
High-Speed Serial Bootloader for PIC16 and PIC18 Devices

Author: E. Schlunder
Microchip Technology Inc.

INTRODUCTION

Microchip's enhanced Flash microcontrollers enable firmware to program itself. This is done by a "bootloader" providing a firmware kernel, residing in the microcontroller. The kernel uses a small portion of program memory not normally used by the firmware's main application.

When the bootloader firmware is activated, a host PC can use a serial protocol to read, write and verify updates to the microcontroller's application firmware. Once the application firmware is programmed, the bootloader cedes control, allowing normal application execution until the bootloader is called.

AN1310 Bootloader Features

The key features of the AN1310, "High-Speed Serial Bootloader for PIC16 and PIC18 Devices" include:

- Small firmware code size (less than 450 instruction words on most devices)
- Automatic baud rate synchronization to the host
- Baud rate flexibility, from 1,200 bps to 3 Mbps for extremely fast programming
- A 16-bit CRC packet and Flash memory verification for quick verification of successful programming, even at low baud rates
- An advanced "write planner" that eliminates unnecessary erase/write transactions
- Support for a wide variety of PIC16 and PIC18 devices through an "essential device characteristics" database
- Optional application remapping that does not require linker script modifications or remapping of interrupt service routines
- A forced bootloader re-entry mechanism requiring minimal start-up delay and no additional I/O pins or application firmware code to re-enter the bootloader
- Optional MCLR Reset control, allowing the host PC application to automatically reset the device for robust bootloader re-entry

- PC software rewritten in C/C++ for the cross-platform, QtSM SDK, enabling Linux host support by recompiling the PC software source code
- A simple, Serial Terminal Application mode, provided by the PC software, that eliminates time wasted by switching between separate bootloader host and serial terminal applications

Note: If a review of the preceding features list indicates that a different bootloader is needed, see "**Alternative References**".

Prerequisites

Before using the serial bootloader, the following is required:

- Familiarity with Configuration bits, compiling and programming PIC[®] microcontrollers
- A development board with a serial port connected to the PIC device's USART1 RX/TX pins
- A PC with a serial port or USB-to-serial adapter
- A traditional programming tool for initially writing the bootloader firmware into the PIC device (such as REAL ICE[™] emulator, PICKIT[™] 3 or MPLAB[®] ICD 3)
- Installation of the MPLAB[®] IDE software
- Installation of the AN1310, high-speed serial bootloader software

The AN1310 high-speed serial bootloader software package (including full source code) can be downloaded from the www.microchip.com/applicationnotes web site.

1. When the Browse Application Notes page appears, go to the **Select a Function** menu and select "Bootloader" under "Programming & Bootloaders".
2. Click the **Search** button.
3. Scroll down to AN1310 and click the compressed file icon in the "Resource Type" column.

IMPLEMENTATION BASICS

This section gives the three basic steps for setting up and using the bootloader for those already familiar with bootloader/application development.

Subsequent sections provide overview and more detailed information relating to these steps. Those sections include:

- “Firmware Overview”
- “Hardware Considerations”
- “Bootloader Mode Considerations”
- “Application Mode Considerations”
- “Software Design”

Step 1: Compile and Program Bootloader Firmware

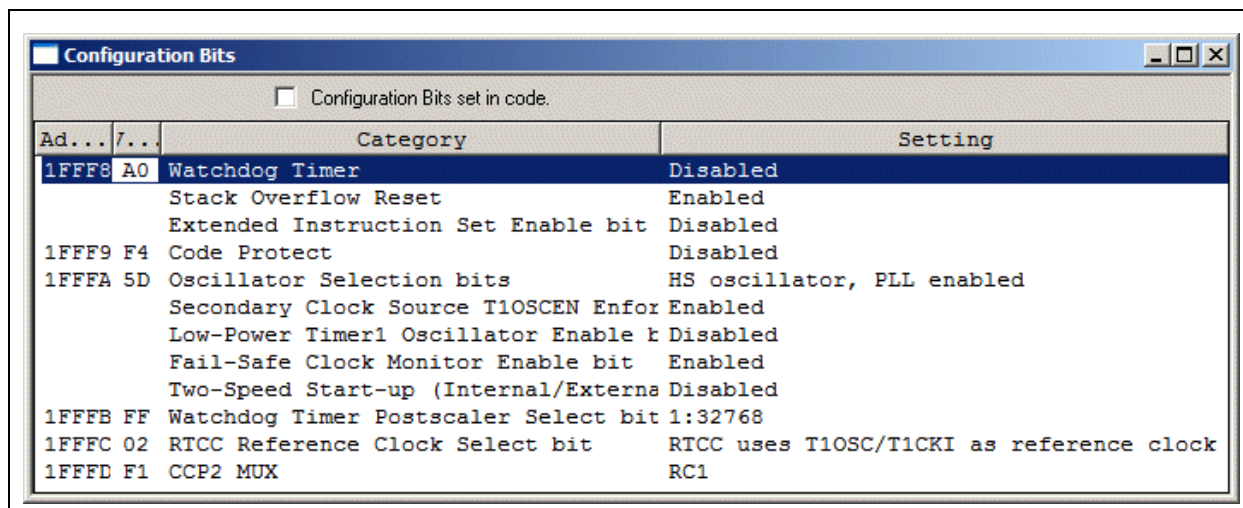
More detailed information on this step is provided in:

- “Firmware Overview”
- “Hardware Considerations”
- “Bootloader Mode Considerations”
- “Software Design”

By default, the serial bootloader installation provides bootloader firmware source code in the PC path:

```
C:\Microchip Solutions\Serial
Bootloader AN1310 vX.XX\PICxx Bootloader\
```

FIGURE 1: CONFIGURATION BITS DIALOG BOX



To compile and program the bootloader firmware:

1. Open the appropriate PIC16 or PIC18 bootloader project in the MPLAB IDE software.
2. Select **Configure>Select Device...** and choose the PIC device to be used (for example, *PIC18F87J90*).
3. To specify Configuration bit settings, select **Configure>Configuration Bits...**

The dialog box, shown in Figure 1, appears.

4. Specify settings for each category.

Table 1 gives suggestions for selected bits that can get the bootloader initially working.

5. Compile and program the bootloader firmware into the microcontroller.

The bootloader must be programmed into the PIC device using an ICSP™ programming tool, such as MPLAB® ICD 3 or a socketed programmer like the MPLAB PM3 Universal Device Programmer.

TABLE 1: CONFIGURATION BIT SUGGESTIONS

Category	Setting
Watchdog Timer ⁽¹⁾	“Disabled”
Extended Instruction Set Enable bit	“Disabled”
Oscillator Selection bits	(Select according to hardware. Higher speeds generally enable more baud rates.)
Fail-Safe Clock Monitor Enable bit	“Enabled”, if available
Low-Voltage Program (LVP)	“Disabled”, if applicable
Table Read-Protect	“Disabled”, if applicable

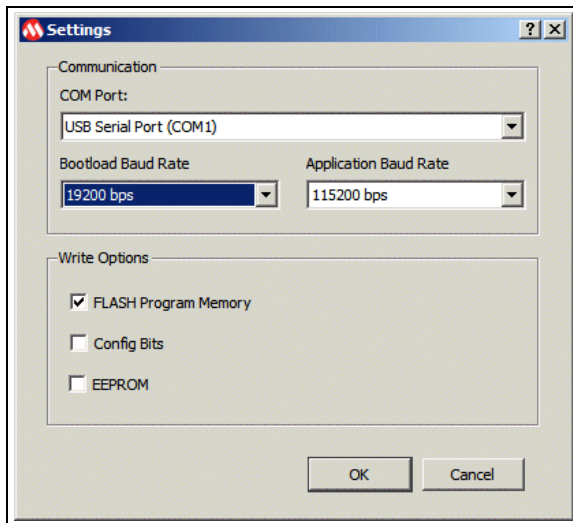
Note 1: On most PIC devices, the Watchdog Timer can be re-enabled in application firmware after start-up.

Step 2: Connect Host to Bootloader

1. Open the serial bootloader host PC software (AN1310ui.exe).
2. If this is the initial setup, configure the serial port and bootloader baud rate by selecting *Program>Settings*.

The dialog box in Figure 2 appears.

FIGURE 2: SETTINGS DIALOG BOX



3. Select values for the “COM Port” and “Bootloader Baud Rate” fields and click **OK**.

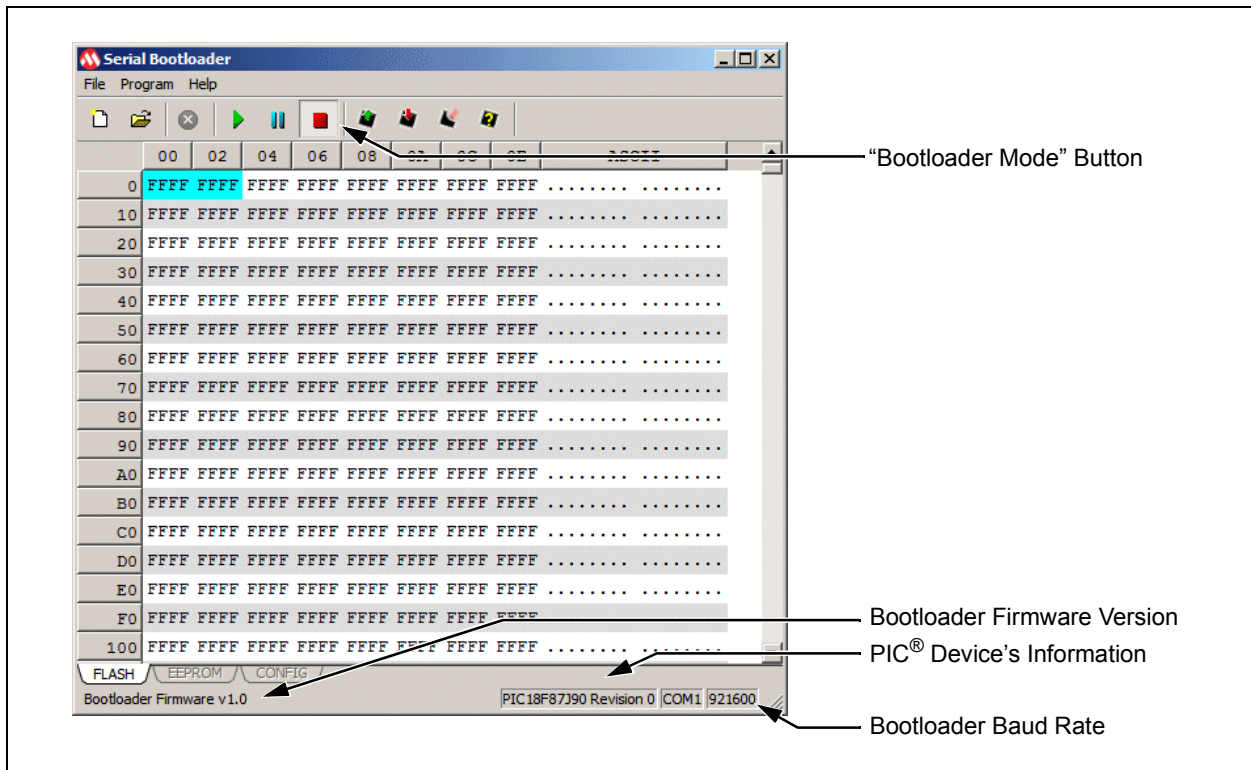
Using a moderate speed “Bootloader Baud Rate”, such as 19.2 kbps, may help with initial communications. Non-standard, high-speed baud rates can be attempted once everything is working.

The software allows serial communication with application firmware after programming. Applications can require a particular baud rate, so a separate “Application Baud Rate” setting is provided. For the example application firmware projects, the 115,200 bps rate is suggested.

4. Connect the PC's serial port to the PIC MCU development board.
5. Go into “Bootloader” mode on the PC by clicking the red bootloader software button, shown in Figure 3, or pressing the PC's <F4> key.

The PC software attempts to communicate with the PIC bootloader firmware, using the specified bootloader baud rate. If communications are established, the PC displays the bootloader firmware revision and the PIC device's information, as shown in Figure 3.

FIGURE 3: SERIAL BOOTLOADER HOST PC APPLICATION



Step 3: Program Application Firmware

More detailed information on this step is provided in:

- “Application Mode Considerations”
- “Software Design”

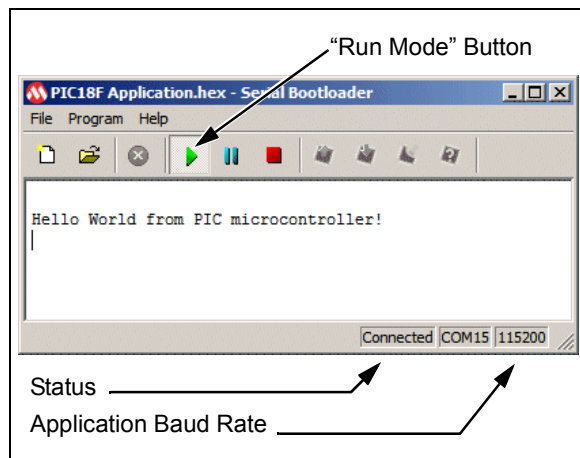
Once the host PIC is connected to the bootloader, the PIC device can be read, written, erased or verified. A sample application firmware project is installed with the software in the path:

```
C:\Microchip Solutions\Serial  
Bootloader AN1310 vX.XX\PICxx Application\
```

1. Open the appropriate project in the MPLAB IDE software, select the desired device, configure the settings and compile the application firmware.
2. Open the resulting application firmware's hex file with the host PC's serial bootloader software.
3. Write the application firmware into the PIC device by clicking the software button with the red down arrow or by pressing the PC's <F6> key.
4. Direct the bootloader to start the application firmware by clicking the software's green “Run Mode” button or pressing <F2> on the PC.

In this mode, the PC software acts as a simple serial terminal application, as shown in Figure 4. This allows two-way text communication with the application firmware through the PIC device's USART1 port.

FIGURE 4: APPLICATION RUN MODE



FIRMWARE OVERVIEW

There are two modes of firmware operation: bootloader and application. When the microcontroller comes out of Reset, the bootloader start-up routine decides whether to enter the bootloader command loop (Bootloader mode) or jump to the application entry point vector (Application mode).

With no intervention, the Bootloader mode is entered if either of the following conditions apply. If these conditions do not apply, the Application mode is entered:

- If application firmware code has not been programmed into the microcontroller, the Bootloader mode is entered.
- If the PIC device's RX pin is at logic level low (the RS-232 “Break” state) when the microcontroller comes out of Reset, Bootloader mode is entered.

Bootloader mode also can be entered manually or automatically through software or hardware.

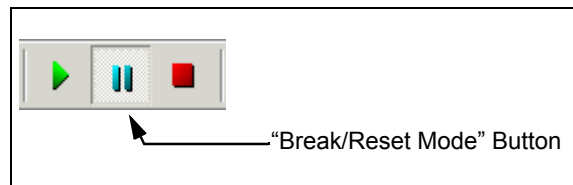
Manual Break and Reset for Re-Entry

The host PC software enables the PIC device's RX pin to be put into the RS-232 “Break” state. This holds the PIC device's RX pin low, forcing the microcontroller into Bootloader mode when it is reset.

To manually enter Bootloader mode:

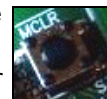
1. Click the software's blue “Break/Reset Mode” button, shown in Figure 5, or press the PC's <F3> key.

FIGURE 5: BREAK/RESET MODE



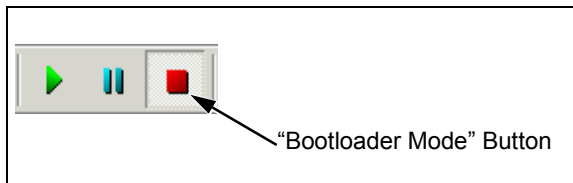
2. Reset the PIC device (so that the bootloader start-up routine is executed) by doing one of the following:

- On the development board, press the MCLR Reset button, shown at right
- Disconnect and reconnect the power



The device resets and the bootloader start-up routine notices the “Break” request on the RX pin. Bootloader mode is entered, even if application firmware has been programmed into the device.

3. Connect to the bootloader on the PC by clicking the software's red “Bootloader Mode” button, shown in Figure 6, or pressing the PC's <F4> key.

FIGURE 6: “BOOTLOADER MODE” BUTTON

The PC software attempts to communicate with the bootloader using the bootload baud rate. If successful, the PC receives the bootloader firmware revision and the PIC device information, shown earlier in Figure 4.

Software Bootloader Re-Entry

The preceding procedure for manually executing the re-entry sequence can be cumbersome when making many incremental application firmware changes during development. An easier alternative is to use the simple software re-entry mechanism, given in the example application firmware project.

That mechanism is shown in Example 1.

EXAMPLE 1: SOFTWARE BOOTLOADER RE-ENTRY

```
while(1)
{
    if(PIR1bits.RCIF)
    {
        if(RCSTAbits.FERR &&
        !PORTCbits.RC7)
        {
            // receiving BREAK
            // state, soft reboot
            // into Bootloader mode.
            Reset();
        }
    }
    (...)
}
```

This code continuously monitors the USART module's state for a framing error. When a framing error occurs, the code verifies that the RXD pin is being held at a logic low level, indicating that the host PC is most likely transmitting an RS-232 Break state to request bootloader re-entry. The application responds by initiating a software Reset of the microcontroller and passing control to the bootloader start-up routine.

PIC16 microcontrollers, however, have no software Reset instruction, so the application jumps to the bootloader start-up vector at address, 0h. To avoid leaving unremovable return addresses on the call stack, jumping to address 0h must be done only from the application's "main()" function.

Hardware Bootloader Re-Entry

The software re-entry procedure is useful for getting started, but is not recommended for robust operation. Should the application code have bugs, the application firmware could lock up and prevent automatic bootloader re-entry for the next code change.

Additionally, an actual framing error, triggered during normal application serial communications, could inadvertently cause unintended re-entry into Bootloader mode. Then, the application could not be restarted without user intervention.

For a more robust, hardware-based bootloader re-entry, the serial port RTS signal can be wired to control the PIC device's MCLR Reset signal. This allows the host PC software to automatically assert Break and Reset signals, as described in "Manual Break and Reset for Re-Entry".

AN1310

HARDWARE CONSIDERATIONS

Figure 7 represents a typical connection diagram for the serial bootloader.

RTS-based $\overline{\text{MCLR}}$ Reset control is optional. When connected, the host PC can use the RTS serial port signal to pull down the PIC device's $\overline{\text{MCLR}}$ pin, allowing the host PC software to automatically reset the device when entering the Bootloader mode.

If using a PIC device that requires high voltage (above V_{DD}) on $\overline{\text{MCLR}}$ to enter the In-Circuit Serial Programming™ (ICSP™) mode, the RTS/ $\overline{\text{MCLR}}$ diode can interfere with traditional ICSP programming/debugging. In this case, using an N-channel metal oxide semiconductor field effect transistor (MOSFET) may be more appropriate. (See Figure 8.)

An RS-232 transceiver chip typically provides an internal pull-down resistor on the incoming TXD/RTS signals. This allows the PIC device to normally see logic level high on its RX pin (the RS-232 "Idle" state) even when disconnected at the serial port, DB9 connector.

FIGURE 7: RX PIN SCHEMATIC

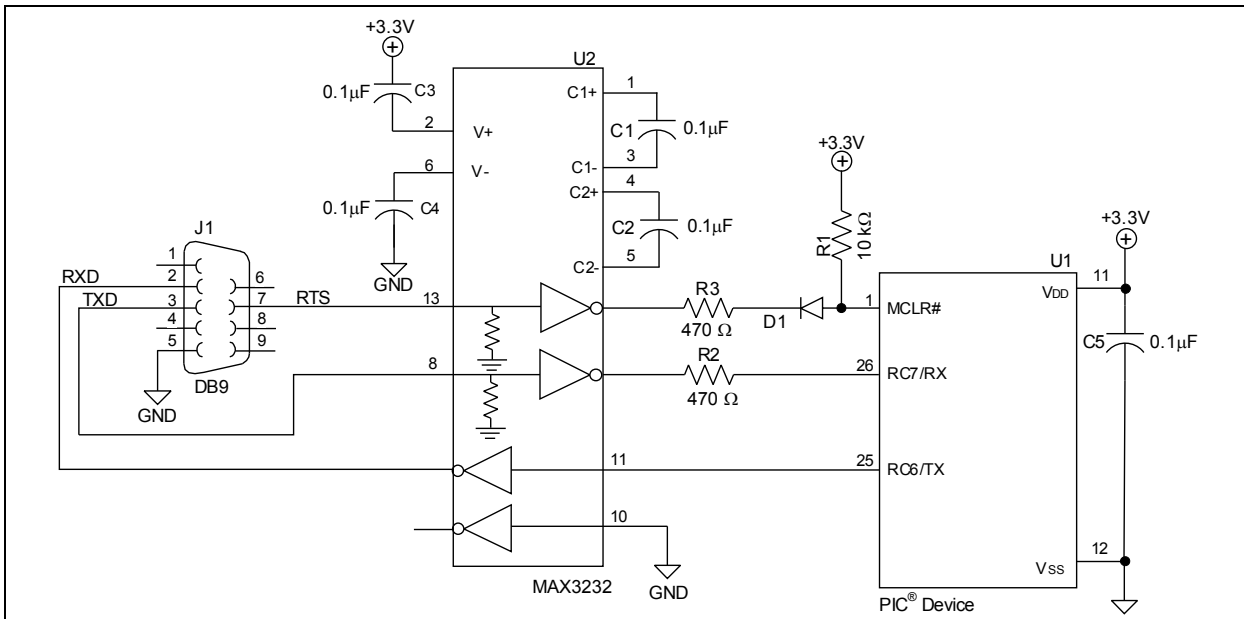
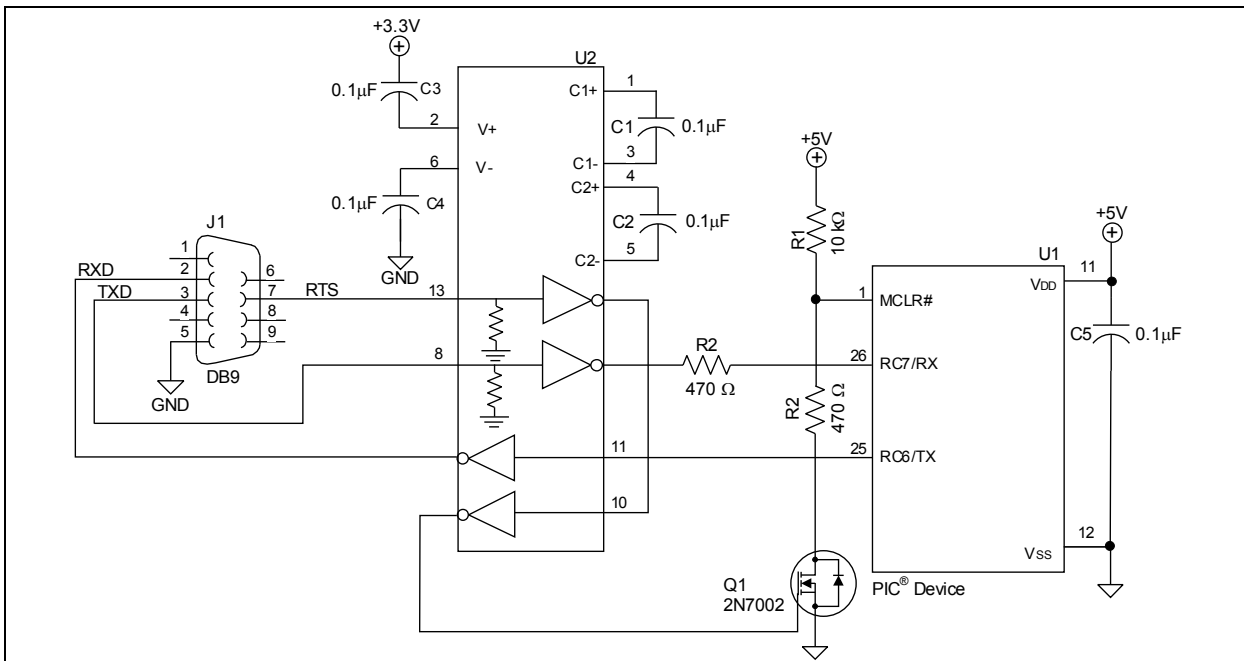


FIGURE 8: MOSFET SCHEMATIC



Entering Application Mode

To ensure the device can reliably enter Application mode, it is suggested that:

- The device be held in Reset long enough for the RX pin to be pulled high, signifying the RS-232 "Idle" state
- Data not be transmitted to the microcontroller during, and immediately after, the device has come out of Reset

Incoming data could be misinterpreted as an RS-232 "Break" state, causing the bootloader start-up routine to enter Bootloader mode.

When using a custom RS-232 transceiver circuit, ensure that the PIC device's RX pin is pulled up to VDD when Idle or disconnected from the host PC serial port. If the PIC device's RX pin is left floating or pulled down, Bootloader mode could be erroneously entered.

High-Speed Baud Rates

The bootloader is capable of operating at baud rates of up to 3 Mbps. Reliably achieving high-speed baud rates requires some considerations in the hardware design.

Traditionally, PCs' serial ports only operate at standard baud rates of up to 115.2 kbps. Manufacturers of USB-to-serial converters may claim higher baud rates, but some incorporate level translator circuits that limit performance to baud rates of 500 kbps or less.

More reliable, high-speed operation may be provided by a bare logic level, USB-to-serial converter circuit wired directly to the PIC device without RS-232 level translators.

Another hardware design consideration is matching the PIC device's clock source frequency to the baud rates available on the host serial port. At high speeds, USB-to-serial converters may have limited steps between available baud rates.

As an example, some of the currently available baud rates for two USB-to-serial converters are shown in Table 2. (For more information, consult Prolific Technology Inc. (PL2303), Future Technology Devices International Limited (FT232BM) or your converter's data sheet.)

TABLE 2: EXAMPLE HIGH-SPEED, USB-TO-SERIAL BAUD RATES

PL2303	FT232BM
6,000,000	3,000,000
3,000,000	2,000,000
2,457,600	1,500,000
1,228,800	1,411,765
921,600	1,333,333
614,400	1,263,158
460,800	1,200,000
230,400	1,142,857
	1,000,000
	750,000
	500,000
	250,000

The PIC device also will have limited steps between available baud rates, dependent on the FOSC clock source used and the PIC device's baud rate formula (see Table 3).

TABLE 3: PIC® DEVICE BAUD RATE FORMULAS

BRG16	BRGH	PIC Baud Rate
0	1	$F_{osc}/[16 (BRG + 1)]$
1	1	$F_{osc}/[4 (BRG + 1)]$

The bootloader firmware source code automatically uses the BRG16 = 1, BRGH = 1 mode on microcontroller devices that support it. This is the most flexible mode for hitting the widest range of possible baud rates.

To avoid miscommunication, the PIC device and serial port baud rates ideally should match within 3%. If a CRC error is detected during communications with the bootloader firmware, the host PC application will halt and display an error status.

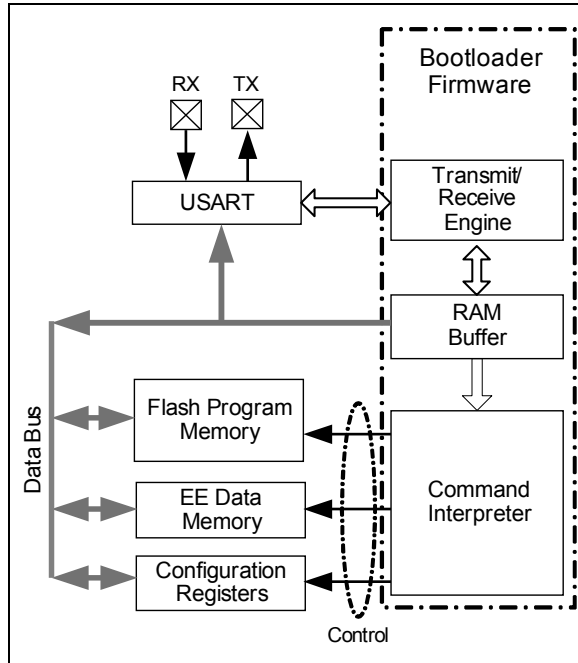
AN1310

Bootloader Firmware Operation

Figure 9 summarizes the essential operating design of the bootloader.

Data is received through the USART module and quickly stored to the RAM buffer to avoid overrun errors. After each packet is received, the integrity of the data is verified using a 16-bit CRC.

FIGURE 9: BOOTLOADER FUNCTIONAL BLOCK DIAGRAM



When a valid request packet has been received, the command interpreter evaluates the command number in the packet to determine what operation needs to be done (such as Erase, Write, Read or Verify). The request is fulfilled and a response is returned through the USART to either Acknowledge completion of the task, or on Read operations, to send back device memory data.

The host PC application is not allowed to send more than one packet at a time. Each packet must be Acknowledged before the next can be sent. This maintains flow control.

For detailed bootloader protocol documentation, see **Appendix A: "Bootloader Protocol" on page 19.**

PIPELINED READS

This bootloader avoids using the RAM buffer during device read operations, unlike the original bootloader document in AN851.

Data read from Flash/EEPROM is streamed directly to the USART module. This has some key advantages:

- The microcontroller RAM size no longer limits the response packet size
- An entire device memory range can be read with a single request transaction, minimizing transaction overhead
- The processor core and the USART can operate in parallel (pipelining), reducing total clock time

USART COMMUNICATIONS

The microcontroller's USART module is used to receive and transmit data. The communications settings are:

- Eight data bits
- No parity
- One Start/Stop bit
- Variable baud rate

An auto-baud routine is used to measure the bit rate of the initial Start-of-Text character (STX or 0Fh). Unlike the original bootloader, the auto-baud routine is not run at the beginning of every packet. Once the baud rate is successfully captured, it is locked in until the bootloader command loop receives an invalid or unexpected request.

Locking the baud rate allows high baud rate data to be received without losing data during slow auto-baud calculations. When an invalid or unexpected request is received, the auto-baud routine is run again.

When switching between baud rates, the bootloader firmware can get locked at the wrong baud rate and the incoming data stream may not trigger the auto-baud routine to be run again. In such situations, a $\overline{\text{MCLR}}$ Reset is required to force the auto-baud routine to run.

When RTS is wired to control $\overline{\text{MCLR}}$, such situations can be handled automatically by the host PC application asserting a $\overline{\text{MCLR}}$ Reset.

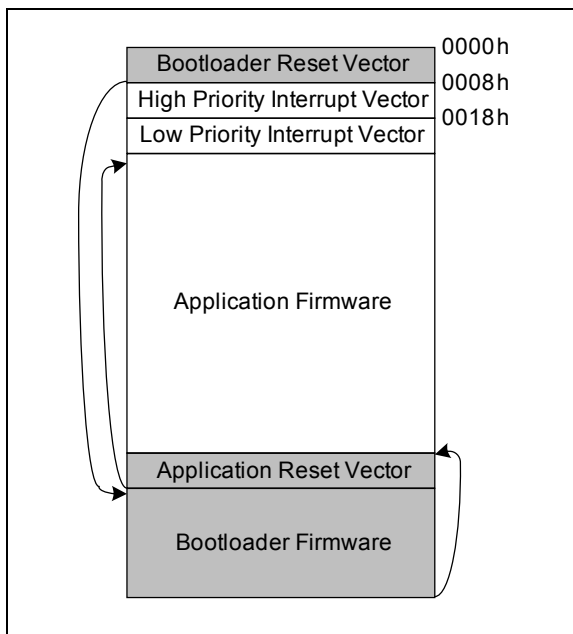
BOOTLOADER MODE CONSIDERATIONS

In the default configuration, bootloader firmware is stored at the end of Flash program memory space. Keeping the bootloader at the end of program memory space allows application firmware to handle interrupts at the normal hardware interrupt vector address. This configuration keeps interrupt latency to a minimum.

The host PC bootloader application will write a “GOTO” as the first instruction at the Reset vector (address, 0000h). This GOTO jumps to the bootloader at start-up. The application firmware’s original Reset vector instruction is automatically moved to reside just before the bootloader block of memory. The bootloader firmware uses this application Reset vector to start the application when the Bootloader mode is not requested (see Figure 10).

On PIC18 devices, this automatic relocation of the Reset vector requires the application firmware to provide a single GOTO instruction as the first instruction at address, 0000h.

FIGURE 10: PROGRAM MEMORY



On PIC16 devices, program memory is paged. Therefore, the PCLATH register needs to be loaded before executing a far GOTO. The host PC bootloader application will write up to four instructions at the Reset vector (address, 0000h) to jump to the bootloader at start-up.

To make room, the first four application firmware instructions are automatically moved by the host PC software to reside just before the bootloader firmware.

Example 2 shows a PIC16 far GOTO code sequence that will be recognized as a valid application Reset vector by the software.

EXAMPLE 2: PIC16 RESET VECTOR

```

ORG 0
ResetVector:
    movlw    high(FarApplication)
    movwf   PCLATH
    goto    FarApplication

    (...)

FarApplication:
    (...)

```

Incremental Bootloading

After the PIC device has been programmed, the host PC application continues to monitor the hex file on the disk. If the application firmware is modified and recompiled, the host PC bootloader application will immediately notice the change. This triggers the host PC application to do an incremental update of the device’s firmware.

Incremental updates compare the last programmed hex file and the new file to determine which bytes of program memory to change. Only memory blocks with changes are reprogrammed. When developing large application projects, this can save significant time and increase productivity.

Write Protection

To ensure that bootloader firmware is always available for updating the device, it is wise to write-protect the bootloader’s block of Flash memory space (the boot block). This can be accomplished through software or hardware.

SOFTWARE WRITE PROTECTION

Software-based boot block write protection is enabled by the USE_SOFTBOOTWP option in the file, bootconfig.inc. With this feature turned on, the bootloader firmware checks each erase and write operation to ensure the affected addresses are outside of the boot block. Erase/write operations within the boot block are silently skipped.

Having write access to Configuration bits is very powerful, but also dangerous. If a design only has enough hardware for operating from the internal oscillator and accidentally changes the Configuration bits to use EC mode, all further operation could be prevented, including Bootloader mode.

Configuration bits can be write-protected in software with the USE_SOFTCONFIGWP option in the include file, bootconfig.inc. PIC16 devices do not have Configuration bit write access.

AN1310

HARDWARE WRITE PROTECTION

Certain PIC18 devices, such as the PIC18F46J11, provide Configuration bits to allow write protection of the end of the Flash memory space.

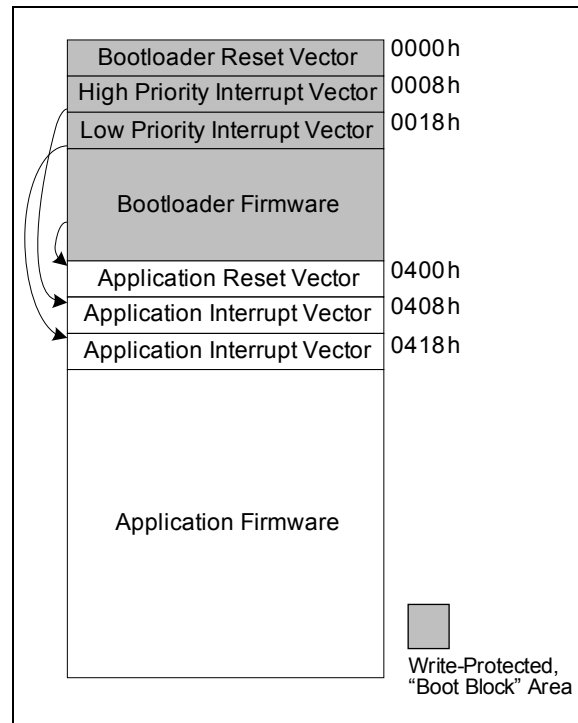
Most devices, however, only provide write protection from the beginning of the Flash memory space. For these devices, it will be necessary to locate the bootloader at address, 0h, in Flash memory. To do that, use the `#define BOOTLOADER_ADDRESS 0` option in the `bootconfig.inc` file.

When located at address, 0h, the bootloader occupies the hardware Reset and interrupt vector addresses. For this layout to work, application firmware must provide remapped Reset and interrupt vectors. The bootloader firmware will “pass-through” hardware Reset and interrupt events to the application firmware at the remapped addresses (see Figure 11).

Note: Some PIC devices provide configuration options that can write-protect application firmware’s program memory regions. Write-protected regions will not be modifiable by the bootloader.

If the pre-existing contents of write-protected regions do not match the new data in the application firmware’s hex file, write protection may prevent the bootloader write operation from completing successfully.

FIGURE 11: MEMORY, WITH WRITE-PROTECTED AREA



The `bootconfig.inc` file defines where the bootloader firmware looks for the following application vectors: `AppVector`, `AppHighIntVector` and `AppLowIntVector`. To optimize usage of Flash memory for a specific application, these addresses can be adjusted. For now, however, leave the addresses at their default values to maintain compatibility with the example application firmware projects.

APPLICATION MODE CONSIDERATIONS

Having bootloader code in Flash memory may require some changes to application firmware. To facilitate that, this application note provides software mechanisms and example application firmware projects that should work “out of the box” on all configurations.

Placing the bootloader firmware towards the end of Flash memory space should require little or no changes to application firmware, due to the automatic Reset vector remapping done by the host PC bootloader software. Interrupt vectors are handled by application firmware at the normal hardware interrupt vector addresses, so no application firmware changes are required in this design. For an example memory map of this design type, see **Figure 10 on page 9**.

If the bootloader code is placed at address, 0h, the application firmware will have to support remapping of the hardware Reset and interrupt vectors to new locations. These new vector locations will be used by “pass-through” bootloader firmware occupying the hardware vector addresses. For an example memory map of this design type, see **Figure 11 on page 10**.

This section discusses how the example application firmware projects were modified to operate with bootloader firmware in both design types.

Note: The example application firmware projects are intended to communicate via UART1 on the PIC device. Unlike the bootloader firmware, no auto-baud code is included.

Before compiling, edit the example source code to provide the correct Baud Rate Generator (BRG) value, based on the clock frequency and the baud rate desired. The source code gives some common values in commented sections.

MPLAB® IDE C18 Applications

When using the MPLAB IDE C18 compiler to develop application firmware, the first Reset vector instruction generated by the C compiler is, by default, a “GOTO”. This allows the bootloader to be used with no code changes required.

The C18 compiler grows application code from the beginning of program memory space, with minimal fragmentation. This allows the bootloader firmware, by default, to stay resident at the end of program memory space without conflicting with application firmware code. As a result, it is usually unnecessary to modify linker scripts to reserve program memory space for the bootloader firmware.

PIC18 MCC18 REMAPPED APPLICATION EXAMPLE

Example remapped application firmware for MCC18 on PIC18 is installed to:

```
C:\Microchip Solutions\Serial
Bootloader AN1310 vX.XX\PIC18 Application\
MCC18 Remapped Application\
```

The C code for this project contains all of the necessary modifications to operate with a bootloader at address, 0h. All of the work is done through C code (see Example 3).

AN1310

EXAMPLE 3: PIC18 REMAPPED APPLICATION C CODE

```
// Prevent application code from
// being written into FLASH
// memory space needed for the
// Bootloader firmware at
// addresses 0 through 3FFh.
#pragma romdata BootloaderProgramMemorySpace = 0x6
const rom char bootloaderProgramMemorySpace[0x400 - 0x6];

// The routine _startup() is
// defined in the C18 startup
// code (usually c018i.c) and
// is usually the first code to be called by a GOTO at the
// normal reset vector of 0.

extern void _startup(void);

// Since the bootloader isn't
// going to write the normal
// reset vector at 0, we have
// to generate our own remapped
// reset vector at the address
// specified in the
// bootloader firmware.

#pragma code AppVector = 0x400
void AppVector(void)
{
    _asm GOTO _startup _endasm
}

// For PIC18 devices the high
// priority interrupt vector is
// normally positioned at
// address 0008h, but the
// bootloader resides there.
// Therefore, the bootloader's
// interrupt vector code is set
// up to branch to our code at
// 0408h.

#pragma code AppHighIntVector = 0x408
void AppHighIntVector(void)
{
    _asm GOTO high_isr _endasm           // branch to the high_isr()
                                        // function to handle priority
                                        // interrupts.
}

#pragma code AppLowIntVector = 0x418
void low_vector(void)
{
    _asm GOTO low_isr _endasm           // branch to the low_isr()
                                        // function to handle low
                                        // priority interrupts.
}

#pragma code                               // return to the default
                                        // code section
```

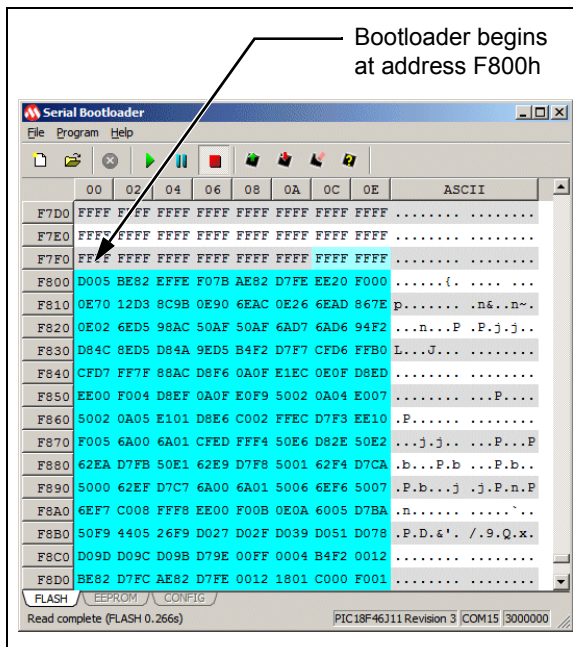
There are no assembly language helper files, project build options or customized linker scripts for remapping application vectors in Example 3. Everything is handled by the code.

HI-TECH C® Applications

The HI-TECH C compiler often fragments application code into portions of program memory space that conflict with the bootloader firmware. To prevent problems with application code conflicting with bootloader firmware sharing the same device program Flash memory, the HI-TECH C project must be modified to reserve program memory space for the bootloader.

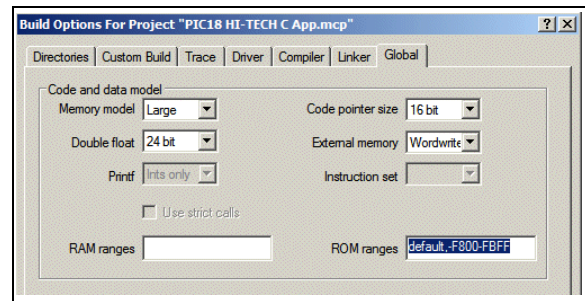
The addresses used by the bootloader firmware can be determined by looking at the Flash memory display inside the host PC bootloader application. While connected to the bootloader, scroll down to the end of Flash memory space where you can find the dark turquoise shaded memory region reserved for the bootloader (see Figure 12).

FIGURE 12: BOOTLOADER MEMORY, HIGHLIGHTED



Add the bootloader region to be reserved in the HI-TECH C project's "Build Options", under the Global tab (see Figure 13).

FIGURE 13: BUILD OPTIONS DIALOG BOX



Set the "ROM ranges" field to something like the following:

```
default, -F800-FBFF
```

The word, "default", tells the compiler to use the entire Flash program memory space of the device, while the address range, "-F800-FBFF," with a minus sign in front, tells the compiler to exclude addresses, F800h through FBFFh, from being used for the application firmware.

This prevents application code from conflicting with the bootloader code.

AN1310

PIC16 HI-TECH C REMAPPED APPLICATION EXAMPLE

By default, example remapped application firmware for HI-TECH C on PIC16 devices is installed to:

```
C:\Microchip Solutions\Serial
Bootloader AN1310 vX.XX\PIC16 Application\
HI-TECH C Remapped Application\
```

At first glance, the C code for this project looks pretty similar to normal application code. However, some slight changes are made:

- An assembly language helper file (`isr.as`) is added to handle Reset and interrupt vector remapping.

This file should be copied into user's projects and can be customized with an alternative to the "ORG 0x400" Reset vector address. The 0x400 address should match the bootloader `AppVector` setting.

- The `Codeoffset` option is reset to 404.

The `Codeoffset` hex number must always be equal to the application Reset vector plus four (`AppVector + 4`). This causes the HI-TECH C compiler to generate Reset and interrupt vector code further into Flash memory space than the default of address 0.

To verify this, open the project in MPLAB IDE, select **Project>Build Options...>Project** and go to the **Linker** tab menu.

- To prevent the HI-TECH C compiler from using the RAM at address, 7Eh and 7Fh, the RAM ranges option is set to `default, -7E-7F`

The bootloader uses access bank RAM at 7Eh and 7Fh to pass the PCLATH and WREG registers, respectively, to the assembly language helper code in the file, `isr.as`. The address numbers used must match the PCLATH_TEMP and W_TEMP address definitions in the bootloader firmware source code.

To verify this, open the project in MPLAB IDE, select **Project>Build Options...>Project** and go to the **Global** tab menu.

Note: Enhanced architecture PIC16F devices provide automatic hardware to save and restore ISR context during interrupt handling. Thus, it is possible to omit the "RAM ranges" setting on enhanced core PIC16 devices.

- The ROM ranges option is set to prevent the HI-TECH C compiler from using Flash memory space from address, 0h, through the end of the bootloader firmware.

Depending on the size of the bootloader firmware and write-protected boot block, the end address to be excluded may be changed to free more Flash memory for application code.

To verify this, go to the same **Global** tab menu.

PIC18 HI-TECH C REMAPPED APPLICATION EXAMPLE

By default, example remapped application firmware for HI-TECH C on PIC18 is installed to:

```
C:\Microchip Solutions\Serial
Bootloader AN1310 vX.XX\PIC18 Application\
HI-TECH C Remapped Application\
```

The code for this project is normal application code. There is no need for an assembly language helper file, as required for PIC16 devices. Instead, only build options need to be configured.

1. Select **Project>Build Options...>Project** and go to the **Linker** tab menu.

The "Codeoffset" field has been set to 400.

This address must be equal to the bootloader defined `AppVector` address in order for the HI-TECH C compiler to generate Reset and interrupt vector code further into Flash memory space than the default of address, 0h.

2. Go to the **Global** tab menu and review the ROM ranges field.

This option has been set to prevent the HI-TECH C compiler from using Flash memory space from address, 0h, through the end of the bootloader firmware.

Note: If the size of the bootloader firmware and write-protected boot block permits, the end address can be adjusted to free more Flash memory for the application code. For now, leave the end address as: `default, -0-3FF`.

On PIC18 devices, there is no need to exclude RAM ranges. The PIC18 bootloader does not use any RAM to save context data when passing through control to your application's interrupt vector code.

SOFTWARE DESIGN

This section discusses the host PC software and write planning for the bootloader application.

Host PC Software

The host PC application accepts the following command-line options:

```
Serial Bootloader.exe [/e] [/p] [/v]
[filename.hex]
```

- /e – Erase the device.
- /p – Program the device with the specified hex file.
- /v – Verify that the device matches the data in the specified hex file.

Software used for building the host PC software includes:

- Qt SDK 4.x – Lesser General Public License software, used unmodified and dynamically linked
- QextSerialPort 1.2 – Public domain software, heavily modified for this application
- SQLite 3.6.x – Public domain software, used unmodified

Using only open source and public domain development tools/libraries facilitates porting the software to other operating system platforms, such as Linux[®], or Macintosh[®]. As of this writing, the software is known to compile and work on Linux. No testing has been done for Macintosh.

Write Planning

The original bootloader application, documented in AN851, used a simple algorithm for writing new firmware: erase all memory and write all memory with new data.

With large memory devices, such as the PIC18F87J11, erasing all of Flash memory can take almost four seconds, followed by the transferring of 128 Kbytes of new Flash data at 115.2 kbps, which can take another 11 seconds or more, at lower baud rates.

When developing new application firmware, typically the target PIC microcontroller has far more Flash memory than what is needed by application firmware. As a result, the earlier bootloader's simple algorithm can become noticeably inefficient.

This serial bootloader incorporates a different algorithm, "write planning". The application firmware hex file is analyzed to produce two lists of memory regions: Erase and Write. These lists form a "Write Plan" that the software uses to issue erase and write commands to the bootloader firmware kernel. For an example Write Plan, see Table 4.

TABLE 4: EXAMPLE WRITE PLAN

Erase List	
Start Address	End Address
1F800h	1F400h
1C00h	0h
20000h	1FC00h
Write List	
Start Address	End Address
1FFC0h	20000h
0h	1AC0h
1F7C0h	1F800h

DEVELOPING THE WRITE PLAN

In the Write Plan process:

1. Regions of Flash memory space are added to the Write List, aligned to the Flash write block boundaries. Any blocks of memory that are found to contain empty data (such as NOPs) are excluded from being added to the Write List.
2. The Write List is copied to create the Erase List, except that memory address regions are aligned to Flash erase block boundaries.

Using Erase and Write Lists makes it easy to reorder the memory regions to be erased or written first, for fail-safe operation in case of interruption during a write operation.

On "J" family devices, where Configuration bits are stored at the end of Flash memory, the Flash erase block with Configuration bits is scheduled to be erased last. The Flash write block containing the new Configuration bits is scheduled to be written immediately afterwards, as the first write transaction.

Scheduling Configuration bits' erase/write minimizes the time that Configuration bits could be left blank on the device.

3. Flash memory is erased in descending order by the erase block address.

This ensures that being interrupted will force the bootloader to stay in Bootloader mode on the next Reset, rather than attempt starting partially erased application firmware.

EXECUTING THE WRITE PLAN

After generating the plan, the Erase List is used to erase all necessary Flash memory and the Write List is used to rewrite necessary Flash memory.

To prevent leaving “junk” data in previously programmed Flash memory regions that the new firmware no longer uses, the following additional steps are taken:

1. The entire Flash memory space is verified by calculating a 16-bit CCITT CRC against each Flash erase block, both on the device and on the host PC.

The calculated CRC values should match, unless a block on the device contains leftover junk data that needs to be erased.

2. The host PC compares CRC values calculated on the device and the desired CRC values calculated on the host PC. When a block needs to be erased to match the desired CRC value, it is added to a new Erase List. If erasing the block will not make it match the desired CRC value, an error is displayed.
3. The new Erase List is executed to clear any leftover junk data from old firmware.

Using CRC numbers enables the bootloader to deal with junk data without having to transfer the entire memory contents back to the host PC or resort to blindly erasing every Flash memory address. This algorithm significantly speeds up reprogramming of devices with prior data.

TABLE READS

Some PIC devices provide Configuration bit options for “Table Read Protect,” which can prevent bootloader table read operations.

The bootloader firmware needs to be able to perform table reads for the following operations:

- Read Flash program memory contents
- Generate CRC values to find junk data for final pass erase
- Generate CRC values to verify correct Flash memory contents
- Read Device and Revision ID numbers to look up essential device characteristics database information

Because this bootloader requires extensive use of table reads, enabling “Table Read Protect” Configuration bits is not recommended.

CODE PROTECTION

Some PIC devices provide “Code-Protect” Configuration bit options. Code-protect is intended to prevent ICSP™ read-out of device memory contents.

When the PIC device is programmed with this bootloader firmware, however, code protection can be easily circumvented. This bootloader performs table reads on device memory contents whether “code-protect” Configuration bits are enabled or not. This could allow the application to be read by an attacker.

For applications where security is a concern, *AN1157*, “A Serial Bootloader for PIC24F Devices” (DS01157), may be more appropriate. An encryption-enabled version of AN1157 may be ordered from Microchip sales offices.

SUPPORTING NEW DEVICES

To enable the serial bootloader to keep pace with the release of new, enhanced microcontrollers, it employs a SQLite™ database on the host PC. The PC software quickly reviews device information when connecting to the bootloader firmware. The database can be updated without recompiling the host PC software.

The bootloader firmware needs to have the same adaptability, but obviously can not use a database. Instead, the bootloader firmware relies on the include file, `DEVICES.INC`, to provide device-specific information at compile time.

With this approach, updating support for a new device simply requires adding the new device information to the database and include files, assuming code compatibility with the new device.

Modifying the SQLite™ Database

By default, the SQL source code to the device database is installed to:

```
C:\Microchip Solutions\Serial
Bootloader AN1310 vX.XX\Device Database\
devices.sql
```


The database currently has two tables: DEVICES and CONFIGWORDS.

To update the SQLite database for a new device:

1. Add a single DEVICES record, using an SQL insert statement like that shown in Example 4.

EXAMPLE 4: SQL INSERT STATEMENT

```
insert into DEVICES values (
    161, -- Device ID (in decimal)
    4,   -- Family ID (2 for PIC16, 4 for
PIC18)
    'PIC18F8722',
    2,   -- Bytes Per Word (FLASH)
    64,  -- Write FLASH Block Size
    64,  -- Erase FLASH Page Size
    '0x0', -- Start address of FLASH
    '0x20000', -- End address of FLASH
    '0xF00000', -- Start address of EEPROM
(use 0 if no EEPROM on your device)
    '0xF00400', -- End address of EEPROM
    '0x200000', -- Start address of User ID
(use 0 if no User ID space on your device)
    '0x200008', -- End address of User ID
    '0x300000', -- Start address of Config
Bits
    '0x30000E', -- End address of Config
Bits
    '0x3FFFFE', -- Start address of Device
Id
    '0x400000', -- End address of Device Id
    '0xFFE0', -- Device Id Mask (so that
Revision ID bits are ignored)
    '0x0', -- Start address of GPR
    '0xF60' -- End address of GPR (tells the
PC software how big packets can be
           -- without overflowing the
device buffer RAM)
);
```

2. If the new device uses configuration fuses, add records to the CONFIGWORDS table for each Configuration Word, as shown in Example 5.

EXAMPLE 5: CONFIGWORDS ADDITIONS

```
insert into CONFIGWORDS values (
    161, -- Device ID
    4,   -- Family ID
    'CONFIG1H', -- Config Name
    3145729, -- Address
    '0x37', -- Default Value
    '0xCF' -- Implemented Bits
);

insert into CONFIGWORDS values (
    161, -- Device ID
    4,   -- Family ID
    'CONFIG2L', -- Config Name
    3145730, -- Address
    '0xFF', -- Default Value
    '0x1F' -- Implemented Bits
);

(...)
```

This allows the verify operation to mask off unused Configuration Word bits to avoid a possible false verify failure.

3. Regenerate the binary devices.db database file, used at run time by the host PC software:

- Go to the following command prompt path:

```
C:\Microchip Solutions\Serial
Bootloader AN1310 vX.XX\Device Database
```

- Run the following command:

```
sqlite3.exe -init devices.sql -batch
..\devices.db .quit
```

The updated devices.db file must be kept in the same folder as the host PC serial bootloader executable file.

Modifying the DEVICES .INC Include File

To update the DEVICES.INC file for a new device:

1. In MPLAB IDE, open the bootloader firmware project file appropriate for the new device.
2. Open the DEVICES.INC file.
3. Add a new #ifdef block for the new device, as shown in Example 6.

The device numbers in the DEVICES.INC file must match those used in the SQL database script (Example 5).

EXAMPLE 6: INCLUDE FILE ADDITION

```
#ifdef __18F8722
#define DEVICEID .161
#define WRITE_FLASH_BLOCKSIZE .64
#define ERASE_FLASH_BLOCKSIZE .64
#define END_FLASH 0x20000
#define END_GPR 0xF60
#endif
```

Simplifying Device Data Collection

The device and family numbers required in the preceding examples can be obtained from the devices' data sheets, programming specifications and reference manuals. The data can be acquired more easily, however, by using Microchip's XML "essential device characteristics" files.

These XML files, used to generate the MPLAB processor-specific header files, MPLAB configuration screens and other items, are in a compressed file stored on the host PC during the MPLAB IDE installation. A "Device Database" utility program can generate the SQL and include information for new devices.

AN1310

To use the utility:

1. Go to the following path and decompress the bundled XML files.

```
C:\Program Files\Microchip\  
MPLAB IDE\Device\*PIC.zip
```

2. Go to the following path and run the executable file, Device Database.exe:

```
C:\Microchip Solutions\  
Serial Bootloader AN1310 vX.XX\
```

The utility's dialog box appears.

3. Select *File>Open PIC Definitions* and select the PIC file(s) for the desired device(s) (for example, PIC18F6520.PIC).

To generate information for more than one device, hold down <Ctrl> as you select the second and subsequent PIC file names, before clicking the **Open** button.

The utility displays the device information on the **devices.inc** tab, as shown in Figure 14.

REFERENCES

Brant Ivey, *AN1157, "A Serial Bootloader for PIC24F Devices"* (DS01157), Microchip Technology Inc., 2008.

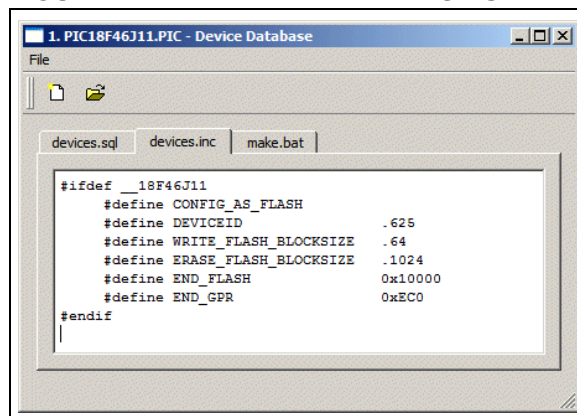
Ross M. Fosler and Rodger Richey, *AN851, "A FLASH Bootloader for PIC16 and PIC18 Devices"* (DS00851), Microchip Technology Inc., 2002.

Alternative References

For information on other bootloaders, visit the following web sites:

- USB bootloaders — <http://www.microchip.com/usb>
- TCPIP bootloaders — <http://www.microchip.com/tcpip>
- dsPIC® DSCs and 32-bit bootloaders — <http://www.microchip.com/pic32libraries>

FIGURE 14: DEVICE DATABASE UTILITY



APPENDIX A: BOOTLOADER PROTOCOL

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

Packet Format

All data packets transmitted from the host PC to the microcontroller follow the basic packet format:

```
[ <STX>... ] <STX> [ <DATA>... ] <CRCL> <CRCH> <ETX> [ <ETX> ]
<...> – Represents a byte
[... ] – Represents an optional or variable number
of bytes
```

The maximum packet length is constrained by the microcontroller's RAM. Host PC software should determine what device is being used and look up the RAM size in the essential device characteristics database to avoid sending more data than can be buffered by the microcontroller. For a working implementation, see the example host PC software source code and `Device.cpp` and the `maxPacketSize()` function.

Most response data packets transmitted from the bootloader firmware, back to the host PC, follow a similar basic packet format:

```
[ <STX>... ] <STX> [ <DATA>... ] <CRCL> <CRCH> <ETX>
```

Maximum response packet length is limited by the number of blocks requested by the host PC software in the command packet.

Control Characters

There are three control characters that have special meaning. Two of them, <STX> and <ETX>, are shown in the previous examples. The remaining character is "Data Link Escape", <DLE> (0x05 in hexadecimal).

TABLE 5: CONTROL CHARACTERS

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

<DLE> is used to escape a byte that could be interpreted as a control character. The bootloader will always accept the byte following a <DLE> as data and will always send a <DLE> before any of the control characters. The data payload and CRC bytes will be escaped by <DLE> characters when the data happens to match a control character.

The <STX> or "Start of TeXt" control character (0x0F in hexadecimal) serves multiple purposes: auto-baud, flow control and packet framing.

For establishing initial communications, the pulse width of the <STX> character is measured by the bootloader firmware to calculate the rate at which the host PC is transmitting data (auto-baud).

The host PC will transmit <STX> characters to the microcontroller up until the microcontroller echoes back an <STX> character of its own. The host PC must not begin transmitting the data payload field until an echoed <STX> character is received back from the microcontroller. If data were to be transmitted immediately, without an echoed <STX> character, data could be lost because the microcontroller firmware is busy performing auto-baud or some other task.

The <ETX> or "End of TeXt" control character (0x04 in hexadecimal) marks the end of the packet. Sending an extra <ETX> can also be used by the host PC software to force a baud rate locked bootloader back into its auto-baud routine. This provides a faster and more reliable resynchronization for baud rate changes made on the host PC.

Cyclic Redundancy Check (CRC)

A 16-bit CCIT CRC algorithm is used to ensure that received data has not been corrupted by the serial communication link. A 16-bit CRC may detect bursts of bad data, up to 15 bits in length, while simple checksum algorithms can be unreliable at detecting multi-bit errors.

The 16-bit CCIT CRC algorithm is the same CRC algorithm specified in the MMC/SD Card SPI protocol commonly used by embedded applications. A free, easy-to-use CRC calculator and source code generator are available from Thomas Pircher's "pycrc" tool:

<http://www.tty1.net/pycrc/>

The bootloader firmware and host PC software calculate the CRC against each byte of the data payload, excluding control characters. If the calculated CRC does not match the transmitted CRC, the packet is assumed to be corrupted and is discarded.

The CRC algorithm is doubly useful for quickly verifying that Flash memory has been programmed correctly. Calculating CRC values against blocks of Flash memory can be done much faster than transferring every byte of Flash memory across a slow serial communications link.

AN1310

Bootloader Commands

Read Bootloader Information

Request

```
<STX> <0x00> <CRCL><CRCH> <ETX>
```

PIC18 Response

```
<STX> <BOOTBYTESL><BOOTBYTESH> <VERSIONL><VERSIONH> <COMMANDMASKH> <COMMANDMASKL:FAMILYID>  
<STARTBOOTL><STARTBOOTH><STARTBOOTU><0x00> <CRCL><CRCH> <ETX>
```

PIC16 Response

```
<STX> <BOOTBYTESL><BOOTBYTESH> <VERSIONL><VERSIONH> <COMMANDMASKH> <COMMANDMASKL:FAMILYID>  
<STARTBOOTL><STARTBOOTH><STARTBOOTU><0x00> <DEVICEIDL><DEVICEIDH> <CRCL><CRCH> <ETX>
```

Details

CRC for the request packet is, conveniently, always 0x0000 for this packet.

- **VERSION** defines the version number of the bootloader firmware.
- **BOOTBYTES** specifies the boot block size, in bytes.
- **STARTBOOT** specifies the starting Flash memory address for the boot block.

Together, **STARTBOOT** and **BOOTBYTES** are used in the host PC software to color the bootloader Flash memory region turquoise.

- **COMMANDMASK** is currently unused on PIC18.

For PIC16, **COMMANDMASKH** will contain 0x01 if the device implements the erase Flash command. Otherwise, it contains 0x00, signifying that the PIC16 device does automatic erases during writes (for example, PIC16F88X family).

- **FAMILYID** is a four-bit number in the least significant nibble, indicating what kind of microcontroller is in use. Currently used family IDs are: **2** – PIC16, **4** – PIC18.
- **DEVICEID** is only transmitted for PIC16 devices. It is a compile-time value indicating exactly which PIC16FXXX part number is being used.

PIC18 parts do not need to transmit the **DEVICEID** here because the host PC application can use the read Flash memory command, below, to read the device ID from configuration memory at address 0x3FFFE.

The host PC application uses the Family ID and Device ID to look up the correct device information in the essential device characteristics database.

Read Flash Memory

Request

```
<STX> <0x01> <ADDRESSL><ADDRESSH><ADDRESU><0x00> <BYTESL><BYTESH> <CRCH><CRCL> <ETX>
```

Response

```
<STX> [DATA...] <CRCL><CRCH> <ETX>
```

Details

- **ADDRESS** is the beginning Flash memory address from which to start reading.
- **BYTES** specifies the number of bytes to read.

Read CRCs of Flash Memory

Request

```
<STX> <0x02> <ADDRESSL><ADDRESSH><ADDRESSU><0x00> <BLOCKSL><BLOCKSH> <CRCL><CRCH> <ETX>
```

Response

```
<STX> [ <CRC1L><CRC1H>...<CRCL><CRCH> ] <ETX>
```

Details

- ADDRESS is the beginning Flash memory address from which to start reading.
- BLOCKS specifies that number of 16-bit CRC words to generate. Each 16-bit CRC is generated on erase Flash block size number of bytes.

Note that the response packet of this command does not provide a CRC of the data payload, which is a little different from most of the packets. This allows the bootloader firmware to avoid having to calculate two different CRCs at the same time, reducing memory use and processing time.

Erase Flash Memory

Request

```
<STX> <0x03> <ADDRESSL><ADDRESSH><ADDRESSU><0x00> <PAGESL> <CRCL><CRCH> <ETX>
```

Response

```
<STX> <0x03> <CRCL><CRCH> <ETX>
```

Details

- ADDRESS is the last Flash memory address to start erasing from. Erases are performed in *descending* address order, which is backwards from normal operations.
This feature can help the bootloader to fail more safely in some situations.
- PAGES specifies the number of erase Flash block size pages of Flash memory.

Write Flash Memory

Request

```
<STX> <0x04> <ADDRESSL><ADDRESSH><ADDRESSU><0x00> <BLOCKSL> [ <DATA>... ] <CRCL><CRCH> <ETX>
```

Response

```
<STX> <0x04> <CRCL><CRCH> <ETX>
```

Details

- ADDRESS is the Flash memory address to start writing to. Writes are performed in ascending address order.
- BLOCKS specifies the number of write Flash block size chunks of Flash memory to write.

Flash memory cells must be erased before writing can be performed on the same cells again.

AN1310

Read EEPROM

Request

```
<STX> <0x05> <ADDRESSL><ADDRESSH><0x00><0x00> <BYTESL><BYTESH> <CRCH><CRCL> <ETX>
```

Response

```
<STX> [ <DATA...> ] <CRCL><CRCH> <ETX>
```

Details

- ADDRESS is the beginning EEPROM address from which to start reading.
- BYTES specifies the number of bytes to read.

Microcontrollers that do not have EEPROM immediately send the response packet with a dummy payload of 0x05.

Write EEPROM

Request

```
<STX> <0x06> <ADDRESSL><ADDRESSH><0x00><0x00> <BYTESL><BYTESH> [ <DATA...> ] <CRCL><CRCH> <ETX>
```

Response

```
<STX> <0x06> <CRCL><CRCH> <ETX>
```

Details

- ADDRESS is the EEPROM address to start writing to. Writes are performed in ascending address order.
- BYTES specifies the number of bytes to write to EEPROM.

EEPROM does not require erasing before rewriting, thus, there is no erase command.

Microcontrollers that do not have EEPROM immediately send the response packet without performing any action.

Write Config Fuses

Request

```
<STX> <0x07> <ADDRESSL><ADDRESSH><0x00><0x00><BYTES> [ <DATA...> ] <CRCL><CRCH> <ETX>
```

Response

```
<STX> <0x07> <CRCL><CRCH> <ETX>
```

Details

- ADDRESS is the Configuration address to start writing to. Writes are performed in ascending address order.
- BYTES specifies the number of bytes to write.

Config bits do not require erasing before rewriting, so there is no erase command.

Run Application Firmware

Request

```
<STX> <0x08> <CRCL><CRCH> <ETX>
```

Response

None.

Details

This command makes the bootloader jump to the application firmware Reset vector. No response is returned because the bootloader is no longer active once application firmware is started.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families 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 Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is 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 products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, PIC³² logo, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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.

© 2010, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==**

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



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:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820