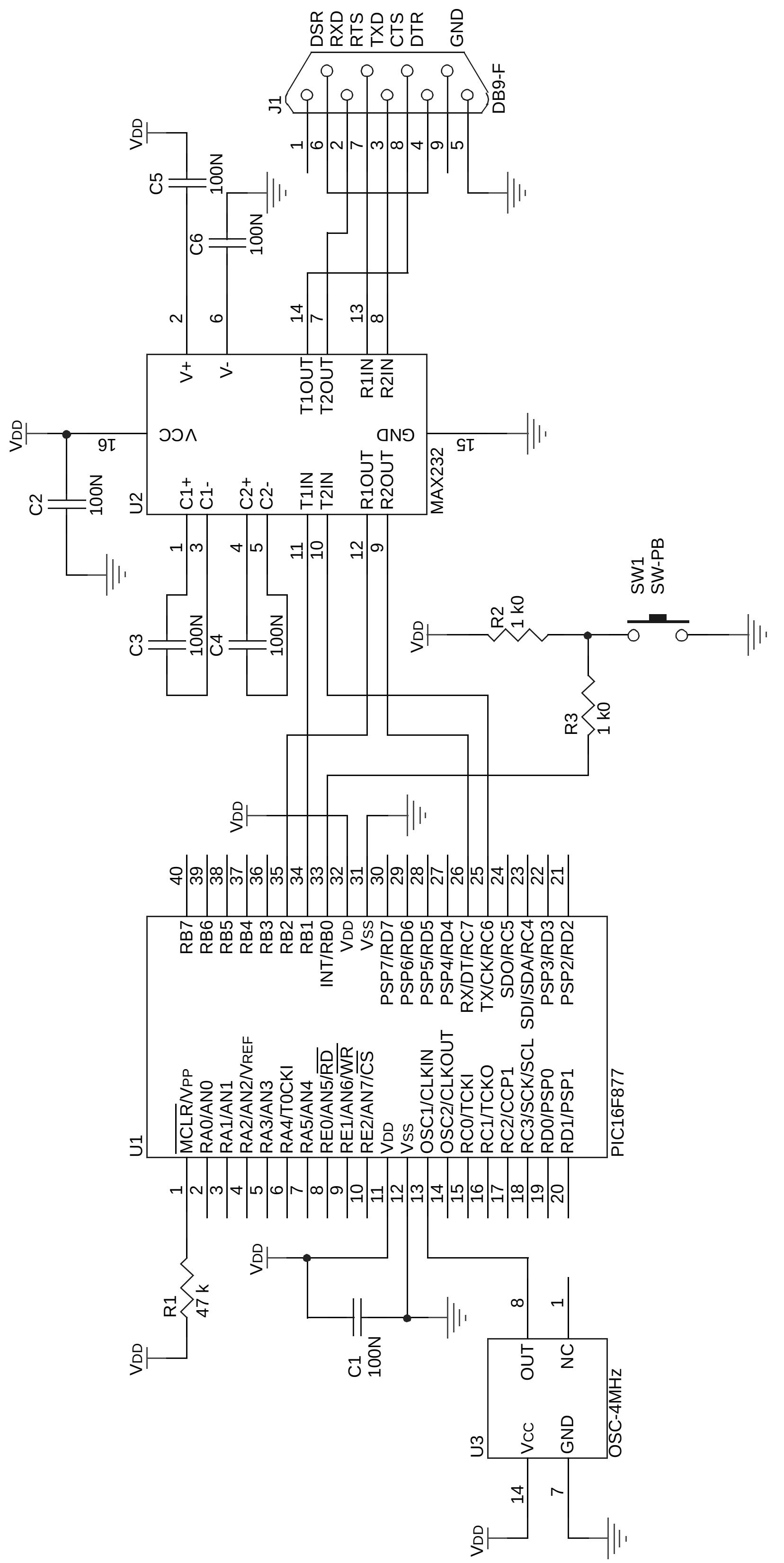


FIGURE 3: SCHEMATIC SHOWING SERIAL PORT AND TEST PIN



 Preliminary © 2000 Microchip Technology Inc.

IMPLEMENTATION

How this Bootloader Works

The boot code in Appendix A implements a bootloader in a PIC16F877 device. It uses the USART to receive data with hardware handshaking, tests a pin to decide if new user code should be received and includes many of the features discussed in this application note.

Integrating User Code and Boot Code

The code at the Reset location (`ResetVector`) writes to PCLATH. To set the page bits, it then jumps to the rest of the boot code in upper memory. The main code is in the upper 224 bytes of memory starting at address `0x1F20` (`StartOfBoot`). The first instructions at this location trap accidental entry into the boot code. The main bootloader routine starts at the address labeled `Main`.

The boot code requires that the user code includes a `goto` instruction in the first four locations after the Reset vector and relocates these four instructions into the boot code section (`StartUserCode`). This simplifies the development of code for use with the bootloader, since the same user code will also run when programmed directly into the chip, without the boot code present. The boot code changes to bank 0 and clears PCLATH before executing the relocated code, so that the normal Reset conditions apply. If a program branch does not occur in the four relocated instructions, then program execution is trapped in an endless loop to avoid any unintended operation.

The boot code must be programmed into the PIC16F877 using conventional programming techniques and the configuration bits are programmed at the same time. The configuration bits are defined with a `__CONFIG` directive and cannot be accessed by the boot code, because they are not mapped into the program memory space. The boot code does not use a Watchdog Timer.

Determining Whether to Load new Code or to Execute User Code

The boot code tests port pin RB0 to determine whether new user code should be downloaded. If a download is required, then the boot code branches to the `Loader` routine that receives the data and writes it into program memory.

If pin RB0 does not indicate that new user code should be loaded, then a program memory location (labeled `CodeStatus`) is read with routine `FlashRead` to determine whether there is valid user code in the device. If there is valid user code, the boot code transfers execution to the user code by branching to location `StartUserCode`. Otherwise, execution is trapped in an endless loop to avoid this error from causing any unintended operation.

Receiving New User Code to Load into Program Memory

The boot code receives the new firmware as a standard Intel® hex file (`INHX8M` format), using the USART in Asynchronous Receiver mode (hex format defined in Appendix B). It is assumed that a PC will be used to send this file via an RS-232 cable, connected to a COM port. Hardware handshaking allows the boot code to stop the PC from transmitting data while FLASH program memory is being written. Since the PICmicro device halts program execution while the FLASH write occurs, it cannot read data from the USART during this time.

Hardware handshaking (described in Appendix C) is implemented using port pin RB1 as the RTS output and RB2 as the CTS input. The USART is set to 8-bit Asynchronous mode at 9600 baud in the `SerialSetup` routine. The `SerialReceive` routine enables transmission with the RTS output and waits until a data byte has been received by the USART, before returning with the data. The `SerialTransmit` routine checks the CTS input until a transmission is allowed and then sends a byte out the USART. This is used for transmitting progress indication data back to the PC.

The boot code receives the hex file, one line at a time and stops transmission after receiving each line, while received data is programmed into program memory.

Decoding the Hex File

The boot code remains in a loop, waiting until a colon is received. This is the first character of a line of the hex file. The following four pairs of characters are received and converted into bytes, by calling the `GetHexByte` routine. The number of bytes (divided by two to get the number of words) and the address (divided by two to get a word address) are saved, and the record type is checked for a data record, or end of file record.

If the record type shows that the line contains program memory data, then this data is received, two pairs of characters at a time (using the `GetHexByte` routine), and is stored in an array. The checksum at the end of the line is received and checked, to verify that there were not any errors in the line.

Once the hex file line has been received, hardware handshaking is used to stop further transmission, while the data is written into the program memory. The `<CR>` and `<LF>` characters that are sent at the end of the line are ignored. This gives the handshaking time to take effect by ignoring the byte being transmitted, when the handshaking signal is asserted. Once the data from the line has been programmed, the following lines are received and programmed in the same way, until the line indicating the end of the file has been received. A success indication ‘S’ is then transmitted out the USART (by the `FileEnd` routine) and the boot code waits for a Reset.

Programming the FLASH Program Memory

Data is written to the FLASH program memory using special function registers. The address is written to the EEADR and EEADRH registers and the first two bytes of data are written to EEDATA and EEDATH. The `FlashWrite` routine is then executed, which writes the data into program memory. The address is then incremented and the next two data bytes are written. This is repeated until all the data from the line of the hex file has been programmed into the FLASH program memory.

Error Handling

There are several things that can go wrong during execution of the boot code or user code, and a number of these error conditions are handled by the boot code. If an error occurs, the boot code traps it by executing an infinite loop, until the user intervenes and resets the device. If an error is detected in the incoming data, then a failure indication ‘F’ is transmitted. This does not occur in the case of an overflow error, or if the data transmission is halted.

If the bootloader is being used for the first time, or if the user code is partially programmed because of a previous error, there might not be valid user code programmed into the microcontroller. The boot code handles this by writing a status word (`0x3fff`) at a location labeled `CodeStatus,` before programming the FLASH device, and then writing a different status word (`0x0000`) to this same location, when programming of the user code has been completed. The boot code tests this location and only starts execution of the user code, if it sees that the user code was successfully programmed. When the boot code is originally programmed into the PICmicro MCU, the status word indicates that there is not valid user code in the device.

The transfer of data can be interrupted. In this case, the boot code waits, trying to receive more data with no time-out, until the user intervenes and resets the device. Noise, or a temporary interruption, may corrupt incoming data. The Intel hex file includes a checksum on each line and the boot code checks the validity of each line by verifying the checksum.

Incorrect use of flow control can result in data being sent to the PIC16F877, while it is not ready to receive data. This can cause an overrun error in the USART. Once an overrun has occurred, the USART will not move any new data into the receive FIFO and the boot code will be stuck in a loop waiting for more data. This effectively traps the error until the user intervenes by resetting the device.

If the user starts transmitting a hex file before the boot code is running, the boot code may miss the first lines of the file. Since all the lines of a hex file have the same format, it is not normally possible to determine whether the line being received is the first line of the hex file. However, since MPASM generates hex files with addresses in ascending order, the first valid line of the hex file should contain the code for the Reset vector which is checked by the boot code.

The user code may try to use program memory locations that contain boot code. This is detected by checking the address being programmed and detecting conflicting addresses. The boot code will not overwrite itself and is not code protected.

Faulty user code, or noise that corrupts the program counter, can cause execution to jump to an unprogrammed memory location and possibly run into the start of the boot code. The first instructions in the boot code are an infinite loop that traps execution into the boot code area.

Because the first four instructions in program memory are relocated in the boot code implementation, there must be a program branch within these four instructions. If there is no program branch, then execution is trapped by the boot code.

Using the Bootloader

The procedure for using the bootloader is as follows:

- On the PC, set up the serial port baud rate and flow control (hardware handshaking).
- Connect the serial port of the PIC16F87X device to the serial port of the PC.
- Press the switch to pull pin RB0 low.
- Power up the board to start the boot code running.
- The switch on RB0 can be released if desired.
- From the PC, send the hex file to the serial port.
- A period ‘.’ will be received from the serial port for each line of the hex file that is sent.
- An ‘S’ or ‘F’ will be received to indicate success or failure.
- The user must handle a failure by resetting the board and starting over.
- Release the switch to set pin RB0 high.
- Power-down the board and power it up to start the user code running.

On the PC, there are several ways to set up the serial port and to transfer data. This also differs between operating systems.

A terminal program allows the user to set up and send data to a serial port. In most terminal programs, an ASCII or text file can be sent and this option should be used to send the hex file. A terminal program will also show data received on the serial port and this allows the user to see the progress ‘.’ indicators and the success ‘S’ or failure ‘F’ indicators. There are many terminal programs available, some of which are available free on the Internet. This boot code was tested using Tera Term Pro, Version 2.3. The user should be aware that some popular terminal programs contain bugs.

 © 2000 Microchip Technology Inc.

A serial port can be set up in a DOS window, using the MODE command and a file can be copied to a serial port, using the COPY command. When using Windows® 95/98, the MODE command does not allow the handshaking signals to be configured. This makes it difficult to use the COM port in DOS. When using Windows NT® or Windows 2000®, the following commands can be used to send a hex file named `filename.hex` to serial port COM1:

```
MODE COM1: BAUD=9600 PARITY=N DATA=8
STOP=1   to=off   xon=off   odsr=off
octs=on dtr=off rts=on idsr=off

COPY filename.hex COM1:
```

Resources Used

The boot code coexists with the user code on the PIC16F877 and many of the resources used by the boot code can also be used by the user code. The boot code uses the resources listed in Table 1.

TABLE 1: RESOURCES USED BY THE BOOT CODE

Resource	Amount
Program memory	224 words
Data memory	72 bytes
I/O pins	5 pins
Peripherals	USART

The program memory used by the boot code cannot be used for user code, although it is possible to call some of the subroutines implemented in the boot code to save space. The user code can use all the data memory.

The USART can be used by the user code with the two I/O pins for the USART and the I/O pins used for handshaking. The I/O pin used to indicate that the boot code should load new user code, is connected to a switch or jumper. This can be isolated with a resistor and used as an output, so that it is possible to use all the I/O pins used by the bootloader.

In summary, all resources used by the boot code, except program memory, can also be used by the user code.

CONCLUSION

Using a bootloader is an efficient way to allow firmware upgrades in the field. Less than 3% of the total program memory is used by the boot code and the entire program memory available on a PIC16F877 can be programmed in less than one minute at 19,200 baud.

The cost of fixing code bugs can be reduced with a bootloader. Products can be upgraded with new features in the field, adding value and flexibility to the products. The ability to upgrade in the field is an added feature and can enhance the value of a product.

© 2000 Microchip Technology Inc.

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the “Company”) for its PICmicro® Microcontroller is intended and supplied to you, the Company’s customer, for use solely and exclusively on Microchip PICmicro Microcontroller products.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

APPENDIX A: SOURCE CODE – FILE BOOT877.ASM

```
MPASM 02.40 Released          BOOT877.ASM   6-26-2000  14:58:44         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

;=============================================================================
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the "Company") for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
; Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;=============================================================================
```

 © 2000 Microchip Technology Inc.

```
                  00023 ;       Filename:       boot877.asm
                  00024 ;==============================================================================
                  00025 ;       Author:         Mike Garbutt
                  00026 ;       Company:        Microchip Technology Inc.
                  00027 ;       Revision:       1.00
                  00028 ;       Date:           26 June 2000
                  00029 ;       Assembled using MPASM V2.40
                  00030 ;==============================================================================
                  00031 ;       Include Files:  p16f877.inc     V1.00
                  00032 ;==============================================================================
                  00033 ;       Boot code to receive a hex file containing user code from a
                  00034 ;       serial port and write it to program memory. Tests a pin to see
                  00035 ;       if code should be downloaded. Receives hex file using USART and
                  00036 ;       hardware handshaking. Does error checking on data and writes to
                  00037 ;       program memory. Waits for reset and then starts user code running.
                  00038 ;==============================================================================
                  00039
                  00040         list p=16f877, st=OFF, x=OFF, n=0
                  00041         errorlevel -302
                  00042         #include <p16f877.inc>
                  00001         LIST
                  00002 ; P16F877.INC  Standard Header File, Version 1.00    Microchip Technology, Inc.
                  00370         LIST
                  00043
2007   3F31       00044         __CONFIG _BODEN_OFF & _CP_OFF & _PWRTE_ON & _WDT_OFF & _WRT_ENABLE_ON & _XT_OSC & _DEBUG_OFF & _
                        CPD_OFF & _LVP_OFF
                  00045
                  00046 ;------------------------------------------------------------------------------
                  00047 ;Constants
                  00048
  00000000        00049 TEST_INPUT      EQU     0       ;Port B Pin 0 input indicates download
  00000001        00050 RTS_OUTPUT      EQU     1       ;Port B Pin 1 output for flow control
  00000002        00051 CTS_INPUT       EQU     2       ;Port B Pin 2 input for flow control
                  00052
  00000019        00053 BAUD_CONSTANT   EQU     0x19    ;Constant for baud generator for 9600 baud
                  00054 ;BAUD_CONSTANT  EQU     0x0c    ;Constant for baud generator for 19200 baud
                  00055                                 ;Fosc is 4MHz
                  00056
                  00057 ;------------------------------------------------------------------------------
                  00058 ;Variables in bank0
                  00059
                  00060                 CBLOCK  0x20
  00000020        00061                 AddressH:       1       ;flash program memory address high byte
  00000021        00062                 AddressL:       1       ;flash program memory address low byte
  00000022        00063                 NumWords:       1       ;number of words in line of hex file
  00000023        00064                 Checksum:       1       ;byte to hold checksum of incoming data
  00000024        00065                 Counter:        1       ;to count words being saved or programmed
```

```
00000025                            TestByte:       1       ;byte to show reset vector code received
00000026                            HexByte:        1       ;byte from 2 incoming ascii characters
00000027                            DataPointer:    1       ;pointer to data in buffer
00000028                            DataArray:      0x40    ;buffer for storing incoming data
                                    ENDC

                  ;------------------------------------------------------------------------------
                  ;Macros to select the register bank
                  ;Many bank changes can be optimised when only one STATUS bit changes

                  Bank0             MACRO                   ;macro to select data RAM bank 0
                                    bcf     STATUS,RP0
                                    bcf     STATUS,RP1
                                    ENDM

                  Bank1             MACRO                   ;macro to select data RAM bank 1
                                    bsf     STATUS,RP0
                                    bcf     STATUS,RP1
                                    ENDM

                  Bank2             MACRO                   ;macro to select data RAM bank 2
                                    bcf     STATUS,RP0
                                    bsf     STATUS,RP1
                                    ENDM

                  Bank3             MACRO                   ;macro to select data RAM bank 3
                                    bsf     STATUS,RP0
                                    bsf     STATUS,RP1
                                    ENDM

                  ;==============================================================================
                  ;Reset vector code

0000                                ORG     0x0000

0000   301F       ResetVector:      movlw   high Main
0001   008A                         movwf   PCLATH          ;set page bits for page3
Message[306]: Crossing page boundary -- ensure page bits are set.
0002   2F2C                         goto    Main            ;go to boot loader

                  ;==============================================================================
                  ;Start of boot code in upper memory traps accidental entry into boot code area

1F20                                ORG     0x1f20          ;Use last part of page3 for PIC16F876/7
                  ;                 ORG     0x0f20          ;Use last part of page1 for PIC16F873/4
                  ;                 ORG     0x0720          ;Use last part of page0 for PIC16F870/1

```

 © 2000 Microchip Technology Inc.

```
1F20   301F     StartOfBoot:    movlw   high TrapError  ;trap if execution runs into boot code
1F21   008A                     movwf   PCLATH          ;set correct page
1F22   2F22     TrapError:      goto    TrapError       ;trap error and wait for reset

                ;-----------------------------------------------------------------------------
                ;Relocated user reset code to jump to start of user code
                ;Must be in bank0 before jumping to this routine

1F23   018A     StartUserCode:  clrf    PCLATH          ;set correct page for reset condition
1F24   0000                     nop                     ;relocated user code replaces this nop
1F25   0000                     nop                     ;relocated user code replaces this nop
1F26   0000                     nop                     ;relocated user code replaces this nop
1F27   0000                     nop                     ;relocated user code replaces this nop
1F28   301F                     movlw   high TrapError1 ;trap if no goto in user reset code
1F29   008A                     movwf   PCLATH          ;set correct page
1F2A   2F2A     TrapError1:     goto    TrapError1      ;trap error and wait for reset

                ;-----------------------------------------------------------------------------
                ;Program memory location to show whether valid code has been programmed

1F2B   3FFF     CodeStatus:     DA      0x3fff          ;0 for valid code, 0x3fff for no code

                ;-----------------------------------------------------------------------------
                ;Main boot code routine
                ;Tests to see if a load should occur and if valid user code exists

                Main:           Bank0                   ;change to bank0 in case of soft reset
1F2E   1C06                     btfss   PORTB,TEST_INPUT ;check pin for boot load
1F2F   2F3A                     goto    Loader          ;if low then do bootload
1F30   27B5                     call    LoadStatusAddr  ;load address of CodeStatus word
1F31   27F6                     call    FlashRead       ;read data at CodeStatus location
                                Bank2                   ;change from bank3 to bank2
1F34   088C                     movf    EEDATA,F        ;set Z flag if data is zero
                                Bank0                   ;change from bank2 to bank0
1F37   1D03                     btfss   STATUS,Z        ;test Z flag
1F38   2F38     TrapError2:     goto    TrapError2      ;if not zero then is no valid code
1F39   2F23                     goto    StartUserCode   ;if zero then run user code

                ;-----------------------------------------------------------------------------
                ;Start of routine to load and program new code

1F3A   01A5     Loader:         clrf    TestByte        ;indicate no reset vector code yet

1F3B   27B5                     call    LoadStatusAddr  ;load address of CodeStatus word
1F3C   303F                     movlw   0x3f            ;load data to indicate no program
1F3D   008E                     movwf   EEDATH
1F3E   30FF                     movlw   0xff            ;load data to indicate no program
```

```
1F3F 008C                     movwf   EEDATA
1F40 27EA                     call    FlashWrite       ;write new CodeStatus word

1F41 27CC                     call    SerialSetup      ;set up serial port

          ;-----------------------------------------------------------------------------
          ;Get new line of hex file starting with ':'
          ;Get first 8 bytes after ':' and extract address and number of bytes

1F42 27DB GetNewLine:         call    SerialReceive    ;get new byte from serial port
1F43 3A3A                     xorlw   ':'              ;check if ':' received
1F44 1D03                     btfss   STATUS,Z
1F45 2F42                     goto    GetNewLine       ;if not then wait for next byte

1F46 01A3                     clrf    Checksum         ;start with checksum zero

1F47 27BC                     call    GetHexByte       ;get number of program data bytes in line
1F48 391F                     andlw   0x1F             ;limit number in case of error in file
1F49 00A2                     movwf   NumWords
1F4A 1003                     bcf     STATUS,C
1F4B 0CA2                     rrf     NumWords,F       ;divide by 2 to get number of words

1F4C 27BC                     call    GetHexByte       ;get upper half of program start address
1F4D 00A0                     movwf   AddressH

1F4E 27BC                     call    GetHexByte       ;get lower half of program start address
1F4F 00A1                     movwf   AddressL

1F50 1003                     bcf     STATUS,C
1F51 0CA0                     rrf     AddressH,F       ;divide address by 2 to get word address
1F52 0CA1                     rrf     AddressL,F

1F53 27BC                     call    GetHexByte       ;get record type
1F54 3A01                     xorlw   0x01
1F55 1903                     btfsc   STATUS,Z         ;check if end of file record (0x01)
1F56 2FAB                     goto    FileDone         ;if end of file then all done

1F57 0826                     movf    HexByte,W
1F58 3A00                     xorlw   0x00
1F59 1D03                     btfss   STATUS,Z         ;check if regular line record (0x00)
1F5A 2FA8                     goto    LineDone         ;if not then ignore line and send '.'

1F5B 30E0                     movlw   0xe0
1F5C 0720                     addwf   AddressH,W       ;check if address < 0x2000
1F5D 1803                     btfsc   STATUS,C         ;which is ID locations and config bits
1F5E 2FA8                     goto    LineDone         ;if so then ignore line and send '.'
```


© 2000 Microchip Technology Inc.

```
                    ;-----------------------------------------------------------------------------
                    ;Get data bytes and checksum from line of hex file

1F5F   3028                            movlw   DataArray
1F60   0084                            movwf   FSR               ;set pointer to start of array
1F61   0822                            movf    NumWords,W
1F62   00A4                            movwf   Counter           ;set counter to number of words

1F63   27BC         GetData:           call    GetHexByte        ;get low data byte
1F64   0080                            movwf   INDF              ;save in array
1F65   0A84                            incf    FSR,F             ;point to high byte

1F66   27BC                            call    GetHexByte        ;get high data byte
1F67   0080                            movwf   INDF              ;save in array
1F68   0A84                            incf    FSR,F             ;point to next low byte

1F69   0BA4                            decfsz  Counter,F
1F6A   2F63                            goto    GetData

1F6B   27BC                            call    GetHexByte        ;get checksum
1F6C   0823                            movf    Checksum,W        ;check if checksum correct
1F6D   1D03                            btfss   STATUS,Z
1F6E   2FB2                            goto    ErrorMessage

1F6F   1486                            bsf     PORTB,RTS_OUTPUT ;set RTS off to stop data being received

                    ;-----------------------------------------------------------------------------
                    ;Get saved data one word at a time to program into flash

1F70   3028                            movlw   DataArray
1F71   0084                            movwf   FSR               ;point to start of array
1F72   0822                            movf    NumWords,W
1F73   00A4                            movwf   Counter           ;set counter to half number of bytes

                    ;-----------------------------------------------------------------------------
                    ;Check if address is in reset code area

1F74   0820         CheckAddress:      movf    AddressH,W        ;checking for boot location code
1F75   1D03                            btfss   STATUS,Z          ;test if AddressH is zero
1F76   2F84                            goto    CheckAddress1     ;if not go check if reset code received

1F77   30FC                            movlw   0xfc
1F78   0721                            addwf   AddressL,W        ;add 0xfc (-4) to address
1F79   1803                            btfsc   STATUS,C          ;no carry means address < 4
1F7A   2F84                            goto    CheckAddress1     ;if not go check if reset code received

1F7B   1425                            bsf     TestByte,0        ;show that reset vector code received
```

```
1F7C   0821                     movf    AddressL,W      ;relocate addresses 0-3 to new location
1F7D   3E24                     addlw   low (StartUserCode + 1) ;add low address to new location
                                Bank2                   ;change from bank0 to bank2
1F80   008D                     movwf   EEADR           ;load new low address
1F81   301F                     movlw   high (StartUserCode + 1) ;get new location high address
1F82   008F                     movwf   EEADRH          ;load high address
1F83   2F9A                     goto    LoadData        ;go get data byte and program into flash

              ;-----------------------------------------------------------------------------
              ;Check if reset code has been received
              ;Check if address is too high and conflicts with boot loader

1F84   1C25   CheckAddress1:    btfss   TestByte,0      ;check if reset vector code received first
1F85   2FB2                     goto    ErrorMessage    ;if not then error

1F86   301F                     movlw   high StartOfBoot ;get high byte of address
1F87   0220                     subwf   AddressH,W
1F88   1C03                     btfss   STATUS,C        ;test if less than boot code address
1F89   2F90                     goto    LoadAddress     ;yes so continue with write
1F8A   1D03                     btfss   STATUS,Z        ;test if equal to boot code address
1F8B   2FB2                     goto    ErrorMessage    ;no so error in high byte of address

1F8C   3020                     movlw   low StartOfBoot ;get low byte of address
1F8D   0221                     subwf   AddressL,W
1F8E   1803                     btfsc   STATUS,C        ;test if less than boot code address
1F8F   2FB2                     goto    ErrorMessage    ;no so error in address

              ;-----------------------------------------------------------------------------
              ;Load address and data and write data into flash

1F90   0820   LoadAddress:      movf    AddressH,W      ;get high address
                                Bank2                   ;change from bank0 to bank2
1F93   008F                     movwf   EEADRH          ;load high address
                                Bank0                   ;change from bank2 to bank0
1F96   0821                     movf    AddressL,W      ;get low address
                                Bank2                   ;change from bank0 to bank2
1F99   008D                     movwf   EEADR           ;load low address

1F9A   0800   LoadData:         movf    INDF,W          ;get low byte from array
1F9B   008C                     movwf   EEDATA          ;load low byte
1F9C   0A84                     incf    FSR,F           ;point to high data byte
1F9D   0800                     movf    INDF,W          ;get high byte from array
1F9E   008E                     movwf   EEDATH          ;load high byte
1F9F   0A84                     incf    FSR,F           ;point to next low data byte

1FA0   27EA                     call    FlashWrite      ;write data to program memory
```


© 2000 Microchip Technology Inc.

```
                         Bank0                    ;change from bank3 to bank0
1FA3  0FA1               incfsz  AddressL,F       ;increment low address byte
1FA4  2FA6               goto    CheckLineDone    ;check for rollover
1FA5  0AA0               incf    AddressH,F       ;if so then increment high address byte

1FA6  0BA4      CheckLineDone:  decfsz  Counter,F ;check if all words have been programmed
1FA7  2F74               goto    CheckAddress     ;if not then go program next word

      ;-----------------------------------------------------------------------------
      ;Done programming line of file

1FA8  302E      LineDone:       movlw   '.'              ;line has been programmed so
1FA9  27E2               call    SerialTransmit   ;transmit progress indicator back
1FAA  2F42               goto    GetNewLine       ;go get next line hex file

      ;-----------------------------------------------------------------------------
      ;Done programming file so send success indicator and trap execution until reset

1FAB  3053      FileDone:       movlw   'S'              ;programming complete so
1FAC  27E2               call    SerialTransmit   ;transmit success indicator back

1FAD  27B5               call    LoadStatusAddr   ;load address of CodeStatus word
1FAE  018E               clrf    EEDATH           ;load data to indicate program exists
1FAF  018C               clrf    EEDATA           ;load data to indicate program exists
1FB0  27EA               call    FlashWrite
1FB1  2FB1      TrapFileDone:   goto    TrapFileDone     ;all done so wait for reset

      ;-----------------------------------------------------------------------------
      ;Error in hex file so send failure indicator and trap error

1FB2  3046      ErrorMessage:   movlw   'F'              ;error occurred so
1FB3  27E2               call    SerialTransmit   ;transmit failure indicator back
1FB4  2FB4      TrapError3:     goto    TrapError3       ;trap error and wait for reset

      ;-----------------------------------------------------------------------------
      ;Load address of CodeStatus word into flash memory address registers
      ;This routine returns in bank2

                LoadStatusAddr: Bank2                    ;change from bank0 to bank2
1FB7  301F               movlw   high CodeStatus  ;load high addr of CodeStatus location
1FB8  008F               movwf   EEADRH
1FB9  302B               movlw   low CodeStatus   ;load low addr of CodeStatus location
1FBA  008D               movwf   EEADR
1FBB  0008               return

      ;-----------------------------------------------------------------------------
      ;Receive two ascii digits and convert into one hex byte
```

```
                  ;This routine returns in bank0

1FBC   27DB       GetHexByte:     call    SerialReceive   ;get new byte from serial port
1FBD   3EBF                       addlw   0xbf            ;add -'A' to Ascii high byte
1FBE   1C03                       btfss   STATUS,C        ;check if positive
1FBF   3E07                       addlw   0x07            ;if not, add 17 ('0' to '9')
1FC0   3E0A                       addlw   0x0a            ;else add 10 ('A' to 'F')
1FC1   00A6                       movwf   HexByte         ;save nibble
1FC2   0EA6                       swapf   HexByte,F       ;move nibble to high position

1FC3   27DB                       call    SerialReceive   ;get new byte from serial port
1FC4   3EBF                       addlw   0xbf            ;add -'A' to Ascii low byte
1FC5   1C03                       btfss   STATUS,C        ;check if positive
1FC6   3E07                       addlw   0x07            ;if not, add 17 ('0' to '9')
1FC7   3E0A                       addlw   0x0a            ;else add 10 ('A' to 'F')
1FC8   04A6                       iorwf   HexByte,F       ;add low nibble to high nibble
1FC9   0826                       movf    HexByte,W       ;put result in W reg
1FCA   07A3                       addwf   Checksum,F      ;add to cumulative checksum
1FCB   0008                       return

                  ;-----------------------------------------------------------------------------
                  ;Set up USART for asynchronous comms
                  ;Routine is only called once and can be placed in-line saving a call and return
                  ;This routine returns in bank0

                  SerialSetup:    Bank0                   ;change from bank3 to bank0
1FCE   1486                       bsf     PORTB,RTS_OUTPUT ;set RTS off before setting as output
                                  Bank1                   ;change from bank0 to bank1
1FD1   1086                       bcf     TRISB,RTS_OUTPUT ;enable RTS pin as output
1FD2   3019                       movlw   BAUD_CONSTANT   ;set baud rate 9600 for 4Mhz clock
1FD3   0099                       movwf   SPBRG
1FD4   1518                       bsf     TXSTA,BRGH      ;baud rate high speed option
1FD5   1698                       bsf     TXSTA,TXEN      ;enable transmission
                                  Bank0                   ;change from bank1 to bank0
1FD8   1618                       bsf     RCSTA,CREN      ;enable reception
1FD9   1798                       bsf     RCSTA,SPEN      ;enable serial port
1FDA   0008                       return

                  ;-----------------------------------------------------------------------------
                  ;Wait for byte to be received in USART and return with byte in W
                  ;This routine returns in bank0

                  SerialReceive:  Bank0                   ;change from unknown bank to bank0
1FDD   1086                       bcf     PORTB,RTS_OUTPUT ;set RTS on for data to be received
1FDE   1E8C                       btfss   PIR1,RCIF       ;check if data received
1FDF   2FDE                       goto    $-1             ;wait until new data
1FE0   081A                       movf    RCREG,W         ;get received data into W
```


© 2000 Microchip Technology Inc.

```
1FE1   0008                        return

                  ;---------------------------------------------------------------------------
                  ;Transmit byte in W register from USART
                  ;This routine returns in bank0

                  SerialTransmit:  Bank0                       ;change from unknown bank to bank0
1FE4   1906                        btfsc   PORTB,CTS_INPUT ;check CTS to see if data can be sent
1FE5   2FE4                        goto    $-1
1FE6   1E0C                        btfss   PIR1,TXIF       ;check that buffer is empty
1FE7   2FE6                        goto    $-1
1FE8   0099                        movwf   TXREG           ;transmit byte
1FE9   0008                        return

                  ;---------------------------------------------------------------------------
                  ;Write to a location in the flash program memory
                  ;Address in EEADRH and EEADR, data in EEDATH and EEDATA
                  ;This routine returns in bank3

                  FlashWrite:      Bank3                       ;change from bank2 to bank3
1FEC   3084                        movlw   0x84            ;enable writes to program flash
1FED   008C                        movwf   EECON1

1FEE   3055                        movlw   0x55            ;do timed access writes
1FEF   008D                        movwf   EECON2
1FF0   30AA                        movlw   0xaa
1FF1   008D                        movwf   EECON2
1FF2   148C                        bsf     EECON1,WR       ;begin writing to flash

1FF3   0000                        nop                     ;processor halts here while writing
1FF4   0000                        nop
1FF5   0008                        return

                  ;---------------------------------------------------------------------------
                  ;Read from a location in the flash program memory
                  ;Address is in EEADRH and EEADR, data returned in EEDATH and EEDATA
                  ;Routine is only called once and can be placed in-line saving a call and return
                  ;This routine returns in bank3 and is called when in bank2

1FF6   301F       FlashRead:       movlw   0x1f            ;keep address within range
1FF7   058F                        andwf   EEADRH,F

                                   Bank3                   ;change from bank2 to bank3
1FFA   3080                        movlw   0x80            ;enable reads from program flash
1FFB   008C                        movwf   EECON1

1FFC   140C                        bsf     EECON1,RD       ;read from flash
```

```
                      00441
1FFD   0000           00442                 nop                     ;processor waits while reading
1FFE   0000           00443                 nop
1FFF   0008           00444                 return
                      00445
                      00446 ;-----------------------------------------------------------------------------
                      00447
                      00448                 END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXX------------- ---------------- ---------------- ----------------
1F00 : ---------------- ---------------- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
1F40 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
1F80 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
1FC0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
2000 : -------X-------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   227
Program Memory Words Free:  7965


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     1 reported,    24 suppressed
```

  © 2000 Microchip Technology Inc.

APPENDIX B: HEX FILE FORMAT

MPASM generates an 8-bit Intel hex file (INHX8M) by default. The lines of this hex file all have the following format:

```
:BBAAAATTHHHH….HHCC
```

A colon precedes each line and is followed by hexadecimal digits in ASCII format.

`BB` is a 2-digit hexadecimal byte count representing the number of data bytes that will appear on the line. This is a number from `0x00` to `0x10` and is always even because the PIC16F87X parts have a 14-bit wide memory and use two bytes for every program memory word.

`AAAA` is a 4-digit hexadecimal address representing the starting byte address of the data bytes that follow. To get the actual program memory word address, the byte address must be divided by two.

`TT` is a 2-digit hexadecimal record type that indicates the meaning of the data on the line. It is `0x00` for a regular data record and `0x01` for an end of file record. The boot code ignores all other record types.

`HH` are 2-digit hexadecimal data bytes that correspond to addresses, incrementing sequentially from the starting address earlier in the line. These bytes come in low byte, high byte pairs, corresponding to each 14-bit program memory word.

`CC` is a 2-digit hexadecimal checksum byte, such that the sum of all bytes in the line including the checksum, is a multiple of 256. The initial colon is ignored.

The code in Example B-1 will generate a line in a hex file as shown in Figure B-1.

EXAMPLE B-1: CODE TO GENERATE A HEX FILE

```
ORG        0x17A

movlw      0xFF
movwf      PORTB
bsf        STATUS,RP0
movwf      TRISA
clrf       TRISB
bcf        STATUS,RP0
```

FIGURE B-1: LINE OF HEX FILE

```
:0C02F400FF308600831685008601831200F
```

Checksum is `0x0F`

```
0x0C + 0x02 + 0xF4 + 0x00 + 0xFF + 0x30 +
0x86 + 0x00 + 0x83 + 0x16 + 0x85 + 0x00 +
0x86 + 0x01 + 0x83 + 0x12 + 0x0F = 0x0500
```

Result of addition(1) mod `256` is zero

Second program memory word is `0x0086`

This corresponds to an instruction `MOVWF 0x06`(2)

First program memory word is `0x30FF`

This corresponds to an instruction `MOVLW 0xFF`

Record type is `0x00` indicating a regular data record

Address of first program memory word is `0x02F4` ÷ `2` = `0x017A`

Number of data bytes is `0x0C`

Number of program memory words is `0x0C` ÷ `2` = `0x06`

Note 1: The calculation to test the checksum adds every byte (pair of digits) in the line of the hex file, including the checksum itself.

2: The label PORTB is defined as `0x06` in the standard include file for the PIC16F877.

 Preliminary © 2000 Microchip Technology Inc.

APPENDIX C: RS-232 HARDWARE HANDSHAKING SIGNALS

Understanding hardware flow control can be confusing, because of the terminology used and the slightly different way that handshaking is now implemented, compared to the original specification.

RS-232 hardware handshaking was specified in terms of communication between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE). The DTE (e.g., computer terminal) was always faster than the DCE (e.g., modem) and could receive data without interruption. The hardware handshaking protocol required that the DTE would request to send data to the DCE (with the request to send RTS signal) and that the DCE would then indicate to the DTE that it was cleared to send data (with the clear to send CTS signal). Both RTS and CTS were, therefore, used to control data flow from the DTE to the DCE.

The Data Terminal Ready (DTR) signal was defined so that the DTE could indicate to the DCE that it was attached and ready to communicate. The Data Set Ready (DSR) signal was defined to enable the DCE to indicate to the DTE that it was attached and ready to communicate. These are higher level signals not generally used for byte by byte control of data flow, although they can be used for this purpose.

Most RS-232 connections use 9-pin DSUB connectors. A DTE uses a male connector and a DCE uses a female connector. The signal names are always in terms of the DTE, so the RTS pin on the female connector of the DCE is an input and is the RTS signal from the DTE.

Over time, the clear distinction between the DTE and DCE has been lost. In many instances, two DTE devices are connected together. In other cases, the DCE device is able to send data at a rate that is too high for the DTE to receive continuously. In practice, the DTR output of the DTE has come to be used to control the flow of data to the DTE and now indicates that the DCE (or other DTE) may send data. It no longer indicates a request to send data to the DCE.

It is common for a DTE to be connected to another DTE (e.g., two computers), and in this case, they will both have male connectors and the cable between them will have two female connectors. This is known as a null modem cable. The cable is usually wired in such a way that each DTE looks like a DCE to the other DTE. To achieve this, the RTS output of one DTE is connected to the CTS input of the other DTE and vice versa. Each DTE device will use its RTS output to allow the other DTE device to transmit data and will check its CTS input to determine whether it is allowed to transmit data.

FIGURE C-1: DTE TO DCE CONNECTION

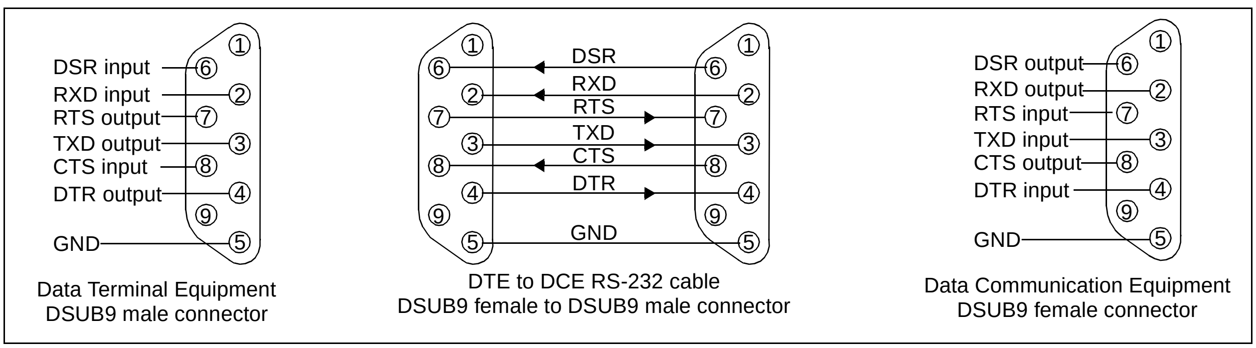
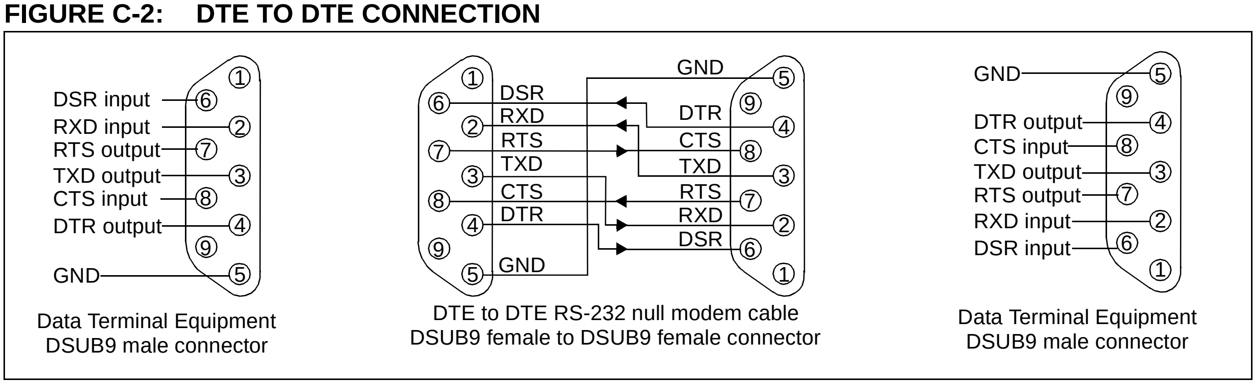


FIGURE C-2: DTE TO DTE CONNECTION



Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rfPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

DNV Certification, Inc.
USA

ANSI * RAB
QMS
ACCREDITED

DNV

DNV MSC
The Netherlands
Accredited by the RvA

ISO 9001 / QS-9000
REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

© 2002 Microchip Technology Inc.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/18/02

© 2002 Microchip Technology Inc.