PIC32MX

PIC32MX Flash Programming Specification

1.0 DEVICE OVERVIEW

This document defines the programming specification for the PIC32MX family of 32-bit microcontrollers. This programming specification is designed to guide developers of external programmer tools. Customers who are developing applications for PIC32MX devices should use development tools that already provide support for device programming.

2.0 PROGRAMMING OVERVIEW

All PIC32MX devices can be programmed via two primary methods – self-programming and external tool programming.

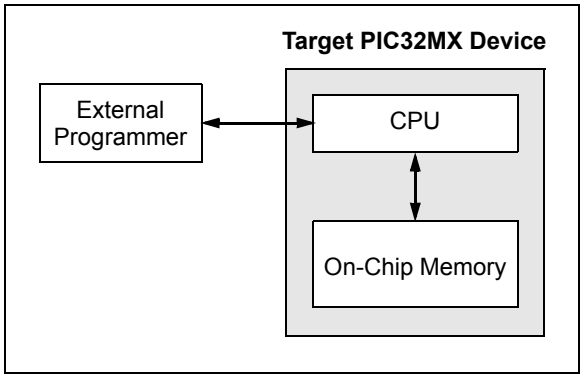
The self-programming method requires that the target device already contains executable code with the logic necessary to complete the programming sequence.

The external tool programming method does not require any code in the target device – it can program all target devices with or without any executable code.

This document describes the external tool programming method. Refer to the individual sections of the *"PIC32 Family Reference Manual"* and the specific device data sheet for more information about using the self-programming method.

An external tool programming setup consists of an external programmer tool and a target PIC32MX device. Figure 2-1 illustrates the block diagram view of the typical programming setup. The programmer tool is responsible for executing necessary programming steps and completing the programming operation.

FIGURE 2-1: PROGRAMMING SYSTEM SETUP



All PIC32MX devices provide two physical interfaces to the external programmer tool:

- 2-wire In-Circuit Serial Programming™ (ICSP™)
- 4-wire Joint Test Action Group (JTAG)

See **Section 4.0 "Connecting to the Device"** for more information.

Either of these methods may use a downloadable Programming Executive (PE). The PE executes from the target device RAM and hides device programming details from the programmer. It also removes overhead associated with data transfer and improves overall data throughput. Microchip has developed a PE that is available for use with any external programmer.

See **Section 16.0 "The Programming Executive"** for more information.

Section 3.0 "Programming Steps" describes high-level programming steps, followed by a brief explanation of each step. Detailed explanations are available in corresponding sections of this document.

More details on programming commands, EJTAG, and DC specs are available in the following sections:

- **Section 18.0 "Configuration Memory and Device ID"**
- **Section 19.0 "TAP Controllers"**
- **Section 20.0 "AC/DC Characteristics and Timing Requirements"**

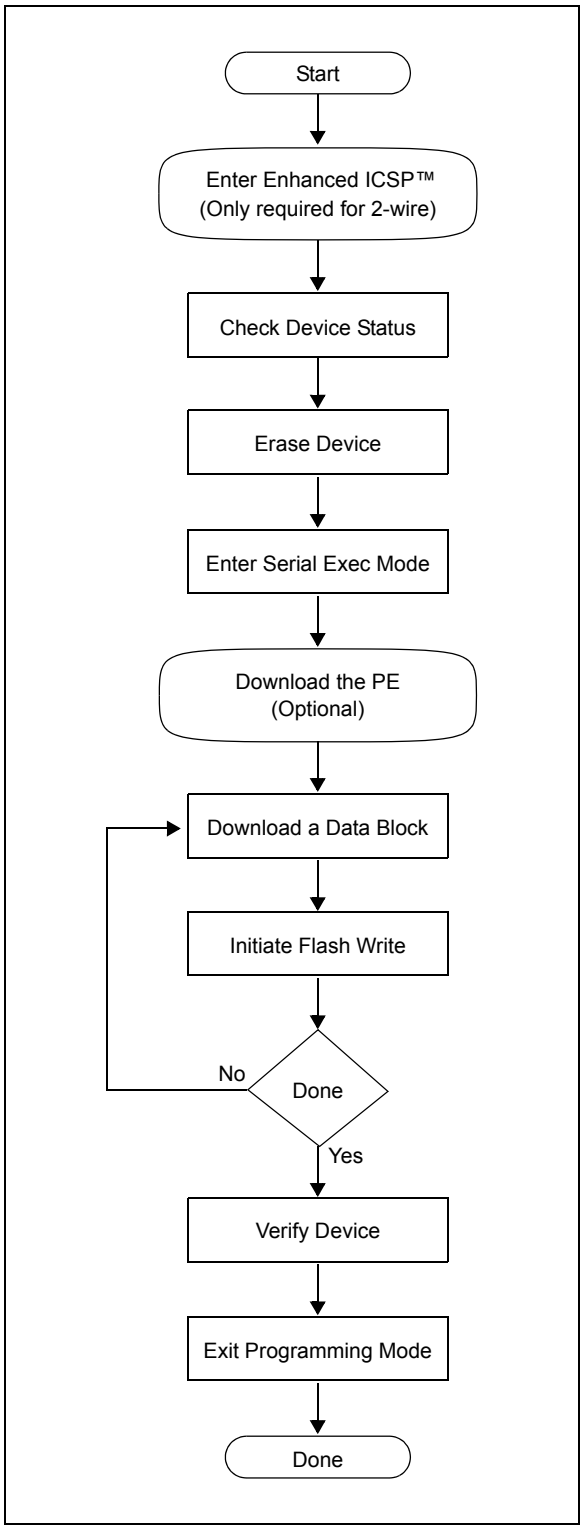
2.1 Assumptions

Both 2-wire and 4-wire interfaces use the EJTAG protocol to exchange data with the programmer. While this document provides a working description of this protocol as needed, advanced users are advised to refer to the *"EJTAG Specification"* (MD00047), which is available from MIPS Technologies, Inc.

3.0 PROGRAMMING STEPS

All tool programmers must perform a common set of steps, regardless of the actual method being used. Figure 3-1 shows the set of steps to program PIC32MX devices.

FIGURE 3-1: PROGRAMMING FLOW



The following sequence lists the steps, with a brief explanation of each step. More detailed information about the steps is available in the following sections.

1. Connect to the Target Device.

   To ensure successful programming, all required pins must be connected to appropriate signals. See **Section 4.0 "Connecting to the Device"** in this document for more information.

2. Place the Target Device in Programming Mode.

   For 2-wire programming methods, the target device must be placed in a special programming mode (Enhanced ICSP™) before executing any other steps.

   Note: For the 4-wire programming methods, Step 2 is not required.

   See **Section 7.0 "Entering Programming Mode"** for more information.

3. Check the Status of the Device.

   Step 3 checks the status of the device to ensure it is ready to receive information from the programmer.

   See **Section 8.0 "Check Device Status"** for more information.

4. Erase the Target Device.

   If the target memory block in the device is not blank, or if the device is code-protected, an erase step must be performed before programming any new data.

   See **Section 9.0 "Erasing the Device"** for more information.

5. Enter Programming Mode.

   Step 5 verifies that the device is not code-protected and boots the TAP controller to start sending and receiving data to and from the PIC32MX CPU.

   See **Section 10.0 "Entering Serial Execution Mode"** for more information.

6. Download the Programming Executive (PE).

   The PE is a small block of executable code that is downloaded into the RAM of the target device. It will receive and program the actual data.

   Note: If the programming method being used does not require the PE, Step 6 is not required.

   See **Section 11.0 "Downloading the Programming Executive (PE)"** for more information.

7. Download the Block of Data to Program.

   All methods, with or without the PE, must download the desired programming data into a block of memory in RAM.

   See **Section 12.0 "Downloading a Data Block"** for more information.

8. Initiate Flash Write.

   After downloading each block of data into RAM, the programming sequence must be started to program it into the target device's Flash memory.

   See **Section 13.0 "Initiating a Flash Row Write"** for more information.

9. Repeat Steps 7 and 8 until all data blocks are downloaded and programmed.
10. Verify the program memory.

    After all programming data and Configuration bits are programmed, the target device memory should be read back and verified for the matching content.

    See **Section 14.0 "Verify Device Memory"** for more information.

11. Exit the Programming mode.

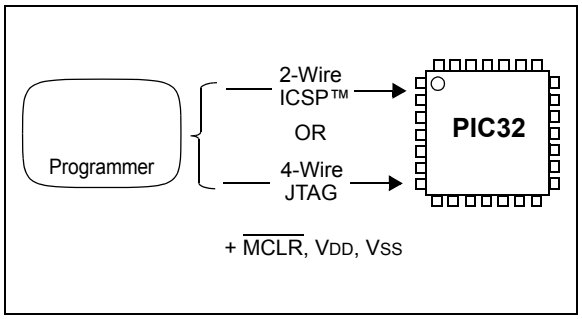
    The newly programmed data is not effective until either power is removed and reapplied to the target device or an exit programming sequence is performed.

    See **Section 15.0 "Exiting Programming Mode"** for more information.

4.0 CONNECTING TO THE DEVICE

The PIC32MX family provides two possible physical interfaces for connecting to and programming the memory contents (Figure 4-1). For all programming interfaces, the target device must be properly powered and all required signals must be connected.

FIGURE 4-1: PROGRAMMING INTERFACES



4.1 4-Wire Interface

One possible interface is the 4-wire JTAG (IEEE 1149.1) port. Table 4-1 lists the required pin connections. This interface uses the following four communication lines to transfer data to and from the PIC32MX device being programmed:

- TCK – Test Clock Input
- TMS – Test Mode Select Input
- TDI – Test Data Input
- TDO – Test Data Output

These signals are described in the following four sections. Refer to the specific device data sheet for the connection of the signals to the chip pins.

4.1.1 TEST CLOCK INPUT (TCK)

TCK is the clock that controls the updating of the TAP controller and the shifting of data through the Instruction or selected Data register(s). TCK is independent of the processor clock with respect to both frequency and phase.

4.1.2 TEST MODE SELECT INPUT (TMS)

TMS is the control signal for the TAP controller. This signal is sampled on the rising edge of TCK.

4.1.3 TEST DATA INPUT (TDI)

TDI is the test data input to the Instruction or selected Data register(s). This signal is sampled on the rising edge of TCK for some TAP controller states.

4.1.4 TEST DATA OUTPUT (TDO)

TDO is the test data output from the Instruction or Data register(s). This signal changes on the falling edge of TCK. TDO is only driven when data is shifted out, otherwise the TDO is tri-stated.

TABLE 4-1: 4-WIRE INTERFACE PINS

Device Pin Name	Pin Type	Pin Description
MCLR	I	Programming Enable
ENVREG	I	Enable for On-Chip Voltage Regulator
VDD and AVDD(1)	P	Power Supply
VSS and AVSS(1)	P	Ground
VCAP	P	CPU logic filter capacitor connection
TDI	I	Test Data In
TDO	O	Test Data Out
TCK	I	Test Clock
TMS	I	Test Mode State

Legend: I = Input O = Output P = Power

Note 1: All power supply and ground pins must be connected, including analog supplies (AVDD) and ground (AVSS).

4.2 2-Wire Interface

Another possible interface is the 2-wire ICSP port. Table 4-2 lists the required pin connections. This interface uses the following 2 communication lines to transfer data to and from the PIC32MX device being programmed:

- PGCx – Serial Program Clock
- PGDx – Serial Program Data

These signals are described in the following two sections. Refer to the specific device data sheet for the connection of the signals to the chip pins.

4.2.1 SERIAL PROGRAM CLOCK (PGCx)

PGCx is the clock that controls the updating of the TAP controller and the shifting of data through the Instruction or selected Data register(s). PGCx is independent of the processor clock, with respect to both frequency and phase.

4.2.2 SERIAL PROGRAM DATA (PGDx)

PGDx is the data input/output to the Instruction or selected Data Register(s), it is also the control signal for the TAP controller. This signal is sampled on the falling edge of PGC for some TAP controller states.

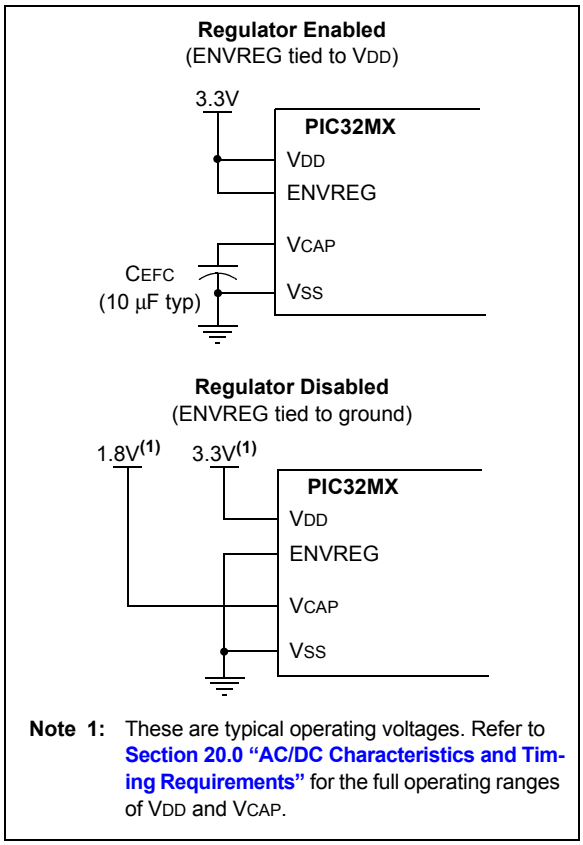
4.3 Power Requirements

All devices in the PIC32MX family are dual voltage supply designs: one supply for the core and peripherals and another for the I/O pins. Some devices contain an on-chip regulator to eliminate the need for two external voltage supplies.

All of the PIC32MX devices power their core digital logic at a nominal 1.8V. This may create an issue for designs that are required to operate at a higher typical voltage, such as 3.3V. To simplify system design, all devices in the PIC32MX family incorporate an on-chip regulator that allows the device to run its core logic from VDD.

The regulator provides power to the core from the other VDD pins. A low-ESR capacitor (e.g., a tantalum capacitor) must be connected to the VCAP pin (Figure 4-2). This helps to maintain the stability of the regulator. The specifications for core voltage and capacitance are listed in **Section 20.0 "AC/DC Characteristics and Timing Requirements"**.

FIGURE 4-2: CONNECTIONS FOR THE ON-CHIP REGULATOR



Note 1: These are typical operating voltages. Refer to **Section 20.0 "AC/DC Characteristics and Timing Requirements"** for the full operating ranges of VDD and VCAP.

TABLE 4-2: 2-WIRE INTERFACE PINS

Device Pin Name	Programmer Pin Name	Pin Type	Pin Description
$\overline{MCLR}$	$\overline{MCLR}$	P	Programming Enable
ENVREG	N/A	I	Enable for On-Chip Voltage Regulator
VDD and AVDD(1)	VDD	P	Power Supply
VSS and AVSS(1)	VSS	P	Ground
VCAP	N/A	P	CPU logic filter capacitor connection
PGC1	PGC	I	Primary Programming Pin Pair: Serial Clock
PGD1	PGD	I/O	Primary Programming Pin Pair: Serial Data
PGC2	PGC	I	Secondary Programming Pin Pair: Serial Clock
PGD2	PGD	I/O	Secondary Programming Pin Pair: Serial Data

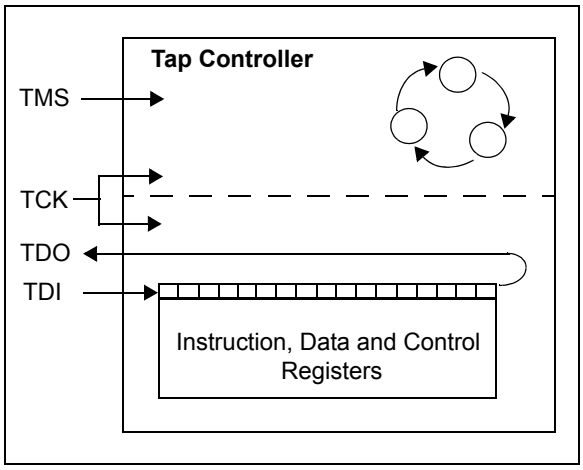
Legend: I = Input O = Output P = Power

Note 1: All power supply and ground pins must be connected, including analog supplies (AVDD) and ground (AVSS).

5.0 EJTAG vs. ICSP

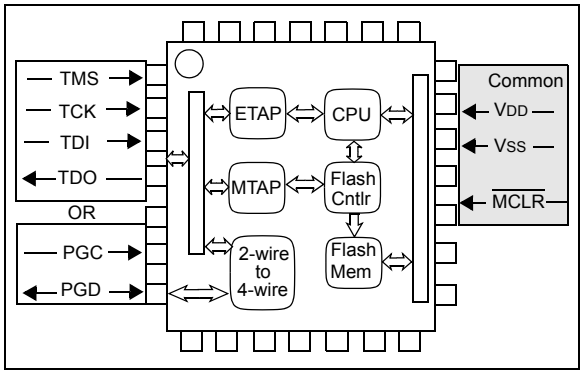
Programming is accomplished via the EJTAG module in the CPU core. EJTAG is connected to either the full set of JTAG pins, or a reduced 2-wire to 4-wire EJTAG interface. In both modes, programming of the PIC32MX Flash memory is accomplished through the ETAP controller. The TAP Controller uses the TMS pin to determine if Instruction or Data registers should be accessed in the shift path between TDI and TDO (see Figure 5-1).

FIGURE 5-1: TAP CONTROLLER



The basic concept of EJTAG that is used for programming is the use of a special memory area called DMSEG (0xFF200000 to 0xFF2FFFFF), which is only available when the processor is running in Debug mode. All instructions are serially shifted into an internal buffer, then loaded into the Instruction register and executed by the CPU. Instructions are fed through the ETAP state machine in 32-bit groups.

FIGURE 5-2: BASIC PIC32MX PROGRAMMING BLOCK



- **ETAP**
  - Serially feeds instructions and data into CPU.
- **MTAP**
  - In addition to the EJTAG TAP (ETAP) controller, the PIC32MX device uses a second proprietary TAP controller for additional operations. The Microchip TAP (MTAP) controller supports two instructions relevant to programming: `MTAP_COMMAND` and TAP switch Instructions. See Table 19-1 for a complete list of Instructions. The `MTAP_COMMAND` instruction provides a mechanism for a JTAG probe to send commands to the device via its Data register.
  - The programmer sends commands by shifting in the `MTAP_COMMAND` instruction via the `SendCommand` pseudo operation, and then sending `MTAP_COMMAND DR` commands via `XferData` pseudo operation (see Table 19-2 for specific commands).
  - The probe does not need to issue an `MTAP_COMMAND` instruction for every command shifted into the Data register.
- **2-Wire to 4-Wire**
  - Converts 2-wire ICSP interface to 4-wire JTAG.
- **CPU**
  - The CPU executes instructions at 8 MHz via the internal oscillator.
- **Flash Controller**
  - The Flash Controller controls erasing and programming of the Flash memory on the device.
- **Flash Memory**
  - The PIC32MX device Flash memory is divided into two logical Flash partitions consisting of the Boot Flash Memory (BFM) and Program Flash Memory (PFM). The Boot Flash Memory map extends from 0x1FC00000 to 0x1FC02FFF, and the Program Flash Memory map extends from 0x1D000000 to 0x1D07FFFF. Code storage begins with the BFM and supports up to 12 Kbytes. It continues with the PFM, which supports up to 512 Kbytes. Table shows the program memory size of each device variant. Each erase block, or page, contains 1K instructions (4 Kbytes) or 256 instructions (1 Kbytes) and each program block, or row, contains 128 instructions (512 bytes) or 32 instructions (128 bytes).
  - The last four implemented program memory locations in BFM are reserved for the device Configuration registers.

TABLE 5-1: CODE MEMORY SIZE

PIC32MX Device	Row Size (Instr. Words)	Page Size (Instr. Words)	Boot Flash Memory Address (Bytes)	Program Flash Memory Address (Bytes)
PIC32MX110F016B	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D003FFF (16 KB)
PIC32MX110F016C	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D003FFF (16 KB)
PIC32MX110F016D	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D003FFF (16 KB)
PIC32MX210F016B	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D003FFF (16 KB)
PIC32MX210F016C	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D003FFF (16 KB)
PIC32MX210F016D	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D003FFF (16 KB)
PIC32MX120F032B	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D007FFF (32 KB)
PIC32MX120F032C	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D007FFF (32 KB)
PIC32MX120F032D	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D007FFF (32 KB)
PIC32MX220F032B	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D007FFF (32 KB)
PIC32MX220F032C	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D007FFF (32 KB)
PIC32MX220F032D	32	256	0x1FC00000-0x1FC00BFF (3 KB)	0x1D000000-0x1D007FFF (32 KB)
PIC32MX320F032H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D007FFF (32 KB)
PIC32MX130F064B	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX130F064C	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX130F064D	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX230F064B	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX230F064C	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX230F064D	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX320F064H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX534F064H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX564F064H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX664F064H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX534F064L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX564F064L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX664F064L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D00FFFF (64 KB)
PIC32MX150F128B	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX150F128C	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX150F128D	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX250F128B	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX250F128C	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX250F128D	32	256	0x1FC00000-0x1FC00FFF (3 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX320F128H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX564F128H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX664F128H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX764F128H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX320F128L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX564F128L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX664F128L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D01FFFF (128 KB)
PIC32MX764F128L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D01FFFF (128 KB)

TABLE 5-1: CODE MEMORY SIZE (CONTINUED)

PIC32MX Device	Row Size (Instr. Words)	Page Size (Instr. Words)	Boot Flash Memory Address (Bytes)	Program Flash Memory Address (Bytes)
PIC32MX340F256H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D03FFFF (256 KB)
PIC32MX575F256H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D03FFFF (256 KB)
PIC32MX675F256H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D03FFFF (256 KB)
PIC32MX775F256H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D03FFFF (256 KB)
PIC32MX360F256L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D03FFFF (256 KB)
PIC32MX575F256L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D03FFFF (256 KB)
PIC32MX675F256L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D03FFFF (256 KB)
PIC32MX775F256L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D03FFFF (256 KB)
PIC32MX575F512H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX675F512H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX695F512H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX775F512H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX795F512H	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX360F512L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX575F512L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX675F512L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX695F512L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX775F512L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)
PIC32MX795F512L	128	1024	0x1FC00000-0x1FC02FFF (12 KB)	0x1D000000-0x1D07FFFF (512 KB)

5.1 4-Wire JTAG Details

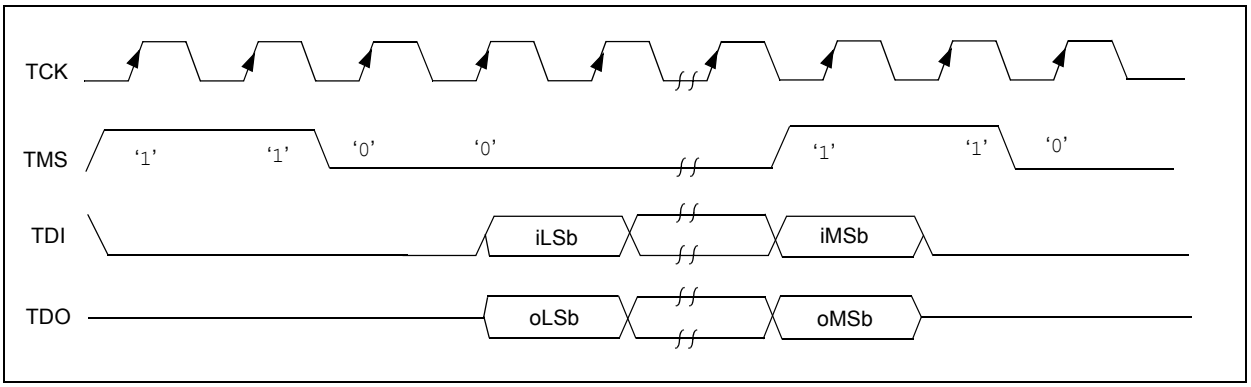
The 4-wire interface uses standard JTAG (IEEE 1149.1-2001) interface signals.

- TCK: Test Clock – drives data in/out
- TMS: Test Mode Select – selects operational mode
- TDI: Test Data In – data into the device
- TDO: Test Data Out – data out of the device

Since only one data line is available, the protocol is necessarily serial (like SPI). The clock input is at the TCK pin. Configuration is performed by manipulating a state machine bit by bit through the TMS pin. One bit of data is transferred in and out per TCK clock pulse at the TDI and TDO pins, respectively. Different instruction modes can be loaded to read the chip ID or manipulate chip functions.

Data presented to TDI must be valid for a chip-specific setup time before, and hold time, after the rising edge of TCK. TDO data is valid for a chip-specific time after the falling edge of TCK (refer to Figure 5-3).

FIGURE 5-3: 4-WIRE JTAG INTERFACE



5.2 2-Wire ICSP Details

In ICSP mode, the 2-wire ICSP signals are time multiplexed into the 2-wire to 4-wire block. The 2-wire to 4-wire block then converts the signals to look like a 4-wire JTAG port to the TAP controller.

There are two possible modes of operation:

- 4-Phase ICSP
- 2-Phase ICSP

5.2.1 4-PHASE ICSP

In 4-Phase ICSP mode, the TDI, TDO and TMS device pins are multiplexed onto PGD in 4 clocks (see Figure 5-4). The Least Significant bit (LSb) is shifted first; and TDI and TMS are sampled on the falling edge of PGC, while TDO is driven on the falling edge of PGC. 4-Phase mode is used for both read and write data transfers.

5.2.2 2-PHASE ICSP

In 2-Phase ICSP mode, the TMS and TDI device pins are multiplexed into PGD in 2 clocks (see Figure 5-5). The LSb is shifted first; and TDI and TMS are sampled on the falling edge of PGC. There is no TDO output provided in this mode. The 2-Phase ICSP mode was designed to accelerate 2-wire, write-only transactions.

Note:	The packet is not actually executed until the first clock of the next packet. To enter 2-Wire, 2-Phase ICSP mode, the TDOEN bit (DDPCON<0>) must be set to '0'.

FIGURE 5-4: 2-WIRE, 4-PHASE

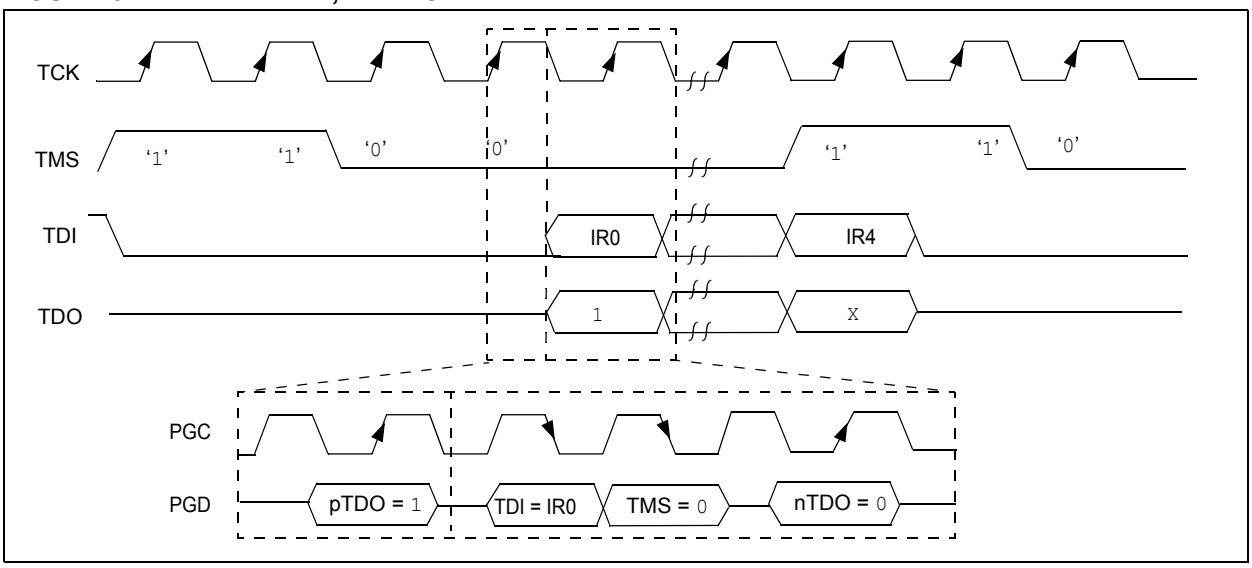
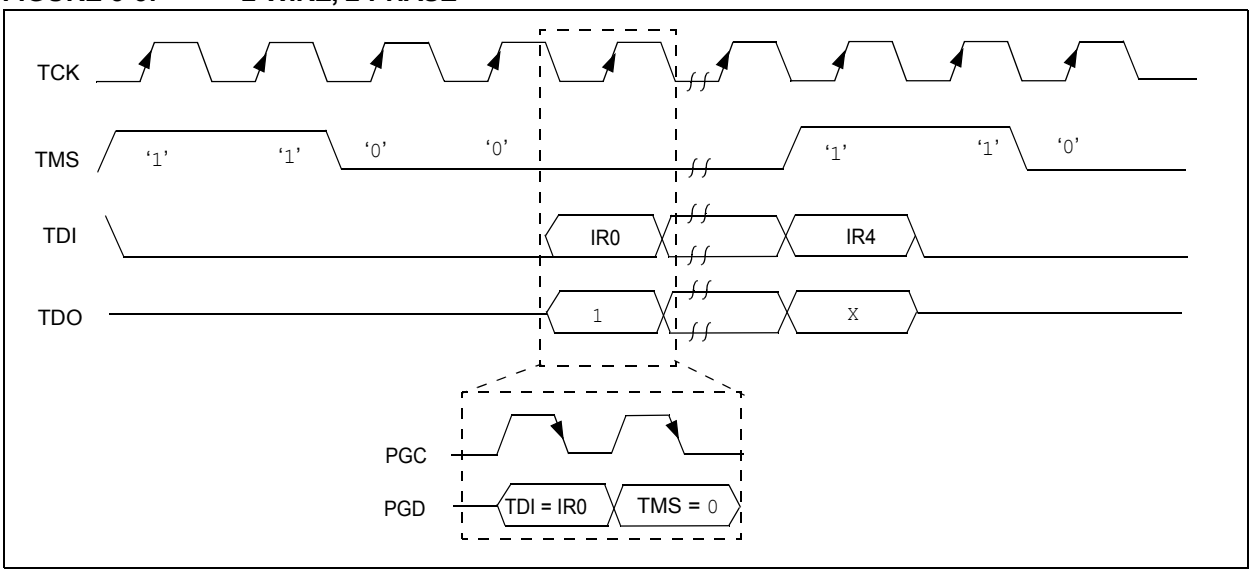


FIGURE 5-5: 2-WIRE, 2-PHASE



6.0 PSEUDO OPERATIONS

To simplify the description of programming details, all operations will be described using pseudo operations. There are several functions used in the pseudocode descriptions. These are used either to make the pseudocode more readable, to abstract implementation-specific behavior, or both. When passing parameters with pseudo operation, the following syntax will be used: 5'h0x03 (i.e., send 5-bit hex value 0x03). These functions are defined in this section, and include the following operations:

- **SetMode** (mode)
- **SendCommand** (command)
- oData = **XferData** (iData)
- oData = **XferFastData** (iData)
- oData = **XferInstruction** (instruction)

6.1 SetMode Pseudo Operation

Format:
SetMode (mode)

Purpose:
To set the EJTAG state machine to a specific state.

Description:
The value of mode is clocked into the device on signal TMS. TDI is set to a '0' and TDO is ignored.

Restrictions:
None.

Example:
SetMode (6'b011111)

FIGURE 6-1: SetMode 4-WIRE

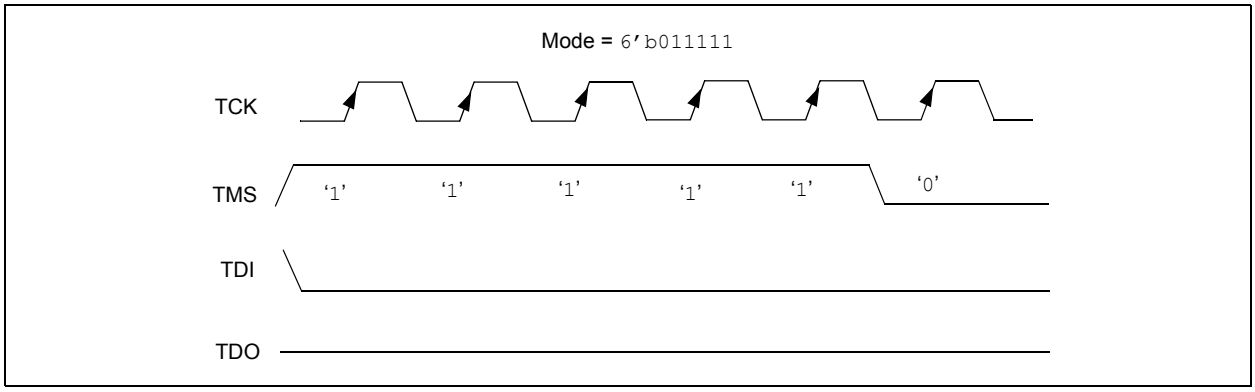
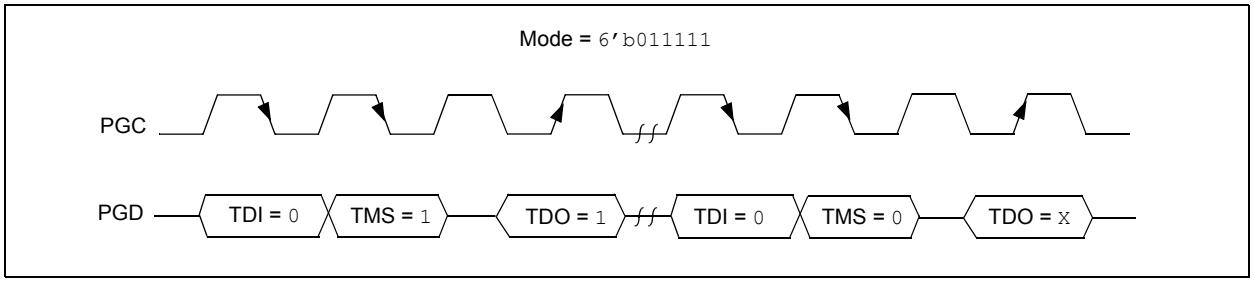


FIGURE 6-2: SetMode 2-WIRE



6.2 `SendCommand` Pseudo Operation

Format:
`SendCommand` (command)

Purpose:
To send a command to select a specific TAP register.

Description (in sequence):

1. The TMS Header is clocked into the device to select the Shift IR state
2. The command is clocked into the device on TDI while holding signal TMS low.
3. The last Most Significant bit (MSb) of the command is clocked in while setting TMS high.
4. The TMS Footer is clocked in on TMS to return the TAP controller to the Run/Test Idle state.

Restrictions:
None.

Example:
`SendCommand` (5'h07)

FIGURE 6-3: `SendCommand` 4-WIRE

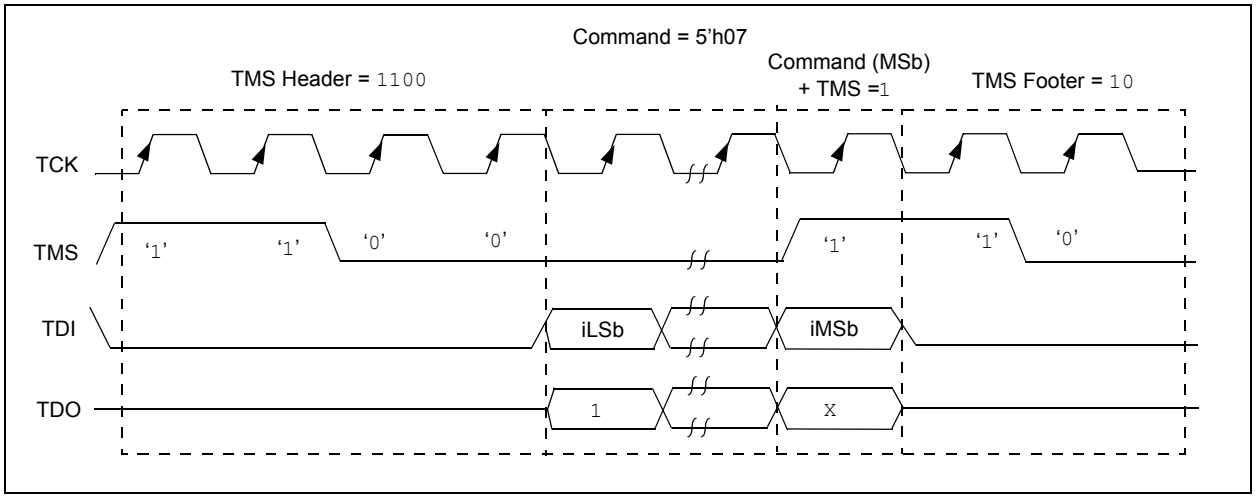
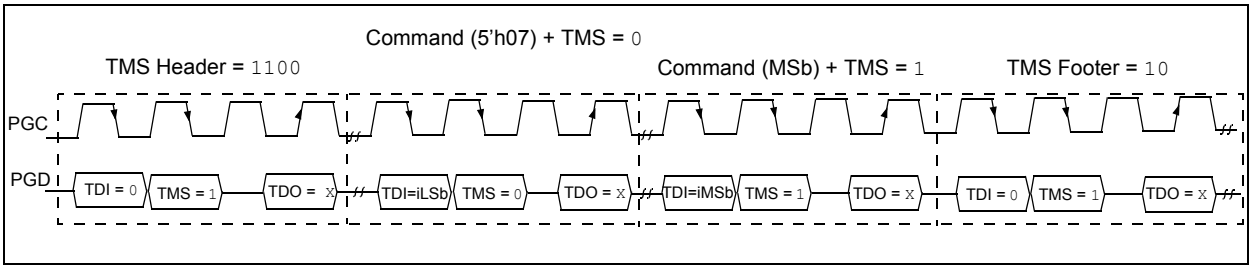


FIGURE 6-4: `SendCommand` 2-WIRE



6.3 `XferData` Pseudo Operation

Format:
oData = **XferData** (iData)

Purpose:
To clock data to and from the register selected by the command.

Description (in sequence):

1. The TMS Header is clocked into the device to select the Shift DR state.
2. The data is clocked in/out of the device on TDI/TDO while holding signal TMS low.
3. The last MSb of the data is clocked in/out while setting TMS high.
4. The TMS Footer is clocked in on TMS to return the TAP controller to the Run/Test Idle state.

Restrictions:
None.

Example:
oData = XferData (32'h12)

FIGURE 6-5: XferData 4-WIRE

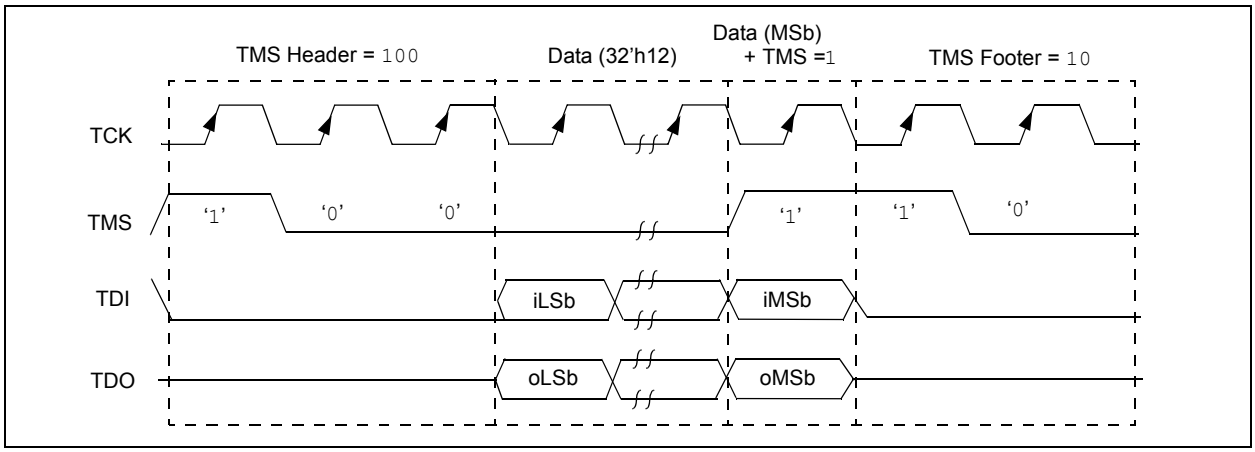
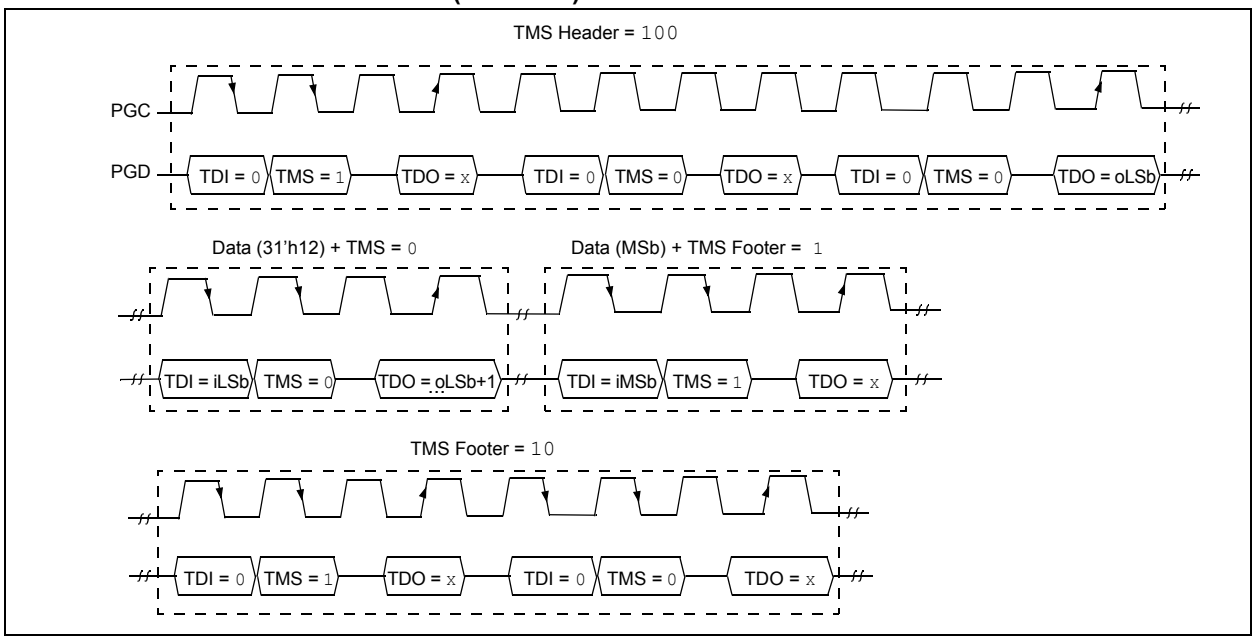


FIGURE 6-6: XferData 2-WIRE (4-PHASE)



6.4 `XferFastData` Pseudo Operation

Format:
oData = **XferFastData** (iData)

Purpose:
To quickly send 32 bits of data in/out of the device.

Description (in sequence):

1. The TMS Header is clocked into the device to select the Shift DR state.

Note:	For 2-wire (4-phase) – on the last clock, the oPrAcc bit is shifted out on TDO while clocking in the TMS Header. If the value of oPrAcc is not '1', the whole operation must be repeated.

2. The input value of the PrAcc bit, which is '0', is clocked in.

Note:	For 2-wire (4-phase) – the TDO during this operation will be the LSb of output data. The rest of the 31 bits of the input data are clocked in and the 31 bits of output data are clocked out. For the last bit of the input data, the TMS Footer = 1 is set.

3. TMS Footer = 10 is clocked in to return the TAP controller to the Run/Test Idle state.

Restrictions:
The SendCommand (ETAP_FASTDATA) must be sent first to select the Fastdata register, as shown in Example 6-1. See Table 19-4 for a detailed descriptions of commands.

Note:	The 2-Phase XferData is only used when talking to the PE. See **Section 16.0 "The Programming Executive"** for more information.

EXAMPLE 6-1: `SendCommand`

```
// Select the Fastdata Register
SendCommand(ETAP_FASTDATA)
// Send/Receive 32-bit Data
oData = XferFastData(32'h12)
```

FIGURE 6-7: XferFastData 4-WIRE

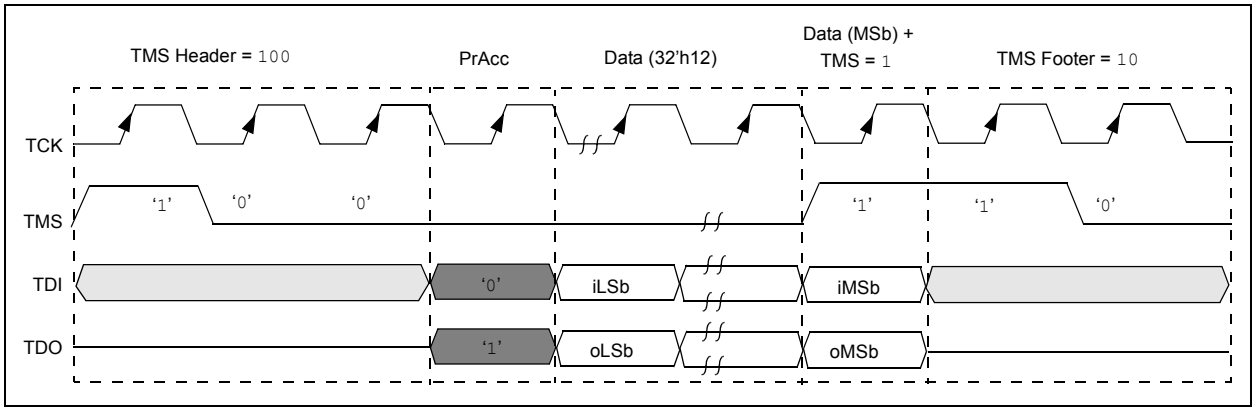


FIGURE 6-8: XferFastData 2-WIRE (2-PHASE)

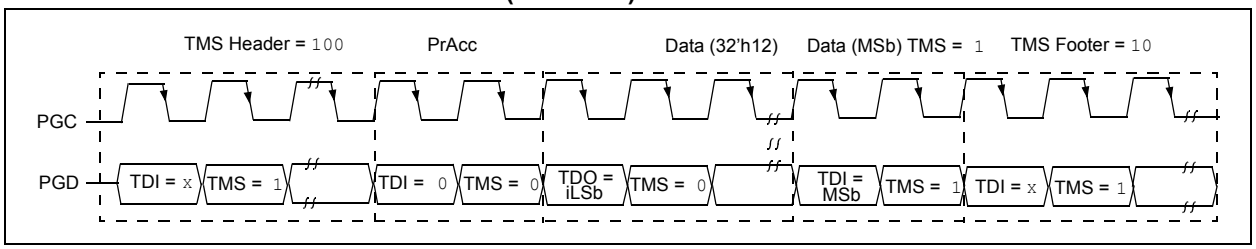
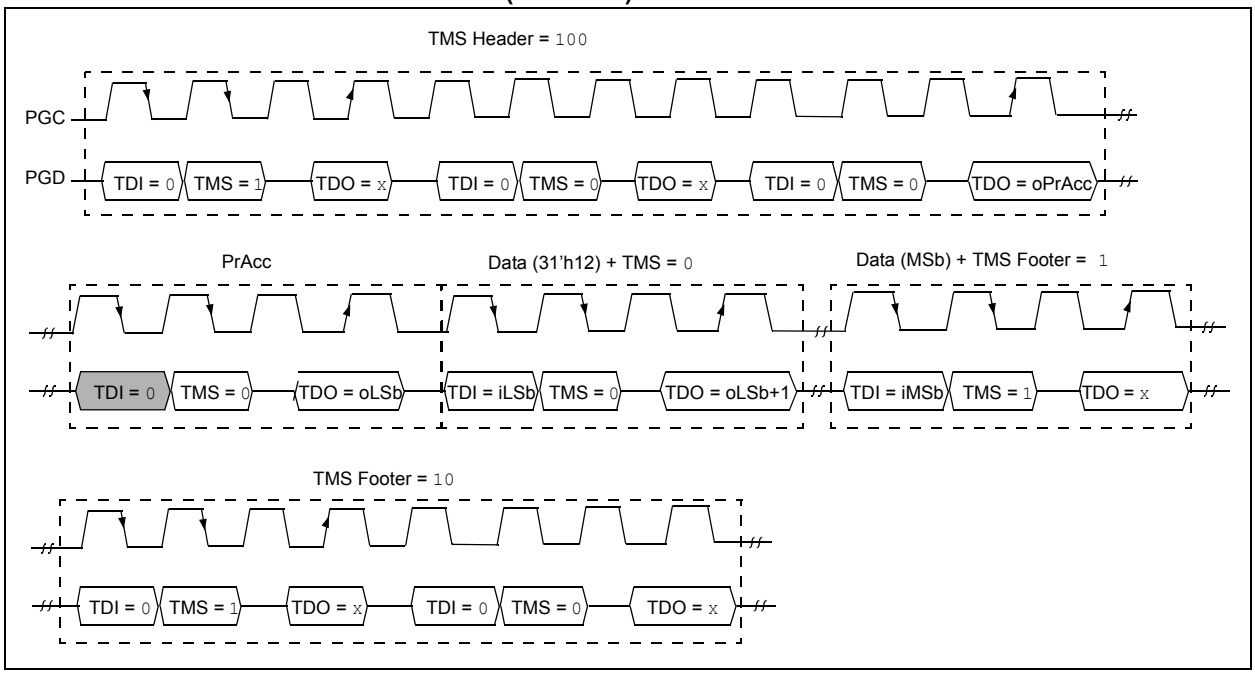


FIGURE 6-9: XferFastData 2-WIRE (4-PHASE)



6.5 `XferInstruction` Pseudo Operation

Format:

`XferInstruction` (instruction)

Purpose:

To send 32 bits of data for the device to execute.

Description:

The instruction is clocked into the device and then executed by CPU.

Restrictions:

The device must be in Debug mode.

EXAMPLE 6-2: `XferInstruction`

```
XferInstruction (instruction)
{
    // Select Control Register
    SendCommand(ETAP_CONTROL);
    // Wait until CPU is ready
    // Check if Processor Access bit (bit 18) is set
    do {
        controlVal = XferData(32'h0x0004C000);
    } while( PrAcc(contorlVal<18>) is not '1' );

    // Select Data Register
    SendCommand(ETAP_DATA);

    // Send the instruction
    XferData(instruction);

    // Tell CPU to execute instruction
    SendCommand(ETAP_CONTROL);
    XferData(32'h0x0000C000);
}
```

7.0 ENTERING PROGRAMMING MODE

For 2-wire programming methods, the target device must be placed in a special programming mode before executing further steps.

Note:	If a 4-wire programming method is used, it is not necessary to enter the programming mode.

The following steps are required to enter Programming mode:

1. The MCLR pin is briefly driven high, then low.
2. A 32-bit key sequence is clocked into PGDx.
3. MCLR is then driven high within a specified period of time and held.

Please refer to **Section 20.0 "AC/DC Characteristics and Timing Requirements"** for timing requirements.

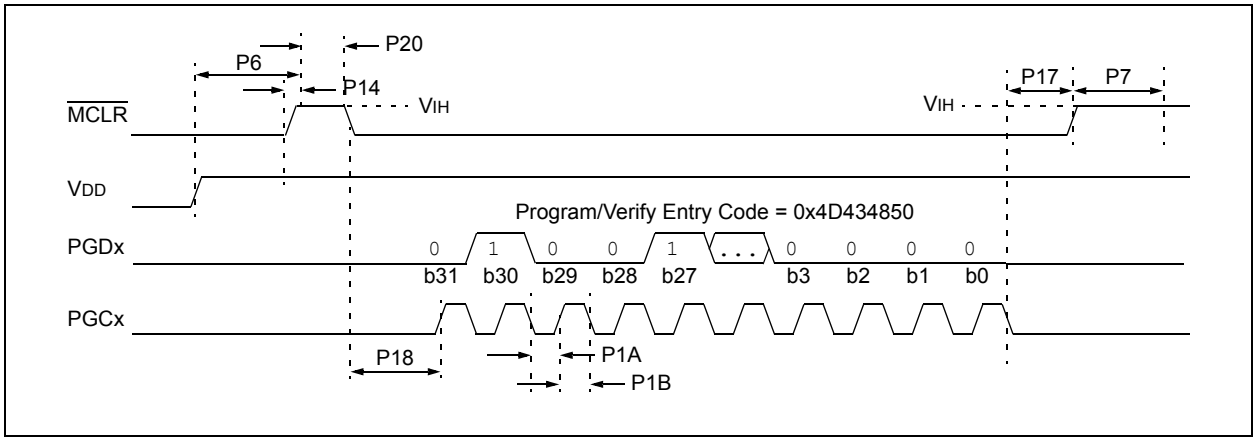
The programming voltage applied to MCLR is VIH, which is essentially VDD, in PIC32MX devices. There is no minimum time requirement for holding at VIH. After VIH is removed, an interval of at least P18 must elapse before presenting the key sequence on PGDx.

The key sequence is a specific 32-bit pattern: '0100 1101 0100 0011 0100 1000 0101 0000' (the acronym 'MCHP', in ASCII). The device will enter Program/Verify mode only if the key sequence is valid. The MSb of the Most Significant nibble must be shifted in first.

Once the key sequence is complete, VIH must be applied to MCLR and held at that level for as long as Programming mode is to be maintained. An interval of at least time P17 and P7 must elapse before presenting data on PGDx. Signals appearing on PGDx before P7 has elapsed will not be interpreted as valid.

Upon successful entry, the program memory can be accessed and programmed in serial fashion. While in Programming mode, all unused I/Os are placed in the high-impedance state.

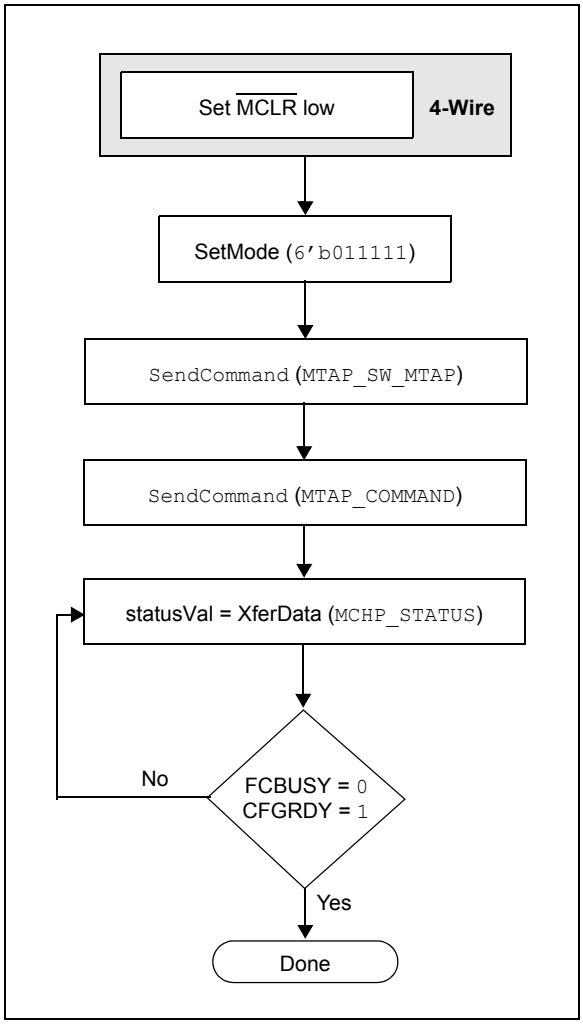
FIGURE 7-1: ENTERING ENHANCED ICSP™ MODE



8.0 CHECK DEVICE STATUS

Before a device can be programmed, the programmer must check the status of the device to ensure that it is ready to receive information.

FIGURE 8-1: CHECK DEVICE STATUS



8.1 4-Wire Interface

Four-wire JTAG programming is a Mission mode operation and therefore the setup sequence to begin programing should be done while asserting MCLR. Holding the device in Reset prevents the processor from executing instructions or driving ports.

The following steps are required to check the device status using the 4-wire interface:

1. Set MCLR pin low.
2. SetMode (`6'b011111`) to force the Chip TAP controller into Run Test/Idle state.
3. `SendCommand` (`MTAP_SW_MTAP`).
4. `SendCommand` (`MTAP_COMMAND`).
5. statusVal = XferData (`MCHP_STATUS`).
6. If CFGRDY (statusVal<3>) is not '`1`' and FCBUSY (statusVal<2>) is not '`0`' GOTO step 5.

8.2 2-Wire Interface

The following steps are required to check the device status using the 2-wire interface:

1. SetMode (`6'b011111`) to force the Chip TAP controller into Run Test/Idle state.
2. `SendCommand` (`MTAP_SW_MTAP`).
3. `SendCommand` (`MTAP_COMMAND`).
4. statusVal = XferData (`MCHP_STATUS`).
5. If CFGRDY (statusVal<3>) is not '`1`' and FCBUSY (statusVal<2>) is not '`0`', GOTO step 4.

Note:	If CFGRDY and FCBUSY do not come to the proper state within 10 ms, the sequence may have been executed wrong or the device is damaged.

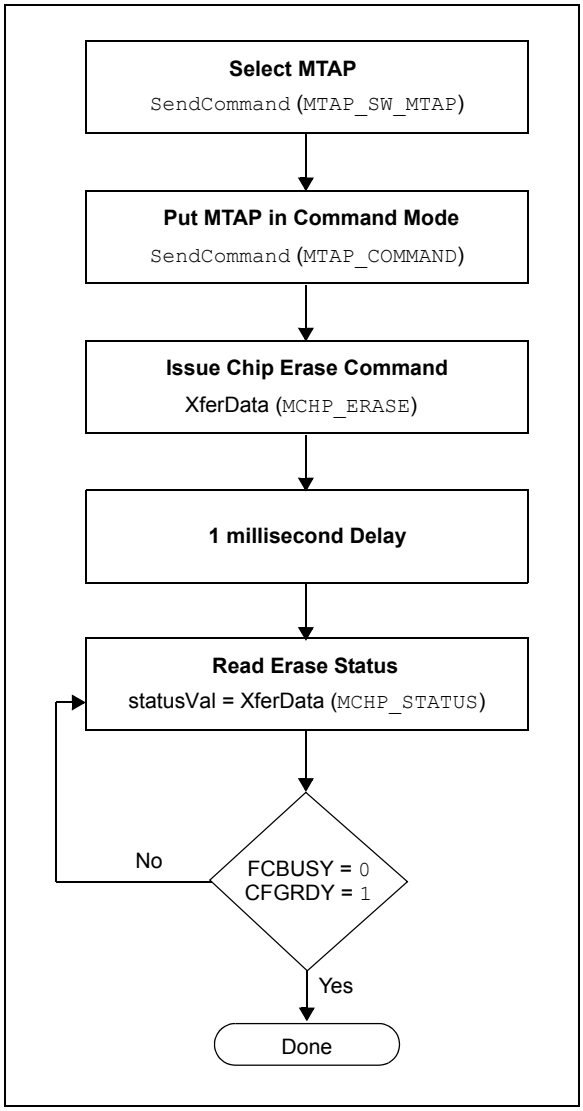
9.0 ERASING THE DEVICE

Before a device can be programmed, it must be erased. The erase operation writes all '1s' to the Flash memory and prepares it to program a new set of data. Once a device is erased, it can be verified by performing a "Blank Check" operation. See **Section 9.1 "Blank Check"** for more information.

The procedure for erasing program memory (Program, Boot, and Configuration memory) consists of selecting the MTAP and sending the MCHP_ERASE command. The programmer then must wait for the erase operation to complete by reading and verifying bits in the MCHP_STATUS value. Figure 9-1 illustrates the process for performing a Chip Erase.

Note:	The Device ID memory locations are read-only and cannot be erased. Therefore, Chip Erase has no effect on these memory locations.

FIGURE 9-1: ERASE DEVICE



The following steps are required to erase a target device:

1. SendCommand (MTAP_SW_MTAP).
2. SendCommand (MTAP_COMMAND).
3. XferData (MCHP_ERASE).
4. delay 1 ms.
5. statusVal = XferData (MCHP_STATUS).
6. If CFGRDY (statusVal<3>) is not '1' and FCBUSY (statusVal<2>) is not '0', GOTO to step 4.

Note:	The Chip Erase operation is a self-timed operation. If the FCBUSY and CFGRDY bits do not become properly set within the specified Chip Erase time, the sequence may have been executed wrong or the device is damaged.

9.1 Blank Check

The term "Blank Check" implies verifying that the device has been successfully erased and has no programmed memory locations. A blank or erased memory location always reads as '1'.

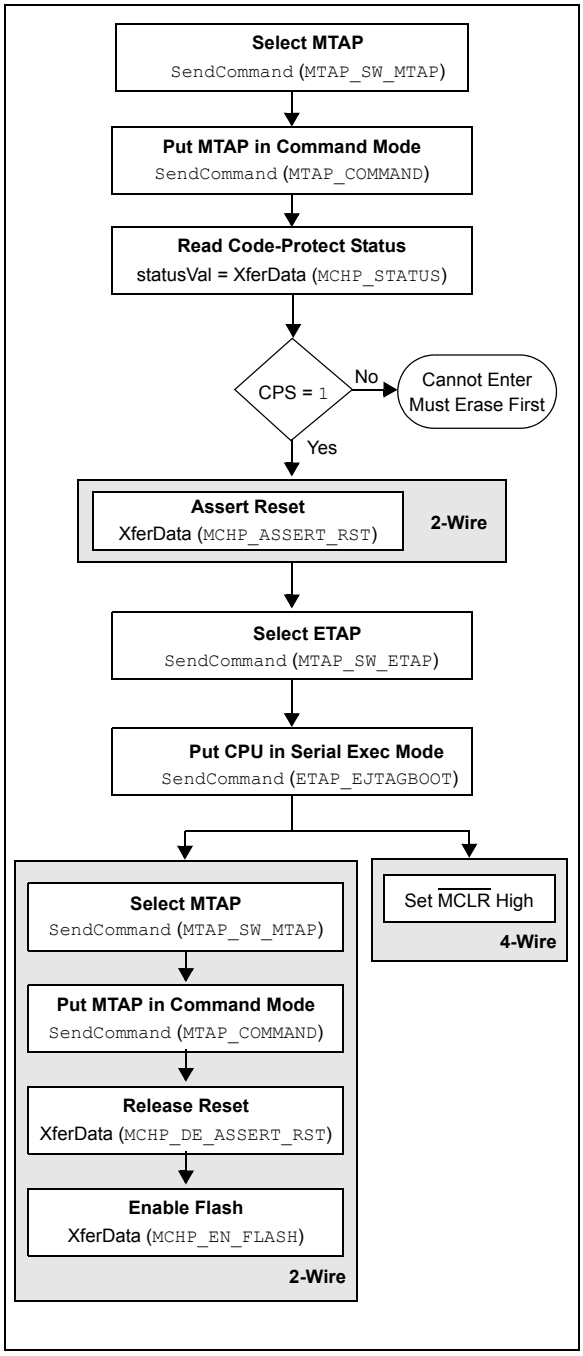
The device Configuration registers are ignored by the Blank Check. Additionally, all unimplemented memory space should be ignored from the Blank Check.

10.0 ENTERING SERIAL EXECUTION MODE

Before a device can be programmed, it must be placed in Serial Execution mode.

The procedure for entering Serial Execution mode consists of verifying that the device is not code-protected. If the device is code-protected, a Chip Erase must be performed. See **Section 9.0 "Erasing the Device"** for details.

FIGURE 10-1: ENTERING SERIAL EXECUTION MODE



10.1 4-Wire Interface

The following steps are required to enter Serial Execution mode:

1. SendCommand (MTAP_SW_MTAP).
2. SendCommand (MTAP_COMMAND).
3. statusVal = XferData (MCHP_STATUS).
4. If CPS (statusVal<7>) is not '1', the device must be erased first.
5. SendCommand (MTAP_SW_ETAP).
6. SendCommand (ETAP_EJTAGBOOT).
7. Set $\overline{\text{MCLR}}$ high.

10.2 2-Wire Interface

The following steps are required to enter Serial Execution mode:

1. SendCommand (MTAP_SW_MTAP).
2. SendCommand (MTAP_COMMAND).
3. statusVal = XferData (MCHP_STATUS).
4. If CPS (statusVal<7>) is not '1', the device must be erased first.
5. XferData (MCHP_ASSERT_RST).
6. SendCommand (MTAP_SW_ETAP).
7. SendCommand (ETAP_EJTAGBOOT).
8. SendCommand (MTAP_SW_MTAP).
9. SendCommand (MTAP_COMMAND).
10. XferData (MCHP_DE_ASSERT_RST).
11. XferData (MCHP_EN_FLASH).

11.0 DOWNLOADING THE PROGRAMMING EXECUTIVE (PE)

The PE resides in RAM memory and is executed by the CPU to program the device. The PE provides the mechanism for the programmer to program and verify PIC32MX devices using a simple command set and communication protocol. There are several basic functions provided by the PE:

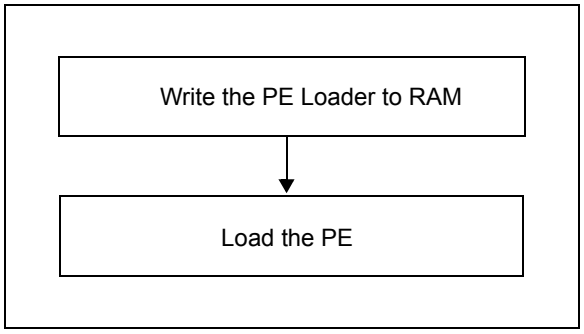
- Read Memory
- Erase Memory
- Program Memory
- Blank Check
- Read Executive Firmware Revision
- Get the cyclic redundancy check (CRC) of Flash memory locations

The PE performs the low-level tasks required for programming and verifying a device. This allows the programmer to program the device by issuing the appropriate commands and data. A detailed description for each command is provided in **Section 16.2 "The PE Command Set"**.

The PE uses the device's data RAM for variable storage and program execution. After the PE has run, no assumptions should be made about the contents of data RAM.

After the PE is loaded into the data RAM, the PIC32MX family can be programmed using the command set shown in Table 16-1.

FIGURE 11-1: DOWNLOADING THE PE



Loading the PE in the memory is a two step process:

1. Load the PE loader in the data RAM. (The PE loader loads the PE binary file in the proper location of the data RAM, and when done, jumps to the programming exec and starts executing it.)
2. Feed the PE binary to the PE loader.

Table 11-1 lists the steps that are required to download the PE.

TABLE 11-1: DOWNLOAD THE PE

Operation	Operand

Step 1: Initialize BMXCON to 0x1f0040. The instruction sequence executed by the PIC32MX core is as follows:
`lui a0,0xbf88`
`ori a0,a0,0x2000 /* address of BMXCON */`
`lui a1,0x1f`
`ori a1,a1,0x40 /* $a1 has 0x1f0040 */`
`sw a1,0(a0) /* BMXCON initialized */`	

`XferInstruction`	`0x3c04bf88`
`XferInstruction`	`0x34842000`
`XferInstruction`	`0x3c05001f`
`XferInstruction`	`0x34a50040`
`XferInstruction`	`0xac850000`

Step 2: Initialize BMXDKPBA to 0x800. The instruction sequence executed by the PIC32MX core is as follows:
`li a1,0x800`
`sw a1,16(a0)`	

`XferInstruction`	`0x34050800`
`XferInstruction`	`0xac850010`

Step 3: Initialize BMXDUDBA and BMXDUPBA to the value of BMXDRMSZ. The instruction sequence executed by the PIC32MX core is as follows:
`lw a1,64(a0) /* load BMXDMSZ */`
`sw a1,32(a0)`
`sw a1,48(a0)`	

`XferInstruction`	`0x8C850040`
`XferInstruction`	`0xac850020`
`XferInstruction`	`0xac850030`

Step 4: Set up PIC32MX RAM address for PE. The instruction sequence executed by the PIC32MX core is as follows:
`lui a0,0xa000`
`ori a0,a0,0x800`	

`XferInstruction`	`0x3c04a000`
`XferInstruction`	`0x34840800`

TABLE 11-1: DOWNLOAD THE PE

Operation	Operand

Step 5: Load the PE_Loader. Repeat this step (Step 5) until the entire PE_Loader is loaded in the PIC32MX memory. In the operands field, "<PE_loader hi++>" represents the MSbs 31 through 16 of the PE loader opcodes shown in Table 11-2. Likewise, "<PE_loader lo++>" represents the LSbs 15 through 0 of the PE loader opcodes shown in Table 11-2. The "++" sign indicates that when these operations are performed in succession, the new word is to be transferred from the list of opcodes of the LPE Loader shown in Table 11-2. The instruction sequence executed by the PIC32MX core is as follows:
`lui a2, <PE_loader hi++>`
`ori a0,a0, <PE_loader lo++>`
`sw a2,0(a0)`
`addiu a0,a0,4`	

XferInstruction	(0x3c06 <PE_loader hi++>)
XferInstruction	(0x34c6 <PE_loader lo++>)
XferInstruction	0xac860000
XferInstruction	0x24840004

Step 6: Jump to the PE_Loader. The instruction sequence executed by the PIC32MX core is as follows:
`lui t9,0xa000`
`ori t9,t9,0x800`
`jr t9`
`nop`	

XferInstruction	0x3c19a000
XferInstruction	0x37390800
XferInstruction	0x03200008
XferInstruction	0x00000000
Step 7: Load the PE using the PE_Loader. Repeat the last instruction of this step (Step 7) until the entire PE is loaded into the PIC32MX memory. In this step, you are given an Intel® Hex format file of the PE that you will parse and transfer a number of 32-bit words at a time to the PIC32MX memory (refer to **Appendix B: "Hex File Format"**). The instruction sequence executed by the PIC32MX is shown in the "Instruction" column of Table 11-2: PE Loader Opcodes.	
SendCommand	ETAP_FASTDATA
XferFastData	PE_ADDRESS (Address of PE program block from PE Hex file)
XferFastData	PE_SIZE (Number of 32-bit words of the program block from PE Hex file)
XferFastData	PE software opcode from PE Hex file (PE Instructions)

TABLE 11-1: DOWNLOAD THE PE

Operation	Operand
Step 8: Jump to the PE. Magic number (0xDEAD0000) instructs the PE_Loader that the PE is completely loaded into the memory. When the PE_Loader sees the magic number, it jumps to the PE.	
XferFastData	0x00000000
XferFastData	0xDEAD0000

TABLE 11-2: PE LOADER OPCODES

Opcode	Instruction
0x3c07dead	lui a3, 0xdead
0x3c06ff20	lui a2, 0xff20
0x3c05ff20	lui a1, 0xff20
	here1:
0x8cc40000	lw a0, 0 (a2)
0x8cc30000	lw v1, 0 (a2)
0x1067000b	beq v1, a3, <here3>
0x00000000	nop
0x1060fffb	beqz v1, <here1>
0x00000000	nop
	here2:
0x8ca20000	lw v0, 0 (a1)
0x2463ffff	addiu v1, v1, -1
0xac820000	sw v0, 0 (a0)
0x24840004	addiu a0, a0, 4
0x1460fffb	bnez v1, <here2>
0x00000000	nop
0x1000fff3	b <here1>
0x00000000	nop
	here3:
0x3c02a000	lui v0, 0xa000
0x34420900	ori v0, v0, 0x900
0x00400008	jr v0
0x00000000	nop

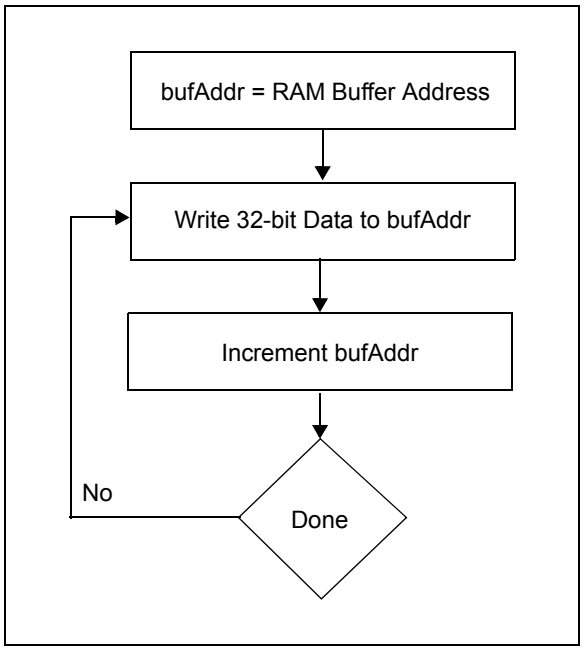
12.0 DOWNLOADING A DATA BLOCK

To program a block of data to the PIC32MX device, it must first be loaded into SRAM.

12.1 Without the PE

To program a block of memory without the use of the PE, the block of data must first be written to RAM. This method requires the programmer to transfer the actual machine instructions with embedded data for writing the block of data to the devices internal RAM memory.

FIGURE 12-1: DOWNLOADING DATA WITHOUT THE PE



The following steps are required to download a block of data:

1. `XferInstruction` (opcode).
2. Repeat Step 1 until the last instruction is transferred to CPU.

TABLE 12-1: DOWNLOAD DATA OPCODES

Opcode	Instruction
Step 1: Initialize SRAM Base Address to 0xA000_0000	
3c10a000	lui $s0, 0xA000;
Step 2: Write the entire row of data to be programmed into system SRAM.	

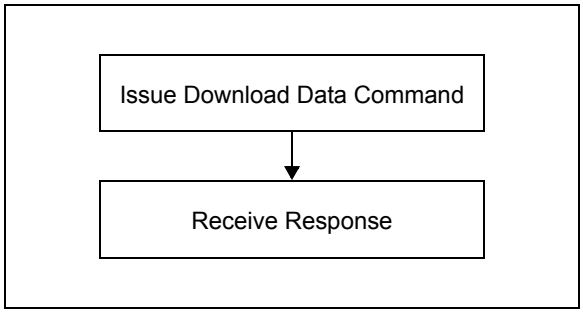
3c08<DATA>
3508<DATA>
ae08<OFFSET>
	lui $t0, <DATA(31:16)>;
	ori $t0, <DATA(15:0)>;
	sw $t0, <OFFSET>($s0);
	// OFFSET increments by 4

Step3: Repeat Step 2 until one row of data has been loaded.	

12.2 With the PE

When using the PE, the code memory is programmed with the `PROGRAM` command (see Table 16-2). The program can program up to one row of code memory starting from the memory address specified in the command. The number of `PROGRAM` commands required to program a device depends on the number of write blocks that must be programmed in the device.

FIGURE 12-2: DOWNLOADING DATA WITH THE PE



The following steps are required to download a block of data using the PE:

1. XferFastData (`PROGRAM|DATA_SIZE`).
2. XferFastData (`ADDRESS`).
3. response = XferFastData (`32'h0x00`).

13.0 INITIATING A FLASH ROW WRITE

Once a row of data has been downloaded into the device's SRAM, the programming sequence must be initiated to write the block of data to Flash memory.

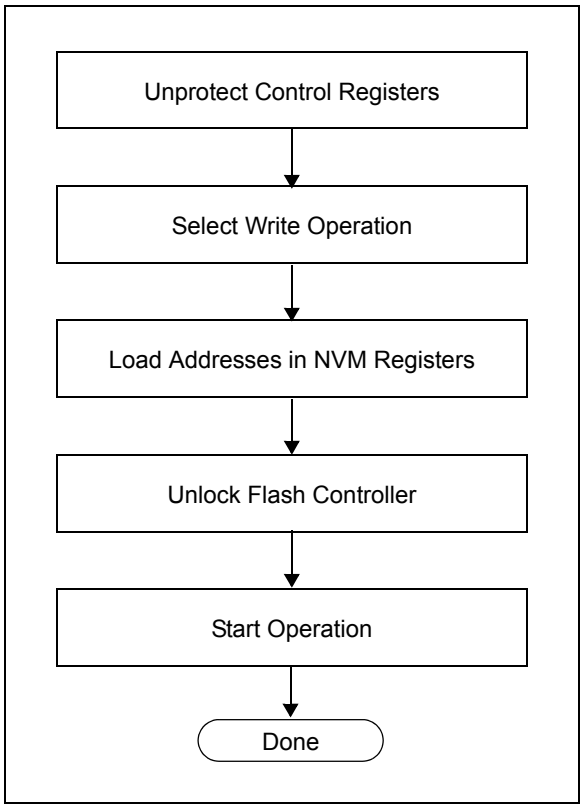
13.1 With the PE

When using PE, the data is immediately written to the Flash memory from the SRAM. No further action is required.

13.2 Without the PE

Flash memory write operations are controlled by the NVMCON register. Programming is performed by setting NVMCON to select the type of write operation and initiating the programming sequence by setting the WR control bit NVMCON<15>.

FIGURE 13-1: INITIATING FLASH WRITE WITHOUT THE PE



The following steps are required to initiate a Flash write:

1. `XferInstruction` (opcode).
2. Repeat Step 1 until the last instruction is transferred to the CPU.

TABLE 13-1: INITIATE FLASH ROW WRITE OPCODES

Opcode	Instruction
Step 1: Initialize some constants.	
3c04bf80	lui a0,0xbf80
3484f400	ori a0,a0,0xf400
34054003	ori a1,$0,0x4003
34068000	ori a2,$0,0x8000
34074000	ori a3,$0,0x4000
3c11aa99	lui s1,0xaa99
36316655	ori s1,s1,0x6655
3c125566	lui s2,0x5566
365299aa	ori s2,s2,0x99aa
3c13ff20	lui s3,0xff20
3c100000	lui s0,0x0000
Step 2: Set NVMADDR with the address of the Flash row to be programmed.	
3c08<ADDR>	lui t0,<FLASH_ROW_ADDR(31:16)>
3508<ADDR>	ori t0,t0,<FLASH_ROW_ADDR(15:0)>
ac880020	sw t0,32(a0)
Step 3: Set NVMSRCADDR with the physical source SRAM address.	
3610<ADDR>	ori s0,s0,<RAM_ADDR(15:0)>
Step 4: Set up NVMCON for write operation and poll LVDSTAT.	
ac850000	sw a1,0(a0)
	delay (6 µs)
	here1:
8C880000	lw t0,0(a0)
31080800	andit0,t0,0x0800
1500fffd	bne t0,$0,<here1>
00000000	nop
Step 5: Unlock NVMCON and start write operation.	
ac910010	sw s1,16(a0)
ac920010	sw s2,16(a0)
ac860008	sw a2,8(a0)
Step 6: Repeatedly read the NVMCON register and poll for WR bit to get cleared.	
	here2:
8c880000	lw t0,0(a0)
01064024	and t0,t0,a2
1500fffd	bne t0,$0,<here2>
00000000	nop

TABLE 13-1: INITIATE FLASH ROW WRITE OPCODES (CONTINUED)

Opcode	Instruction
Step 7: Wait at least 500 ns after seeing a '0' in NVMCON<15> before writing to any NVM registers. This requires inserting NOP in the execution. Example: The following example assumes that the core is executing at 8 MHz; therefore, four NOP instructions equate to 500 ns.	
00000000	nop
00000000	nop
00000000	nop
00000000	nop
Step 8: Clear NVMCON.WREN bit.	
ac870004	sw a3,4(a0)
Step 9: Check the NVMCON.WRERR bit to ensure that the program sequence completed successfully. If an error occurs, jump to the error-processing routine.	
8c880000	lw t0,0(a0)
30082000	andit0,zero,0x2000
1500<ERR_ PROC>	bne t0, $0, <err_proc_offset>
00000000	nop

14.0 VERIFY DEVICE MEMORY

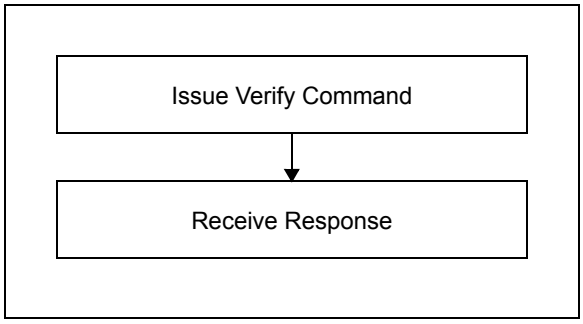
The verify step involves reading back the code memory space and comparing it against the copy held in the programmer's buffer. The Configuration registers are verified with the rest of the code.

Note:	Because the Configuration registers include the device code protection bit, code memory should be verified immediately after writing (if code protection is enabled). This is because the device will not be readable or verifiable if a device Reset occurs after the code-protect bit has been cleared.

14.1 Verifying Memory with the PE

Memory verify is performed using the `GET_CRC` command (see Table 16-2) as shown below.

FIGURE 14-1: VERIFYING MEMORY WITH THE PE



The following steps are required to verify memory using the PE:

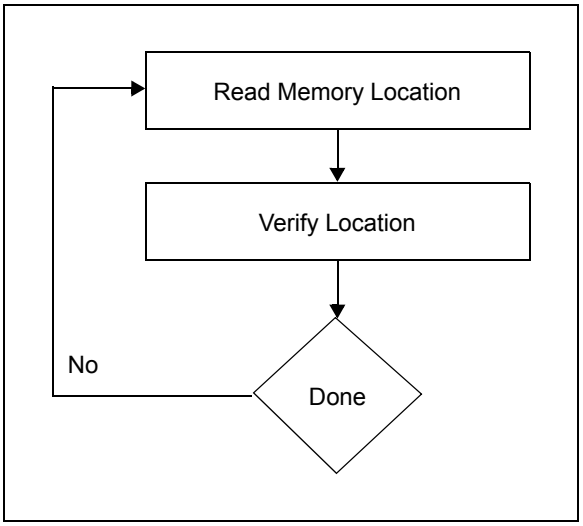
1. XferFastData (`GET_CRC`).
2. XferFastData (`start_Address`).
3. XferFastData (length).
4. valCkSum = XferFastData (32'h0x0).

Verify that valCkSum matches the checksum of the copy held in the programmer's buffer.

14.2 Verifying Memory without the PE

Reading from Flash memory is performed by executing a series of read accesses from the Fastdata register. Table 19-4 shows the EJTAG programming details, including the address and opcode data for performing processor access operations.

FIGURE 14-2: VERIFYING MEMORY WITHOUT THE PE



The following steps are required to verify memory:

1. `XferInstruction` (opcode).
2. Repeat Step 1 until the last instruction is transferred to the CPU.
3. Verify that valRead matches the copy held in the programmer's buffer.
4. Repeat Steps 1-3 for each memory location.

TABLE 14-1: VERIFY DEVICE OPCODES

Opcode	Instruction
Step 1: Initialize some constants.	
3c04bf80	lui $s3, 0xFF20
Step 2: Read memory Location.	
3c08<ADDR> 3508<ADDR>	lui $t0,<FLASH_WORD_ADDR(31:16)> ori $t0,<FLASH_WORD_ADDR(15:0)>
Step 3: Write to Fastdata location.	
8d090000 ae690000	lw $t1, 0($t0) sw $t1, 0($s3)
Step 4: Read data from Fastdata register 0xFF200000.	
Step 5: Repeat Steps 2-4 until all configuration locations are read.	

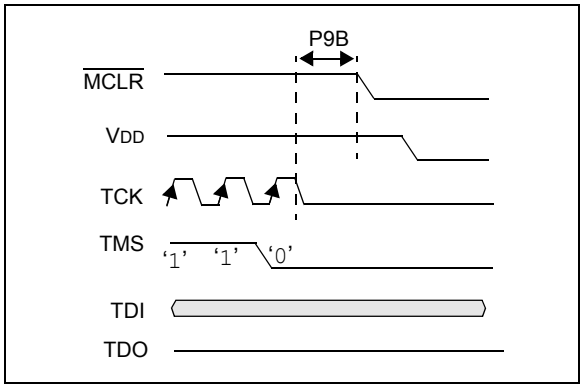
15.0 EXITING PROGRAMMING MODE

Once a device has been properly programmed, the device must be taken out of Programming mode to start proper execution of its new program memory contents.

15.1 4-Wire Interface

Exiting Test mode is done by removing VIH from MCLR, as illustrated in Figure 15-1. The only requirement for exit is that an interval, P9B, should elapse between the last clock and program signals before removing VIH.

FIGURE 15-1: 4-WIRE EXIT TEST MODE



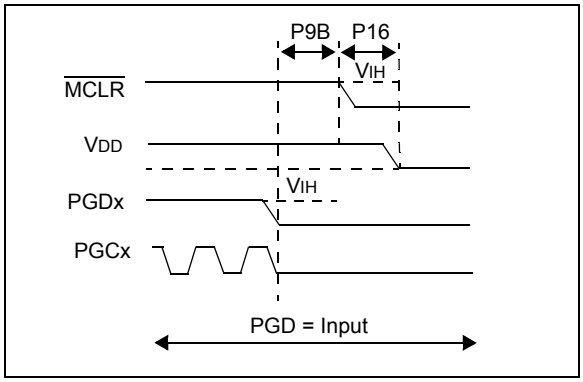
The following steps are required to exit Test mode:

1. SetMode (5'b11111).
2. Assert MCLR.
3. Remove power (if the device is powered).

15.2 2-Wire Interface

Exiting Test mode is done by removing VIH from MCLR, as illustrated in Figure 15-2. The only requirement for exit is that an interval, P9B, should elapse between the last clock and program signals on PGCx and PGDx before removing VIH.

FIGURE 15-2: 2-WIRE EXIT TEST MODE



The following list provides the actual steps required to exit test mode:

1. SetMode (5'b11111).
2. Assert MCLR.
3. Issue a clock pulse on PGCx.
4. Remove power (if the device is powered).

16.0 THE PROGRAMMING EXECUTIVE

16.1 PE Communication

The programmer and the PE have a master-slave relationship, where the programmer is the master programming device and the PE is the slave.

All communication is initiated by the programmer in the form of a command. The PE is able to receive only one command at a time. Correspondingly, after receiving and processing a command, the PE sends a single response to the programmer.

16.1.1 2-WIRE ICSP EJTAG RATE

In Enhanced ICSP mode, the PIC32MX family devices operate from the internal Fast RC oscillator, which has a nominal frequency of 8 MHz. To ensure that the programmer does not clock too fast, it is recommended that a 1 MHz clock be provided by the programmer.

16.1.2 COMMUNICATION OVERVIEW

The programmer and the PE communicate using the EJTAG Address, Data and Fastdata registers. In particular, the programmer transfers the command and data to the PE using the Fastdata register. The programmer receives a response from the PE using the Address and Data registers. The pseudo operation of receiving a response is shown in the `GetPEResponse` pseudo operation below:

Format:

response = `GetPEResponse()`

Purpose:

Enables the programmer to receive the 32-bit response value from the PE.

EXAMPLE 16-1: `GetPEResponse` EXAMPLE

```
WORD GetPEResponse()
{
        WORD response;

// Wait until CPU is ready
SendCommand(ETAP_CONTROL);
// Check if Proc. Access bit (bit 18) is set
do {
      controlVal=XferData(32'h0x0004C000 );
} while( PrAcc(contorlVal<18>) is not '1' );

// Select Data Register
SendCommand(ETAP_DATA);
// Receive Response
response = XferData(0);
// Tell CPU to execute instruction
SendCommand(ETAP_CONTROL);
XferData(32'h0x0000C000);
// return 32-bit response
return response;
}
```

The typical communication sequence between the programmer and the PE is shown in Table 16-1.

The sequence begins when the programmer sends the command and optional additional data to the PE, and the PE carries out the command.

When the PE has finished executing the command, it sends the response back to the programmer.

The response may contain more than one response. For example, if the programmer sent a `READ` command, the response will contain the data read.

TABLE 16-1: COMMUNICATION SEQUENCE FOR THE PE

Operation	Operand
Step 1: Send command and optional data from programmer to the PE.	
`XferFastData`	`(Command | data len)`
`XferFastData..`	`optional data..`
Step 2: Programmer reads the response from the PE.	
`GetPEResponse`	`response`
`GetPEResponse..`	`response..`

16.2 The PE Command Set

The PE command set is shown in Table 16-2. This table contains the opcode, mnemonic, length, time-out and short description for each command. Functional details on each command are provided in **Section 16.2.3 "ROW_PROGRAM Command"** through **Section 16.2.14 "CHANGE_CFG Command"**.

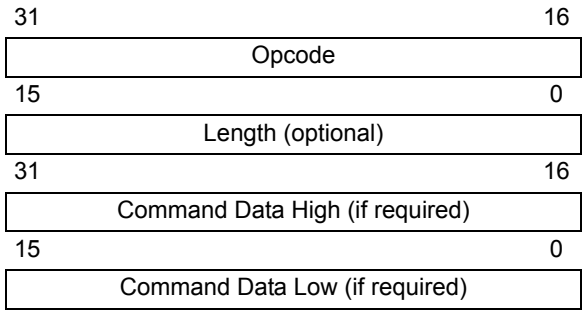
The PE sends a response to the programmer for each command that it receives. The response indicates if the command was processed correctly. It includes any required response data or error data.

16.2.1 COMMAND FORMAT

All PE commands have a general format consisting of a 32-bit header and any required data for the command (see Figure 16-1). The 32-bit header consists of a 16-bit opcode field, which is used to identify the command, a 16-bit command length field. The length field indicates the number of bytes to be transferred, if any.

Note:	Some commands have no Length information, however, the Length field must be sent and the programming executive will ignore the data.

FIGURE 16-1: COMMAND FORMAT



The command in the Opcode field must match one of the commands in the command set that is listed in Table 16-2. Any command received that does not match a command the list returns a NACK response, as shown in Table 16-3.

The PE uses the command Length field to determine the number of bytes to read from or to write to. If the value of this field is incorrect, the command is not be properly received by the PE.

TABLE 16-2: PE COMMAND SET

Opcode	Mnemonic	Length(1) (32-bit words)	Description
0x0	ROW_PROGRAM(2)	2	Program one row of Flash memory at the specified address.
0x1	READ	2	Read N 32-bit words of memory starting from the specified address. (N < 65536).
0x2	PROGRAM	130	Program Flash memory starting at the specified address.
0x3	WORD_PROGRAM	3	Program one word of Flash memory at the specified address.
0x4	CHIP_ERASE	1	Chip Erase of entire chip.
0x5	PAGE_ERASE	2	Erase pages of code memory from the specified address.
0x6	BLANK_CHECK	1	Blank Check code.
0x7	EXEC_VERSION	1	Read the PE software version.
0x8	GET_CRC	2	Get the CRC of Flash memory.
0x9	PROGRAM_CLUSTER	3	Programs the specified number of bytes to the specified address.
0xA	GET_DEVICEID	1	Returns the hardware ID of the device.
0xB	CHANGE_CFG(3)	2	Used by the probe to set various configuration settings for the PE.

Note 1: Length does not indicate the length of data to be transferred. Length indicates the size of the command itself, including 32-bit header.

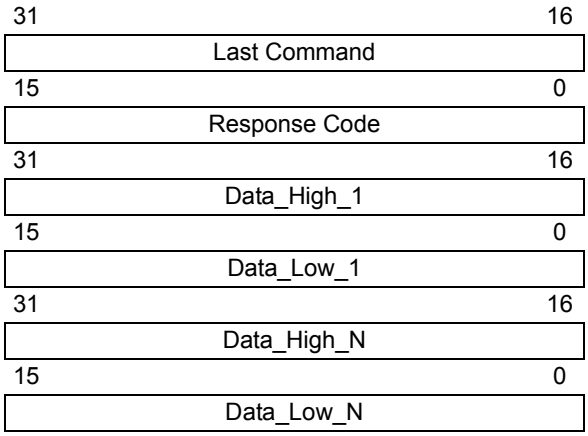
2: Refer to Table 5-1 for the row size for each device.

3: This command is not available in PIC32MX1XX/2XX devices.

16.2.2 RESPONSE FORMAT

The PE response set is shown in Table 16-3. All PE responses have a general format consisting of a 32-bit header and any required data for the response (see Figure 16-2).

FIGURE 16-2: RESPONSE FORMAT



16.2.2.1 Last_Cmd Field

Last_Cmd is a 16-bit field in the first word of the response and indicates the command that the PE processed. It can be used to verify that the PE correctly received the command that the programmer transmitted.

16.2.2.2 Response Code

The response code indicates whether the last command succeeded or failed, or if the command is a value that is not recognized. The response code values are shown in Table 16-3.

TABLE 16-3: RESPONSE VALUES

Opcode	Mnemonic	Description
0x0	PASS	Command successfully processed
0x2	FAIL	Command unsuccessfully processed
0x3	NACK	Command not known

16.2.2.3 Optional Data

The response header may be followed by optional data in case of certain commands such as read. The number of 32-bit words of optional data varies depending on the last command operation and its parameters.

16.2.3 ROW_PROGRAM COMMAND

The ROW_PROGRAM command instructs the PE to program a row of data at a specified address.

The data to be programmed to memory, located in command words Data_1 through Data_128, must be arranged using the packed instruction word format shown in Table 16-4.

FIGURE 16-3: ROW_PROGRAM COMMAND

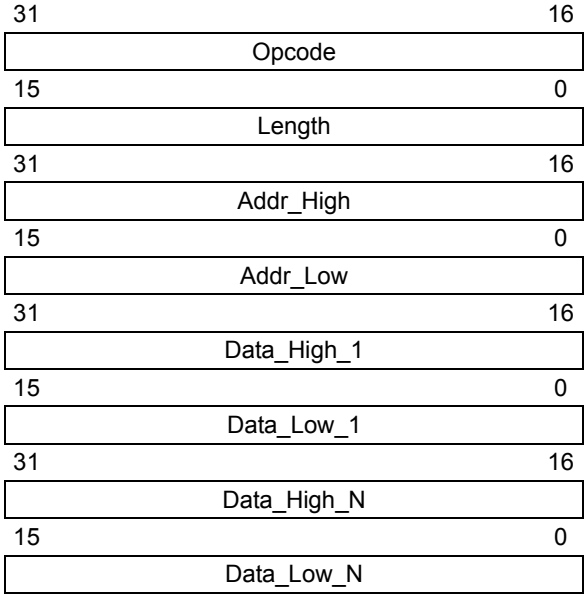
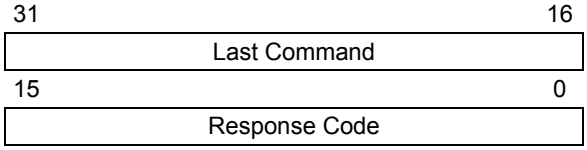


TABLE 16-4: ROW_PROGRAM FORMAT

Field	Description
Opcode	0x0
Length	128
Addr_High	High 16 bits of 32-bit destination address
Addr_Low	Low 16 bits of 32-bit destination address
Data_High_1	High 16 bits data word 1
Data_Low_1	Low 16 bits data word 1
Data_High_N	High 16 bits data word 2 through 128
Data_Low_N	Low 16 bits data word 2 through 128

Expected Response (1 word):

FIGURE 16-4: ROW_PROGRAM RESPONSE



16.2.4 READ COMMAND

The READ command instructs the PE to read the instruction Length field that contains the number of 32-bit words of Flash memory, including Configuration Words, starting from the 32-bit address specified by the Addr_Low and Addr_High fields. This command can only be used to read 32-bit data. All data returned in response to this command uses the packed data format that is shown in Table 16-5.

FIGURE 16-5: READ COMMAND

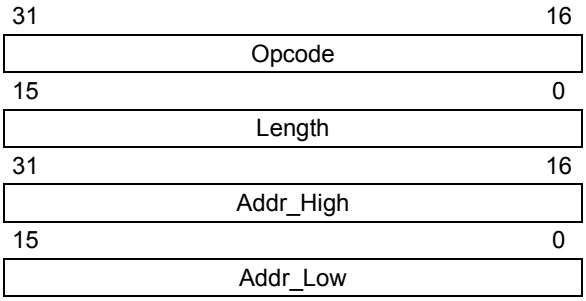
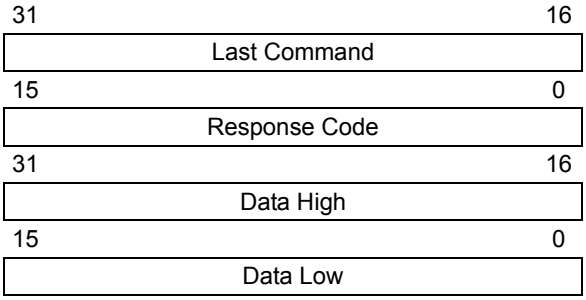


TABLE 16-5: READ FORMAT

Field	Description
Opcode	0x1
Length	Number of 32-bit words to read (max. of 65535)
Addr_Low	Low 16 bits of 32-bit source address
Addr_High	High 16 bits of 32-bit source address

Expected Response:

FIGURE 16-6: READ RESPONSE



Note:	Reading unimplemented memory will cause the PE to reset. Please ensure that only memory locations present on a particular device are accessed.

16.2.5 PROGRAM COMMAND

The PROGRAM command instructs the PE to program Flash memory, including Configuration Words, starting from the 32-bit address specified in the Addr_Low and Addr_High fields. The address must be aligned to a 512-byte boundary (aligned to Flash row size). Also, the length must be a multiple of 512 bytes (multiple of the Flash row size).

FIGURE 16-7: PROGRAM COMMAND

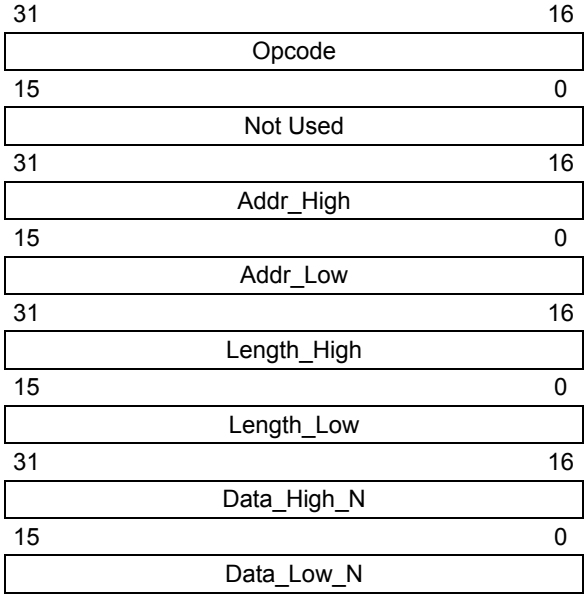


TABLE 16-6: PROGRAM FORMAT

Field	Description
Opcode	0x2
Addr_Low	Low 16 bits of 32-bit destination address
Addr_High	High 16 bits of 32-bit destination address
Length_Low	Low 16 bits of Length
Length_High	High 16 bits Length
Data_Low_N	Low 16 bits data word 2 through N
Data_High_N	High 16 bits data word 2 through N

There are three programming scenarios:

1. The length of the data to be programmed is 512 bytes.
2. The length of the data to be programmed is 1024 bytes.
3. The length of the data to be programmed is larger than 1024 bytes.

When the data length is equal to 512 bytes, the PE receives the 512-byte block of data from the probe and immediately sends the response for this command back to the probe.

When the data length is equal to 1024 bytes, the PE receives the first two 512-byte blocks of data from the probe sequentially. The PE sends the response with the status of the write operation for the first 512-byte block back to the probe, followed immediately by the status of the write operation for the second 512-byte block.

If the data to be programmed is larger than 1024 bytes, the PE receives the first two 512-byte blocks of data from the probe sequentially. The PE sends the response for the first 512-byte block of data back to the probe. The PE receives the third 512-byte block probe and sends the response for the second 512-bye block back to the probe. Successive blocks from the probe and subsequent responses to the probe are received and sent same way. After receiving the last 512-byte block from the probe, the PE sends the response for the second-to-last block to the probe, followed by the response for the last block.

If the PE encounters an error in programming any of the blocks, it sends a failure status to the probe. On receiving the failure status, the probe must stop sending data. The PE does not receive any other data for this command from the probe. The process is illustrated in Figure 16-9.

> **Note:** If the PROGRAM command fails, the programmer should read the failing row using the READ command from the Flash memory. Next, the programmer should compare the row received from Flash memory to its local copy, word-by-word, to determine the address where Flash programming fails.

The response for this command is a little different than the response for other commands. The 16 MSbs of the response contain the 16 LSbs of the destination address, where the last block is programmed. This helps the probe and the PE maintain proper synchronization of sending, and receiving, data and responses.

Expected Response (1 word):

FIGURE 16-8: PROGRAM RESPONSE

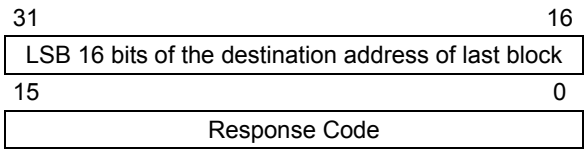
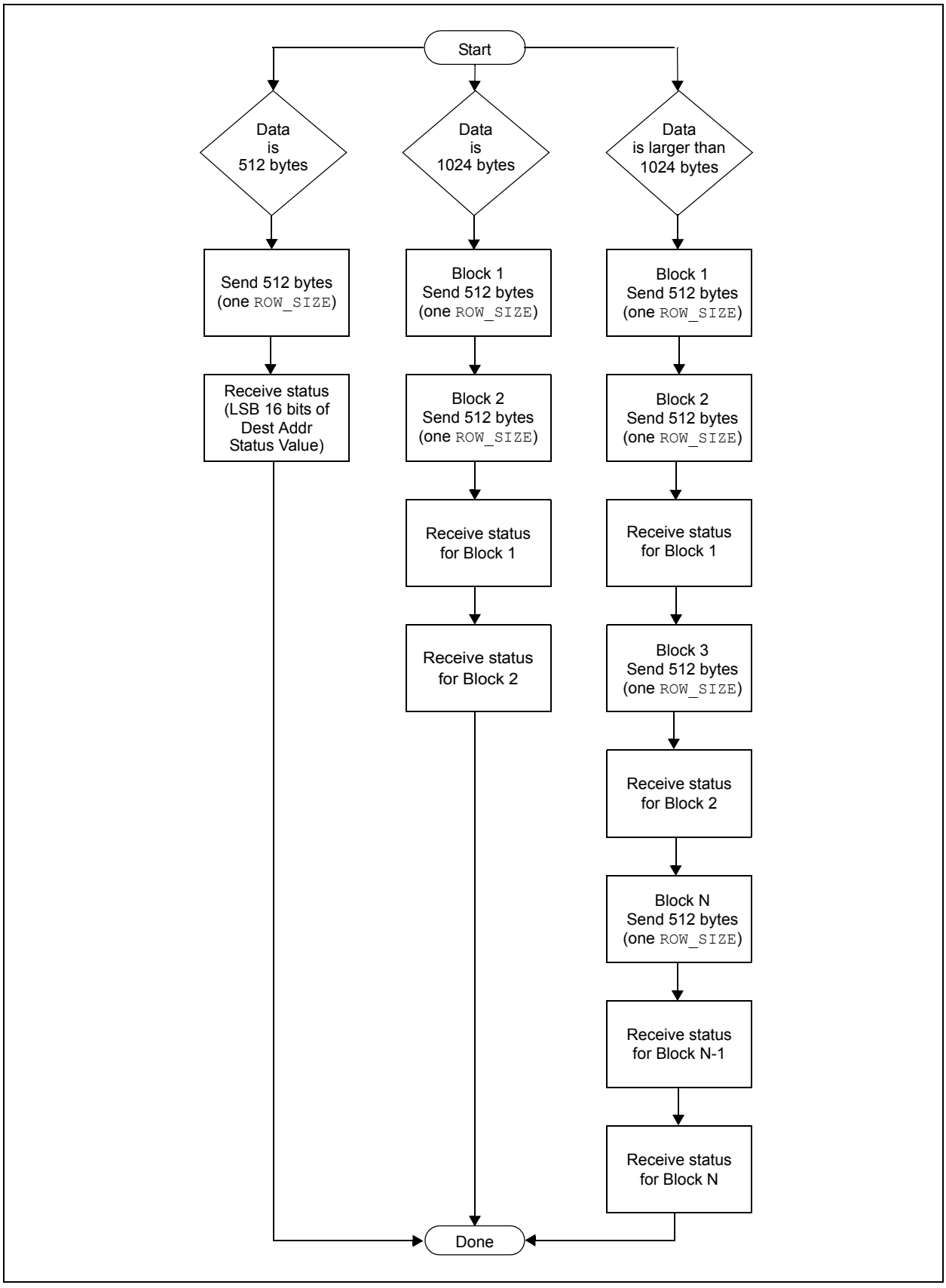


FIGURE 16-9: PROGRAM COMMAND ALGORITHM



16.2.6 WORD_PROGRAM COMMAND

The WORD_PROGRAM command instructs the PE to program a 32-bit word of data at the specified address.

FIGURE 16-10: WORD_PROGRAM COMMAND

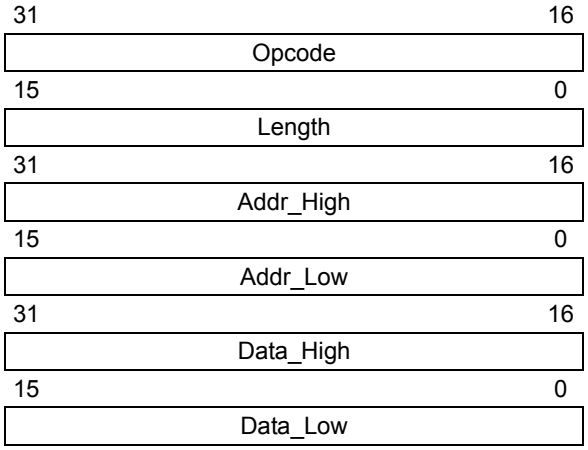
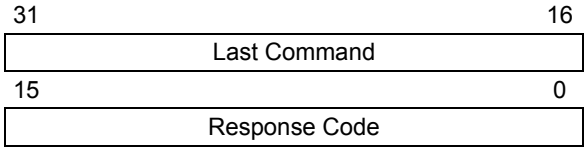


TABLE 16-7: WORD_PROGRAM FORMAT

Field	Description
Opcode	0x3
Length	2
Addr_High	High 16 bits of 32-bit destination address
Addr_Low	Low 16 bits of 32-bit destination address
Data_High	High 16 bits data word
Data_Low	Low 16 bits data word

Expected Response (1 word):

FIGURE 16-11: WORD_PROGRAM RESPONSE



16.2.7 CHIP_ERASE COMMAND

The CHIP_ERASE command erases the entire chip, including the configuration block.

After the erase is performed, the entire Flash memory contains 0xFFFF_FFFF.

FIGURE 16-12: CHIP_ERASE COMMAND

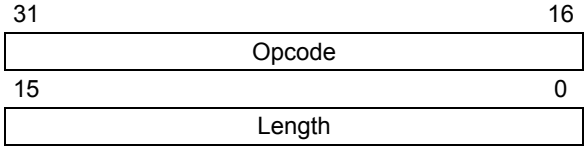
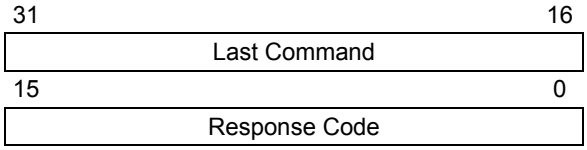


TABLE 16-8: CHIP_ERASE FORMAT

Field	Description
Opcode	0x4
Length	Ignored
Addr_Low	Low 16 bits of 32-bit destination address
Addr_High	High 16 bits of 32-bit destination address

Expected Response (1 word):

FIGURE 16-13: CHIP_ERASE RESPONSE



16.2.8 PAGE_ERASE COMMAND

The PAGE_ERASE command erases the specified number of pages of code memory from the specified base address. Depending on the device, the specified base address must be a multiple of 0x400 or 0x100.

After the erase is performed, all targeted words of code memory contain 0xFFFF_FFFF.

FIGURE 16-14: PAGE_ERASE COMMAND

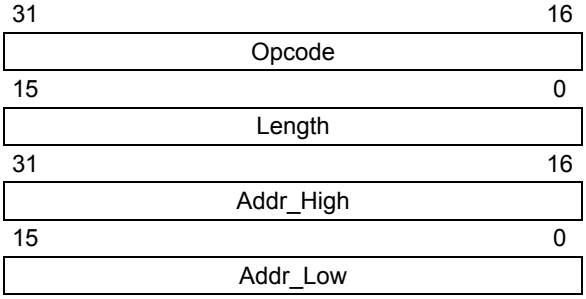
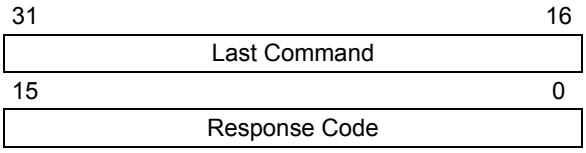


TABLE 16-9: PAGE_ERASE FORMAT

Field	Description
Opcode	0x5
Length	Number of pages to erase
Addr_Low	Low 16 bits of 32-bit destination address
Addr_High	High 16 bits of 32-bit destination address

Expected Response (1 word):

FIGURE 16-15: PAGE_ERASE RESPONSE



16.2.9 BLANK_CHECK COMMAND

The BLANK_CHECK command queries the PE to determine whether the contents of code memory and code-protect Configuration bits (GCP and GWRP) are blank (contains all '1's).

FIGURE 16-16: BLANK_CHECK COMMAND

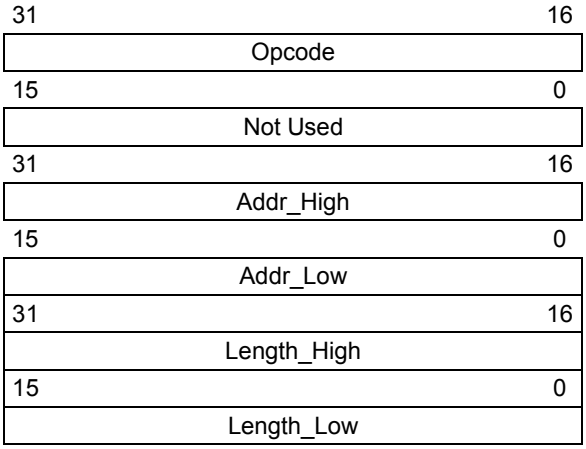
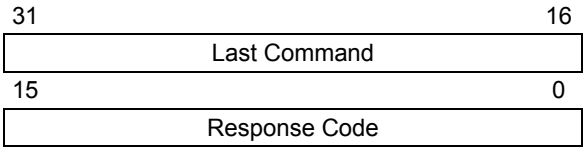


TABLE 16-10: BLANK_CHECK FORMAT

Field	Description
Opcode	0x6
Length	Number of program memory locations to check in terms of bytes
Address	Address where to start the Blank Check

Expected Response (1 word for blank device):

FIGURE 16-17: BLANK_CHECK RESPONSE



16.2.10 EXEC_VERSION COMMAND

EXEC_VERSION queries for the version of the PE software stored in RAM.

FIGURE 16-18: EXEC_VERSION COMMAND

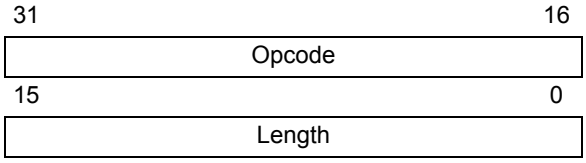
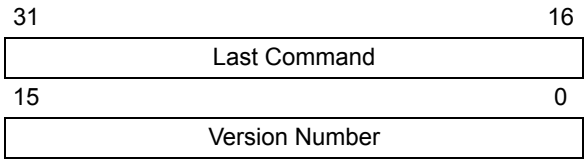


TABLE 16-11: EXEC_VERSION FORMAT

Field	Description
Opcode	0x7
Length	Ignored

Expected Response (1 word):

FIGURE 16-19: EXEC_VERSION RESPONSE



16.2.11 GET_CRC COMMAND

GET_CRC calculates the CRC of the buffer from the specified address to the specified length, using the table look-up method. The CRC details are as follows:

- CRC-CCITT, 16-bit
- Polynomial: X^16+X^12+X^5+1, hex 0x00011021
- Seed: 0xFFFF
- Most Significant Byte (MSB) shifted in first

Note 1: In the response, only the CRC Least Significant 16 bits are valid.

2: The PE will automatically determine if the hardware CRC is available and use it by default. The hardware CRC is not available on PIC32MX1XX/2XX devices.

FIGURE 16-20: GET_CRC COMMAND

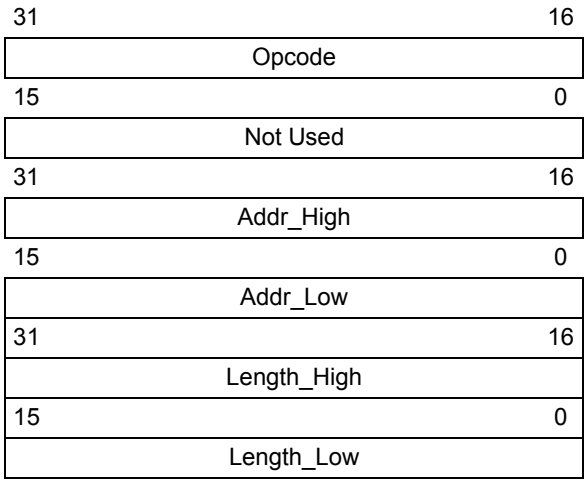
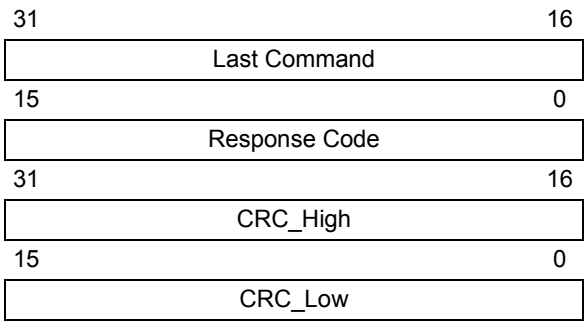


TABLE 16-12: GET_CRC FORMAT

Field	Description
Opcode	0x8
Address	Address where to start calculating the CRC
Length	Length of buffer on which to calculate the CRC, in number of bytes

Expected Response (2 words):

FIGURE 16-21: GET_CRC RESPONSE



16.2.12 PROGRAM_CLUSTER COMMAND

PROGRAM_CLUSTER programs the specified number of bytes to the specified address. The address must be 32-bit aligned, and the number of bytes must be a multiple of a 32-bit word.

FIGURE 16-22: PROGRAM_CLUSTER COMMAND

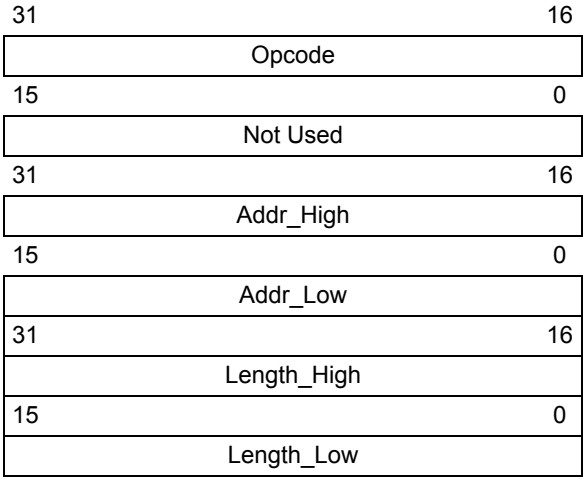


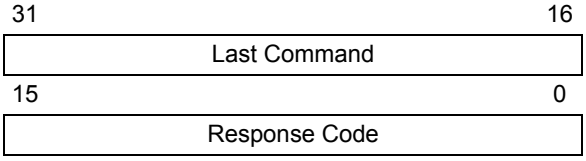
TABLE 16-13: PROGRAM_CLUSTER FORMAT

Field	Description
Opcode	0x9
Address	Start address for programming
Length	Length of area to program in number of bytes

Note:	If the PROGRAM_CLUSTER command fails, the programmer should read the failing row using the READ command from the Flash memory. Next, the programmer should compare the row received from Flash memory to its local copy word-by-word to determine the address where Flash programming fails.

Expected Response (1 word):

FIGURE 16-23: PROGRAM_CLUSTER RESPONSE



16.2.13 GET_DEVICEID COMMAND

The GET_DEVICEID command returns the hardware ID of the device.

FIGURE 16-24: GET_DEVICEID COMMAND

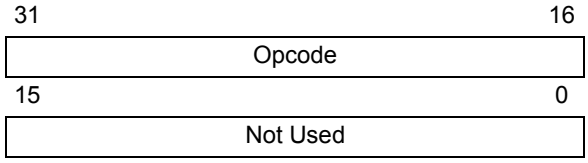
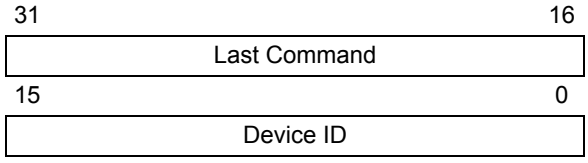


TABLE 16-14: GET_DEVICEID FORMAT

Field	Description
Opcode	0xA

Expected Response (1 word):

FIGURE 16-25: GET_DEVICEID RESPONSE



16.2.14 CHANGE_CFG COMMAND

CHANGE_CFG is used by the probe to set various configuration settings for the PE. Currently, the single configuration setting determines which of the following calculation methods the PE should use:

- Software CRC calculation method
- Hardware calculation method

FIGURE 16-26: CHANGE_CFG COMMAND

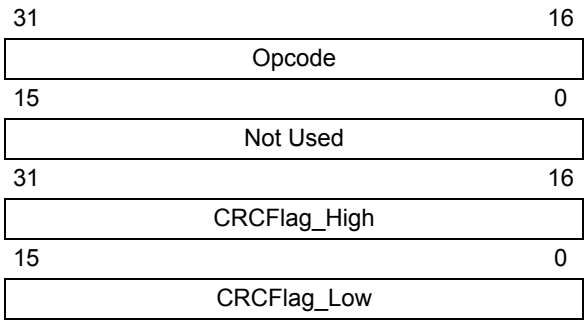
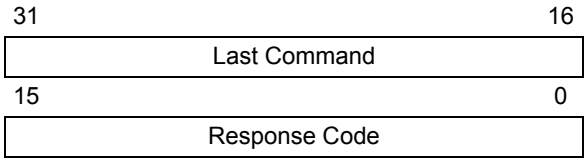


TABLE 16-15: CHANGE_CFG FORMAT

Field	Description
Opcode	0xB
CRCFlag	If the value is '0', the PE uses the software CRC calculation method. If the value is '1', the PE uses the hardware CRC unit to calculate the CRC.

Expected Response (1 word):

FIGURE 16-27: CHANGE_CFG RESPONSE



Note:	The command, CHANGE_CFG, is not available in PIC32MX1XX/2XX devices since only software CRC is available.

17.0 CHECKSUM

17.1 Theory

The checksum is calculated as the 32-bit summation of all bytes (8-bit quantities) in program Flash, boot Flash (except device Configuration Words), the Device ID register with applicable mask, and the device Configuration Words with applicable masks. Next, the 2's complement of the summation is calculated. This final 32-bit number is presented as the checksum.

17.2 Mask Values

The mask value of a device Configuration is calculated by setting all the unimplemented bits to '0' and all the implemented bits to '1'.

For example, Register 17-1 shows the DEVCFG0 register of the PIC32MX360F512L device. The mask value for this register is:

`mask_value_devcfg0` = 0x110FF00B

Table 17-1 lists the mask values of the four device Configuration registers and Device ID registers to be used in the checksum calculations.

REGISTER 17-1: DEVCFG0 REGISTER OF PIC32MX360F512L

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	r-0	r-1	r-1	R/P-1	r-1	r-1	r-1	R/P-1
	—	—	—	CP	—	—	—	BWP
23:16	r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1
	—	—	—	—	PWP19	PWP18	PWP17	PWP16
15:8	R/P-1	R/P-1	R/P-1	R/P-1	r-1	r-1	r-1	r-1
	PWP15	PWP14	PWP13	PWP12	—	—	—	—
7:0	r-1	r-1	r-1	r-1	R/P-1	r-1	R/P-1	R/P-1
	—	—	—	—	ICESEL	—	DEBUG<1:0>	

Legend:	P = Programmable bit	r = Reserved bit	
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

TABLE 17-1: DEVICE CONFIGURATION REGISTER MASK VALUES OF CURRENTLY SUPPORTED PIC32 DEVICES

Device	DEVCFG0	DEVCFG1	DEVCFG2	DEVCFG3	DEVID
All PIC32MX1XX devices	0x1100FC1F	0x03DFF7A7	0x00078777	0xF0C0FFFF	0x0FFFF000
All PIC32MX2XX devices	0x1100FC1F	0x03DFF7A7	0x00078777	0xF0C0FFFF	0x0FFFF000
All PIC32MX3XX devices	0x110FF00B	0x009FF7A7	0x00070077	0x0000FFFF	0x000FF000
All PIC32MX4XX devices	0x110FF00B	0x009FF7A7	0x00078777	0x0000FFFF	0x000FF000
PIC32MX534F064H	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x0FFFF000
PIC32MX534F064L	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x0FFFF000
PIC32MX564F064H	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x0FFFF000
PIC32MX564F064L	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x0FFFF000
PIC32MX564F128H	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x0FFFF000
PIC32MX564F128L	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x0FFFF000
PIC32MX575F256H	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x000FF000
PIC32MX575F256L	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x000FF000
PIC32MX575F512H	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x000FF000
PIC32MX575F512L	0x110FF00F	0x009FF7A7	0x00078777	0xC407FFFF	0x000FF000
PIC32MX664F064H	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x0FFFF000
PIC32MX664F064L	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x0FFFF000
PIC32MX664F128H	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x0FFFF000
PIC32MX664F128L	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x0FFFF000
PIC32MX675F256H	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x000FF000
PIC32MX675F256L	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x000FF000
PIC32MX675F512H	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x000FF000
PIC32MX675F512L	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x000FF000
PIC32MX695F512H	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x000FF000
PIC32MX695F512L	0x110FF00F	0x009FF7A7	0x00078777	0xC307FFFF	0x000FF000
PIC32MX764F128H	0x110FF00F	0x009FF7A7	0x00078777	0xC707FFFF	0x0FFFF000
PIC32MX764F128L	0x110FF00F	0x009FF7A7	0x00078777	0xC707FFFF	0x0FFFF000
PIC32MX775F256H	0x110FF00F	0x009FF7A7	0x00078777	0xC707FFFF	0x000FF000
PIC32MX775F256L	0x110FF00F	0x009FF7A7	0x00078777	0xC707FFFF	0x000FF000
PIC32MX775F512H	0x110FF00F	0x009FF7A7	0x00078777	0xC707FFFF	0x000FF000
PIC32MX775F512L	0x110FF00F	0x009FF7A7	0x00078777	0xC707FFFF	0x000FF000
PIC32MX795F512H	0x110FF00F	0x009FF7A7	0x00078777	0xC707FFFF	0x000FF000
PIC32MX795F512L	0x110FF00F	0x009FF7A7	0x00078777	0xC707FFFF	0x000FF000

17.3 Algorithm

An example of a high-level algorithm for calculating the checksum for a PIC32 device is illustrated in Figure 17-1 to demonstrate one method to derive a checksum. This is merely an example of how the actual calculations can be accomplished, the method that is ultimately used is left to the discretion of the software developer.

As stated earlier, the PIC32 checksum is calculated as the 32-bit summation of all bytes (8-bit quantities) in program Flash, boot Flash (except device Configuration Words), the Device ID register with applicable mask, and the device Configuration Words with applicable masks.

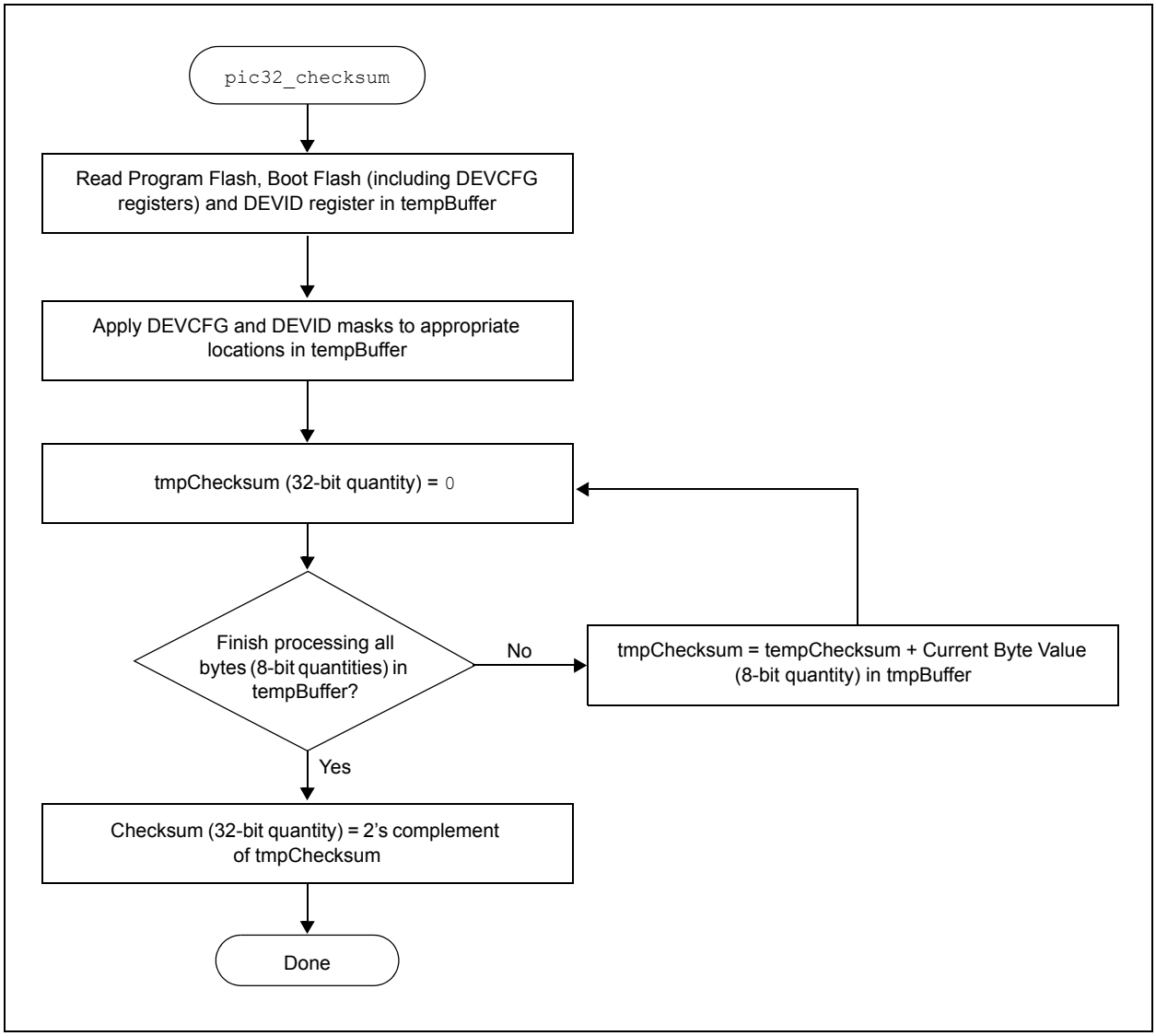
Next, the 2's complement of the summation is calculated. This final 32-bit number is presented as the checksum.

The mask values of the device Configuration and Device ID registers are derived as described in the previous section, **Section 17.2 "Mask Values"**.

Another noteworthy point is that the last four 32-bit quantities in boot Flash are the device Configuration registers. An arithmetic AND operation of these device Configuration register values is performed with the appropriate mask value, before adding their bytes to the checksum.

Similarly, an arithmetic AND operation of the Device ID register is performed with the appropriate mask value, before adding its bytes to the checksum.

FIGURE 17-1: HIGH-LEVEL ALGORITHM FOR CHECKSUM CALCULATION



The formula to calculate for the checksum for a PIC32 device is provided in Equation 17-1.

EQUATION 17-1: CHECKSUM FORMULA

$$Checksum = 2's \text{ complement } (PF + BF + DCR + DIR)$$

Where,

PF = 32-bit summation of all bytes in Program Flash

BF = 32-bit summation of all bytes in Boot Flash, except device Configuration registers

$DCR = \sum_{X=0}^{3}$ 32-bit summation of bytes $(MASK_{DEVCFGX} \& DEVCFGx)$

DIR = 32-bit summation of bytes $(MASK_{DEVID} \& DEVID)$

$MASK_{DEVCFGX}$ = mask value from Table 17-1

$MASK_{DEVID}$ = mask value from Table 17-1

17.4 Example of Checksum Calculation

The following sections 17.4.1- 17.4.5 demonstrate a checksum calculation for the PIC32MX360F512L device using Equation 17-1.

The following assumptions are made for the purpose of this checksum calculation example:

- Program Flash and Boot Flash are in the erased state (all bytes are 0xFF)
- Device Configuration is in the default state of the device (no configuration changes are made)

To begin, each item on the right-hand side of the equation (PF, BF, DCR, DIR) is individually calculated. After those values have been derived, the final value of the checksum can be determined.

17.4.1 CALCULATING FOR “PF” IN THE CHECKSUM FORMULA

The size of Program Flash is 512 KB, which equals 524288 bytes. Since the program Flash is assumed to be in erased state, the value of “PF” is resolved through the following calculation:

PF = 0xFF + 0xFF + … 524288 times

PF = 0x7F80000 (32-bit number)

17.4.2 CALCULATING FOR “BF” IN THE CHECKSUM FORMULA

The size of the Boot Flash is 12 KB, which equals 12288 bytes. However, the last 16 bytes are device Configuration registers, which are treated separately. Therefore, the number of bytes in boot Flash that we consider in this step is 12272. Since the boot Flash is assumed to be in erased state, the value of “BF” is resolved through the following calculation:

BF = 0xFF + 0xFF + … 12272 times

BF = 0x002FC010 (32-bit number)

17.4.3 CALCULATING FOR “DCR” IN THE CHECKSUM FORMULA

Since the device Configuration registers are left in their default state, the value of the appropriate DEVCFG register – as read by the PIC32 core, its respective mask value, the value derived from applying the mask, and the 32-bit summation of bytes (all as shown in Table 17-2) provide the total of the 32-bit summation of bytes.

From Table 17-2, the value of “DCR” is:

DCR = 0x000005D4 (32-bit number)

TABLE 17-2: DCR CALCULATION EXAMPLE

Register	POR Default Value	Mask	POR Default Value & Mask	32-Bit Summation of Bytes
DEVCFG0	0x7FFFFFFF	0x110FF00B	0x110FF00B	0x0000011B
DEVCFG1	0xFFFFFFFF	0x009FF7A7	0x009FF7A7	0x0000023D
DEVCFG2	0xFFFFFFFF	0x00070077	0x00070077	0x0000007E
DEVCFG3	0xFFFFFFFF	0x0000FFFF	0x0000FFFF	0x000001FE
			Total of the 32-bit Summation of Bytes =	**0x000005D4**

17.4.4 CALCULATING FOR “DIR” IN THE CHECKSUM FORMULA

The value of Device ID register, its mask value, the value derived from applying the mask, and the 32-bit summation of bytes are shown in Table 17-3.

From Table 17-3, the value of “DIR” is:

DIR = 0x00000083 (32-bit number.)

TABLE 17-3: DIR CALCULATION EXAMPLE

Register	POR Default Value	Mask	POR Default Value & Mask	32-Bit Summation of Bytes
DEVID	0x00938053	0x000FF000	0x00038000	0x00000083

17.4.5 COMPLETING THE PIC32 CHECKSUM CALCULATION

The values derived in previous sections (PF, BF, DCR, DIR) are used to calculate the checksum value. First, perform the 32-bit summation of the PF, BF, DCR and DIR as derived in previous sections and store it in a variable, called *temp*, as shown in Example 17-1.

EXAMPLE 17-1: CHECKSUM CALCULATION PROCESS

1. First, *temp* = PF + BF + DCR + DIR, which translates to:
   temp = 0x7F80000 + 0x002FC010 + 0x000005D4 + 0x00000083
2. Adding all four values results in *temp* being equal to 0x0827C667
3. Next, the 1’s complement of *temp*, called *temp1*, is calculated:
   temp1 = 1’s complement (*temp*), which is now equal to 0xF7D83998
4. Finally, the 2’s complement of *temp* is the checksum:
   Checksum = 2’s complement (*temp*), which is Checksum = *temp1* + 1, resulting in 0xF7D83999

17.4.6 CHECKSUM VALUES WHILE DEVICE IS CODE-PROTECTED

Since the device Configuration Words are not readable while the PIC32 devices are in code-protected state, the checksum values are zeros for all devices.

18.0 CONFIGURATION MEMORY AND DEVICE ID

PIC32MX devices include several features intended to maximize application flexibility and reliability, and minimize cost through elimination of external components. These features are configurable through specific configuration bits for each device. Refer to the specific device data sheet for a full list of available features and configuration bit settings.

Table 18-1 shows the Device ID register. See Table 18-4 for a full list of Device ID and revision number for specific devices.

TABLE 18-1: DEVID SUMMARY

Virtual Address	Name	Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BF80_F220	DEVID	31:24	VER<3:0>				DEVID<27:24>			
		23:16	DEVID<23:16>							
		15:8	DEVID<15:8>							
		7:0	DEVID<7:0>							

18.1 Device Configuration

In PIC32MX devices, the Configuration Words select various device configurations. These Configuration Words are implemented as volatile memory registers and must be loaded from the nonvolatile programmed Configuration data mapped in the last four words (32-bit x 4 words) of boot Flash memory, DEVCFG0-DEVCFG3. These are the four locations an external programming device programs with the appropriate Configuration data (see Table 18-2 and Table 18-3).

TABLE 18-2: DEVCFG LOCATIONS

Configuration Word	Address
DEVCFG0	0xBFC0_2FFC
DEVCFG1	0xBFC0_2FF8
DEVCFG2	0xBFC0_2FF4
DEVCFG3	0xBFC0_2FF0

TABLE 18-3: DEVCFG LOCATIONS FOR PIC32MX1X0 AND PIC32MX20X DEVICES ONLY

Configuration Word	Address
DEVCFG0	0x1FC0_2FFC
DEVCFG1	0x1FC0_2FF8
DEVCFG2	0x1FC0_2FF4
DEVCFG3	0x1FC0_2FF0

On Power-on Reset (POR), or any Reset, the Configuration Words are copied from the boot Flash memory to their corresponding Configuration registers. A Configuration bit can only be programmed = 0 (unprogrammed state = 1).

During programming, a Configuration Word can be programmed a maximum of two times before a page erase must be performed.

After programming the Configuration Words, the device must be reset to ensure that the Configuration registers are reloaded with the new programmed data.

18.1.1 CONFIGURATION REGISTER PROTECTION

To prevent inadvertent Configuration bit changes during code execution, all programmable Configuration bits are write-once. After a bit is initially programmed during a power cycle, it cannot be written to again. Changing a device configuration requires changing the Configuration data in the boot Flash memory, and cycling power to the device.

To ensure integrity of the 128-bit data, a comparison is made between each Configuration bit and its stored complement continuously. If a mismatch is detected, a Configuration Mismatch Reset is generated, which causes a device Reset.

TABLE 18-4: DEVICE IDs AND REVISION

Device	DEVID Register Value	Revision ID and Silicon Revision
PIC32MX110F016B	0x04A07053	0x0 – A0 Revision
PIC32MX110F016C	0x04A09053	
PIC32MX110F016D	0x04A0B053	
PIC32MX120F032B	0x04A06053	
PIC32MX120F032C	0x04A08053	
PIC32MX120F032D	0x04A0A053	
PIC32MX130F064B	0x04D07053	
PIC32MX130F064C	0x04D09053	
PIC32MX130F064D	0x04D0B053	
PIC32MX150F128B	0x04D06053	
PIC32MX150F128C	0x04D08053	
PIC32MX150F128D	0x04D0A053	
PIC32MX210F016B	0x04A01053	
PIC32MX210F016C	0x04A03053	
PIC32MX210F016D	0x04A05053	
PIC32MX220F032B	0x04A00053	
PIC32MX220F032C	0x04A02053	
PIC32MX220F032D	0x04A04053	
PIC32MX230F064B	0x04D01053	
PIC32MX230F064C	0x04D03053	
PIC32MX230F064D	0x04D05053	
PIC32MX250F128B	0x04D00053	
PIC32MX250F128C	0x04D02053	
PIC32MX250F128D	0x04D04053	
PIC32MX360F512L	0x0938053	0x3 – B2 Revision 0x4 – B3 Revision 0x5 – B4 Revision 0x5 – B6 Revision
PIC32MX360F256L	0x0934053	
PIC32MX340F128L	0x092D053	
PIC32MX320F128L	0x092A053	
PIC32MX340F512H	0x0916053	
PIC32MX340F256H	0x0912053	
PIC32MX340F128H	0x090D053	
PIC32MX320F128H	0x090A053	
PIC32MX320F064H	0x0906053	
PIC32MX320F032H	0x0902053	
PIC32MX460F512L	0x0978053	
PIC32MX460F256L	0x0974053	
PIC32MX440F128L	0x096D053	
PIC32MX440F256H	0x0952053	
PIC32MX440F512H	0x0956053	
PIC32MX440F128H	0x094D053	
PIC32MX420F032H	0x0942053	

TABLE 18-4: DEVICE IDs AND REVISION (CONTINUED)

Device	DEVID Register Value	Revision ID and Silicon Revision
PIC32MX795F512L	0x4307053	0x0 – A0 Revision 0x1 – A1 Revision
PIC32MX795F512H	0x430E053	
PIC32MX775F512L	0x4306053	
PIC32MX775F512H	0x430D053	
PIC32MX775F256L	0x4312053	
PIC32MX775F256H	0x4303053	
PIC32MX764F128L	0x4417053	
PIC32MX764F128H	0x440B053	
PIC32MX695F512L	0x4341053	
PIC32MX695F512H	0x4325053	
PIC32MX675F512L	0x4311053	
PIC32MX675F512H	0x430C053	
PIC32MX675F256L	0x4305053	
PIC32MX675F256H	0x430B053	
PIC32MX664F128L	0x4413053	
PIC32MX664F128H	0x4407053	
PIC32MX664F064L	0x4411053	
PIC32MX664F064H	0x4405053	
PIC32MX575F512L	0x430F053	
PIC32MX575F512H	0x4309053	
PIC32MX575F256L	0x4333053	
PIC32MX575F256H	0x4317053	
PIC32MX564F128L	0x440F053	
PIC32MX564F128H	0x4403053	
PIC32MX564F064L	0x440D053	
PIC32MX564F064H	0x4401053	
PIC32MX534F064H	0x4400053	
PIC32MX534F064L	0x440C053	

18.2 Device Code-Protection Bit (CP)

The PIC32MX features a single device Code-Protection bit (CP). CP, when programmed = 0, protects boot Flash and program Flash from being read or modified by an external programming device. When code-protection is enabled, only the Device ID and User ID registers are available to be read by an external programmer. However, Boot Flash and program Flash memory are not protected from self-programming during program execution when code-protection is enabled.

18.3 Program Write-Protection Bits (PWP)

In addition to a device Code-Protection bit, the PIC32MX also features Program Write-Protection bits (PWP) to prevent boot Flash and program Flash memory regions from being written during code execution.

Boot Flash memory is write-protected with a single Configuration bit, BWP (DEVCFG0<24>), when programmed = 0.

Program Flash memory can be write-protected entirely or in selectable page sizes using Configuration bits PWP<7:0> (BCFG0<19:12>). A page of program Flash memory is 4096 bytes (1024 words) or 1024 bytes (256 words). The PWP bits represent the 1's complement of the number of protected pages. For example, programming PWP bits = 0xFF selects 0 pages to be write-protected, effectively disabling the program Flash write protection. Programming PWP bits = 0xFE selects the first page to be write-protected. When enabled, the write-protected memory range is inclusive from the beginning of program Flash memory (0xBD00_0000) up through the selected page.

Note:	The PWP bits represent the 1's complement of the number of protected pages.

The amount of program Flash memory available for write protection depends on the family device variant.

19.0 TAP CONTROLLERS

TABLE 19-1: MCHP TAP INSTRUCTIONS

Command	Value	Description
MTAP_COMMAND	5'h07	TDI and TDO connected to MCHP Command Shift register (See Table 19-2).
MTAP_SW_MTAP	5'h04	Switch TAP controller to MCHP TAP controller.
MTAP_SW_ETAP	5'h05	Switch TAP controller to EJTAG TAP controller.
MTAP_IDCODE	5'h01	Select Chip Identification Data register.

19.1 Microchip TAP Controllers (MTAP)

19.1.1 MTAP_COMMAND INSTRUCTION

MTAP_COMMAND selects the MCHP Command Shift register. See Table 19-2 for available commands.

19.1.1.1 MCHP_STATUS INSTRUCTION

MCHP_STATUS returns the 8-bit Status value of the Microchip TAP controller. Table 19-3 shows the format of the Status value returned.

19.1.1.2 MCHP_ASERT_RST INSTRUCTION

MCHP_ASERT_RST performs a persistent device Reset. It is similar to asserting and holding MCLR with the exception that test modes are not detected. Its associated Status bit is DEVRST.

19.1.1.3 MCHP_DE_ASERT_RST INSTRUCTION

MCHP_DE_ASERT_RST removes the persistent device Reset. It is similar to de-asserting MCLR. Its associated Status bit is DEVRST.

19.1.1.4 MCHP_ERASE INSTRUCTION

MCHP_ERASE performs a Chip Erase. The CHIP_ERASE command sets an internal bit that requests the Flash Controller to perform the erase. Once the controller becomes busy, as indicated by FCBUSY (Status bit), the internal bit is cleared.

19.1.1.5 MCHP_FLASH_ENABLE INSTRUCTION

MCHP_FLASH_ENABLE sets the FAEN bit, which controls processor accesses to the Flash memory. The FAEN bit's state is returned in the field of the same name. This command has no effect if CPS = 0. This command requires a NOP to complete.

19.1.1.6 MCHP_FLASH_DISABLE INSTRUCTION

MCHP_FLASH_DISABLE clears the FAEN bit which controls processor accesses to the Flash memory. The FAEN bit's state is returned in the field of the same name. This command has no effect if CPS = 0. This command requires a NOP to complete.

19.1.2 MTAP_SW_MTAP INSTRUCTION

MTAP_SW_MTAP switches the TAP instruction set to the MCHP TAP instruction set.

19.1.3 MTAP_SW_ETAP INSTRUCTION

MTAP_SW_ETAP effectively switches the TAP instruction set to the EJTAG TAP instruction set. It does this by holding the EJTAG TAP controller in the Run Test/Idle state until a MTAP_SW_ETAP instruction is decoded by the MCHP TAP controller.

19.1.4 MTAP_IDCODE INSTRUCTION

MTAP_IDCODE returns the value stored in the DEVID register.

TABLE 19-2: MTAP_COMMAND DR COMMANDS

Command	Value	Description
MCHP_STATUS	8'h00	NOP and return Status.
MCHP_ASERT_RST	8'hD1	Requests the reset controller to assert device Reset.
MCHP_DE_ASERT_RST	8'hD0	Removes the request for device Reset, which causes the reset controller to de-assert device Reset if there is no other source requesting Reset (i.e., MCLR).
MCHP_ERASE	8'hFC	Cause the Flash controller to perform a Chip Erase.
MCHP_FLASH_ENABLE	8'hFE	Enables fetches and loads to the Flash (from the processor).
MCHP_FLASH_DISABLE	8'hFD	Disables fetches and loads to the Flash (from the processor).

TABLE 19-3: MCHP STATUS VALUE

Bit Range	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
7:0	CPS	0	NVMERR(1)	0	CFGRDY	FCBUSY	FAEN	DEVRST

bit 7 **CPS:** Code-Protect State bit
1 = Device is not code-protected
0 = Device is code-protected

bit 6 **Unimplemented:** Read as '0'

bit 5 **NVMERR:** NVMCON Status bit(1)
1 = An Error occurred during NVM operation
0 = An Error did not occur during NVM operation

bit 4 **Unimplemented:** Read as '0'

bit 3 **CFGRDY:** Code-Protect State bit
1 = Configuration has been read and CP is valid
0 = Configuration has not been read

bit 2 **FCBUSY:** Flash Controller Busy bit
1 = Flash controller is busy (Erase is in progress)
0 = Flash controller is not busy (either erase has not started or it has finished)

bit 1 **FAEN:** Flash Access Enable bit
This bit reflects the state of CFGCON.FAEN.
1 = Flash access is enabled
0 = Flash access is disabled (i.e., processor accesses are blocked)

bit 0 **DEVRST:** Device Reset State bit
1 = Device Reset is active
0 = Device Reset is not active

Note 1: This bit is available in PIC32MX1X0 and PIC32MX2X0 devices only.

TABLE 19-4: EJTAG TAP INSTRUCTIONS

Command	Value	Description
ETAP_ADDRESS	5'h08	Select Address register.
ETAP_DATA	5'h09	Select Data register.
ETAP_CONTROL	5'h0A	Select EJTAG Control register.
ETAP_EJTAGBOOT	5'h0C	Set EjtagBrk, ProbEn and ProbTrap to '1' as Reset value.
ETAP_FASTDATA	5'h0E	Selects the Data and Fastdata registers.

19.2 EJTAG TAP Controller

19.2.1 ETAP_ADDRESS COMMAND

ETAP_ADDRESS selects the Address register. The read-only Address register provides the address for a processor access. The value read in the register is valid if a processor access is pending, otherwise the value is undefined.

The two or three Least Significant Bytes (LSBs) of the register are used with the Psz field from the EJTAG Control register to indicate the size and data position of the pending processor access transfer. These bits are not taken directly from the address referenced by the load/store.

19.2.2 ETAP_DATA COMMAND

ETAP_DATA selects the Data register. The read/write Data register is used for opcode and data transfers during processor accesses. The value read in the Data register is valid only if a processor access for a write is pending, in which case the Data register holds the store value. The value written to the Data register is only used if a processor access for a pending read is finished afterwards; in which case, the data value written is the value for the fetch or load. This behavior implies that the Data register is not a memory location where a previously written value can be read afterwards.

19.2.3 ETAP_CONTROL COMMAND

ETAP_CONTROL selects the Control register. The EJTAG Control register (ECR) handles processor Reset and soft Reset indication, Debug mode indication, access start, finish and size, and read/write indication. The ECR also provides the following features:

- Controls debug vector location and indication of serviced processor accesses
- Allows a debug interrupt request
- Indicates processor Low-Power mode
- Allows implementation-dependent processor and peripheral Resets

The EJTAG Control register is not updated/written in the Update-DR state unless the Reset occurred; that is ROCC (bit 31) is either already '0' or is written to '0' at the same time. This condition ensures proper handling of processor accesses after a Reset.

Reset of the processor can be indicated through the ROCC bit in the TCK domain a number of TCK cycles after it is removed in the processor clock domain in order to allow for proper synchronization between the two clock domains.

Bits that are R/W in the register return their written value on a subsequent read, unless other behavior is defined.

Internal synchronization ensures that a written value is updated for reading immediately afterwards, even when the TAP controller takes the shortest path from the Update-DR to Capture-DR state.

19.2.4 ETAP_EJTAGBOOT COMMAND

The Reset value of the EjtagBrk, ProbTrap and ProbEn bits follows the setting of the internal EJTAGBOOT indication.

If the EJTAGBOOT instruction has been given, and the internal EJTAGBOOT indication is active, then the Reset value of the three bits is set (1), otherwise the Reset value is clear (0).

The results of setting these bits are:

- Setting the EjtagBrk causes a Debug interrupt exception to be requested right after the processor Reset from the EJTAGBOOT instruction
- The debug handler is executed from the EJTAG memory because ProbTrap is set to indicate debug vector in EJTAG memory at 0x FF20 0200
- Service of the processor access is indicated because ProbEn is set

Therefore, it is possible to execute the debug handler right after a processor Reset from the EJTAGBOOT instruction, without executing any instructions from the normal Reset handler.

19.2.5 ETAP_FASTDATA COMMAND

The width of the Fastdata register is 1 bit. During a fast data access, the Fastdata register is written and read (i.e., a bit is shifted in and a bit is shifted out). During a fast data access, the Fastdata register value shifted in specifies whether the fast data access should be completed or not. The value shifted out is a flag that indicates whether the fast data access was successful or not (if completion was requested). The FASTDATA access is used for efficient block transfers between the DMSEG segment (on the probe) and target memory (on the processor). An “upload” is defined as a sequence that the processor loads from target memory and stores to the DMSEG segment. A “download” is a sequence of processor loads from the DMSEG segment and stores to target memory. The “Fastdata area” specifies the legal range of DMSEG segment addresses (0xFF20.0000-0xFF20.000F) that can be used for uploads and downloads. The Data and Fastdata registers (selected with the FASTDATA instruction) allow efficient completion of pending Fastdata area accesses.

During Fastdata uploads and downloads, the processor will stall on accesses to the Fastdata area. The PrAcc (processor access pending bit) will be 1 indicating the probe is required to complete the access. Both upload and download accesses are attempted by shifting in a zero SPrAcc value (to request access completion) and shifting out SPrAcc to see if the attempt will be successful (i.e., there was an access pending and a legal Fastdata area address was used).

Downloads will also shift in the data to be used to satisfy the load from the DMSEG segment Fastdata area, while uploads will shift out the data being stored to the DMSEG segment Fastdata area.

As noted above, two conditions must be true for the Fastdata access to succeed. These are:

- PrAcc must be 1 (i.e., there must be a pending processor access).
- The Fastdata operation must use a valid Fastdata area address in the DMSEG segment (0xFF20.0000 to 0xFF20.000F).

20.0 AC/DC CHARACTERISTICS AND TIMING REQUIREMENTS

TABLE 20-1: AC/DC CHARACTERISTICS AND TIMING REQUIREMENTS

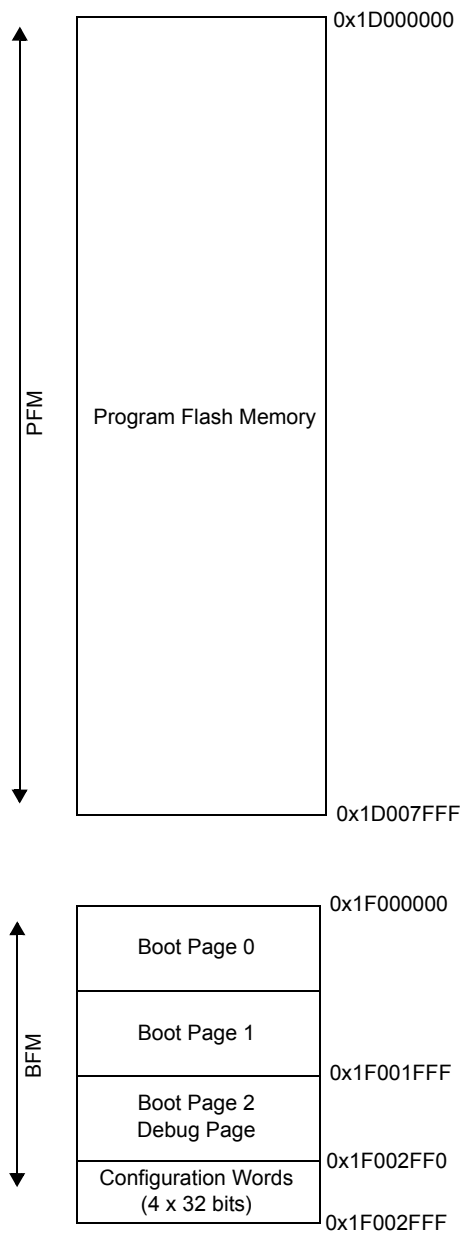
Standard Operating Conditions Operating Temperature: 0ºC to +70ºC. Programming at +25ºC is recommended.						
Param. No.	**Symbol**	**Characteristic**	**Min.**	**Max.**	**Units**	**Conditions**
D111	VDD	Supply Voltage During Programming	2.3V	3.60	V	Normal programming(1,2)
D112	IPP	Programming Current on $\overline{MCLR}$	—	5	µA	—
D113	IDDP	Supply Current During Programming	—	40	mA	—
D114	IPEAK	Instantaneous Peak Current During Start-up	—	100	mA	—
D031	VIL	Input Low Voltage	VSS	0.2 VDD	V	—
D041	VIH	Input High Voltage	0.8 VDD	VDD	V	—
D080	VOL	Output Low Voltage	—	0.4	V	IOL = 8.5 mA @ 3.6V
D090	VOH	Output High Voltage	1.4	—	V	IOH = -3.0 mA @ 3.6V
D012	CIO	Capacitive Loading on I/O pin (PGDx)	—	50	pF	To meet AC specifications
D013	CF	Filter Capacitor Value on VCAP	1	10	µF	—
P1	TPGC	Serial Clock (PGCx) Period	100	—	ns	—
P1A	TPGCL	Serial Clock (PGCx) Low Time	40	—	ns	—
P1B	TPGCH	Serial Clock (PGCx) High Time	40	—	ns	—
P6	TSET2	VDD ↑ Setup Time to $\overline{MCLR}$ ↑	100	—	ns	—
P7	THLD2	Input Data Hold Time from $\overline{MCLR}$ ↑	500	—	ns	—
P9A	TDLY4	PE Command Processing Time	40	—	µs	—
P9B	TDLY5	Delay between PGDx ↓by the PE to PGDx Released by the PE	15	—	µs	—
P11	TDLY7	Chip Erase Time	80	—	ms	—
P12	TDLY8	Page Erase Time	20	—	ms	—
P13	TDLY9	Row Programming Time	2	—	ms	—
P14	TR	$\overline{MCLR}$ Rise Time to Enter ICSP™ mode	—	1.0	µs	—
P15	TVALID	Data Out Valid from PGCx ↑	10	—	ns	—
P16	TDLY8	Delay between Last PGCx ↓and $\overline{MCLR}$ ↓	0	—	s	—
P17	THLD3	$\overline{MCLR}$ ↓to VDD ↓	—	100	ns	—
P18	TKEY1	Delay from First $\overline{MCLR}$ ↓to First PGCx ↑ for Key Sequence on PGDx	40	—	ns	—
P19	TKEY2	Delay from Last PGCx ↓for Key Sequence on PGDx to Second $\overline{MCLR}$ ↑	40	—	ns	—
P20	TMCLRH	$\overline{MCLR}$ High time	—	500	µs	—

Note 1: See **Section 4.3 "Power Requirements"** for more information.

2: VDD must also be supplied to the AVDD pins during programming. AVDD and AVSS should always be within ±0.3V of VDD and VSS, respectively.

APPENDIX A: PIC32MX FLASH MEMORY MAP

FIGURE A-1: FLASH MEMORY MAP



Note: The memory map shown is for reference only. Refer to the specific device data sheet for the memory map for your device.

APPENDIX B: HEX FILE FORMAT

Flash programmers process the standard HEX format used by the Microchip development tools. The format supported is the Intel® HEX32 Format (INHX32). Please refer to Appendix A in the *"MPASM Users Guide"* (DS33014) for more information about hex file formats.

The basic format of the hex file is:

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

Each data record begins with a 9-character prefix and always ends with a 2-character checksum. All records begin with '`:`', regardless of the format. The individual elements are described below.

- `BB` - is a two-digit hexadecimal byte count representing the number of data bytes that appear on the line. Divide this number by two to get the number of words per line.
- `AAAA` - is a four-digit hexadecimal address representing the starting address of the data record. Format is high byte first followed by low byte. The address is doubled because this format only supports 8 bits. Divide the value by two to find the real device address.
- `TT` - is a two-digit record type that will be '00' for data records, '01' for end-of-file records and '04' for extended-address record.
- `HHHH` - is a four-digit hexadecimal data word. Format is low byte followed by high byte. There will be `BB`/2 data words following `TT`.
- `CC` - is a two-digit hexadecimal checksum that is the 2's complement of the sum of all the preceding bytes in the line record.

Because the Intel hex file format is byte-oriented, and the 16-bit program counter is not, program memory sections require special treatment. Each 24-bit program word is extended to 32 bits by inserting a so-called "phantom byte". Each program memory address is multiplied by 2 to yield a byte address.

As an example, a section that is located at 0x100 in program memory will be represented in the hex file as 0x200.

The hex file will be produced with the following contents:

```
:020000040000fa
:040200003322110096
:00000001FF
```

Notice that the data record (line 2) has a load address of 0200, while the source code specified address 0x100. Note also that the data is represented in "little-endian" format, meaning the Least Significant Byte appears first. The phantom byte appears last, just before the checksum.

APPENDIX C: REVISION HISTORY

Revision A (August 2007)

This is the initial released version of this document.

Revision B (February 2008)

Update records for this revision are not available.

Revision C (April 2008)

Update records for this revision are not available.

Revision D (May 2008)

Update records for this revision are not available.

Revision E (July 2009)

This version of the document includes the following additions and updates:

- Minor changes to style and formatting have been incorporated throughout the document
- Added the following devices:
  - PIC32MX565F256H
  - PIC32MX575F512H
  - PIC32MX675F512H
  - PIC32MX795F512H
  - PIC32MX575F512L
  - PIC32MX675F512L
  - PIC32MX795F512L
- Updated $\overline{\text{MCLR}}$ pulse line to show active-high (P20) in Figure 7-1
- Updated Step 7 of Table 11-1 to clarify repeat of the *last* instruction in the step
- The following instructions in Table 13-1 were updated:
  - Seventh, ninth and eleventh instructions in Step 1
  - All instructions in Step 2
  - First instruction in Step 3
  - Third instruction in Step 4
- Added the following devices to Table 17-1:
  - PIC32MX565F256H
  - PIC32MX575F512H
  - PIC32MX575F512L
  - PIC32MX675F512H
  - PIC32MX675F512L
  - PIC32MX795F512H
  - PIC32MX795F512L
- Updated address values in Table 17-2

Revision E (July 2009) (Continued)

- Added the following devices to Table 17-5:
  - PIC32MX565F256H
  - PIC32MX575F512H
  - PIC32MX675F512H
  - PIC32MX795F512H
  - PIC32MX575F512L
  - PIC32MX675F512L
  - PIC32MX795F512L
- Added Notes 1-3 and the following bits to the DEVCFG - Device Configuration Word Summary and the DEVCFG3: Device Configuration Word 3 (see Table 18-1 and Register):
  - FVBUSIO
  - FUSBIDIO
  - FCANIO
  - FETHIO
  - FMIIEN
  - FPBDIV<1:0>
  - FJTAGEN
- Updated the DEVID Summary (see Table 18-1)
- Updated ICESEL bit description and added the FJTAGEN bit in DEVCFG0: Device Configuration Word 0 (see Register 16-1)
- Updated DEVID: Device and Revision ID register
- Added Device IDs and Revision table (Table 18-4)
- Added $\overline{\text{MCLR}}$ High Time (parameter P20) to Table 20-1
- Added **Appendix B: "Hex File Format"** and **Appendix D: "Revision History"**

Revision F (April 2010)

This version of the document includes the following additions and updates:

- The following global bit name changes were made:
  - NVMWR renamed as WR
  - NVMWREN renamed as WREN
  - NVMERR renamed as WRERR
  - FVBUSIO renamed as FVBUSONIO
  - FUPLLEN renamed as UPLLEN
  - FUPLLIDIV renamed as UPLLIDIV
  - POSCMD renamed as POSCMOD
- Updated the PIC32MX family data sheet references in the fourth paragraph of **Section 2.0 "Programming Overview"**
- Updated the note in **Section 5.2.2 "2-Phase ICSP"**
- Updated the Initiate Flash Row Write Opcodes and instructions (see steps 4, 5 and 6 in Table 13-1)

Revision F (April 2010) (Continued)

- Added the following devices:
  - PIC32MX534F064H
  - PIC32MX534F064L
  - PIC32MX564F064H
  - PIC32MX564F064L
  - PIC32MX564F128H
  - PIC32MX564F128L
  - PIC32MX575F256L
  - PIC32MX664F064H
  - PIC32MX664F064L
  - PIC32MX664F128H
  - PIC32MX664F128L
  - PIC32MX675F256H
  - PIC32MX675F256L
  - PIC32MX695F512H
  - PIC32MX605F512L
  - PIC32MX764F128H
  - PIC32MX764F128L
  - PIC32MX775F256H
  - PIC32MX775F256L
  - PIC32MX775F512H
  - PIC32MX775F512L

Revision G (August 2010)

This revision of the document includes the following updates:

- Updated Step 3 in Table 11-1: Download the PE
- Minor corrections to formatting and text have been incorporated throughout the document

Revision H (April 2011)

This version of the document includes the following additions and updates:

- Updates to formatting and minor typographical changes have been incorporated throughout the document
- The following devices were added:
  - PIC32MX110F016B
  - PIC32MX110F016C
  - PIC32MX110F016D
  - PIC32MX120F032B
  - PIC32MX120F032C
  - PIC32MX120F032D
  - PIC32MX210F016B
  - PIC32MX210F016C
  - PIC32MX210F016D
  - PIC32MX220F032B
  - PIC32MX220F032C
  - PIC32MX220F032D
- The following rows were added to Table 17-1:
  - PIC32MX1X0
  - PIC32MX2X0
- Added a new sub section **Section 17.4.6 “Checksum Values While Device Is Code-Protected”**
- Removed Register 18-1 through Register 18-5.
- Removed Table 17-2
- Removed Section 17.5 “Checksum for PIC32 Devices” and its sub sections
- The Flash Program Memory Write-Protect Ranges table was removed (formerly Table 18-4)
- Added DEVCFG Locations for PIC32MX1X0 and PIC32MX20X Devices Only (see Table 18-3)
- In **Section 18.0 “Configuration Memory and Device ID”**, removed Table 18-1 and updated Table 18-2: DEVID Summary as Table 18-1
- Added the NVMERR bit to the MCHP Status Value table (see Table 19-3)
- The following Silicon Revision and Revision ID are added to Table 18-4:
  - 0x5 - B6 Revision
  - 0x1 - A1 Revision
- Added a note to the Flash Memory Map (see Figure A-1)
- Added **Appendix C: “Flash Program Memory Data Sheet Clarification”**

Revision J (August 2011)

Note: The revision history in this document intentionally skips from Revision H to Revision J to avoid confusing the uppercase letter "I" (EY) with the lowercase letter "l" (EL).

This revision includes the following updates:

- All occurences of VCORE/VCAP have been changed to VCAP
- Updated the fourth paragraph of **Section 2.0 "Programming Overview"**
- Removed the column, Programmer Pin Name, from the 2-Wire Interface Pins table and updated the Pin Type for MCLR (see Table 4-2)
- Added the following new devices to the Code Memory Size table (see Table 5-1) and the Device IDs and Revision table (see Table 18-4):
  - PIC32MX130F064B
  - PIC32MX130F064C
  - PIC32MX130F064D
  - PIC32MX150F128B
  - PIC32MX150F128C
  - PIC32MX150F128D
  - PIC32MX230F064B
  - PIC32MX230F064C
  - PIC32MX230F064D
  - PIC32MX250F128B
  - PIC32MX250F128C
  - PIC32MX250F128D
- Added Row Size and Page Size columns to the Code Memory Size table (see Table 5-1)
- Updated the PGCx signal in Entering Enhanced ICSP Mode (see Figure 7-1)
- Updated the Erase Device block diagram (see Figure 9-1)
- Added a new step 4 to the process to erase a target device in **Section 9.0 "Erasing the Device"**
- Updated the MCLR signal in 2-Wire Exit Test Mode (see Figure 15-2)
- Updated the PE Command Set with the following commands and modified Note 2 (see Table 16-2):
  - PROGRAM_CLUSTER
  - GET_DEVICEID
  - CHANGE_CFG
- Added a second note to **Section 16.2.11 "GET_CRC Command"**
- Updated the Address and Length descriptions in the PROGRAM_CLUSTER Format (see Table 16-13)
- Added a note after the CHANGE_CFG Response (see Figure 16-27)
- Updated the DEVCFG0 and DEVCFG1 values for All PIC32MX1XX and All PIC32MX2XX devices in Table 17-1
- The following changes were made to the AC/DC Characteristics and Timing Requirements (Table 20-1):
  - Updated the Minimum value for parameter D111 (VDD)
  - Added parameter D114 (IPEAK)
  - Removed parameters P2, P3, P4, P4A, P5, P8 and P10
- Removed Appendix C: "Flash Program Memory Data Sheet Clarification"
- Minor updates to text and formatting were incorporated throughout the document

NOTES:

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.

Trademarks

ISBN: 978-1-61341-544-3

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2009

Microchip received ISO/TS-16949:2009 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://www.microchip.com/support
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

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

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: 86-10-8569-7000
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 - Hangzhou
Tel: 86-571-2819-3187
Fax: 86-571-2819-3189

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: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-330-9305

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

08/02/11