AN1157

A Serial Bootloader for PIC24F Devices

Author: Brant Ivey
Microchip Technology Inc.

INTRODUCTION

One of the advantages of Microchip's PIC® microcontrollers with self-programmable enhanced Flash memory is the ability to implement a bootloader. This allows designers to implement applications that can be updated many times over, potentially extending the application's useful lifetime.

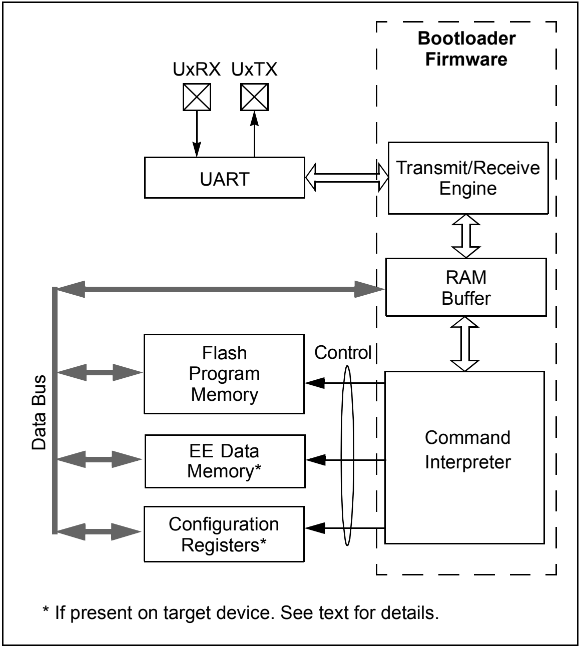
This application note describes a serial bootloader for 16-bit PIC24F devices using the UART module as a communication channel. The bootloader application uses the communication protocols originally outlined in Microchip Application Note *AN851, "A Flash Bootloader for PIC16 and PIC18 Devices"*. Some modifications to the original protocol have been made to maintain compatibility with the PIC24 architecture. It has also been redesigned to accommodate the current generation of PIC24FJ Flash microcontrollers, as well as the next generation of PIC24F devices.

FIRMWARE

Basic Operation

Figure 1 summarizes the essential firmware design of the bootloader. Data is received through the UART module and passed through the transmit/receive engine. The engine filters and parses the data, storing the information into a data buffer in RAM. The command interpreter evaluates the command information within the buffer to determine what should be done (e.g., Is the data written into memory? Is data read from memory? Does the firmware version need to be read?). Once the operation is performed, reply data is passed back to the transmit/receive engine to be transmitted back to the source, closing the software flow control loop.

FIGURE 1: BOOTLOADER FUNCTIONAL BLOCK DIAGRAM



COMMUNICATIONS

The microcontroller's UART module is used to receive and transmit data; it is configured to be compatible with RS-232 communications. The device can be set up in an application to bootload from a computer through its standard serial interface. The following default communication settings are used:

- 8 Data Bits
- No Parity
- 1 Stop Bit
- Automatic Baud Rate Detection

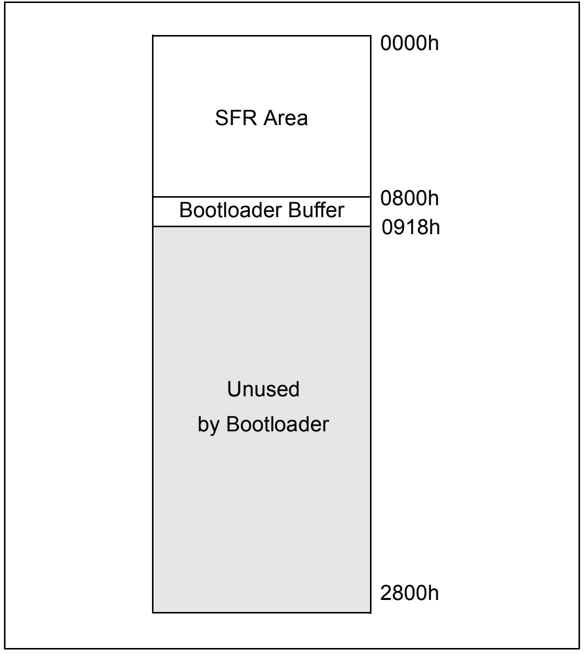
The baud rate is detected using the UART module's hardware auto-baud functionality. For more information on this feature, refer to the UART chapter of the *"PIC24F Family Reference Manual"* (**Section 21. "UART"** (DS39708)).

THE RECEIVE/TRANSMIT BUFFER

All data is moved through a buffer (referred to as the receive/transmit buffer). The buffer is a maximum of 261 bytes deep. This is the maximum packet length supported by the protocol. Figure 2 shows an example of the mapping of the buffer within PIC24FJXXXGAXXX devices.

A useful feature of the receive/transmit buffer is that it maintains the data from the last received packet. This allows commands to be easily repeated by sending an empty packet (a packet with no command code or data). If an empty packet is received, the data in the buffer will be reused as if it were just received.

FIGURE 2: DATA MEMORY USAGE ON PIC24FJXXXGAXXX DEVICES



COMMAND INTERPRETER

The command interpreter decodes and executes various protocol commands. A complete list of the commands is provided in **Appendix A: "PIC24F Serial Bootloader Command Summary"**. The commands allow for read, write and erase operations on all types of nonvolatile memory: program Flash, data EEPROM and Configuration bits. Additionally, there are commands for special operations, such as repeating the last command, replicating the data and resetting the device.

Note:	The command set is designed to support future devices and not all devices implement all types of memory. Compile-time options are provided to disable unsupported commands to save code space. Be sure to check which types of memory are implemented on the device being used and set the configuration accordingly.

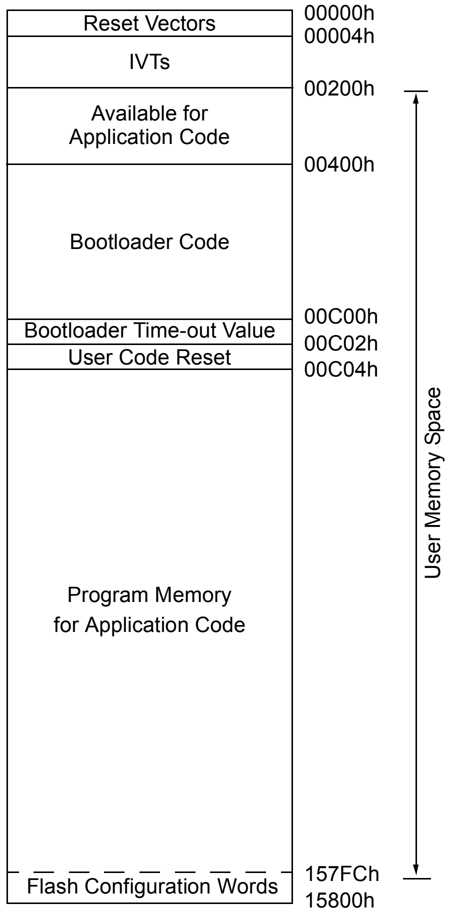
Memory Organization

PROGRAM MEMORY USAGE

The bootloader requires between 2 Kbytes and 3.5 Kbytes of program memory, depending on the configured feature set and compiler optimization. In default configuration, the bootloader is designed so that it can be placed anywhere in memory; by default it is located starting at address 000400h (shown in Figure 3).

The bootloader should not be placed in the same 512-byte page of memory as the interrupt vectors or Configuration Words. This is to allow the bootloader to modify these locations without erasing itself. If possible, the bootloader should be placed in a hardware protected "boot block" area on devices which support this feature. The boot block can be protected from device self writes and erases so that the bootloader will not be at risk of corrupting itself. Software block protection is provided in the bootloader to provide this functionality for devices that do not have built-in boot block protection.

FIGURE 3: PROGRAM MEMORY MAP FOR THE BOOTLOADER IN PIC24FJ128GA0XX DEVICES



Note: Memory areas not shown to scale.

RESET VECTORS

To ensure that the bootloader is always accessible, it is necessary to run the bootloader first after a device Reset. Therefore, the device's hardware Reset vector (addresses 000000h and 000002h) is used to store the bootloader's Reset vector. If the bootloader's Reset vector is ever overwritten, the device would no longer be able to enter Boot mode.

For the bootloader to call the user application, it must store the user application's Reset vector in memory. The constant, `USER_PROG_RESET`, defined in `config.h`, tells the bootloader where the Reset vector is remapped. When a user application is being programmed, the bootloader will automatically relocate the user Reset vector from 000000h to the location defined in this constant. The bootloader will read this location on device start-up and branch to the address stored here.

When using boot block protection, it is necessary to locate the user application Reset vector outside of the hardware boot block, so that it can be erased and written by the bootloader. Optionally, the application linker script can be modified to remove the Reset vector. If this is done, the user application will need to manually place its own Reset vector in the remapped location.

INTERRUPT VECTORS

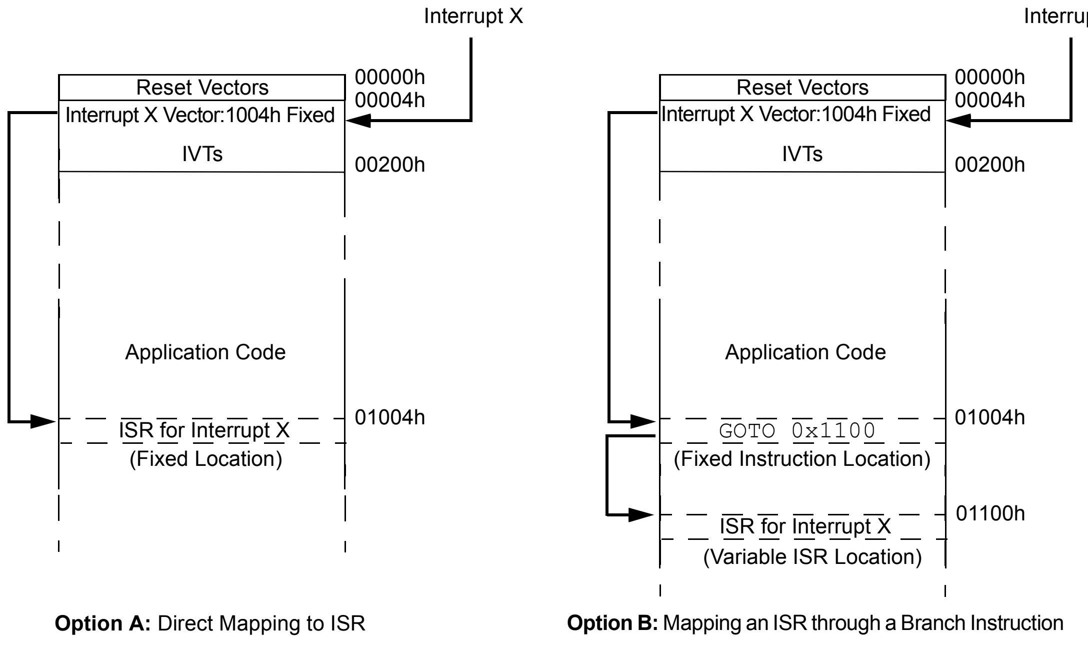
Because the Interrupt Vector Tables (IVTs) are located in the same page of memory as the device's Reset vector, it is good practice to remap the vector tables to a new location, so that the Reset vector does not need to be erased to update interrupt locations. Remapping the IVTs is required when hardware boot block protection is enabled. This is done by modifying bootloader and application linker scripts. The bootloader linker script entry for the interrupt tables indicates what addresses will be placed in the IVT.

The entries for the interrupts being used should be modified to indicate the locations in the user application of the interrupt functions, so that the service routine can be called directly (Option A in Figure 4). If the locations of the interrupts are not known at the time the bootloader is compiled, change the addresses in the linker script to locations where branch instructions can be placed as part of user code. The user application should then place branch instructions in these locations to vector to the interrupt routines (Option B in Figure 4). Note that this method increases the interrupt latency by two instruction cycles to allow for the extra branch instruction to execute.

The application linker script must be modified to remove the interrupt vectors, to prevent the bootloader from erasing the remapped interrupt vectors. Alternatively, you can enable the Bootloader mode which prevents the vector table from being erased, so that the user application linker script does not need to be modified.

Example linker scripts where these modifications have been done have been provided in the folder, `gld`. Modified linker examples are shown in **Appendix C: "Example Linker Scripts for Use with the PIC24F Bootloader"**.

FIGURE 4: TECHNIQUES FOR REMAPPING INTERRUPT VECTORS



Note: Memory areas not shown to scale.

Communication Protocol

The bootloader employs a basic communication protocol that is robust, simple to use and easy to implement. This protocol was originally specified in AN851. It has been slightly modified to improve compatibility with the 16-bit PIC24F architecture by increasing the maximum packet size.

PACKET FORMAT

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

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

where each <...> represents a byte and [...] represents the data field. The start of a packet is indicated by two 'Start of TeXt' control characters (<STX>), and is terminated by a single 'End of TeXt' control character (<ETX>). The last byte before the <ETX> is always a checksum, which is the two's complement of the Least Significant Byte of the sum of all data bytes. The data field is limited to 261 data bytes. This length is used in order to allow a full row of data to be received at a time. If more bytes are received, then the packet is ignored until the next <STX> pair is received.

Note: Although the protocol supports 261 bytes of data, the specific device that contains the bootloader firmware may not have a sufficiently large data memory to support the largest packet size. Refer to the data sheet for the particular device for more information.

COMMUNICATION CONTROL CHARACTERS

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

TABLE 1: CONTROL CHARACTERS

Control	Value	Description
<STX>	55h	Start of TeXt
<ETX>	04h	End of TeXt
<DLE>	05h	Data Link Escape

The <DLE> is used to identify a value that could be interpreted in the data field as a control character. Within the data field, the bootloader will always accept the byte following a <DLE> as data, and will always send a <DLE> before any of the three control characters. For example, if a byte of value, 55h, is transmitted as part of the data field, rather than as the <STX> control character, the <DLE> character is inserted before the <STX>. This is called "byte stuffing".

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

COMMANDS

The data field for each packet contains one command and its associated data. The commands are detailed in **Appendix A: "PIC24F Serial Bootloader Command Summary"**.

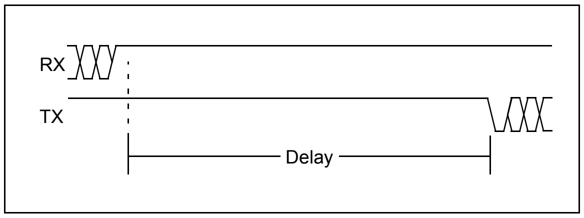
COMMAND RESPONSE LATENCY

Flow control is built into the protocol. Thus, for every received command (except `RESET`), there is a response. If there is no response, then one (or more) of the following has happened:

- the data was corrupted (bad checksum)
- the packet was never received
- the data field was too long
- a `RESET` was executed

So, how long do you wait before deciding a problem has occurred? The response latency (Figure 5) is dependent on the amount of data sent, the command being executed and the clock frequency. For read commands, the latency is highly dependent on the clock frequency and the size of the packet. For a small packet at high frequency, the response is almost immediate, typically about a few microseconds. For large packets, the latency could be hundreds of microseconds. In general, read commands require very little time compared to write commands. Write commands are mostly dependent on internally timed write cycles.

FIGURE 5: RECEIVE TO TRANSMIT LATENCY



BOOTING A DEVICE

Entering and Leaving Boot Mode

With the bootloader firmware loaded, there are two distinct modes of operation: Boot mode and User mode. A 1-byte value should be placed at the location specified by `DELAY_TIME_ADDR`, which is defined in `config.h`. A value of FFh indicates that the bootloader will remain in Boot mode and not switch to User mode without a command to do so. Thus, a new part with no valid user code will automatically enter Boot mode until user code is successfully programmed. Any other value indicates a number of seconds the bootloader will wait prior to switching to User mode. A value of 0 will immediately run User mode. The delay time will not be written into Flash by the bootloader until the `VERIFY_OK` command is sent. This command indicates to the bootloader that the user application has been successfully programmed, and is done in order to prevent possible bootloader lockout.

To leave Boot mode, a device Reset (hardware or software) must be initiated or a `RESET` command must be sent.

Note:	The bootloader will not wait the correct amount of time before entering User mode if the User mode instruction speed (FCY) is not defined correctly.

Program Memory Operations

The bootloader supports reads, writes and erases to Flash memory. Depending on the command being used, there is a minimum amount of data which can be referenced. Reading Flash is done at the instruction level. Each instruction has three bytes of program data and one byte of zeroes, called the "phantom byte", for a total of four bytes. Write operations are performed in 64 instruction blocks called rows (256 bytes). Erase instructions are performed in blocks of 8 rows called pages (2048 bytes). For detailed information on the PIC24F Flash program memory, refer to the PIC24F program memory section in the *"PIC24F Family Reference Manual"* or device data sheets.

When programming memory on PIC24F devices, first the page needs to be erased. Most locations in Flash operate such that erase operations set bits and program operations can only clear bits. Therefore, it is not recommended to write to a location multiple times without erasing. Additionally, this means that it is not possible to recover from a situation where the bootloader is partially or completely corrupted by writes or erases to the memory space it occupies. For this reason, it is recommended to use either software or hardware boot block protection.

Data EEPROM Operations

Some PIC24F devices have built-in data Flash memory EEPROM. The bootloader allows data Flash to be read and erased at a word level, 2 bytes at a time. Erases are done on a word level prior to performing a write.

Not all devices have data EEPROM. The functions that support this can be removed through the bootloader configuration at compile time.

Device Configuration

PIC24F devices implement device configuration in one of two ways: Flash Configuration Words and Configuration bits. Note that having access to the device configuration, though useful, is potentially dangerous. If the configuration is changed to a mode not supported in the application, such as switching from a high-speed crystal to the LPRC oscillator, it will cause the system to function incorrectly, and in some cases, break the bootloader's ability to fix the configuration. For this reason, care should be taken when performing configuration changes.

FLASH CONFIGURATION WORDS

Devices with Flash Configuration Words store their configuration information in the last few instructions in user program memory. These values are copied into volatile registers in the configuration memory space when the part resets. Since they are a part of user memory, a page erase performed on the final page of Flash memory also erases the device configuration. If the Flash Configuration Words are changed during run time, the device continues to run using its original configuration until a Reset occurs.

The PIC24F serial bootloader treats the configuration information on these devices as normal user memory, and programs it along with of the last row of Flash memory. A bootloader configuration option is provided to protect the last page of memory from erases and writes to prevent configuration corruption.

CONFIGURATION BITS

Devices using Configuration bits implement device configuration in 8-bit registers in the configuration memory space, starting at address F80000h. Configuration bits are read and written in single bits and do not need to be erased prior to writing. However, some of the Configuration bits are unidirectional bits, and cannot be self-programmed back to a '`1`' if they are set to '`0`'. An example is any of the code protection bits, which cannot be disabled using device self-programming.

For devices with Configuration bits, configuration information is read and written separately from Flash program memory with separate commands. These commands can optionally be removed from the bootloader for devices that use Flash Configuration Words instead of Configuration bits to save program space.

WRITING APPLICATION CODE

The bootloader operates as a separate entity, which means that an application can be developed with very little concern about what the bootloader is doing. This is as it should be; the bootloader should be dormant code until an event initiates a boot operation. Under ideal circumstances, bootloader code should never be running during an application's intended normal operation. When developing an application with a resident bootloader, some basic principles must be kept in mind.

Remapped Vectors

Reset and interrupt vectors for the user application are remapped outside of the normal vector space. The Reset vector for a user application should be left at address 00h. The bootloader will automatically take the value being programmed to this address and relocate it to the user Reset vector location specified in the configuration file.

The interrupt Reset vectors, which are relocated using the bootloader linker script, can be used in two ways: by pointing directly to the address of an interrupt routine with a known address when the bootloader is being compiled or by pointing to a location with a `GOTO` instruction that will call the proper interrupt routine (see Figure 4). This can be done either by modifying the user application linker script or by creating a constant at the correct locations in memory that contain the correct opcode for the `GOTO` instruction.

Bootloader Re-Entry

If it is necessary to re-enter Boot mode from the application, this can be done in two ways. Either the part can be reset using a `RESET` instruction or external Reset, or the application code can branch to the start location of the bootloader. If it is desirable to maintain the current SFR values, using the branch option over a Reset may be preferable. Note that any SFRs that are used by the bootloader will still be modified by the bootloader's operation.

When entering Boot mode by branching to the start of the bootloader, the Reset indicator bits in the RCON register on the device should be cleared in user code.

The bootloader reads these bits to determine if Boot mode entry is the result of a device Reset or if it was intentionally called by the user. If the bootloader is entered through the user code, it will ignore the value of the bootloader entry delay and stay in Boot mode.

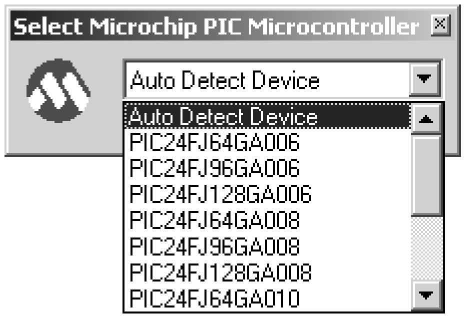
EXAMPLE SOFTWARE

The Microchip PIC24F Quick Programmer (P24QP) is a simple application designed to run on a Windows® PC as a device programmer. It is provided along with the PIC24F serial bootloader to perform basic device programming operations over a serial RS-232 interface. This application can be used as an example on which to base custom programmers. An overview of its logic design is provided in **Appendix D: "P24QP Application Flow Diagrams"**.

Selecting a Device

When P24QP first launches, the device selection dialog box, seen in Figure 6, will appear. This box allows the user to either manually select what device will be used or allow the program to automatically detect a device when connecting.

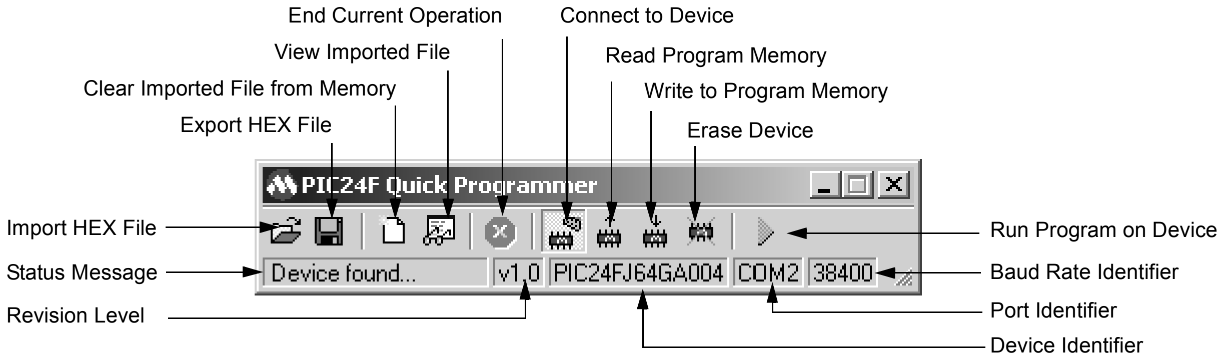
FIGURE 6: DEVICE SELECTION



The Main Toolbar

The main toolbar (Figure 7) provides basic commands and information about the device.

FIGURE 7: QUICK PROGRAMMER TOOLBAR



CONNECTING

Before anything can happen, communications to the attached device must be opened. This is done with the Connect to Device button. If automatic detection was selected, then the software will read the device ID and try to match it with device information provided in `P24QP.ini`. If a device is manually selected, then the settings for that particular device are forced. In either event, the device identity is shown in the Device Identifier area.

READING/WRITING/ERASING

The Read Device, Write Device and Erase Device buttons are used for reading, writing and erasing the attached device. The Read Device button tells the program to read the entire device. The Write Device button writes the data imported from a HEX file that is contained in the quick programmer's data files. The Erase Device button erases all pages of program memory used in the HEX file. On devices with Flash Configuration Words, a dialog box is presented before an erase to confirm if the last page of memory should be erased. If 'No' is selected, the last page of memory (and the Configuration Words) will be omitted from the erase.

Note: The PIC24F quick programmer automatically verifies the programmed memory areas after performing a write. The bootloader will not program the user Reset Vector Pointer or Boot mode entry delay during the normal write cycle. Due to this, these sections should be ignored during normal verification. In order for verification to occur without errors, ensure that `P24QP.ini` contains the correct addresses for these sections.

IMPORT/EXPORT HEX

Basic file import and export operations are available. The Microchip PIC24F Quick Programmer uses formatted text files to store data, which can be re-used over multiple sessions. Importing converts the HEX file into a formatted text file, replacing the current data. Exporting reads the current data files and saves them to a specified HEX file. The program uses the formatted text file for storage and display. When importing a file, always be certain that the HEX file is padded and aligned to a 16-byte boundary. MPLAB® IDE automatically pads to 16 bytes when an integer multiple of 16 bytes of data is selected on a 16-byte boundary when using the Export feature. Note that HEX files generated by compiling user code may or may not be aligned correctly, so they will not necessarily work with the P24QP. The P24QP works with the Intel HEX 32 Format (INHX32).

VIEW/CLEAR MEMORY

The View Data and Clear Data buttons allow the user to view or clear the data that was imported or read from the device. The program does not include any type of text viewer and uses the viewer specified in the `PIC24QP.ini` file. By default, the viewer used in Microsoft® Windows® is Notepad.

RUN MODE

Once the desired data is written into the device, selecting the Normal Execution Mode button will put the device in User mode. It does this by executing the bootloader Reset command.

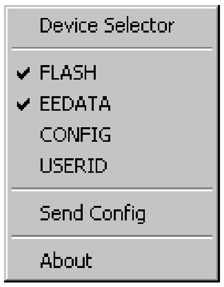
PORT AND BAUD RATE SELECTION

The default serial port and its baud rate are specified in the `PIC24QP.ini` file. The user may change these settings while the application is running by right-clicking on either the port indicator or the baud rate indicator. A menu of valid options that the user may select from (COM ports or baud rates) will appear.

Menu Options

Right-clicking on the status or the toolbar displays a pop-up menu that gives access to some settings and advanced operations. Figure 8 shows the menu options available.

FIGURE 8: MENU OPTIONS



DEVICE SELECTOR

This menu option gives the user the ability to reselect a device, or select a new device (see **"Selecting a Device"** and Figure 7).

MEMORY ACCESS

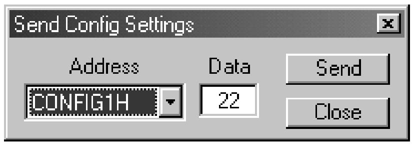
The memory types are either checked or unchecked to determine use. As an example, Figure 9 shows access to Flash program memory and data memory, while access to configuration memory is ignored. Memory types that are not available on a device will not be included on the list.

SEND CONFIGURATION

The check access for configuration in Figure 9 is for read operations only, due to the danger imposed by writing all Configuration bits sequentially. The “Send Config Settings” dialog box (Figure 9) is used to write Configuration register settings.

Selecting a Configuration register label from the Address list box will read from the current device data at that address. The value in the Data field can be edited and then written back to the device by clicking on the Send button.

FIGURE 9: SETTING CONFIGURATION BITS


Send Config Settings

Address: CONFIG1H

Data: 22

PIC24QP Files

Besides the main executable file for the quick programmer application (`P24QP.exe`), two additional files provide configurability and extended functionality.

A configuration file (`P24QP.ini`) is used to store the configuration settings for each device and configure the operation of the application. Details are shown in Example 1.

The dynamic library file, `PICBOOT.dll`, is used with the PC side application to provide an interface for a PC side programming application and the serial communications. This file automatically handles the communication protocols for the serial bootloader and provides a few useful functions for interfacing to the device. A complete list of the application function call implemented in the file is provided in Table 2. The source for this is also provided in order to allow users to modify it.

EXAMPLE 1: TYPICAL `P24QP.ini` FILE

```
;default settings for P24QP.exe

[PIC24FBOOT]
;data files for storing application data & editor to use
eedata="EEDATA.TXT"
config="CONFIG.TXT"
progmem="PROGMEM.TXT"
errorlog="ERROR.TXT"
editor="notepad.exe"

;default serial communications settings
portindex=2
bitrateindex=6
commtimeout=1000
maxpacketsize=261
maxretry=3

;debug flag
debuglevel=0

;show device selector on load and autodetect
selectdevwin=0
devicemode=0

;default locations of user reset pointer and BL entry delay
userresetvector="100"
bootdelay="102"

;supported devices
[DEVICELIST]
0="PICUNKNOWN"
...
xxxx="PIC24Fxxxx"

;example settings for device
;note - example only - refer to device specific settings
;all addresses are in hexidecimal format
[PICUNKNOWN]

;write, read, and erase block sizes (bytes)
writeblock=256
readblock=4
eraseblock=2048

;device feature set: 0 = Unknown, 1 = PIC24F, 2 = PIC24FJ
devicetype=0

;maximum packet size in bytes and bytes per address
maxpacketsize=261
bytesperaddr=2

;minimum and maximum valid memory addresses
pmrangelow="000000"
pmrangehigh="0157FF"
eerangelow="7FFE00"
eerangehigh="7FFFFF"
cfgrangelow="F80000"
cfgrangehigh="F80011"

;configuration bit names and locations
F80000="CFGBYTE1"
...
F80010="CFGBYTE9"

;user reset pointer and BL entry delay
;uncomment following lines to use device specific locations
;userresetvector="100"
;bootdelay="102"

;Device specific settings
[PIC24Fxxxx]
....
```

TABLE 2: FUNCTIONS IMPLEMENTED IN `PICBOOT.dll`

Function Name	Description
`SendPacket()`	Sends a packet of data with a maximum of 261 bytes (default) to the PIC® MCU. Formats the packet by adding STX, ETX and DLE bytes.
`GetPacket()`	Waits for a full, correct packet to be sent from PIC MCU. Returns error if packet is not correct.
`OpenPIC()`	Opens COM port and provides handle.
`ClosePIC()`	Closes COM port.
`SendGetPacket()`	Sends a packet to the PIC MCU, then waits for a complete response.
`ReadPIC()`	Reads the indicated memory area and amount from PIC MCU.
`WritePIC()`	Writes the indicated memory area and amount to PIC MCU.
`VerifyPIC()`	Reads indicated memory from PIC MCU and compares it to provided data.
`ErasePIC()`	Erases the indicated memory area and amount from PIC MCU.

OPERATION EXAMPLE

Here is a brief overview of how to use the PIC24F serial bootloader with the provided P24QP programmer.

1. Modify the bootloader linker script as desired (see **Appendix C: "Example Linker Scripts for Use with the PIC24F Bootloader"**). Set bootloader configuration options (see **Appendix B: "Bootloader Compile-Time Options"**).
2. Program the bootloader into the device.
3. Modify the linker script for the application (see **Appendix C: "Example Linker Scripts for Use with the PIC24F Bootloader"**). Make certain that the user application does not occupy the same memory space as the bootloader.
4. Set up the device's Configuration bits for the application.
5. Compile the application.
6. Export the HEX file from user application using *File > Export* in MPLAB IDE.
7. Modify the `P24QP.ini` configuration file as needed.
8. Execute the PIC24F quick programmer (`P24QP.exe`) and program the device:
   a) Select device or use auto-detect.
   b) Set up the COM port and baud rate.
   c) Connect to the device.
   d) Import the HEX file to program.
   e) Erase the device. If the device has Flash Configuration Words and the application contains configuration information or data in the final page, click "yes" at the prompt.
   f) Write the application data into the device.
   g) On devices with Configuration bits, enter "Send Config" menu and load Configuration bits as desired.

TROUBLESHOOTING

Issue: The PIC24F quick programmer displays "No firmware version available" or "Device Found" with device "PICUNKNOWN" when I attempt to connect to the device.

Solutions: This usually occurs when the P24QP is not able to communicate successfully with the PIC MCU UART. First, try changing the baud rate to see if an incorrect or incompatible baud rate was used. If the PIC MCU is unable to communicate at any baud rate, check the bootloader firmware to ensure that the correct clock frequency and clock source are specified. Also, reset the device and attempt to connect, ensuring that the bootloader has not timed out and jumped into user code.

Issue: P24QP is incorrectly formatting the data from my HEX file and is not programming the device properly.

Solutions: P24QP is designed to use HEX files exported from MPLAB IDE using the *File > Export* command. P24QP may not work with files generated compiling with C30, as the address formats and page alignment differ. Note that only the Intel HEX 32 format is supported (INHX32), not the Intel Split HEX Format (INHX8S).

Issues: Some device configuration settings are incompatible with the bootloader. Do not use the following settings in Bootload mode:

- **General Code Write-Protect:** This prevents the bootloader from being able to change Flash program memory. Do not write-protect sections of the program memory that the bootloader needs to change.
- **WDT Windowed Mode:** The bootloader resets the WDT frequently consistently and does not detect what the WDT time-out settings are. Therefore, it cannot ensure that it will not reset the device if it is running in Windowed mode. The user application can run with the Watchdog Timer using Windowed mode but the WDT must be disabled before entering the bootloader.
- **WDT with a Time-out Shorter than 32 ms:** This is to allow a worst-case page erase time to execute. If set to be shorter, it cannot be ensured that all write/erase instructions can successfully execute without timing out the WDT. The user application can run with a shorter Watchdog Timer time-out, but the WDT must be disabled before entering the bootloader. This can be done by manually enabling and disabling the WDT using the software Watchdog Timer enable bit.

Issue: Compiling the bootloader results in an error such as "Link Error: Region Program is Full".

Solutions: This occurs when the feature set used with the bootloader is too large for the memory size allocated in the linker script. To fix this issue, reduce code size by increasing the compiler optimization level or removing bootloader features. Another option is to increase the size of the bootloader in the linker script. Note that if this is done, the user application linker scripts should be modified to start user code at a higher location in memory as well.

APPENDIX A: PIC24F SERIAL BOOTLOADER COMMAND SUMMARY

TABLE A-1: BOOTLOADER COMMANDS

Name	Number	Description	Command Device [data field]	Response [data field]
RESET	ANY	Reset the device	[<COM><0x00>]	None
RD_VER	00h	Read bootloader version Information	[<0x00><0x02>]	[<0x00><0x02><VERL><VERH>]
RD_FLASH	01h	Read <LEN> instructions from program memory	[<0x01><LEN><ADDRL><ADDRH><ADDRU>]	[<0x01><LEN><ADDRL><ADDRH><ADDRU>... LEN blocks of Data...]
WT_FLASH	02h	Write <LEN> rows to program memory	[<0x02><LEN><ADDRL><ADDRH><ADDRU> ...LEN blocks of Data...]	[<0x02>]
ER_FLASH	03h	Erase <LEN> pages of program memory	[<0x03><LEN><ADDRL><ADDRH><ADDRU>]	[<0x03>]
RD_EEDATA	04h	Read <LEN> words from EE data memory	[<0x04><LEN><ADDRL><ADDRH><0x00>]	[<0x04><LEN><ADDRL><ADDRH><0x00>... LEN blocks of Data...]
WT_EEDATA	05h	Write <LEN> words to data EEPROM	[<0x05><LEN><ADDRL><ADDRH><0x00>...LEN blocks of Data...]	[<0x05>]
RD_CONFIG	06h	Read <LEN> bytes from configuration memory	[<0x06><LEN><ADDRL><0x00><0x30>]	[<0x06><LEN><ADDRL><0x00><0x30>... LEN bytes of Data...]
WT_CONFIG	07h	Write <LEN> bytes to configuration memory	[<0x07><LEN><ADDRL><0x00><0x30> ...LEN bytes of Data...]	[<0x07>]
VERIFY_OK	08h	Indicates to bootloader that user application has been successfully written; bootloader will write time-out value to Flash	[<0x08>]	[<0x08>]
REPEAT	COM	Repeat last command	[Empty data field]	Refer to the appropriate command response for the last command sent
REPLICATE	COM	Write old buffer data to another area	[<COM><LEN><ADDRL><ADDRH><ADDRU>] (<COM> is any write command)	[<COM>]

APPENDIX B: BOOTLOADER COMPILE-TIME OPTIONS

The PIC24F serial bootloader has a number of compile-time options that can be used to set its run-time configuration, as well as optimize its performance with different applications. The options fall into five categories:

Version Identifiers (Table B-1) are used to track the bootloader revision level.

Device Dependent Options (Table B-2) are configuration options that are enabled or disabled based on the hardware capabilities of the device in use. Refer to the specific device data sheet for implemented device features.

Bootloader Feature Options (Table B-3) are user-selectable configuration options, enabled depending on what additional features are desired in the bootloader. Disabling unwanted features reduces code size.

Bootloader Operational Options (Table B-4) are configuration options that are dependent on the application being used with the bootloader. These should be reconfigured any time the bootloader is implemented with a new user application.

Device Flash Memory Options (Table B-5) are configuration options that are device dependent. These must be reviewed and changed any time the bootloader is implemented on a new device.

TABLE B-1: VERSION IDENTIFIERS

Configuration Setting	Value	Description
`MAJOR_VERSION`	0x01	Major revision of bootloader firmware.
`MINOR_VERSION`	0x00	Minor revision of bootloader firmware.

TABLE B-2: DEVICE DEPENDENT CONFIGURATION OPTIONS

Configuration Setting	Description
`DEV_HAS_WORD_WRITE`	Use if the device has word write capability. This applies to all PIC24FJ devices.
`DEV_HAS_PPS`	Use if the device has Peripheral Pin Select (PPS) feature to map UART I/O.
`DEV_HAS_CONFIG_BITS`	Use if device has Configuration bits instead of Flash Configuration Words (see **"Device Configuration"** section). Enables configuration memory commands and removes Flash Configuration Word protection.
`DEV_HAS_EEPROM`	Use if device has internal data Flash to enable EEPROM read and write commands.

TABLE B-3: BOOTLOADER FEATURE OPTIONS

Configuration Setting	Description
`USE_BOOT_PROTECT`	Enables bootloader block protection. This is used to provide software protection of the bootloader in devices not using hardware boot block protection.
`USE_RUNAWAY_PROTECT`	Enables runaway code protection. This uses key sequences which are checked before writes and erases to ensure that a runaway code scenario will not corrupt Flash memory.
`USE_CONFIGWORD_PROTECT`	Enables Configuration Word protection. This is used on devices with Flash Configuration Words to prevent writes and erases to the last page of memory.
`USE_VECTOR_PROTECT`	Enables Reset and interrupt vector protection. This prevents the first page of memory from being written to or erased. This setting is intended for use when it is necessary to remap the interrupt vectors on devices without boot block protection.
`USE_HI_SPEED_BRG`	Enables High-Speed UART mode. Sets U2MODE.BRGH = `1`.
`USE_WORKAROUNDS`	This is provided to support workarounds for UART silicon errata.

TABLE B-4: BOOTLOADER CONFIGURATION OPTIONS

Configuration Setting	Value	Description
`FCY`	16000000	Instruction clock frequency, FCY = FOSC/2.
`MAX_PACKET_SIZE`	261	Maximum size, in bytes, of a packet of data.
`USER_PROG_RESET`	0x100	Address of user Reset vector: the value in this location will be the address where user code begins.
`DELAY_TIME_ADDR`	0x102	Address of bootloader entry delay: the value in this location will be the entry delay setting of the bootloader.
`BOOT_ADDR_LOW`	0x400	Starting address of bootloader. Used when software boot protection is enabled. Should be page-aligned even if bootloader is not to allow for erase protection.
`BOOT_ADDR_HI`	0xBFF	Ending address of bootloader. Used when software boot protection is enabled. Should be page-aligned to the end of last page bootloader occupies.
`PPS_UTX_PIN`	RPOR12.RP25R	Maps UART TX pin for PPS devices.
`PPS_URX_PIN`	19	Maps UART RX function for PPS devices.

TABLE B-5: FLASH MEMORY CONFIGURATION OPTIONS

Configuration Setting	Value	Description
`PM_INSTR_SIZE`	4	Bytes per instruction.
`PM_ROW_SIZE`	256	User Flash row size in bytes.
`PM_PAGE_SIZE`	2048	User Flash page size in bytes.
`CM_ROW_SIZE`	1	Configuration row size in bytes.
`CONFIG_WORD_1`	0xABFE	Location of Flash Configuration Word 1. Used for configuration protection.
`CONFIG_WORD_2`	0xABFC	Location of Flash Configuration Word 2. Used for configuration protection.
`CONFIG_WORD_3`	0xABFA	Location of Flash Configuration Word 3. Used for configuration protection. **Note:** Not present in all PIC24F devices.
`PM_PAGE_ERASE`	0x4042	Opcode to load into NVMCON register to perform a Flash page erase.
`PM_ROW_WRITE`	0x4001	Opcode to load into NVMCON register to perform a Flash row write.
`PM_WORD_WRITE`	0x4003	Opcode to load into NVMCON register to perform a Flash word write. Only used when bootloader is configured to use word write capability.
`EE_WORD_WRITE`	0x4004	Opcode to load into NVMCON register to perform a data Flash word write. Only used when bootloader is configured to use EEPROM capability.
`EE_ROW_WRITE`	0x4006	Opcode to load into NVMCON register to perform a data Flash row write. Only used when bootloader is configured to use EEPROM capability.
`CONFIG_WORD_WRITE`	0x4004	Opcode to load into NVMCON register to perform a Configuration bit write. Only used when bootloader is configured to use Configuration bits.

APPENDIX C: EXAMPLE LINKER SCRIPTS FOR USE WITH THE PIC24F BOOTLOADER

EXAMPLE C-1: EXAMPLE LINKER SCRIPT FOR BOOTLOADER

```
/*
** Linker script outline for PIC24F bootloader (for PIC24FJ64GA004 device)
*/

OUTPUT_ARCH("24FJ64GA004")
/*EXTERN(__resetPRI)*/  //no data init needed, only load no-init starupt
EXTERN(__resetALT)

/*
** Memory Regions
*/
MEMORY
{
  data  (a!xr) : ORIGIN = 0x800,      LENGTH = 0x2000
  reset        : ORIGIN = 0x0,        LENGTH = 0x4
  ivt          : ORIGIN = 0x4,        LENGTH = 0xFC
  aivt         : ORIGIN = 0x104,      LENGTH = 0xFC
  program (xr) : ORIGIN = 0x400,      LENGTH = 0x800 /*start & length of BL*/
  config2      : ORIGIN = 0xABFC,     LENGTH = 0x2
  config1      : ORIGIN = 0xABFE,     LENGTH = 0x2
}

__CONFIG2 = 0xABFC;
__CONFIG1 = 0xABFE;

__IVT_BASE  = 0x4;
__AIVT_BASE = 0x104;
__DATA_BASE = 0x800;
__CODE_BASE = 0x400;   /* Starting location of bootloader */
.
.
.
/*
** Interrupt Vector Table
*/
.ivt __IVT_BASE :
  {
    …
    /* Int Vector Remap Method 1 - point to application's ISR location */
    LONG( ABSOLUTE(0xF00)); /*Location of Interrupt ISR*/
    …
    /* Int Vector Remap Method 2 -point to a location in a jump table */
    LONG(ABSOLUTE(0x1004)); /*Location of jump table goto instruction*/
    …
  } >ivt
```

EXAMPLE C-2: EXAMPLE LINKER SCRIPT FOR THE USER APPLICATION

```
/*
** Linker script outline for PIC24F bootloader user application
** (for PIC24FJ64GA004 device)
*/

OUTPUT_ARCH("24FJ64GA004")
EXTERN(__resetPRI)
EXTERN(__resetALT)

/*
** Memory Regions
*/
MEMORY
{
  data  (a!xr) : ORIGIN = 0x800,         LENGTH = 0x2000
  reset        : ORIGIN = 0x0,           LENGTH = 0x4
  ivt          : ORIGIN = 0x4,           LENGTH = 0xFC
  aivt         : ORIGIN = 0x104,         LENGTH = 0xFC

  /*Starting location and length of user program  */
  program (xr) : ORIGIN = 0xC00,         LENGTH = 0x9FFC

  config2      : ORIGIN = 0xABFC,        LENGTH = 0x2
  config1      : ORIGIN = 0xABFE,        LENGTH = 0x2

  /*
  ** Section for storing user app reset vector and BL time out value.  This
  ** section should be defined to ensure it is not overwritten by compiler.
  */
  BLreset   : ORIGIN = 0x100, LENGTH = 0x4
}

__CONFIG2 = 0xABFC;
__CONFIG1 = 0xABFE;

__IVT_BASE  = 0x4;
__AIVT_BASE = 0x104;
__DATA_BASE = 0x800;
__CODE_BASE = 0xC00;
.
.
.
/*
** Outputs the BLreset section into the hex file at the area specified in ** memory map above.
This is where the bootloader, by default, stores its ** user reset and entry mode timer.
*/
.BLreset :
  {
(.BLreset);
  } >BLreset
.
.
.
```

APPENDIX D: P24QP APPLICATION FLOW DIAGRAMS

FIGURE D-1: P24QP APPLICATION, PAGE 1 (MAIN LOOP)

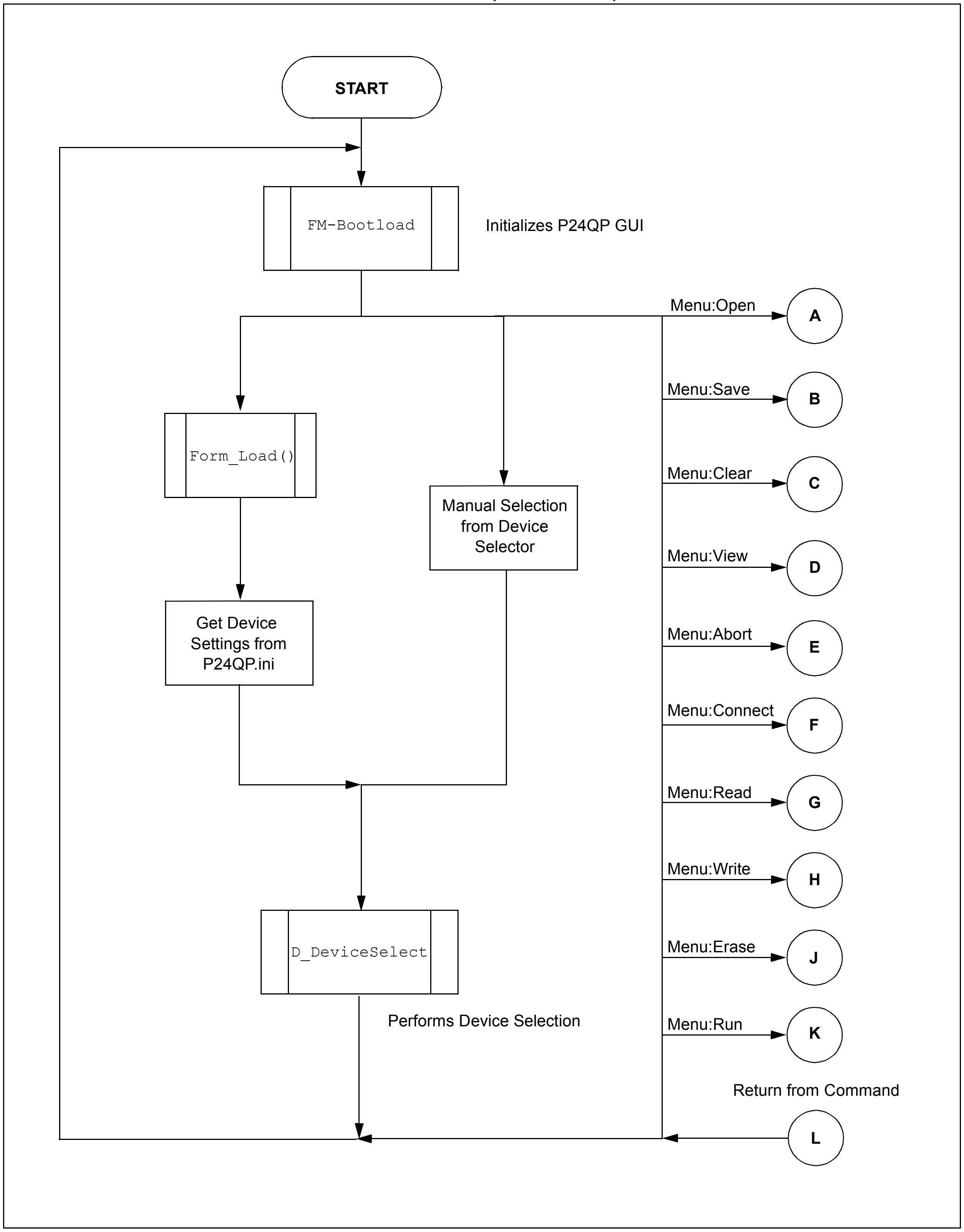


FIGURE D-2: P24QP FLOW, PAGE 2 (OPEN, SAVE AND CLEAR ROUTINES)

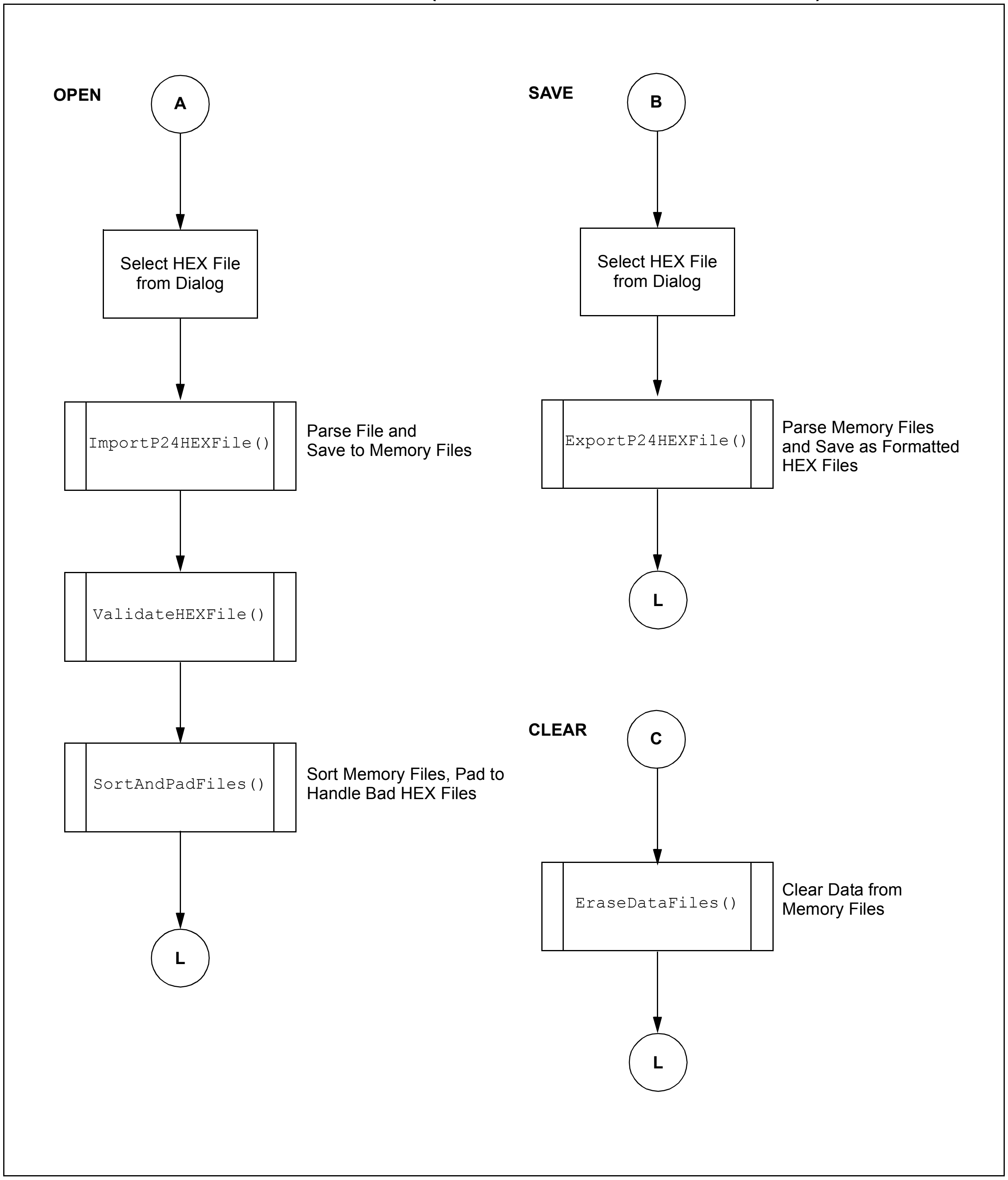

OPEN
A
Select HEX File from Dialog
ImportP24HEXFile()
Parse File and Save to Memory Files
ValidateHEXFile()
SortAndPadFiles()
Sort Memory Files, Pad to Handle Bad HEX Files
L
SAVE
B
Select HEX File from Dialog
ExportP24HEXFile()
Parse Memory Files and Save as Formatted HEX Files
L
CLEAR
C
EraseDataFiles()
Clear Data from Memory Files
L

FIGURE D-3: P24QP FLOW, PAGE 3 (VIEW, ABORT, READ AND WRITE)

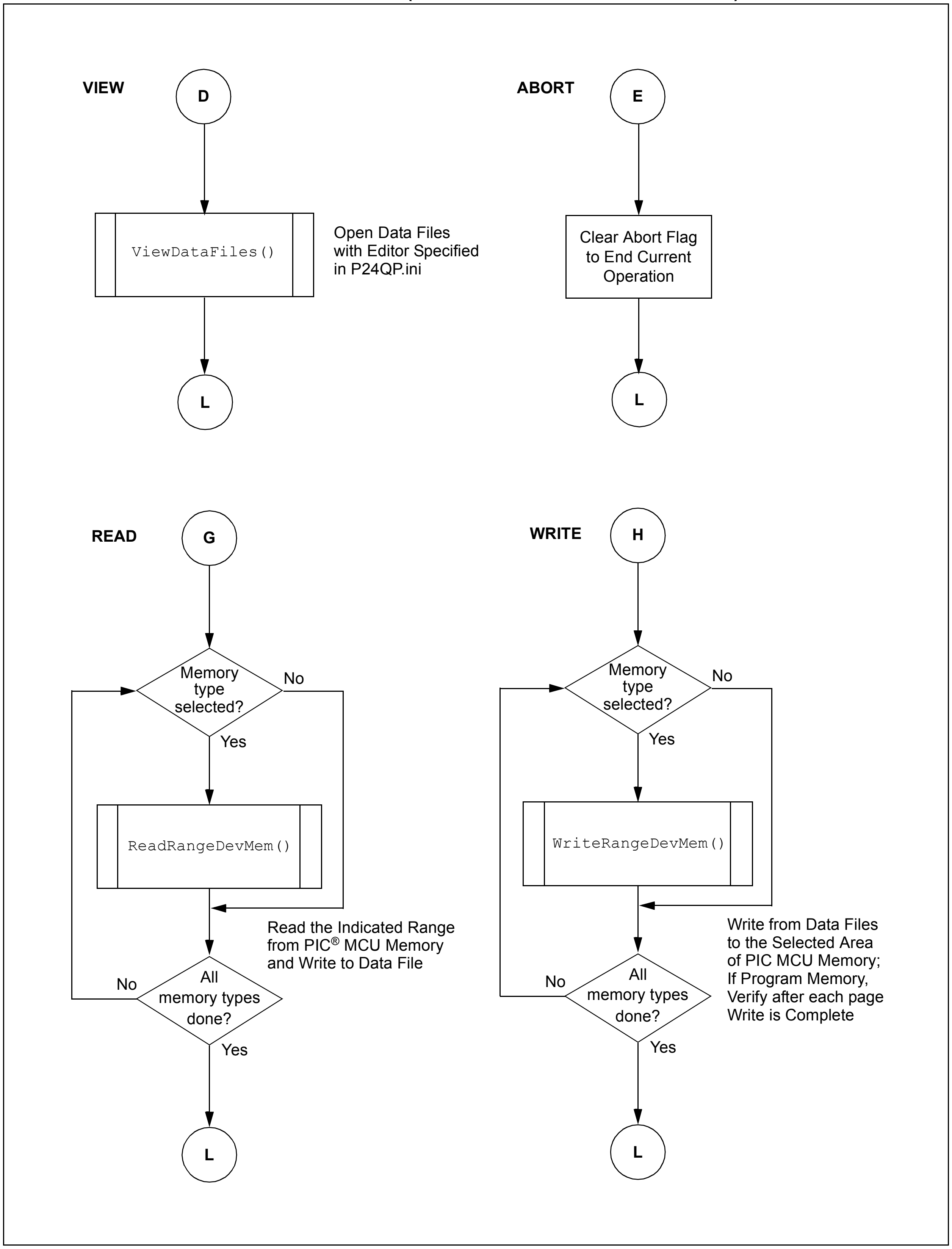

VIEW
D
ViewDataFiles()
Open Data Files with Editor Specified in P24QP.ini
L
ABORT
E
Clear Abort Flag to End Current Operation
L
READ
G
Memory type selected?
No
Yes
ReadRangeDevMem()
Read the Indicated Range from PIC® MCU Memory and Write to Data File
All memory types done?
No
Yes
L
WRITE
H
Memory type selected?
No
Yes
WriteRangeDevMem()
Write from Data Files to the Selected Area of PIC MCU Memory; If Program Memory, Verify after each page Write is Complete
All memory types done?
No
Yes
L

FIGURE D-4: P24QP FLOW, PAGE 4 (ERASE)

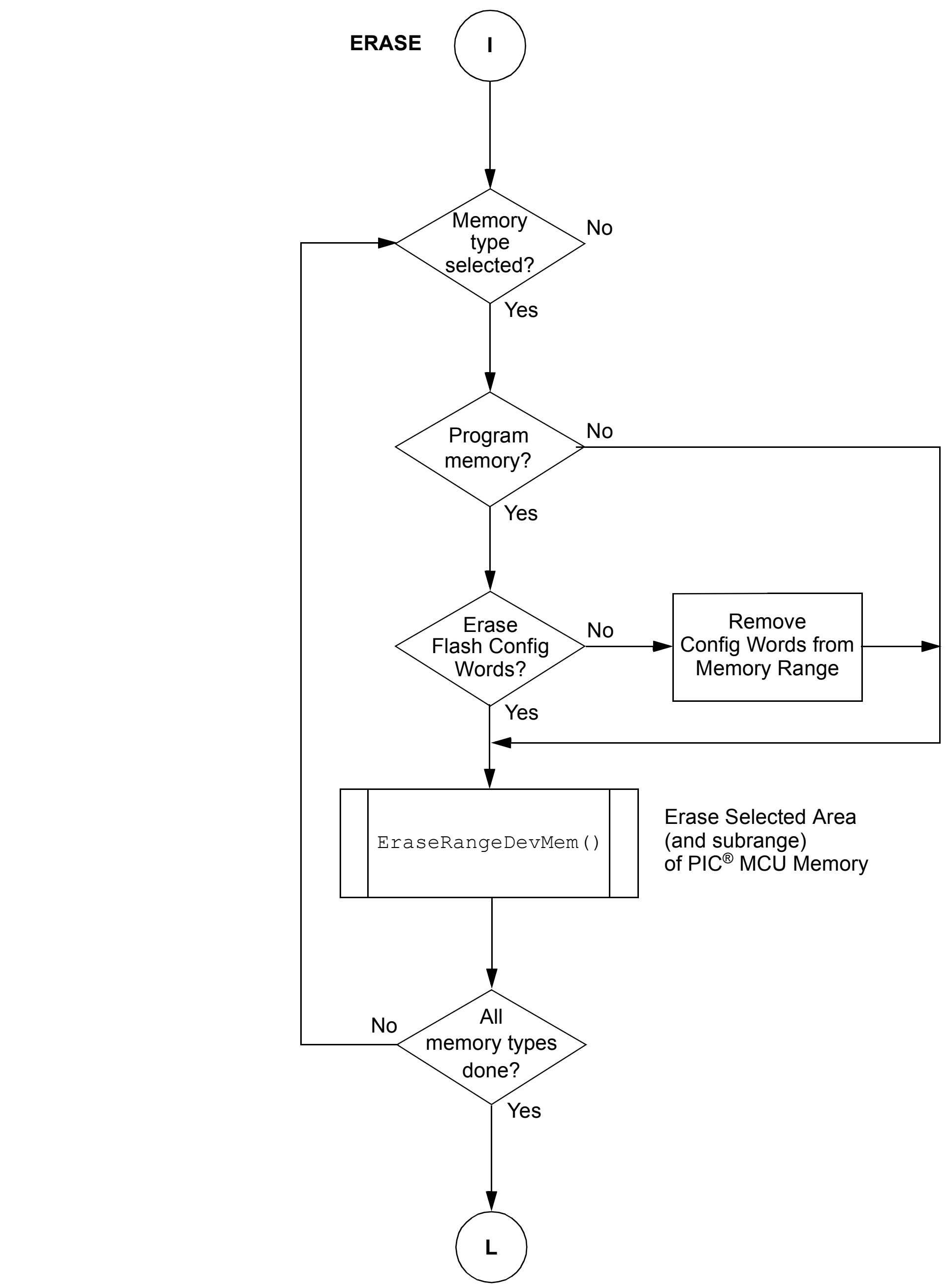

ERASE
I
Memory type selected?
No
Yes
Program memory?
No
Yes
Erase Flash Config Words?
No
Remove Config Words from Memory Range
Yes
EraseRangeDevMem()
Erase Selected Area (and subrange) of PIC® MCU Memory
All memory types done?
No
Yes
L

FIGURE D-5: P24QP FLOW, PAGE 5 (CONNECT)

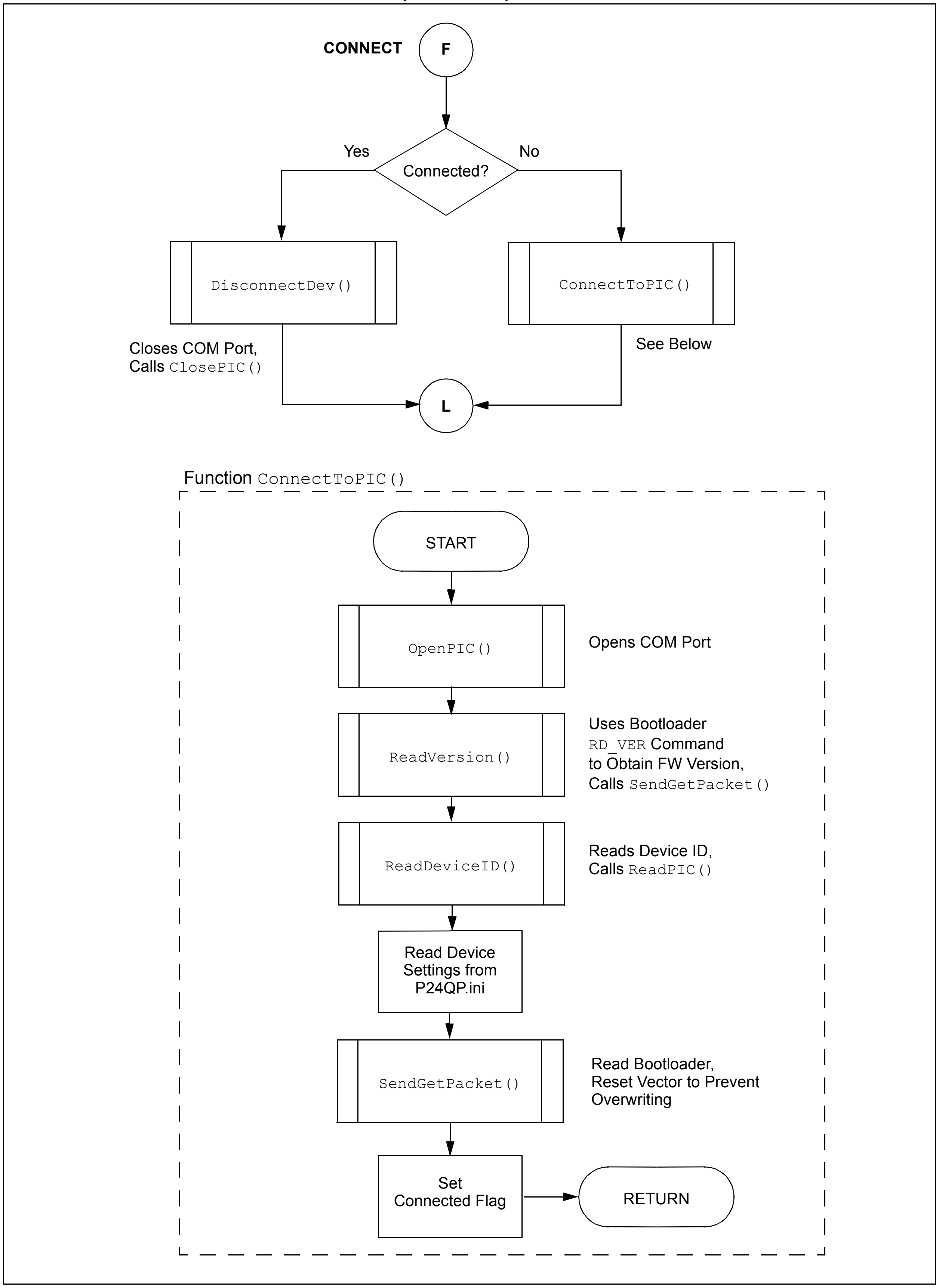
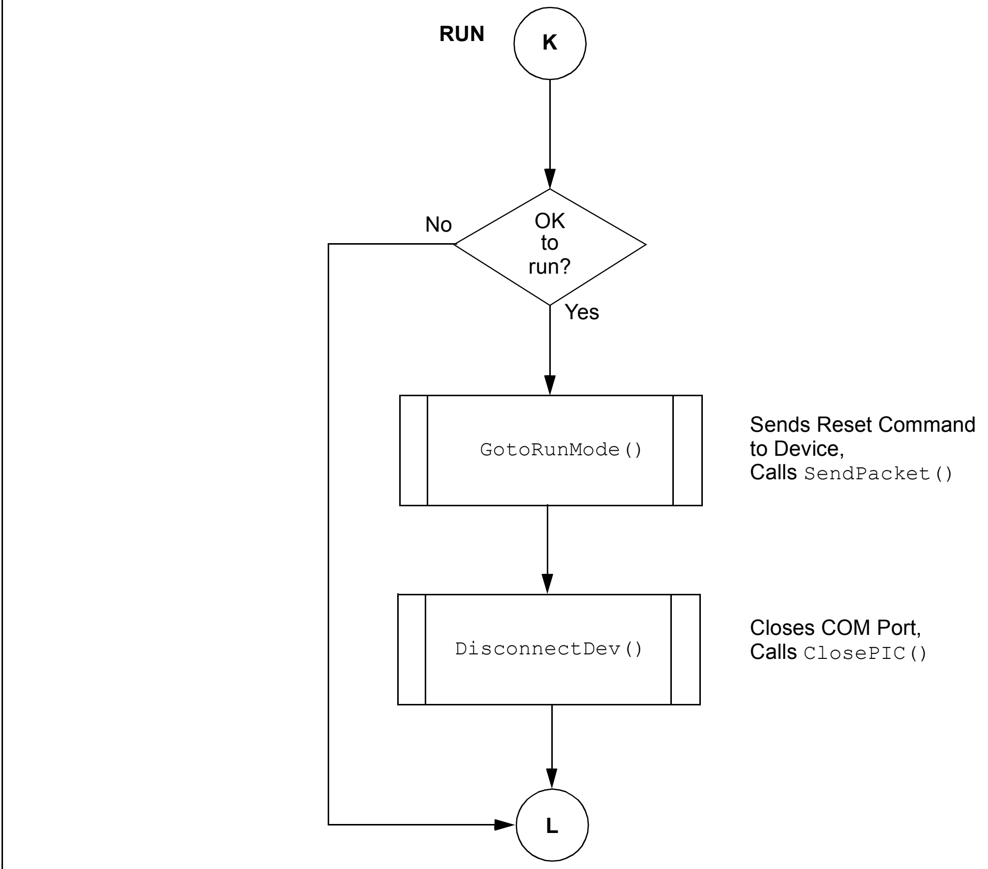


FIGURE D-6: P24QP FLOW, PAGE 6 (RUN)


RUN
K
OK to run?
No
Yes
GotoRunMode()
Sends Reset Command to Device, Calls SendPacket()
DisconnectDev()
Closes COM Port, Calls ClosePIC()
L

APPENDIX E: SOFTWARE DISCUSSED IN THIS APPLICATION NOTE

All of the software covered in this application note (the source code for the bootloaders, the PIC24F quick programmer and all associated project files) is available as a single WinZip archive file. The archive may be downloaded from the Microchip corporate web site at:

www.microchip.com

NOTES:

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:
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

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

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

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

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

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: 886-3-572-9526
Fax: 886-3-572-6459

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

01/02/08