AN750

Self-Programming the PIC18C452 OTP

Author: *Tim Rovnak*
Microchip Technology Incorporated

INTRODUCTION

You've decided on the Microchip PIC18C452 8-bit microcontroller. Its ample program memory space of 32 Kbytes, operating speed of 40 MHz, and extensive set of peripherals and I/Os fit your design perfectly. Your application is small to medium in volume and so you choose the PIC18C452 OTP. It is a standard product with flexible quantities and short lead times. You can appreciate an inexpensive solution that allows for the most recent firmware and yet maintains a rapid time to market. Your decision has the confidence of being in familiar territory; at some point in time, you've tested your design by inserting a programmed OTP part. The big question you now ask yourself is, "How should we program all those parts we're about to buy?"

Here are your options to program the PIC18C452 OTP:

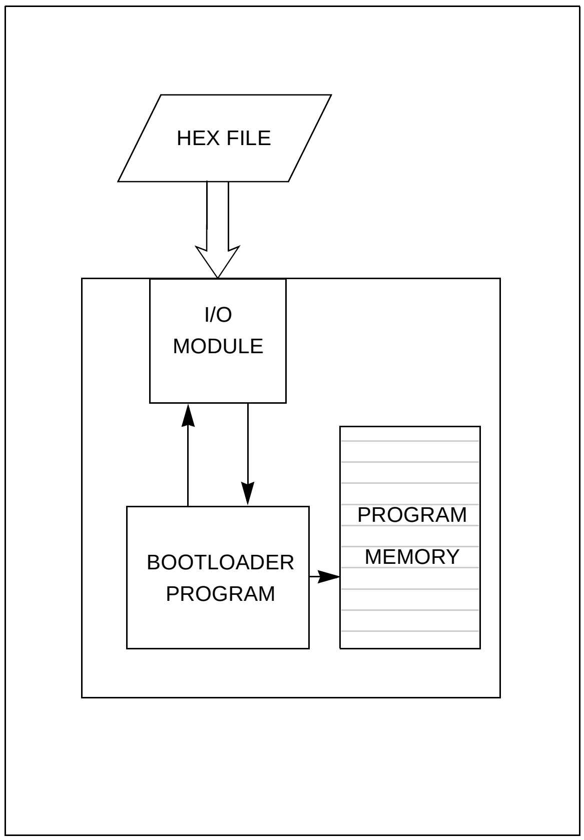
1. Microchip programs the part with user code before shipping to customer. This is called Quick-Turn-Programming (QTP). The cost and time required for this service is minimal.
2. Program blank parts in the production line using a programmer and assemble programmed parts into hardware. This allows more flexibility than QTP parts in terms of firmware design changes.
3. Implement In-Circuit Serial Programming™ (ICSP™). This allows the customer to assemble boards with a blank device and program the microcontroller just before shipping. This method has a minor impact on hardware design. To accomplish programming the user code, special equipment and software are necessary.
4. Implement self-programming capability. This involves a two-step programming process. The first step is to program a bootloader into the device. This can be accomplished by an in-house programmer, or by Microchip via QTP. The board is then assembled with the bootloader programmed device. Using the bootloader, the user code is then programmed while in circuit. Like ICSP, this can occur just before shipping the final assembly. The bootloader takes up some program memory space and minimal hardware must be added, but this method is very simple to do. No special equipment or software are necessary to program the user code.

This application note describes how to self-program the PIC18C452 OTP (option 4). It should be noted that not all microcontrollers have this ability. The PIC18C452 can program itself through a feature that uses special instructions called Table Reads and Table Writes.

BOOTLOADER OVERVIEW

The bootloader program is at the heart of a self-programming PIC18C452 application. The function of the bootloader is to process executable lines of code from the outside world and then program them into the memory space from which the CPU fetches instructions. Figure 1 describes the generic environment of the bootloader. When the target memory space is of the EPROM variety, such as in the PIC18C452, the bootloader will only need to program it once. When the bootloader completes its task, the microcontroller is ready to perform its desired function.

FIGURE 1: BOOTLOAD ENVIRONMENT



SELF-PROGRAMMING BOOTLOADER DESIGN STRATEGY

The self-programming bootloader should be designed with as little impact as possible on both the firmware and hardware sides of the project. This bootloader routine must be compact and the parts list small and standard.

The self-programming bootloader is designed with simplicity in mind. The ultimate goal is to provide a reliable method to program the PIC18C452 with standard equipment and protocols available to even the smallest of operations.

Firmware Design

One of the best features a bootloader program can have is transparency. What this means is the user should be able to freely develop code for the PIC18C452, with little concern for the workings of the bootloader. The only real issues should be the small decrease in available program memory and the impact of extra hardware. The designer should not be forced into rearranging placement of the user code and should not have to add any special branching within the user code, just to allow it to run on a bootloader part. The user code should also be able to expect RESET defaults once the bootloader finishes execution. This allows the development of user code to be as clean as possible.

In order to allow the user code to be developed with minimal constraints, the bootloader is designed in the following manner. The bootloader is placed near the end of program memory. Space for four words (eight bytes) is left unprogrammed at the very end. A `GOTO` statement is placed at the RESET vector, forcing execution to the start of the bootloader. When the bootloader executes self-programming, it takes the user code RESET vector and programs it into the four empty spaces at the end. The rest of the code is placed normally. Table 1 describes what happens to the program memory map after installing the bootloader and then programming the user code.

TABLE 1: PROGRAM MEMORY WITH BOOTLOADER AND USER CODE FOR PIC18C452 OTP

Address	Blank Part from Microchip	First Program Bootloader	Final Program User Code
0000h	RESET Vector	RESET Vector - GOTO BOOTLOADER	RESET Vector - GOTO BOOTLOADER
0008h	High Priority Interrupt	High Priority Interrupt	High Priority Interrupt - USER
0018h	Low Priority Interrupt Program Memory	Low Priority Interrupt Program Memory	Low Priority Interrupt - USER Program Memory - USER
7C5Ch 7FF7h		BOOTLOADER PROGRAM	BOOTLOADER PROGRAM
7FF8h 7FFFh			USER RESET VECTOR

 Preliminary

Hardware Design

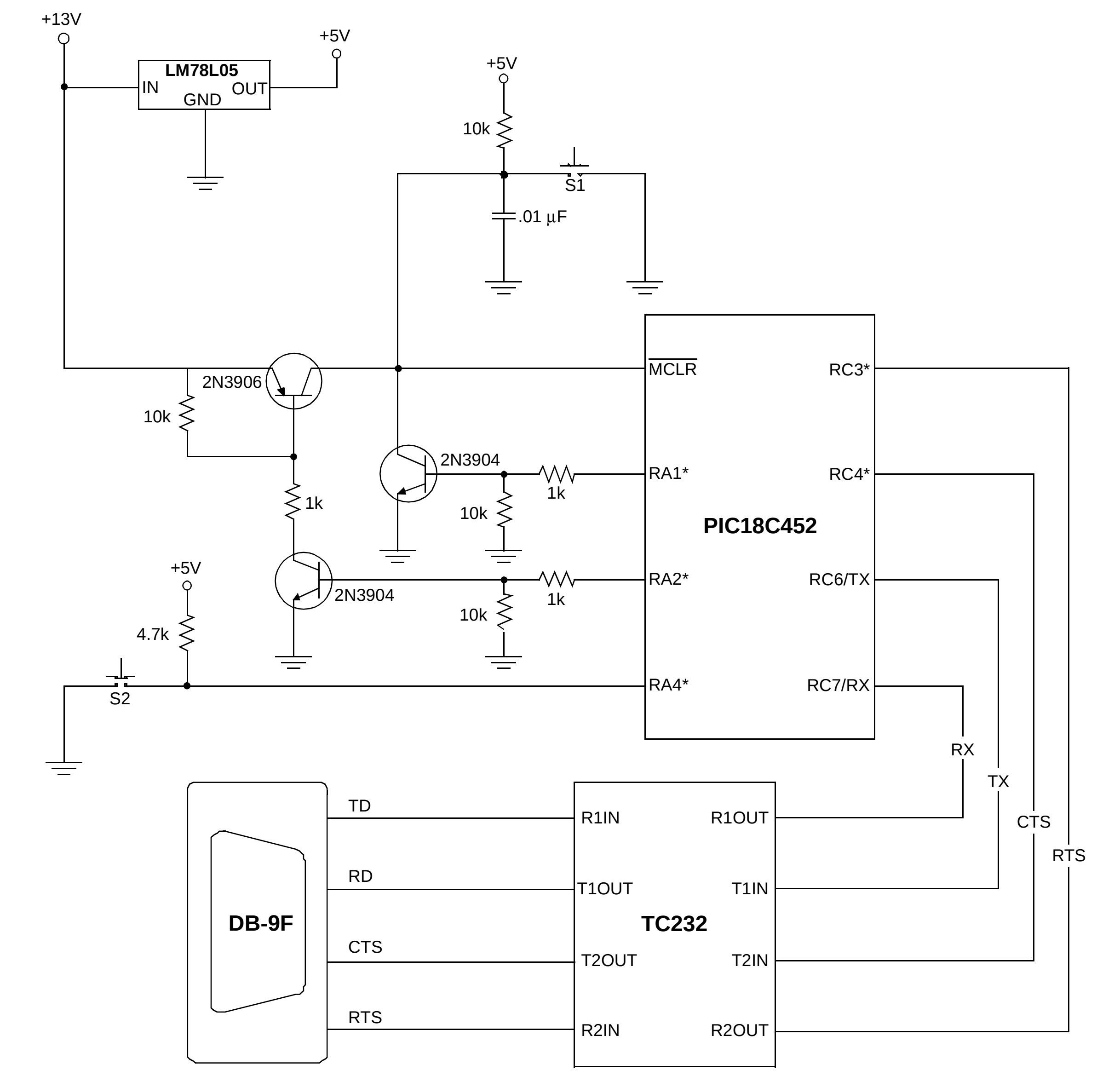
The hardware design for the self-programming boot-loader is based on two criteria:

- Programming power supply
- HEX file transmission method

A 13V power supply (VPP) must be made available to the circuit at appropriate times for programming. This can easily be accomplished by switching transistors controlled by I/O lines on the PIC18C452. The HEX file must be sent via a simple and reliable communication medium.

This can be accomplished through one of the interface modules on the PIC18C452. The choices are I^2C™, SPI™, or USART. The USART was chosen for several reasons. When configured in Asynchronous Receiver mode, it can be used as a standard RS-232 port. Hardware flow control is generated on I/O lines RC3 and RC4 (programmed as RTS and CTS). This is because it is necessary to pause data transmission while the PIC18C452 is busy programming. The USART requires only one additional component, a level shifter. This requirement can be met by a TC232, which is inexpensive and widely available. Also, a PC becomes available as a good download platform, using a serial port and terminal software. Figure 2 shows a guide schematic for the hardware design.

FIGURE 2: SELF-PROGRAMMING THE PIC18C452 OTP GUIDE SCHEMATIC



* Arbitrary, based on application.

IMPLEMENTATION

This section details the firmware and hardware issues and the specific implementations of the self-programming PIC18C452 OTP.

Programming the Bootloader

The bootloader is installed by a programmer. At this time, the configuration bits will need to be set. This is because the self-programming algorithm applies only to user program memory space (addresses 0000h-7FFFh), and the configuration bits are located at addresses 300000h-300006h. Since the configuration bits are based on the hardware design, setting them at this point poses no problem. Last minute firmware changes should not affect the hardware design. The Watchdog Timer Enable bit (CONFIG2H<0>) must be given some extra consideration, however. If the user code requires the Watchdog Timer to be enabled, the bootloader must be modified to accommodate it. This is because the programming pulse cannot be interrupted by anything other than it's intended source, in order to guarantee good programming margins. See the *Long Write* section for more information. An alternative is to disable the Watchdog Timer in the configuration bits and enable it in software by WDTCON<0>.

To Boot or Not To Boot

Upon power-up or RESET, the program execution always vectors to the bootloader. The beginning of the bootloader is located at memory address 7C5Ch. The bootloader first checks for an indication that it should enter the programming part of its code. In this application, push-button S2 provides the indication.

If S2 is not pressed, it is assumed that the part is either already programmed, or the outside world is not quite ready to transmit. In either case, execution will jump to the RESET vector of the user code, 7FF8h, which is located at the end of the bootloader code. In an empty part, there are no user RESET code instructions to execute, so the processor will simply execute `NOPs`, wrap around to 0000h, jump to the top of the bootloader, where it will try again. If the user code is in place, normal RESET vectoring will take execution to the beginning of user code and the bootloader will not be accessed again until another power-up or RESET.

If S2 is pressed on power-up or RESET, the program execution will continue with the bootloader. Before programming any data, however, it must be verified that the part is indeed empty. This is accomplished by reading a particular location in program memory, 7FF6h. If that location has not been previously programmed, the process of receiving and programming data begins. Otherwise, the part is not empty and the bootloader will require user input to proceed any further.

It is important to leave all peripherals and I/O's in their RESET default states before testing S2, because the user code may expect RESET defaults in its execution.

Download Protocol

The protocol to download the HEX file is RS-232 with hardware flow control. CTS (Clear to Send) and RTS (Request to Send) are hardware handshaking lines that become useful in this application. When the PC sends data to the application, it must wait an undefined amount of time while the bootloader programs the cells. RTS tells the microcontroller that the PC would like to send more data. CTS tells the PC that the microcontroller is done programming and is now ready for more data.

CTS and RTS are implemented by PORTC pins, RC3 and RC4, respectively. RC6 and RC7 are configured in USART mode as TX and RX, respectively. The USART module on the PIC18C452 is set to 9600 baud. The terminal software is set to 9600 baud, 8 data bits, no parity, one STOP bit, and hardware control.

The bootloader receives the data one line at a time. Each line is buffered into RAM and a checksum is performed before any programming is done. This is to ensure that the transmission was successful.

HEX File Format

The bootloader expects HEX data in the INHX8M format. Please refer to Appendix A in the MPASM™ User's Guide (DS33014), for more information on HEX file formats. The format of a line of HEX is as follows:

```
:BBAAAATTHHHH...HHHHCC
```

A record begins with a colon '`:`'. The contents of the record are as follows:

- `BB` - # of data bytes
- `AAAA` - starting address of data record
- `TT` - record type (`00` = data, `01` = EOF, `04` = extended address)
- `HHHH` - HEX data word
- `CC` - checksum

The data in a HEX record is in ASCII format; seven bits per character represent a binary number. These characters are converted to binary within the bootloader before programming.

The address range of the INMX8M format is 64 Kbytes.

Programming an EPROM Cell

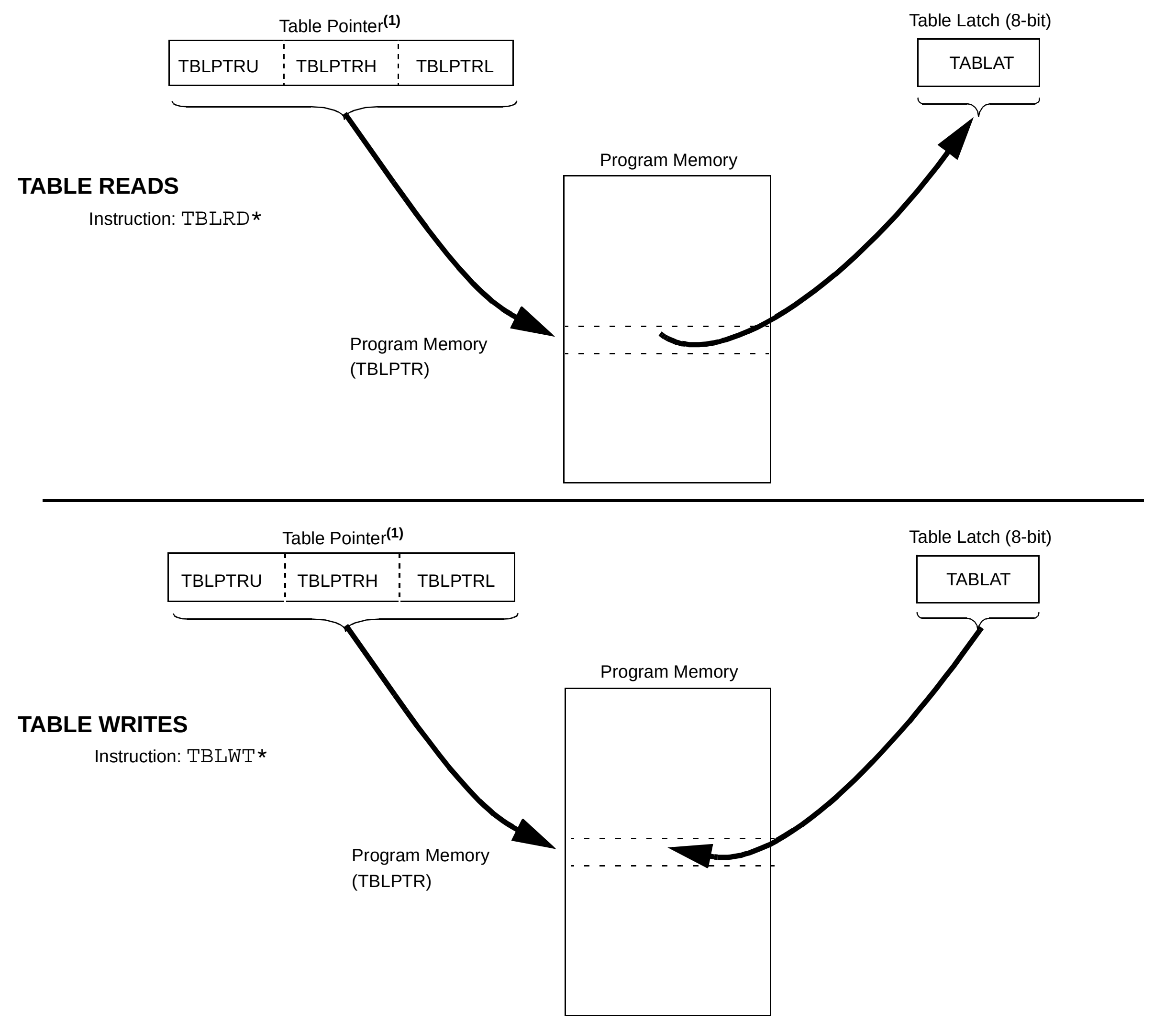
To write an EPROM location, initially apply a programming voltage pulse for the minimum programming time, as defined in the data sheet. For the PIC18C452, the programming voltage is between 12.75V and 13.25V, and the minimum time is 100 µS. After one programming pulse, the respective program memory location is checked. If the data did not program successfully, another program pulse is sent. A maximum of 25 programming pulses may be needed to program a particular program memory word. If, after 25 programming pulses, the word is not successfully programmed, a program failure must be reported.

Over-programming completes the process. This is accomplished by applying the VPP pulse to the memory location three times longer than was determined during the initial write/verify stage. This will ensure a solid programming margin on the EPROM cells.

Table Reads and Table Writes

Table Reads and Writes (`TBLRD`, `TBLWT`) are instructions that move data between data memory space and program memory space. The Table Latch (TABLAT) is an 8-bit register used to hold data during transfers between program memory and data memory. The Table Pointer (TBLPTR) addresses the byte in program memory being read or written (see Figure 3).

FIGURE 3: TABLE READS AND WRITES



Note 1: Table Pointer points to a byte in program memory.

Long Write

Writing to program memory space in the PIC18C452 is accomplished by executing a `TBLWT` instruction as a 'long write'. Program words consist of two bytes and the TABLAT register is only one byte wide. Therefore, two `TBLWTs` are necessary to program one word. When long writes are enabled, and a `TBLWT` is made to an even program memory address (TBLPTR<0>=0), the contents of TABLAT are transferred to a holding register. When a `TBLWT` is made to an odd program memory address (TBLPTR<0>=1), TABLAT is written to that address and the holding register is written to the corresponding even address.

Before executing the long write, it would be a good idea to disable or clear the WDT, so the controller is not unintentionally interrupted while the cells are being programmed.

The basic procedure to perform a long write follows:

1. Set the LWRT bit in the RCON register (see Register 1).
2. Enable one interrupt; this will be used to terminate the long write.
3. Set up the interrupt to trigger at the appropriate time.
4. Drive the $\overline{\text{MCLR}}$/VPP pin to the programming voltage.
5. Execute a `TBLWT` for the lower byte of the word.
6. Execute a `TBLWT` for the upper byte of the word; this initiates the long write.
7. The controller is halted while the long write is executed.
8. The interrupt terminates the long write and execution resumes.
9. $\overline{\text{MCLR}}$/VPP pin may be released back to VDD.
10. Execute a `TBLRD` to verify the memory location.

REGISTER 1: RCON REGISTER (ADDRESS: FD0h)

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
IPEN	LWRT	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (16CXXX Compatibility mode)

bit 6 **LWRT:** Long Write Enable
1 = Enable `TBLWT` to internal program memory
0 = Disable `TBLWT` to internal program memory.

Note: Only cleared on a POR or $\overline{\text{MCLR}}$.
This bit has no effect on `TBLWTs` to external program memory.

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{\text{RI}}$:** `RESET` Instruction Flag bit
1 = No `RESET` instruction occurred
0 = A `RESET` instruction occurred

bit 3 **$\overline{\text{TO}}$:** Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 2 **$\overline{\text{PD}}$:** Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status bit
1 = No Power-on Reset occurred
0 = A Power-on Reset occurred
(must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit
1 = No Brown-out Reset nor POR occurred
0 = A Brown-out Reset or POR occurred
(must be set in software after a Brown-out Reset occurs)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

Status and Errors

During normal operation, the user will receive four basic status messages:

1. A prompt to download the HEX file.
2. A series of periods ('.'), each indicating a successfully programmed line of code.
3. A program success message followed by the maximum write count value.

Note:	The maximum write count value indicates whether any cells required more than one programming pulse. If this value is greater than '1', it is suggested to verify the programming pulse period and the power supply.

4. A prompt to initiate a RESET, which clears the Long Write bit (LWRT) and begins user code execution.

In the case of an unsuccessful bootload, the following error messages are transmitted:

- **Not Empty** – Before programming, it was determined that the part was not empty. User input is requested to proceed with programming.
- **Checksum Error** – A line of the HEX file was received but does not match its checksum. Either the HEX file is incorrect, or the transmission was faulty. The bootloader reports the address of the HEX line of code that caused the checksum error.
- **Program Error** – Using standard programming procedure, a cell was unable to program correctly. The bootloader provides the address of the bad program memory location before halting.
- **Overwrite Condition** – The HEX file is too big to fit into available program memory. Bootloader halts.
- **Overrun Error** – During transmission, the USART reported an overrun. Because data may have been lost, the bootloader halts.

Cutting Corners

This bootloader contains many features that will be useful in getting a system up and running. Once all the system issues have been resolved, it may be appropriate to free up some resources.

Additional program memory can be gained by reducing the size and number of error messages returned to the user. Calls and returns can be replaced by in-line code. The empty part check can also be removed. In this case, a generic program error will be the only indication of a problem. Further, RA1 (the self-reset line) can be set free, if programming and system test stations are in different locations.

The code, in its present form, occupies 930 bytes (465 words) of program memory. Adopting the corner cutting methods above should free up an additional 200-300 bytes, reducing bootloader program memory usage to 2 percent.

SOURCE CODE

Appendix A contains flow charts for the bootloader program code.

Appendix B contains the actual code.

CONCLUSION

There are many ways to implement a self-programming algorithm for the PIC18C452 OTP. Design requirements and production resources will dictate the best method. The material presented here offers a simple and reliable, yet debug friendly solution to self-programming the PIC18C452 OTP.

RESOURCE USAGE

The impact on user application resources from the PIC18C452 OTP self-programming application is defined below:

Program Memory (bytes) 930

Data Memory (bytes)... 0

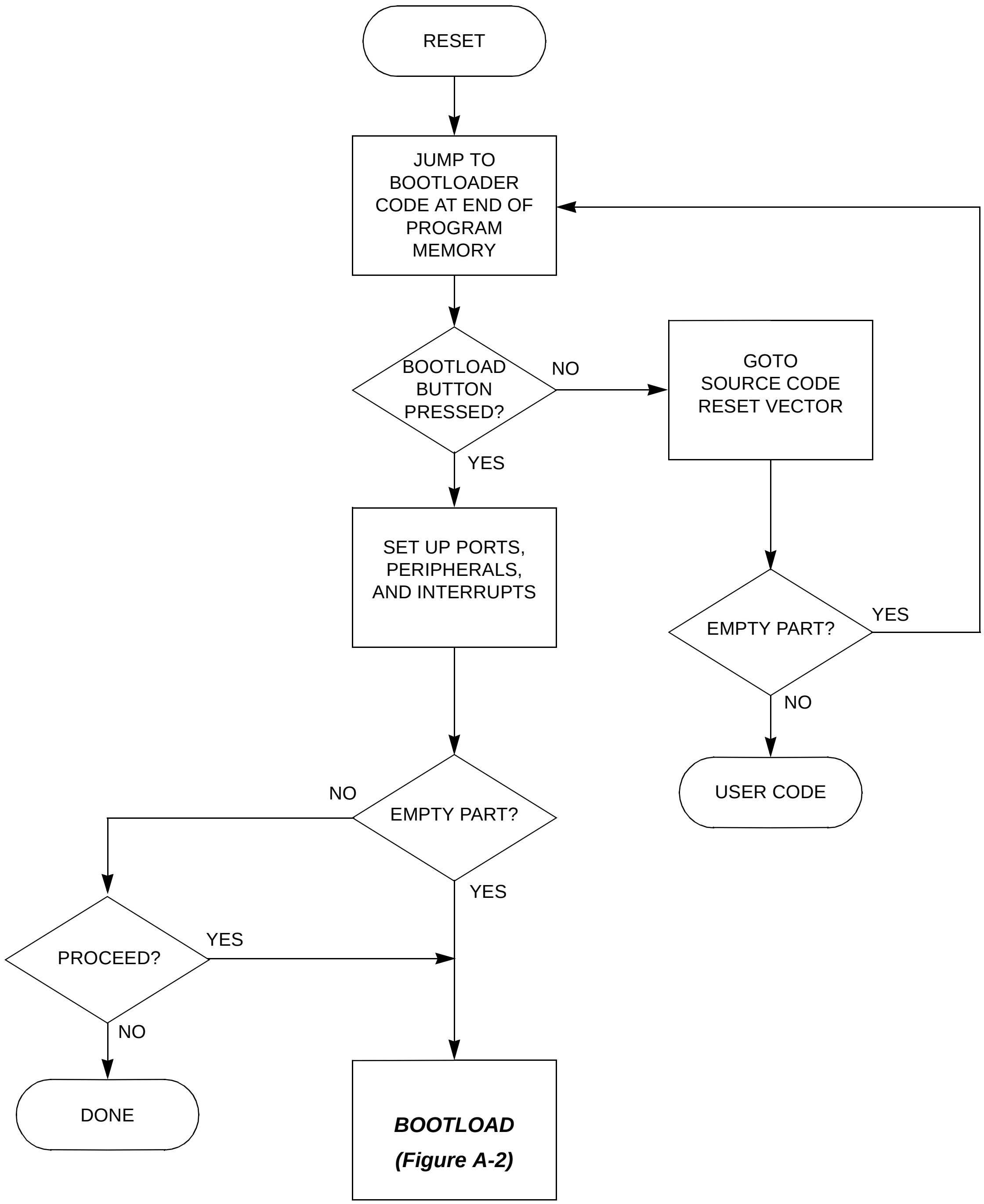
I/O pins.. 7

REFERENCES

Please refer to Appendix A in the MPASM™ User's Guide (DS33014) for more information on HEX file formats.

APPENDIX A: BOOTLOADER PROGRAM FLOW CHART

FIGURE A-1: BOOTLOADER PROGRAM FLOW CHART



 Preliminary

FIGURE A-2: BOOTLOAD ALGORITHM FLOW CHART

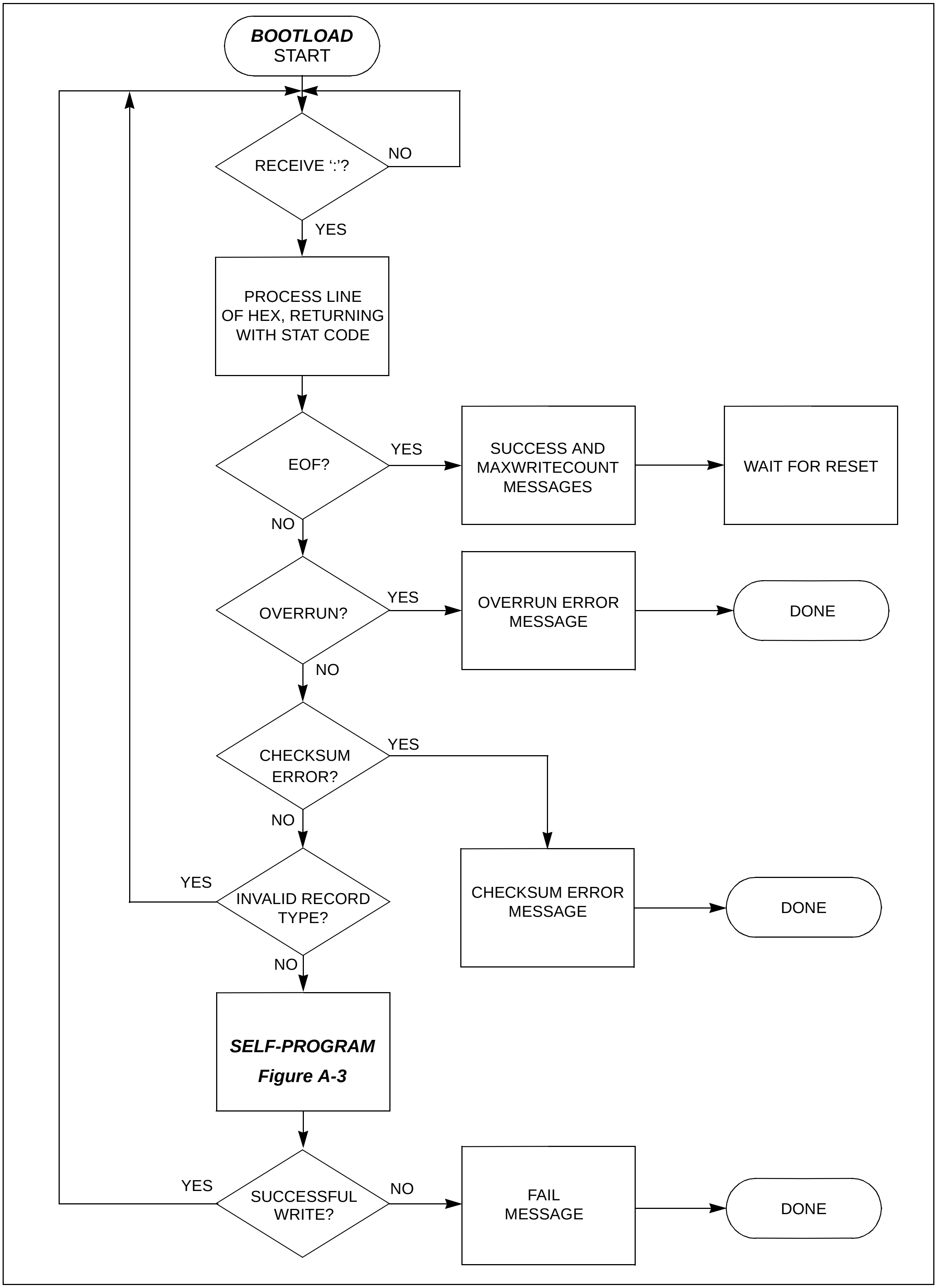
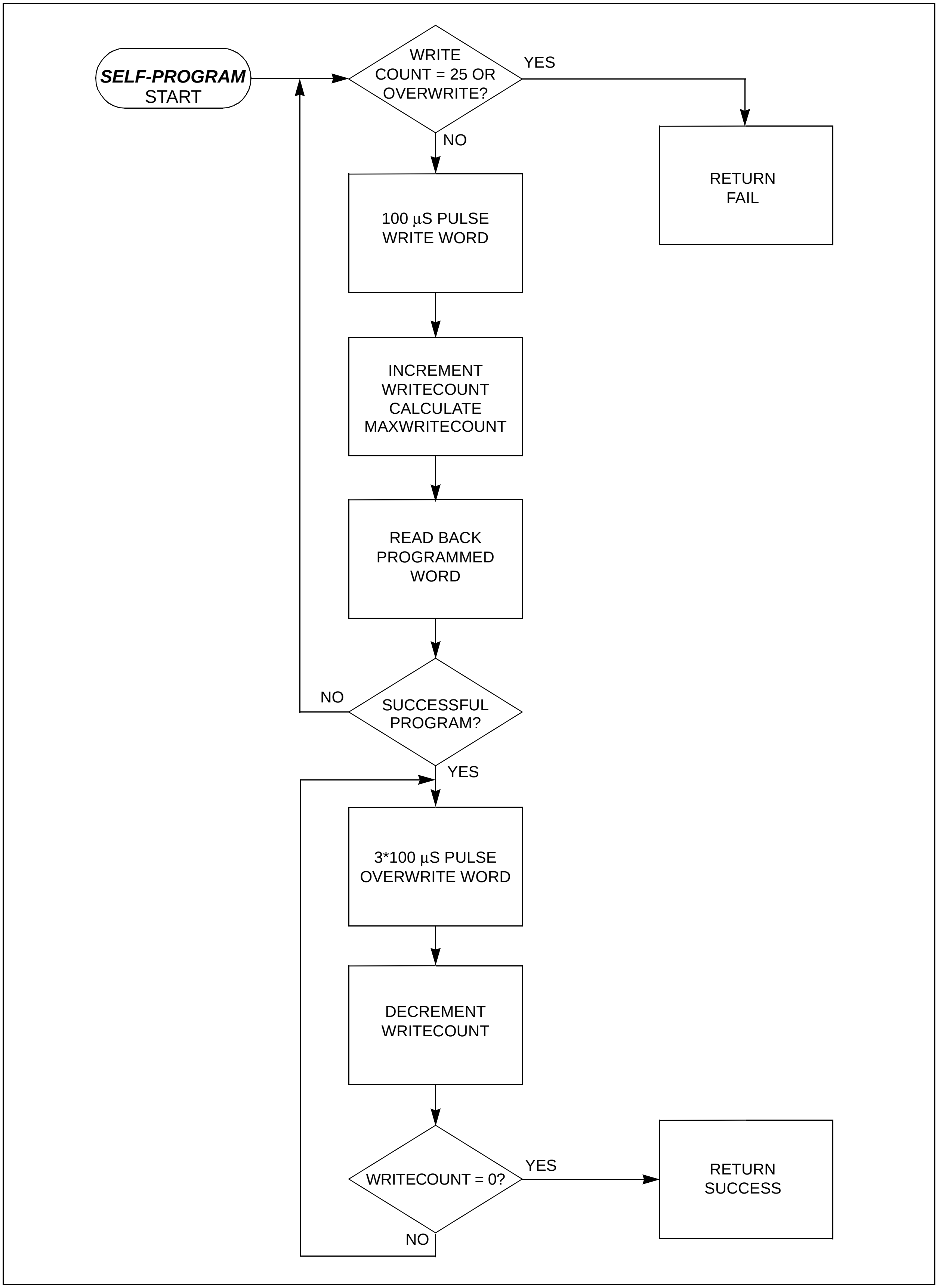


FIGURE A-3: SELF-PROGRAM ALGORITHM FLOW CHART



 Preliminary

APPENDIX B: SOURCE CODE

```
;-----------------------------------------------------------------------------------
;       SELF-PROGRAMMING THE PIC18C452
;
;
;     At start-up, the code checks for user input (button press) and
;     either loads hex file from the USART and programs it to internal
;     program memory or it vectors to user code reset. If the bootloader
;     is executed successfully, a hardware reset is forced on the part
;     and user code execution begins.
;     This program assumes that the user hex code is in INHX8M format
;     AND the user reset vector is emdedded entirely within 1 line
;     of hex.  However, user hex code lines can be non-sequential.
;     This program is installed at the end of program memory space.
;
;
;     Filename:     18cself.asm
;     Author:       Tim Rovnak
;     Version:      1.1 (1/01)
;     Hardware:     Modified PICDEM-2
;                   PICSTART PLUS Programmer V2.10.00
;     Software:     MPLAB V5.11.00
;     Osc:          16.00000 MHz
;     Size:         930 bytes
;
;
;-----------------------------------------------------------------------------------

;-----------------------------------------------------------------------------------
; List file format, Include files
;-----------------------------------------------------------------------------------

          list      P = 18C452      ; set processor type
          list      n = 0           ; supress page breaks in list file
          #include  <P18C452.INC>   ; Processor Include file

;-----------------------------------------------------------------------------------
; CONFIG bits:
;  CONFIG bits are set when programming the bootloader into a blank part.
;  They are determined by the user application.
;  Bootloader program could be modified to program CONFIG bits from hex
;  file in INHX32 format for 32-bit addressing.
;-----------------------------------------------------------------------------------

;         __CONFIG  _CONFIG0, _CP_OFF_0
;         __CONFIG  _CONFIG1, _OSCS_OFF_1 & _ECIO_OSC_1
;         __CONFIG  _CONFIG2, _BOR_OFF_2 & _BORV_25_2 & _PWRT_ON_2
;         __CONFIG  _CONFIG3, _WDT_OFF_3 & _WDTPS_128_3
;         __CONFIG  _CONFIG5, _CCP2MX_ON_5
;         __CONFIG  _CONFIG6, _STVR_ON_6
```

```
;-------------------------------------------------------------------------------
; Compile Constant Definitions:
;-------------------------------------------------------------------------------

#define   RsetSig   PORTA,1               ; Post-program reset signal
#define   BurnSig   PORTA,2               ; Program voltage signal
#define   BootSig   PORTA,4               ; SW2 button line

#define   RTS       PORTC,3               ; RTS line for USART
#define   CTS       PORTC,4               ; CTS line for USART
#define   baud      D'96000'              ; Baud rate

#define   FOSC      D'16000000'           ; Osc. frequency

; tick = (1/(Fosc/4))*prescale = 2uSec for prescale of 8
; timer primer equation:  time = (0x100 - PRIMER)*tick
#define   usec016   0xF8                  ; TMROL primer for 16uSec @16MHz
#define   usec050   0xE7                  ; TMROL primer for 50uSec @16MHz
#define   usec100   0xCE                  ; TMROL primer for 100uSec @16MHz
#define   usec150   0xB5                  ; TMROL primer for 150uSec @16MHz
#define   usec300   0x6A                  ; TMROL primer for 300uSec @16MHz

; Processor dependent placements
; PIC18C452:
#define   BootLoc   0x7C5C                ; Placement of Boot code
#define   EmptLoc   0x7FF6                ; Placement of Empty indicator
#define   RsetLoc   0x7FF8                ; Placement of user reset vector

#define   BuffLen   D'20'                 ; Size of input buffer


;-------------------------------------------------------------------------------
; RAM storage declarations:
;-------------------------------------------------------------------------------

       CBLOCK      0x00
       RXBuffer:   BuffLen                 ; Location for storage of line of Hex
       Tempa                               ; Temps for ASCII to Hex conversion
       Tempb                               ;
       ByteCnt                             ; # of bytes in current line
       ByteCntCpy                          ;
       AddrH                               ; High Byte of address of current line
       AddrL                               ; Low Byte of address of current line
       RecType                             ; Record type of current line
       HexDataH                            ; High byte of current word
       HexDataL                            ; Low byte of current word
       LineCheckSum                        ; Checksum holder for current line
       Adr0Count                           ; Reset vector word counter
       Adr0Flag                            ; Reset vector indicator
       Stat                                ; 'retlw' Status code holder
       WrtCnt                              ; # of writes current word counter
       MaxWrtCnt                           ; Maximum # writes counter
       AsciiCmp                            ; Compare for Hx2ASCII
       ENDC

;-------------------------------------------------------------------------------
; Macros:
;-------------------------------------------------------------------------------

Ltblptr   MACRO     baseaddr               ; Load TBLPTR with address
          movlw     LOW(baseaddr)          ;
          movwf     TBLPTRL                ;
          movlw     HIGH(baseaddr)         ;
          movwf     TBLPTRH                ;
```



```
         movlw     Upper(baseaddr)        ;
         movwf     TBLPTRU                ;
         ENDM

;-------------------------------------------------------------------------------
; Vectors:
;  Only Reset. Don't touch interrupts; leave for user code.
;-------------------------------------------------------------------------------

Rset     org       0x000000               ; Reset vector location
         goto      BootLdr                ; goto startup code

HPvect   org       0x000008               ; High priority interrupt vector

LPvect   org       0x000018               ; Low priority interrupt vector


;-------------------------------------------------------------------------------
; Bootloader:
;  Place at end of program memory space with 4 empty words reserved
;  for source code reset vector.  Run 'Setup' after checking switch
;  in order to leave Micro in default state for user code.
;  Stat codes:     0x01 - 100% successful program
;                  0x02 - Overrun Error
;                  0x03 - Checksum Error
;                  0x04 - Invalid Record Type
;                  0x05 - Good Checksum...Program next line of code
;                  0x06 - Good single line Program...Get next line of code
;                  0x07 - Program Error
;                  0x08 - Overwrite Bootcode error
;-------------------------------------------------------------------------------

BootLdr  org       BootLoc                ; Place appropriately

;        clrwdt                           ; Clear the watchdog timer if enabled.

         ; Check user input----------------------------------------------------
         btfsc     BootSig                ; Is SW2 being pressed?
         goto      RsetSrc                ;  No: goto RsetSrc

         ; Setup part, verify empty, ready to download calls------------------------
         rcall     Setup                  ; Setup ports, peripherals
         rcall     ChkEmpt                ; Check for empty part, prog. PrgEmpt
         sublw     0x07                   ; Test if code 0x07, bad prog.
         bz        PrgErr                 ;
         Ltblptr   RdyMsg                 ; 'Ready' message
         rcall     TXmsg                  ;

GoLoop   ; Fill buffer w/ 1 line of code calls--------------------------------------
         rcall     GetLine                ; 1 line of code in RXBuffer
         movwf     Stat                   ; Move WREG return value into Stat
         dcfsnz    Stat                   ; Decr. Stat, skip cause >0x01
         bra       Success                ;  Stat was 0x01, EOF and Success
         dcfsnz    Stat                   ; Decr. Stat, skip cause >0x02
         bra       OvrErr                 ;  Stat was 0x02, Overrun Error
         dcfsnz    Stat                   ; Decr. Stat, skip cause >0x03
         bra       ChkErr                 ;  Stat was 0x03, CheckSum Error
         dcfsnz    Stat                   ; Decr. Stat, skip cause >0x04
         bra       GoLoop                 ;  Stat was 0x04, bad RecType, Go back
                                          ;  otherwise 0x05, continue...

         ; Write 1 line of code to PMEM calls---------------------------------------
         movlw     LOW(RXBuffer)          ; Reset FSR0 to top of RXBuffer
         movwf     FSR0L                  ;
         movlw     HIGH(RXBuffer)         ;
```

```
        movwf     FSR0H                  ;
        rcall     Wrt                    ; Write to Program Memory
        sublw     0x07                   ; Test if code <,>, or = 0x07
        bn        OWtErr                 ; WREG 0x08 means overwrite.
        bz        PrgErr                 ; Else must be 0x07 Program error
        Ltblptr   DotMsg                 ; '.' indicating 1 good line prg.
        rcall     TXmsg                  ;
        bra       GoLoop                 ;  good program 0x06

        ; Done Traps and Messages---------------------------------------------------
Success Ltblptr   ScsMsg                 ; 'Success, Max. write..' message
        rcall     TXmsg                  ;
        movf      MaxWrtCnt,W            ;
        rcall     TXbyte                 ; Send Max. write count to PC
        Ltblptr   RstMsg                 ; 'Reset? [y]' message
        rcall     TXmsg
        rcall     GetASCII               ; Get a byte in RCREG and compare to
        movlw     'y'                    ;  'y', loop until true
        cpfseq    RCREG                  ;
        bra       $-6                    ;
        bsf       RsetSig                ; BYE-BYE

OWtErr  Ltblptr   OWtMsg                 ; 'Overwrite..' message
        rcall     TXmsg
        bra       $

OvrErr  Ltblptr   OvrMsg                 ; 'Overrun..' message
        rcall     TXmsg
        bra       $

PrgErr  Ltblptr   PgEMsg                 ; 'Program fail at..' message
        rcall     TXmsg                  ;
        bra       ADRmsg

ChkErr  Ltblptr   ChkMsg                 ; 'Checksum err at..' message
        rcall     TXmsg                  ;

ADRmsg  movf      AddrH,W                ; Send Address bytes to PC
        rcall     TXbyte
        movf      AddrL,W
        rcall     TXbyte
        bra       $

        ;--------------------------------------------------------------------------
Setup   ; Setup Ports, Initialize variables, send Ready message
        ;--------------------------------------------------------------------------

        ; Setup ports--------------------------------------------------------------
        clrf      PORTA                  ; Clear PORTA output latch
        movlw     B'11111001'            ; Make RA1 (rset), RA2 (burn) outputs,
        movwf     TRISA                  ;  RA4 (SW2) and rest inputs
        clrf      PORTB                  ; Clear PORTB output latch
        clrf      TRISB                  ; PORTB pins all outputs for LEDs
        clrf      PORTC                  ; Clear PORTC output latch
        movlw     B'10101111'            ; RC6=TX output, RC4=CTS output,
        movwf     TRISC                  ;  rest inputs (inc. RTS,RX)
        bsf       CTS                    ; Not ready for a send yet.

        ; Setup TIMER0-------------------------------------------------------------
        movlw     B'01000010'            ; Init. TMR0 to off, 8-bit, int. clk,
        movwf     T0CON                  ;  pre-scaler, 1:8 pre-scaler value.
                                         ;  @16Mhz=> (1/4Mhz)*8 = 2usec Tick
        bsf       INTCON,TMR0IE          ; Enable TMR0 overflow interrupt

        ; Setup USART--------------------------------------------------------------
```



```
        movlw     D'25'                 ; Load baud rate generator for 9.6kbd
        movwf     SPBRG                 ;  @ 16Mhz device frequency.
        movlw     B'00100000'           ; Enable USART transmit, set baud
        movwf     TXSTA                 ; rate generator for low speed.
        movlw     B'10010000'           ; Enable USART for continuous reception,
        movwf     RCSTA                 ;  enable USART

        ; Init. vars, enable Long Writes---------------------------------------------
        movlw     0x3A                  ;
        movwf     AsciiCmp              ;
        clrf      WrtCnt                ; Initialize WrtCnt, MaxWrtCnt to 0
        clrf      MaxWrtCnt             ;
        bsf       RCON,LWRT             ; Enable Long writes to PMEM
        movlw     D'1'                  ; Init ByteCntCpy to 1 for WrtLoop
        movwf     ByteCntCpy            ;  to program EmptLoc
        clrf      HexDataH              ; For Emptchk also
        clrf      HexDataL              ;
        return                          ;

        ;-------------------------------------------------------------------------------
ChkEmpt ; Check byte in program memory indicating if part has been
        ; previously programmed. Prompt user to continue.
        ;-------------------------------------------------------------------------------

        Ltblptr   EmptLoc              ; Set TBLPTR to EmptLoc
        TBLRD*                         ; Read EmptLoc Hi-byte
        movf      TABLAT,W             ; Store in W
        btfss     STATUS,Z             ; Test if zero
        bra       WrtLoop              ; Go write PrgEmpt code (0x00)
                                       ; Returns 0x06 or 0x07 to main when done
        Ltblptr   NtEMsg               ; 'Not empty..proceed? [y]' message
        rcall     TXmsg                ;
        rcall     GetASCII             ; Get a byte in RCREG and compare to
        movlw     'y'                  ;  'y', loop until true
        cpfseq    RCREG                ;
        bra       $-6                  ;
        retlw     0x06                 ; Return a dummy 0x06 to continue

        ;-------------------------------------------------------------------------------
GetLine ; Receive line of HEX from PC through USART and place in
        ; RXBuffer. Performs error checking and returns error/success
        ; code.
        ;-------------------------------------------------------------------------------

        ; Get ASCII byte, look for start of line (':')-------------------------------
        rcall     GetASCII             ; Get a byte in RCREG and compare to
        movlw     ':'                  ;  ':', loop until true
        cpfseq    RCREG                ;
        bra       GetLine              ;

        ; Initialize Address 0 control and LFSR0 each new line------------------------
        movlw     D'4'                 ; Initialize Adr0Count to maximum
        movwf     Adr0Count            ;  size of Source Reset vector
        movlw     D'2'                 ; Initialize Adr0Flag to 2 in order
        movwf     Adr0Flag             ;  to decr. test 2 bytes of address
        movlw     LOW(RXBuffer)        ; Point to the RXBuffer using FSR0
        movwf     FSR0L                ;  NOTE: Avoid 'lfsr'!
        movlw     HIGH(RXBuffer)       ;
        movwf     FSR0H                ;

        ; Get byte count------------------------------------------------------------
        rcall     GetHex8              ; Conv. to Hex
        movwf     ByteCnt              ;
        movwf     LineCheckSum         ; Add to Checksum
        rrncf     ByteCnt,F            ; Divide by 2 for word count
```

```
         movff     ByteCnt,ByteCntCpy    ; Make copy for write routine

         ; Get Hi address byte----------------------------------------------------------
         rcall     GetHex8               ; Conv. to Hex
         movwf     AddrH                 ; Hibyte of address
         bnz       notz1                 ; Test for Hibyte of Reset vector(0x00)
         decf      Adr0Flag,F            ; Decr. flag ...could be reset
notz1    addwf     LineCheckSum,F        ; Add to Checksum

         ; Get Lo address byte----------------------------------------------------------
         rcall     GetHex8               ; Conv. to Hex
         movwf     AddrL                 ; Lobyte of address
         bnz       notz2                 ; Test for Lobyte of Reset vector(0x00)
         decf      Adr0Flag,F            ; Decr. flag ...could be reset
notz2    addwf     LineCheckSum,F        ; Add to checksum

         ; Set Address of Table pointer-----------------------------------------------
         movff     AddrL,TBLPTRL         ; Default TBLPTR to Addrl and AddrH
         movff     AddrH,TBLPTRH         ;
         clrf      TBLPTRU               ; Clear TBLPTR upper-byte
         negf      Adr0Flag              ; Test if Adr0Flag is set
         bnz       GetRec                ;  No-> Keep default, branch to GetRec
         Ltblptr   RsetSrc               ;  Yes-> change write address to
         ;         RsetSrc temporarily

GetRec   ; Get record type--------------------------------------------------------------
         rcall     GetHex8               ; Conv. Record type to Hex
         movwf     RecType               ;
         addwf     LineCheckSum,F        ; Add to Checksum

GetData  ; Get data bytes loop----------------------------------------------------------
         ; Chk EOL
         movf      ByteCnt,F             ; Check ByteCnt
         bz        ChkChkSm              ;  If 0->calculate LineCheckSum
         ; Lo
         rcall     GetHex8               ; Get LoByte and store in RXBuffer,
         movwf     POSTINC0              ;  incr. FSR0, and then add to
         addwf     LineCheckSum,F        ;  LineCheckSum
         ; Hi
         rcall     GetHex8               ; Get HiByte and store in RXBuffer,
         movwf     POSTINC0              ;  incr. FSR0, and then add to
         addwf     LineCheckSum,F        ;  LineCheckSum
         decf      ByteCnt,F             ; Decr. line Byte counter
         bra       GetData               ; Get next word

ChkChkSm ; Test for overrun, verify Checksum then return status code-----------------
         rcall     GetHex8               ; Convert last ASCII byte to hex
         addwf     LineCheckSum,F        ; Add to Checksum
         btfsc     RCSTA,OERR            ; Check overrun bit in RCSTA
         retlw     0x02                  ; Return 0x02 for overrun
         tstfsz    LineCheckSum          ; CheckSum=0->test for EOF, then return
         retlw     0x03                  ; Return code 0x03 for CheckSum Error
         movf      RecType,F             ; Check Record type
         bnz       Eof                   ; Branch to Eof if not 0x00
         retlw     0x05                  ; Return code 0x05 for continue
Eof      decfsz    RecType,F             ; Decr. Record type
         retlw     0x04                  ; Return code 0x04 for invalid format
         retlw     0x01                  ; EOF All good, No burn. code 0x01

         ;-------------------------------------------------------------------------
Wrt      ; Write 1 line of data to Internal Program Memory
         ;-------------------------------------------------------------------------

         ; Load current RXBuffer value to HexDataH/L-------------------------------
         movff     POSTINC0,HexDataH     ; Move RXBuffer data to HexDataH/L
```

 Preliminary

```
        movff     POSTINC0,HexDataL       ;

ChkOvw  ; Check for overwrite condition--------------------------------------------
        movlw     HIGH(BootLoc)           ; Load W Hi-byte Boot code location
        subwf     AddrH,W                 ; Subtract W from AddrH, result in W
        bn        WrtLoop                 ; Branch to WrtLoop if AddrH is lower
                                          ;  than HIGH(BootLoc)
        btfss     STATUS,Z                ; Go test Lo if Hi-bytes are eq.
        retlw     0x08                    ; Return Error cause AddrH is greater
                                          ;  than HIGH(BootLoc)
        movlw     LOW(BootLoc)            ; Load W Lo-byte Boot code location
        cpfslt    AddrL                   ; Compare with AddrL, Skip to Wrtloop
        retlw     0x08                    ;  if AddrL is less than LOW(BootLoc)
                                          ;  else return Error

WrtLoop ; Write 1 word to Program Memory-------------------------------------------
;       clrwdt                            ; Clear the watchdog timer if enabled.

        movlw     D'25'                   ; Load W with max # of wrt attempts
        cpfslt    WrtCnt                  ; Compare with WrtCnt
        retlw     0x07                    ;  WrtCnt=25-> Code 0x07 bad progr.
        bsf       BurnSig                 ; ON Programming Voltage
        movlw     usec100                 ; Load W with 100uSec delay value
        rcall     DoWrt                   ; Do Write
        bcf       BurnSig                 ; OFF Programming voltage
        incf      WrtCnt,F                ; Incr. Write Counter

        ; Verify write-------------------------------------------------------------
        TBLRD*                            ; Read back first PMEM byte written
        movf      TABLAT,W                ; Store in W
        cpfseq    HexDataH                ; Compare to original data
        bra       WrtLoop                 ;  Not Equal-> ReWrite
        TBLRD+*                           ; Read back second PMEM byte written
        movf      TABLAT,W                ; Store in W
        cpfseq    HexDataL                ; Compare to original data
        bra       WrtLoop                 ;  Not Equal-> ReWrite
        TBLRD*-                           ; Move pointer back to 1st Pbyte
        movf      MaxWrtCnt,W             ; Move MaxWrtCnt to W
        cpfslt    WrtCnt                  ; If WrtCnt>MaxWrtCnt, Max.=WrtCnt
        movff     WrtCnt,MaxWrtCnt        ;

OverPrg ; 3*WrtCnt over-programming-----------------------------------------------
;       clrwdt                            ; Clear the watchdog timer if enabled.

        bsf       BurnSig                 ; ON Programming Voltage
        movlw     usec300                 ; Load W with 300uSec delay value
        rcall     DoWrt                   ; Do Write
        decfsz    WrtCnt,F                ; Decr. Write Counter
        bra       OverPrg                 ;  Not 0-> Keep writing
        bcf       BurnSig                 ; OFF Programming voltage
        TBLRD*+                           ; Dummmy reads
        TBLRD*+                           ;

        ; Update vars then burn next word OR return------------------------------
        incf      AddrL,F                 ; Incr. AddrL
        infsnz    AddrL,F                 ; Incr. AddrL, skip if not 0
        incf      AddrH,F                 ;  0-> Overflow, inc. AddrH
        negf      Adr0Flag                ; Was it Reset vector?
        bnz       ChkCnt                  ;  No-> Check Byte count
        decfsz    Adr0Count,F             ; Decr. Adr0Count. Last Reset vector word?
        bra       ChkCnt                  ;  No-> Check Byte count
        movff     AddrL,TBLPTRL           ; Point TBLPTR to Addrl and AddrH again
        movff     AddrH,TBLPTRH           ;
        clrf      TBLPTRU                 ;
        movlw     D'2'                    ; Reinit Adr0Flag to 0x02
```

```
         movwf     Adr0Flag              ;
ChkCnt   decfsz    ByteCntCpy            ; Check ByteCntCpy
         bra       Wrt                   ; Go back to write
         retlw     0x06                  ; Code 0x06 for Good Programming


;------------------------------------------------------------------------------
; Subroutines:
;------------------------------------------------------------------------------

TXmsg   ; Transmit Message from PMEM to PC thru USART-------------------------------
         btfss     PIR1,TXIF             ; Wait until the USART is not busy.
         bra       TXmsg                 ;
         tblrd*+                         ; Read byte from PMEM, incr. Table ptr
         movf      TABLAT,W              ; Check if byte read is 0
         bz        TXDone                ;  0-> done
         movwf     TXREG                 ; Put byte in Transmit register
         bra       TXmsg                 ;  and loop back for next byte
TXDone   return                          ; ----------------------------------------

TXbyte   ; Transmit byte from Wreg to PC thru USART---------------------------------
         rcall     Hx2ASCII              ; convert W to 2 ASCII bytes (TempA/B)
         btfss     PIR1,TXIF             ; Wait until the USART is not busy.
         bra       $-2                   ;
         movff     Tempa,TXREG           ; Put byte in Transmit register
         nop                             ;
         btfss     PIR1,TXIF             ; Wait until the USART is not busy.
         bra       $-2                   ;
         movff     Tempb,TXREG           ; Put byte in Transmit register
         return                          ; ----------------------------------------

Hx2ASCII ; Convert Hex byte to 2 ASCII bytes----------------------------------------
         ; Return with Tempa High, Tempb low
         movwf     Tempa                 ; Keep copy of HEX in Tempa
         andlw     0x0F                  ; Mask out lower nibble
         addlw     0x30                  ; add 0x30
         cpfsgt    AsciiCmp              ; If W less than 0x3A, done
         addlw     0x07                  ;  Else, add 0x07, then done
         movwf     Tempb                 ;
         swapf     Tempa,W               ; Swap nibbles Tempa, place in Wreg
         andlw     0x0F                  ; Mask out lower nibble
         addlw     0x30                  ; add 0x30
         cpfsgt    AsciiCmp              ; If W less than 0x3A, done
         addlw     0x07                  ;  Else, add 0x07, then done
         movwf     Tempa                 ;
         return                          ; ---------------------------------------

GetASCII ; Receive ASCII byte thru USART using CTS/RTS (H/W cntrl)-----------------
         btfsc     RTS                   ; Check RTS=0-> PC wants to send data
         bra       GetASCII              ;  If RTS=1-> PC not sending, check again.
         bcf       CTS                   ; Set CTS=0-> micro Clear to Send
RXwait   btfss     PIR1,RCIF             ; Test Recv. interrupt flag
         bra       RXwait                ;  not set, keep checking
         bsf       CTS                   ; Set CTS=1-> micro NOT Clear To Send
         movf      RCREG,W               ; Move Recv. buffer to W, CLEARS RCIF
         return                          ; ---------------------------------------

GetHex8  ; Receive 2 ASCII chars, convert to one 8-bit HEX #----------------------
         ; Hi byte
         rcall     GetASCII              ; Get 1st ASCII char thru USART
         rcall     ASCII2Hx              ; Convert to Hex
         movwf     Tempa                 ; Move to Tempa
         swapf     Tempa,F               ; swap nibbles in Tempa
                                         ; Lo byte
         rcall     GetASCII              ; Get 2nd ASCII char thru USART
```

 Preliminary

```
         rcall     ASCII2Hx             ; Covert to Hex
         iorwf     Tempa,F              ; combine nibbles into 1 byte
         movf      Tempa,W              ; move result to W
         return                         ; -------------------------------------

ASCII2Hx  ; Convert ASCII byte of data to binary----------------------------------
         movwf     Tempb                ; Move W to Tempb
         movlw     '0'                  ; Load W with ASCII '0', (0x30)
         subwf     Tempb,F              ; Subtract from Tempb
         movlw     0xF0                 ; Load W B'11110000'
         andwf     Tempb,W              ; Mask out Tempb Upper nibble
         bz        DoneASC              ; If 0-> We had a number, now it's good.
         movlw     'A'-'0'-0x0a         ; Had a letter, subtract off additional
         subwf     Tempb,F              ;  amount
DoneASC  movf      Tempb,W              ;
         return                         ; -------------------------------------

DoWrt    ; PMEM write with built-in delay----------------------------------------
         movwf     TMR0L                ; Prime TMR0L
         bcf       INTCON,TMR0IF        ; Clear TMR0 overflow flag
         bsf       T0CON,TMR0ON         ; ON TMR0
         movff     HexDataH,TABLAT      ;
         TBLWT*+                        ;
         movff     HexDataL,TABLAT      ;
         TBLWT*-                        ; Should pause here until TMR0 interrupt
         bcf       T0CON,TMR0ON         ; OFF TMR0
         return                         ; -------------------------------------


;---------------------------------------------------------------------------------
; Messages:
;---------------------------------------------------------------------------------

DotMsg   data      ".",0
RdyMsg   data      "\r\nReady to Receive Hex File.\r\n",0
PgEMsg   data      "\r\nProgram Failed at: 0x",0
ScsMsg   data      "\r\nProgram Success! Maximum Write Count: 0x",0
ChkMsg   data      "\r\nChecksum Error in Address Block: 0x",0
OvrMsg   data      "\r\nOverrun Error.",0
NtEMsg   data      "\r\nPart Not Empty. Proceed? [y]",0
RstMsg   data      "\r\nReset? [y]",0
OWtMsg   data      "\r\nOverwrite Bootcode Error.",0

;---------------------------------------------------------------------------------
; Empty part indicator will be programmed here
;---------------------------------------------------------------------------------

         ORG       EmptLoc
         data      0xFFFF

;---------------------------------------------------------------------------------
; RESET Vector for source code will be programmed here
;---------------------------------------------------------------------------------

RsetSrc  ORG       RsetLoc              ; Space for 4 program words
         res       8                    ;  to be programmed by Bootloader

         end                            ; End of File AND End of PMEM
```

NOTES:

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.

MICROCHIP

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

Austin
Analog Product Sales
8303 MoPac Expressway North
Suite A-201
Austin, TX 78759
Tel: 512-345-2030 Fax: 512-345-6085

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

Boston
Analog Product Sales
Unit A-8-1 Millbrook Tarry Condominium
97 Lowell Road
Concord, MA 01742
Tel: 978-371-6400 Fax: 978-371-0050

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

Dayton
Two Prestige Place, Suite 130
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

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

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

Mountain View
Analog Product Sales
1300 Terra Bella Avenue
Mountain View, CA 94043-1836
Tel: 650-968-9241 Fax: 650-967-1590

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 Beijing Office
Unit 915
New China Hong Kong Manhattan Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Shanghai
Microchip Technology Shanghai Office
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

Hong Kong
Microchip Asia Pacific
RM 2101, 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 Intl. Inc.
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

ASIA/PACIFIC (continued)

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
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 Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

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

Germany
Analog Product Sales
Lochhamer Strasse 13
D-82152 Martinsried, Germany
Tel: 49-89-895650-0 Fax: 49-89-895650-22

Italy
Arizona 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

01/30/01