PIC32MM FAMILIES

PIC32MM Families Flash Programming Specification

1.0 DEVICE OVERVIEW

This document defines the programming specification for the PIC32MM families of 32-bit microcontrollers. This programming specification is designed to guide developers of external programmer tools. Customers who are developing applications for PIC32MM devices should use development tools that already provide support for device programming.

This document includes programming specifications for the following devices:

- PIC32MM0016GPL020
- PIC32MM0032GPL020
- PIC32MM0064GPL020
- PIC32MM0016GPL028
- PIC32MM0032GPL028
- PIC32MM0064GPL028
- PIC32MM0016GPL036
- PIC32MM0032GPL036
- PIC32MM0064GPL036
- PIC32MM0064GPM028
- PIC32MM0128GPM028
- PIC32MM0256GPM028
- PIC32MM0064GPM036
- PIC32MM0128GPM036
- PIC32MM0256GPM036
- PIC32MM0064GPM048
- PIC32MM0128GPM048
- PIC32MM0256GPM048
- PIC32MM0064GPM064
- PIC32MM0128GPM064
- PIC32MM0256GPM064

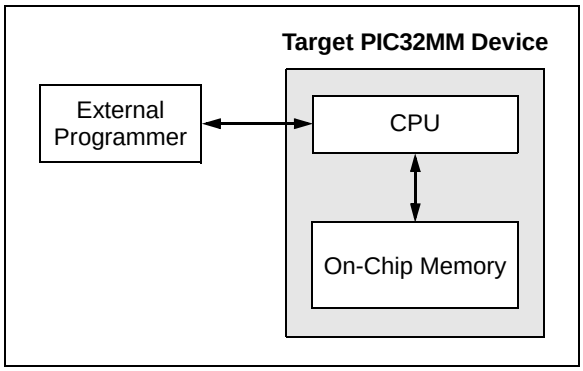
The major topics of discussion include:

2.0 PROGRAMMING OVERVIEW

When in development of a programming tool, it is necessary to understand the internal Flash program operations of the target device and the Special Function Registers (SFRs) used to control Flash programming, as these same operations and registers are used by an external programming tool and its software. These operations and control registers are described in the **“Flash Program Memory”** chapter in the specific device data sheet, and the related section in the *“PIC32 Family Reference Manual”*. It is highly recommended that these documents be used in conjunction with this programming specification.

An external tool programming setup consists of an external programmer tool and a target PIC32MM device. Figure 2-1 illustrates a typical programming setup. The programmer tool is responsible for executing necessary programming steps and completing the programming operation.

FIGURE 2-1: PROGRAMMING SYSTEM SETUP



2.1 Programming Interfaces

All PIC32MM 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 information on programming commands, EJTAG and DC specifications are available in the following sections:

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

2.2 Enhanced JTAG (EJTAG)

The 2-wire ICSP and 4-wire JTAG 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 Imagination Technologies Limited web site (www.imgtec.com) for more information.

2.3 Data Sizes

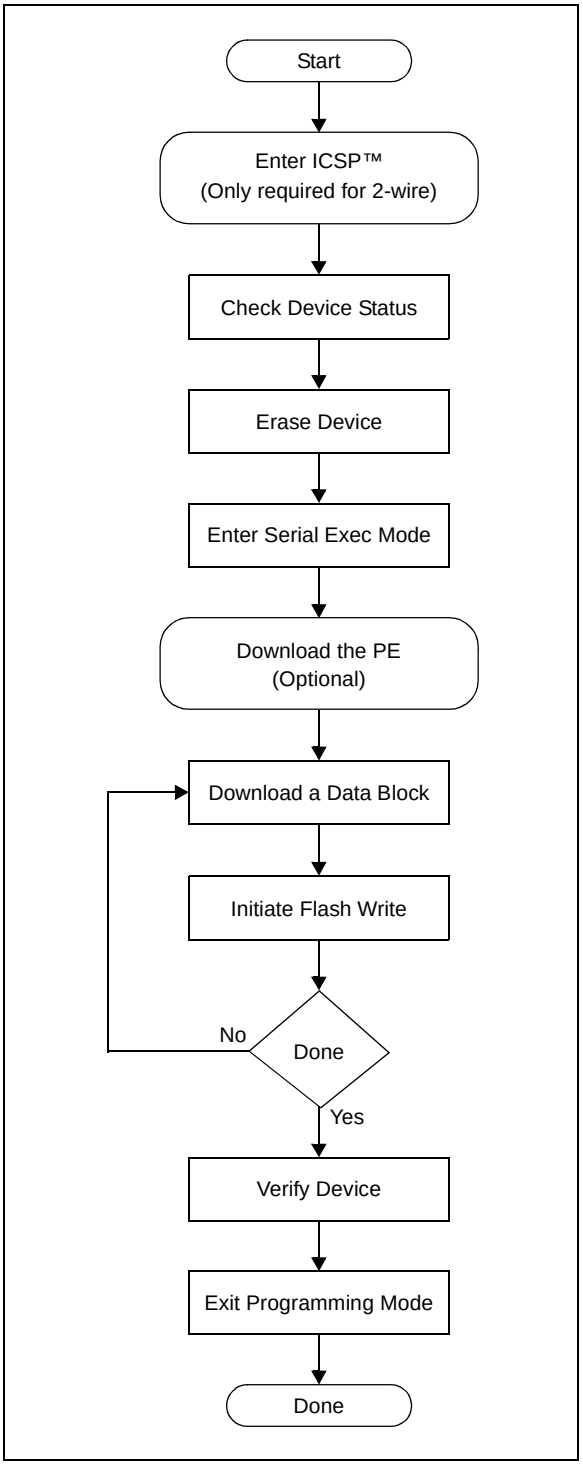
Data sizes are defined as follows:

- Double Word: 64 bits
- One Word: 32 bits
- One-Half Word: 16 bits
- One-Quarter Word: 8 bits
- One Byte: 8 bits

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 PIC32MM devices.

FIGURE 3-1: PROGRAMMING FLOW



The following sequence lists the programming 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 before executing any other steps.

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

   See **Section 7.0 "Entering 2-Wire ICSP 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 PIC32MM 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.

   For more information, see **Section 11.0 "Downloading the Programming Executive (PE)"**.

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.

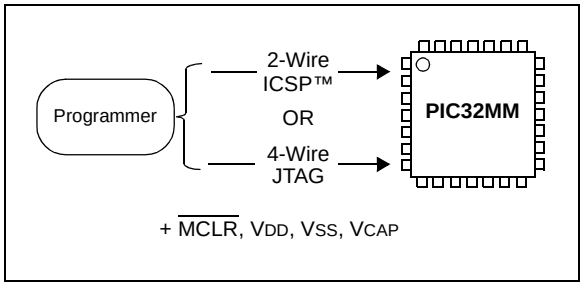
4.0 CONNECTING TO THE DEVICE

The PIC32MM family provides two possible physical interfaces for connecting and programming the memory contents (see Figure 4-1). For all programming interfaces, the target device must be powered and all required signals must be connected. In addition, the interface must be enabled, either through its Configuration bit, as in the case of the JTAG 4-wire interface, or though a special initialization sequence, as is the case of the 2-wire ICSP interface.

The JTAG interface is enabled by default in blank devices shipped from the factory.

Enabling ICSP is described in **Section 7.0 "Entering 2-Wire ICSP Mode"**.

FIGURE 4-1: PROGRAMMING INTERFACES



4.1 4-Wire JTAG 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 PIC32MM device being programmed:

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

Refer to the specific device data sheet for the connection of the signals to the device 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 Register (IR) or selected Data registers. 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 Register or selected Data register. 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 Register or Data registers. This signal changes on the falling edge of TCK. TDO is only driven when data is shifted out; otherwise, TDO is tri-stated.

TABLE 4-1: 4-WIRE INTERFACE PINS

Device Pin Name	Pin Type	Pin Description
MCLR	I	Programming Enable
VDD and AVDD(1)	P	Power Supply
VSS and AVSS(1)	P	Ground
VCAP	P	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 ICSP Interface

Another possible interface is the 2-wire ICSP port. Table 4-2 lists the required pin connections. This interface uses the following two communication lines to transfer data to and from the PIC32MM device being programmed:

- Serial Program Clock (PGECx)
- Serial Program Data (PGEDx)

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 (PGECx)

PGECx is the clock that controls the updating of the TAP controller and the shifting of data through the Instruction Register or selected Data registers. PGECx is independent of the processor clock with respect to both frequency and phase.

4.2.2 SERIAL PROGRAM DATA (PGEDx)

PGEDx is the data input/output to the Instruction Register or selected Data Registers. It is also the control signal for the TAP controller. This signal is sampled on the falling edge of PGECx for some TAP controller states.

TABLE 4-2: 2-WIRE INTERFACE PINS

Device Pin Name	Programmer Pin Name	Pin Type	Pin Description
$\overline{MCLR}$	$\overline{MCLR}$	P	Programming Enable
V_{DD} and AV_{DD}(1)	V_{DD}	P	Power Supply
V_{SS} and AV_{SS}(1)	V_{SS}	P	Ground
V_{CAP}(2)	N/A	P	Filter Capacitor Connection
PGECx(2)	PGEC	I	Programming Pin Pair: Serial Clock
PGEDx(2)	PGED	I/O	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 (AV_{DD}) and ground (AV_{SS}).

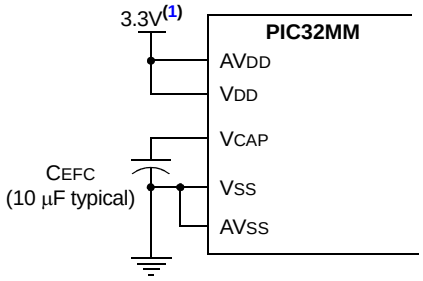
2: Any pair of programming pins available on a particular device may be used, however, they must be used as a pair (PGED1 must be used with PGEC1, and so on).

4.3 Power Requirements

Devices in the PIC32MM family are dual voltage supply designs. There is one supply for the core and another for peripherals and I/O pins. All devices contain an on-chip regulator for the lower voltage core supply to eliminate the need for an additional external regulator. An external capacitor, called VCAP, is required for proper device operation.

Refer to **Section 20.0 "AC/DC Characteristics and Timing Requirements"** and the **"Electrical Characteristics"** chapter(s) in the specific device data sheet for the power requirements for your device.

FIGURE 4-2: POWER CONNECTIONS



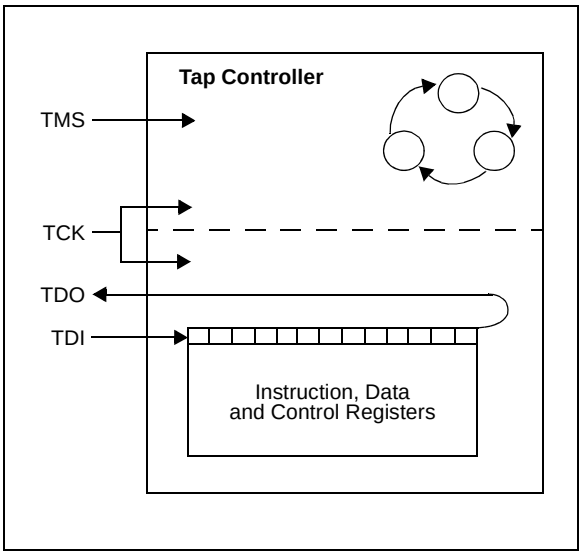
Note 1: These are typical operating voltages. Refer to **Section 20.0 "AC/DC Characteristics and Timing Requirements"** for the full operating ranges of VDD and VCAP.

5.0 4-WIRE JTAG vs. ICSP

Programming is accomplished through 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 for ICSP mode. In both modes, programming the PIC32MM Flash memory is accomplished through the ETAP controller. The TAP controller uses the TMS pin to determine if the Instruction Register or Data registers should be accessed in the shift path between TDI and TDO (see Figure 5-1).

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, and then loaded into the Instruction Register and executed by the CPU. Instructions are fed through the ETAP state machine in 32-bit words.

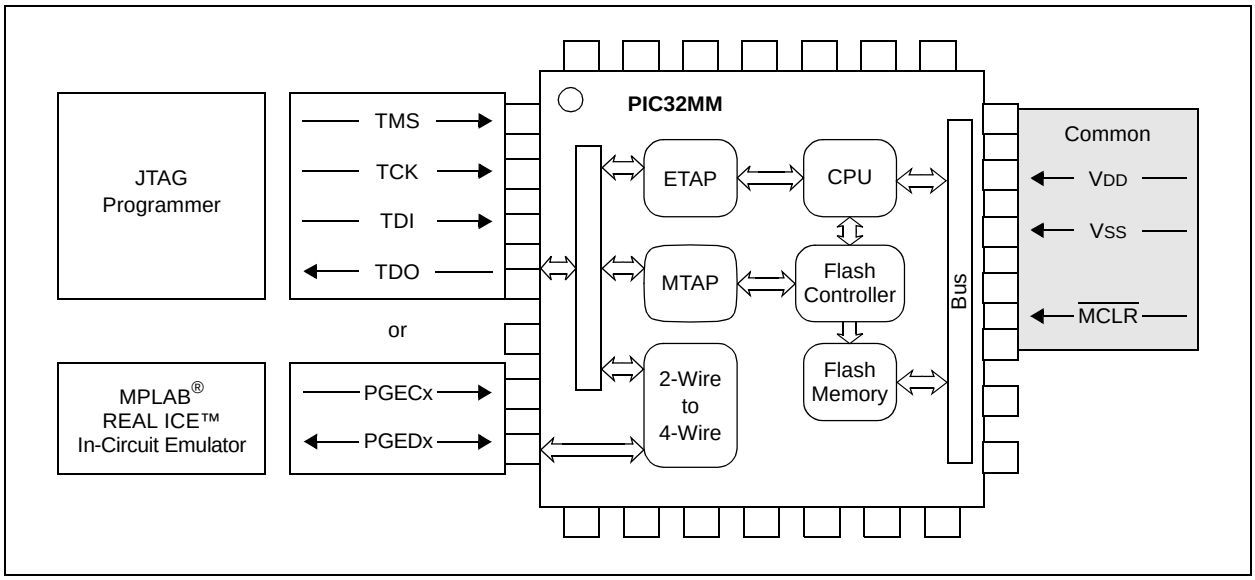
FIGURE 5-1: TAP CONTROLLER



5.1 Programming Interface

Figure 5-2 shows the basic programming interface in PIC32MM devices. Descriptions of each interface block are provided in subsequent sections.

FIGURE 5-2: BASIC PIC32MM PROGRAMMING INTERFACE BLOCK DIAGRAM



5.1.1 ETAP

This block serially feeds instructions and data into the CPU.

5.1.2 MTAP

In addition to the EJTAG TAP (ETAP) controller, the PIC32MM 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 through its Data register.

The programmer sends commands by shifting in the `MTAP_COMMAND` instruction through the `SendCommand` pseudo operation and then sending `MTAP_COMMAND DR` commands through the `XferData` pseudo operation (see Table 19-2 for specific commands).

The probe does not need to issue a `MTAP_COMMAND` instruction for every command shifted into the Data register.

5.1.3 2-WIRE TO 4-WIRE

This block converts the 2-wire ICSP interface to the 4-wire JTAG interface.

5.1.4 CPU

The CPU executes instructions at 8 MHz through the internal oscillator.

5.1.5 FLASH CONTROLLER

The Flash controller controls erasing and programming of the Flash memory on the device.

5.1.6 FLASH MEMORY

The PIC32MM device Flash memory is divided into two logical Flash partitions, consisting of the Boot Flash Memory (BFM) and Program Flash Memory (PFM). The BFM begins at address, 0x1FC00000, and the PFM begins at address, 0x1D000000. Each Flash partition is divided into pages, which represent the smallest block of memory that can be erased. Depending on the device, page sizes are 256 words (1024 bytes), 512 words (2048 bytes), 1024 words (4096 bytes) or 4096 words (16,384 bytes). Row size indicates the number of words that are programmed with the Row Program command. There are typically 8 rows within a page; therefore, devices with 256, 512, 1024 and 4096 word page sizes have 32, 64, 128 and 512 word row sizes, respectively. Table 5-1 shows the PFM, BFM, row and page size of each device family.

Some memory locations of the BFM are reserved for the device Configuration registers; see **Section 18.0 "Configuration Memory Device ID and Unique Device Identifier"** for more information.

TABLE 5-1: CODE MEMORY SIZE

PIC32MM Device	Row Size (Words)	Page Size (Words)	Boot Flash Memory Address (Bytes)(1)	Programming Execution(2,3)
PIC32MM0016/0032/0064/ 0128/0256GPL0XX	64	512	0x1FC00000-0x1FC016FF (5.75K)	RIPE_20_aabbcc.hex

Note 1: The Program Flash Memory address ranges are based on Program Flash size and are as follows:

- 0x1D000000-0x1D003FFF (16 Kbytes)
- 0x1D000000-0x1D007FFF (32 Kbytes)
- 0x1D000000-0x1D00FFFF (64 Kbytes)
- 0x1D000000-0x1D01FFFF (128 Kbytes)
- 0x1D000000-0x1D03FFFF (256 Kbytes)
- 0x1D000000-0x1D07FFFF (512 Kbytes)
- 0x1D000000-0x1D0FFFFF (1024 Kbytes)
- 0x1D000000-0x1D1FFFFF (2048 Kbytes)

All Program Flash Memory sizes are not supported by each family.

2: The Programming Executive can be obtained from the related product page on the Microchip web site, or it can be located in the following MPLAB® X IDE installation folders:
…\Microchip\MPLABX\mplab_ide\mplablibs\modules\ext\REALICE.jar
…\Microchip\MPLABX\mplab_ide\mplablibs\modules\ext\ICD3.jar
…\Microchip\MPLABX\mplab_ide\mplablibs\modules\ext\PICKIT3.jar

3: The last characters of the file name, aabbcc, vary based on the revision of the file.

5.2 4-Wire JTAG Details

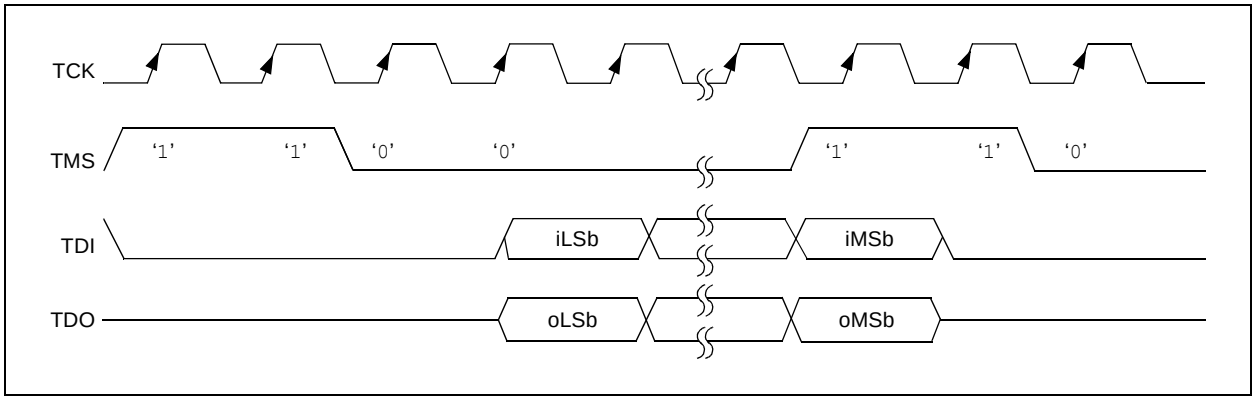
The 4-wire interface uses standard JTAG (IEEE 1149.1-2001) interface signals.

- TCK: Test Clock – drives data in/out
- TMS: Test Mode Select – selects operational mode
- TDI: Test Data Input – data into the device
- TDO: Test Data Output – 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. 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



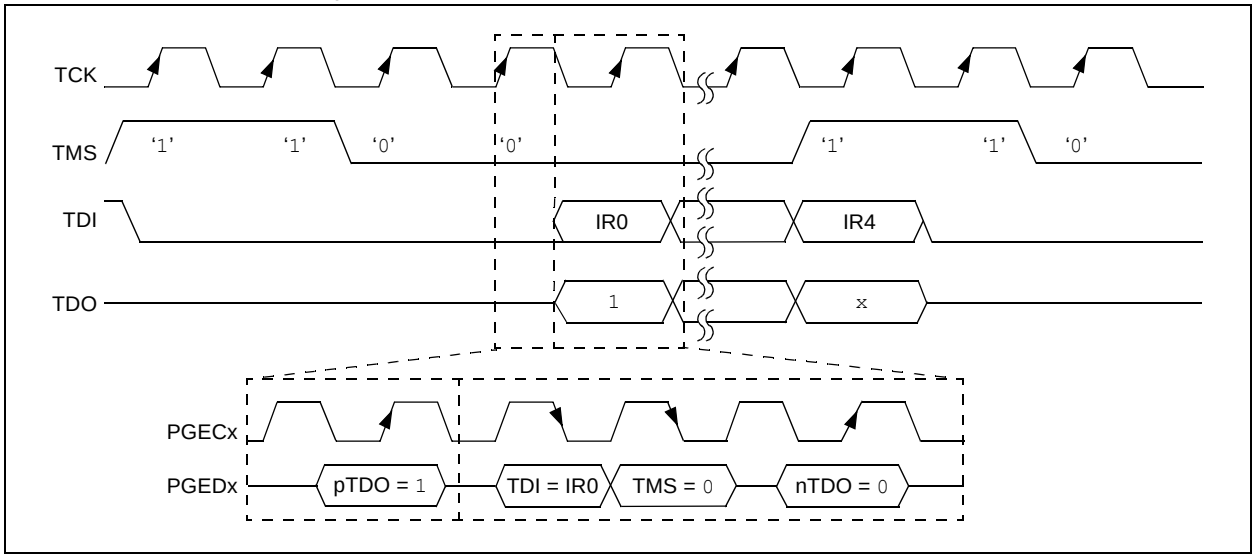
5.3 2-Wire ICSP Details

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.

5.3.1 4-PHASE ICSP

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

FIGURE 5-4: 2-WIRE, 4-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 to either make the pseudo-code more readable, to abstract implementation-specific behavior or both. When passing parameters with pseudo operation, the following syntax will be used:

- `5'h0x03` – sends 5-bit hexadecimal value of 3
- `6'b011111` – sends 6-bit binary value of 31

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) *
  oData = **`ReadFromAddress`** (address)

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

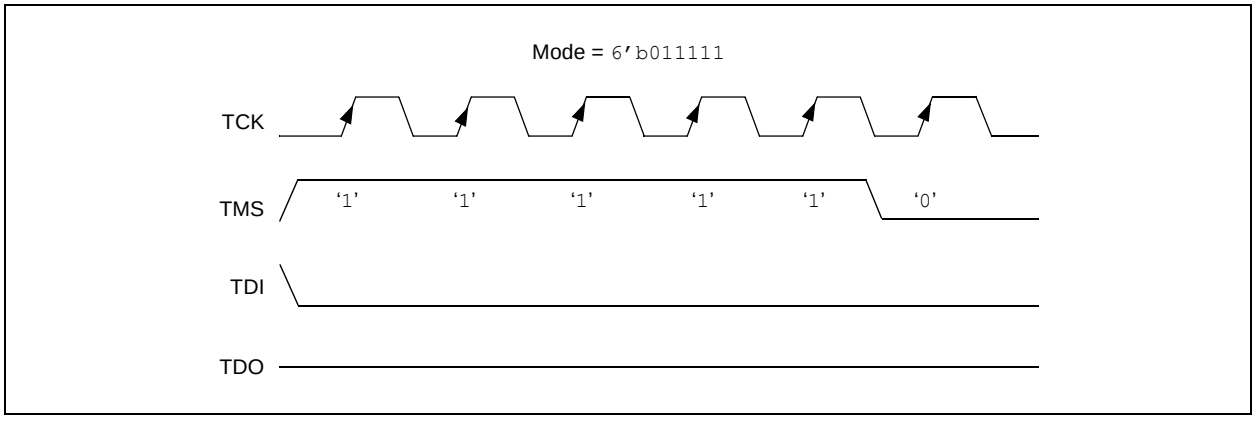
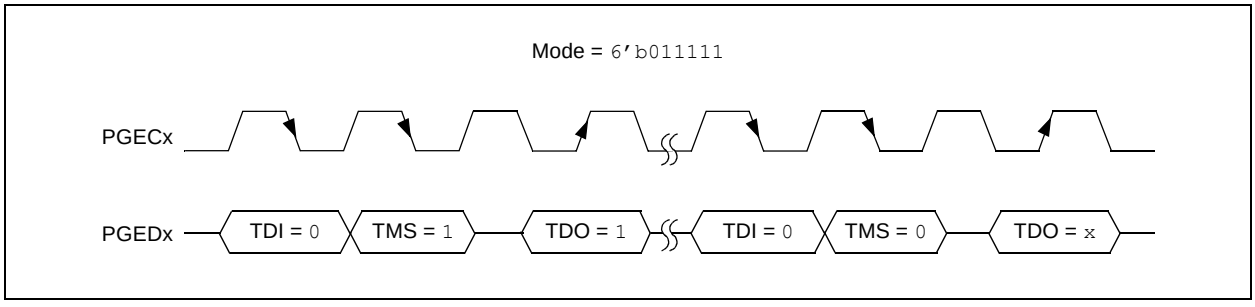


FIGURE 6-2: `SetMode` 2-WIRE



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 the TMS signal 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'h0x07)`

FIGURE 6-3: SendCommand 4-WIRE

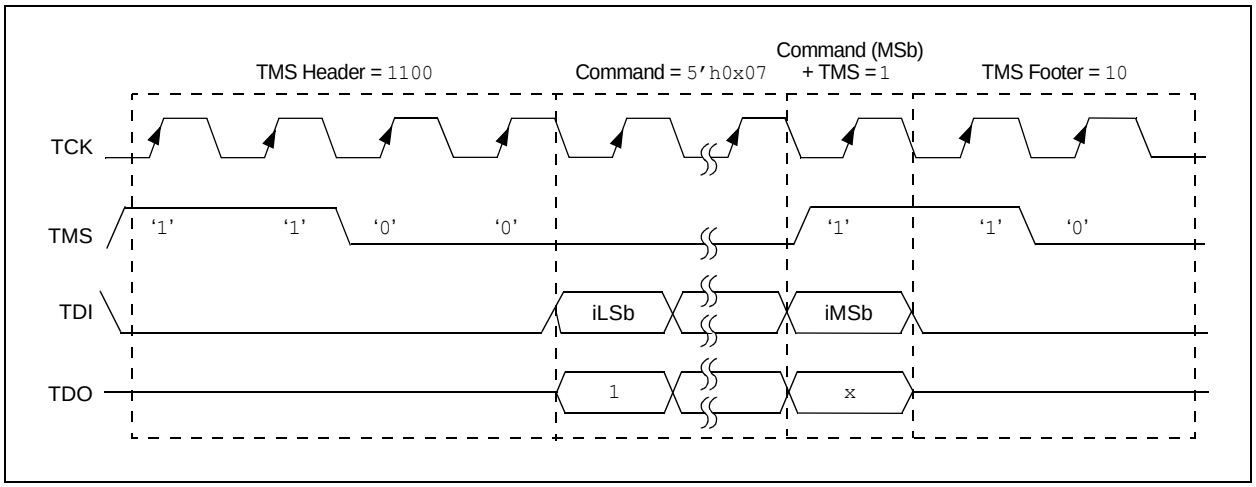
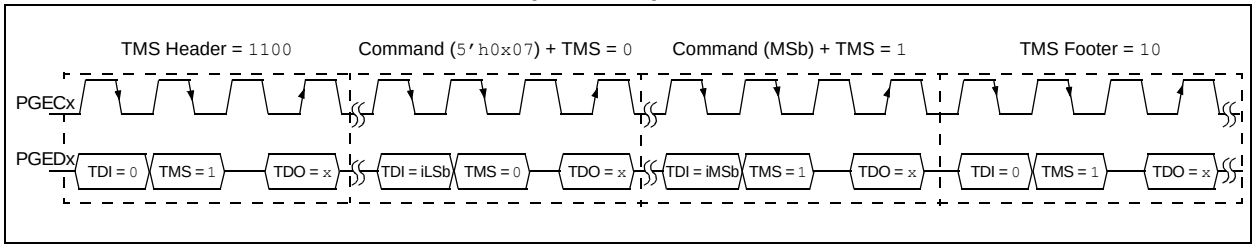


FIGURE 6-4: SendCommand 2-WIRE (4-PHASE)



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 the TMS signal 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'h0x12)

FIGURE 6-5: XferData 4-WIRE

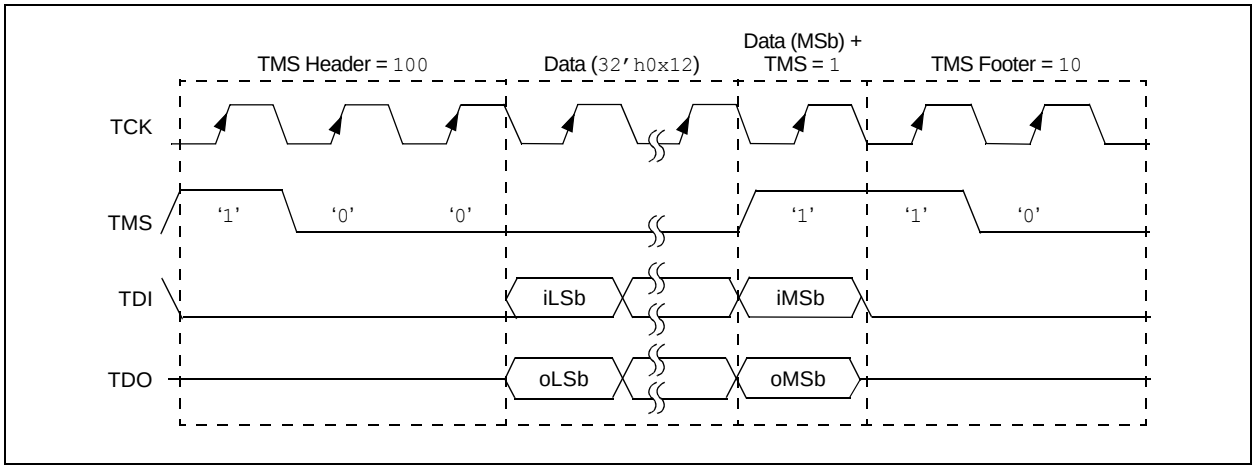
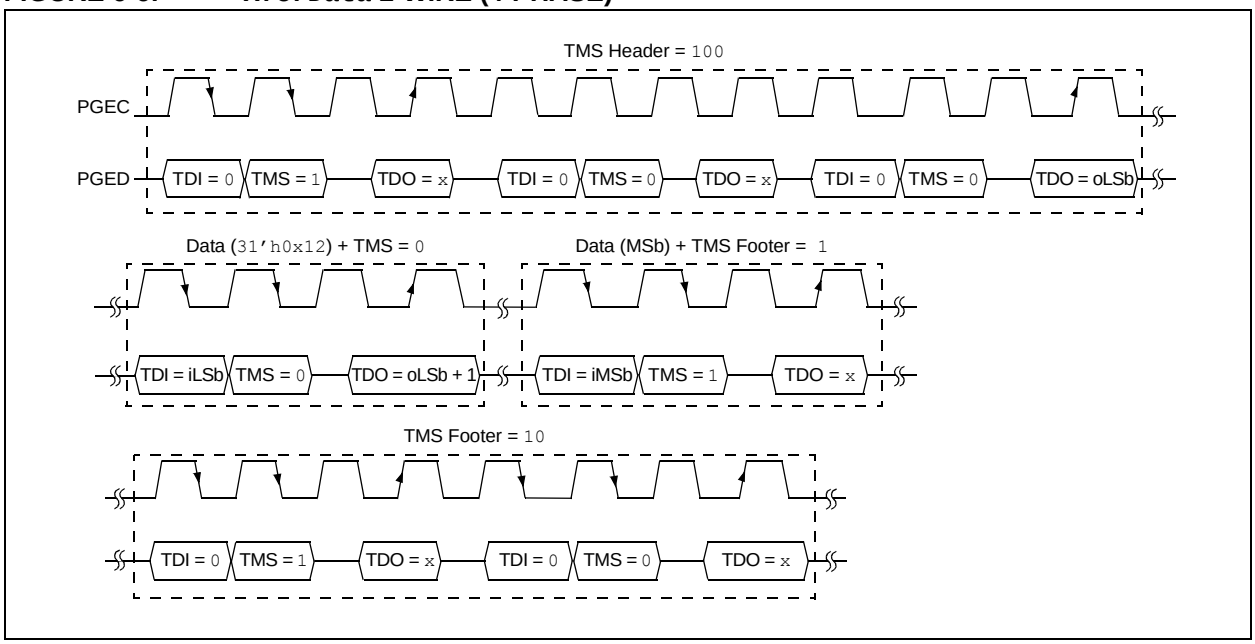


FIGURE 6-6: XferData 2-WIRE (4-PHASE)



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 detailed descriptions of commands.

EXAMPLE 6-1: SendCommand

```
// Select the Fastdata Register
SendCommand(ETAP_FASTDATA)
// Send/Receive 32-bit Data
oData = XferFastData(32'h0x12)
```

FIGURE 6-7: XferFastData 4-WIRE

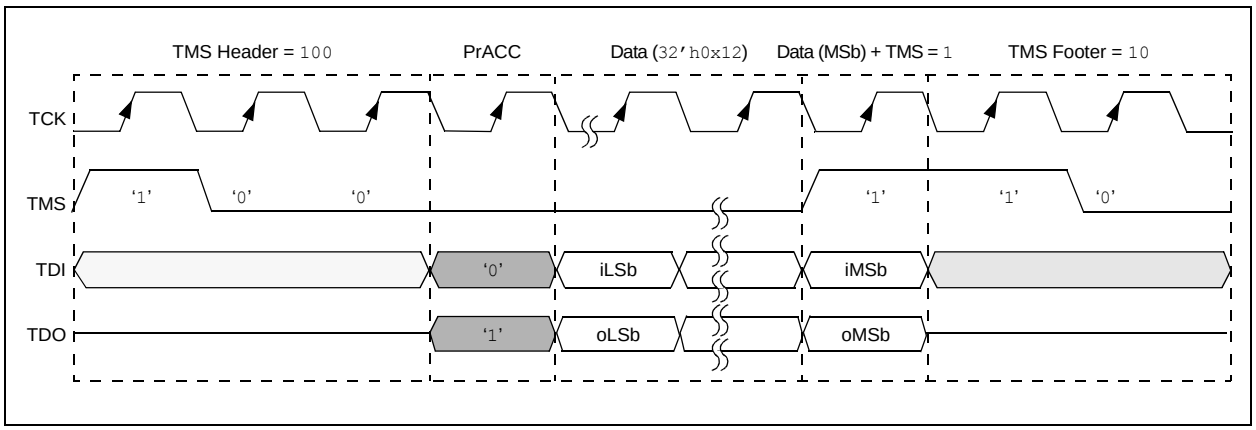
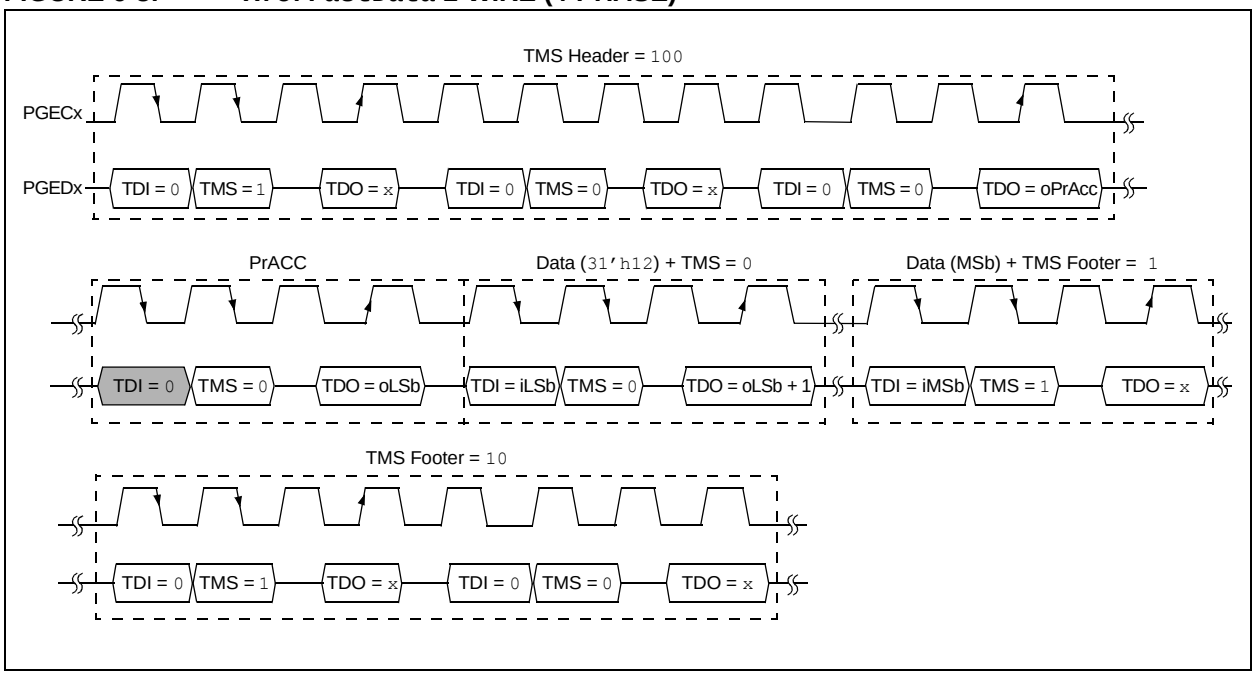


FIGURE 6-8: `XferFastData` 2-WIRE (4-PHASE)



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 the CPU.

Restrictions:
The device must be in Debug mode.

EXAMPLE 6-2: `XferInstruction` FOR PIC32MM DEVICES

```
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);
}
```

6.6 ReadFromAddress Pseudo Operation

Format:

`oData = ReadFromAddress` (address)

Purpose:

To send 32 bits of data to the device memory.

Description:

The 32-bit data is read from the memory at the address specified in the “address” parameter.

Restrictions:

The device must be in Debug mode.

EXAMPLE 6-3: ReadFromAddress

```
ReadFromAddress (address)
{

// Load Fast Data register address to s3
instruction = 0x000041B3;
instruction |= (0xff200000&0xffff0000);
XferInstruction(instruction);   // lui s3, <FAST_DATA_REG_ADDRESS(31:16)> - set address of fast data register

// Load memory address to be read into t0
instruction = 0x000041A8;
instruction |= (address&0xffff0000);
XferInstruction(instruction);   // lui t0, <DATA_ADDRESS(31:16)> - set address of data
instruction = 0x00005108;
instruction |= (address<<16)&0xffff0000;
XferInstruction(instruction);   // ori t0, <DATA_ADDRESS(15:0)> - set address of data

// Read data
XferInstruction(0x0000FD28);    // lw  t1, 0(t0)

// Store data into Fast Data register
XferInstruction(0x0000F933);    // sw  t1, 0(s3) - store data to fast data register
XferInstruction(0x0C000C00);    // 2 nops

// Shift out the data
SendCommand(ETAP_FASTDATA);
oData = XferFastData(32'h0x00000000);

return oData;

}
```

7.0 ENTERING 2-WIRE ICSP MODE

Any pair of the 2-wire PGEDx and PGECx pins can be used for programming; however, they must be used as a pair. PGED1 must be used with PGEC1, and so on.

Note:	If using the 4-wire JTAG interface, the following procedure is not necessary.

The following steps are required to enter 2-Wire ICSP mode:

1. The $\overline{\text{MCLR}}$ pin is briefly driven high, then low.
2. A 32-bit key sequence is clocked into PGEDx.
3. $\overline{\text{MCLR}}$ is then driven high within a specified period of time and held.

Refer to **Section 20.0 "AC/DC Characteristics and Timing Requirements"** for timing details.

The programming voltage applied to $\overline{\text{MCLR}}$ is V_{IH}, which is essentially V_{DD} in PIC32MM devices. There is no minimum time requirement for holding at V_{IH}. After V_{IH} is removed, an interval of at least P18 must elapse before presenting the key sequence on PGEDx.

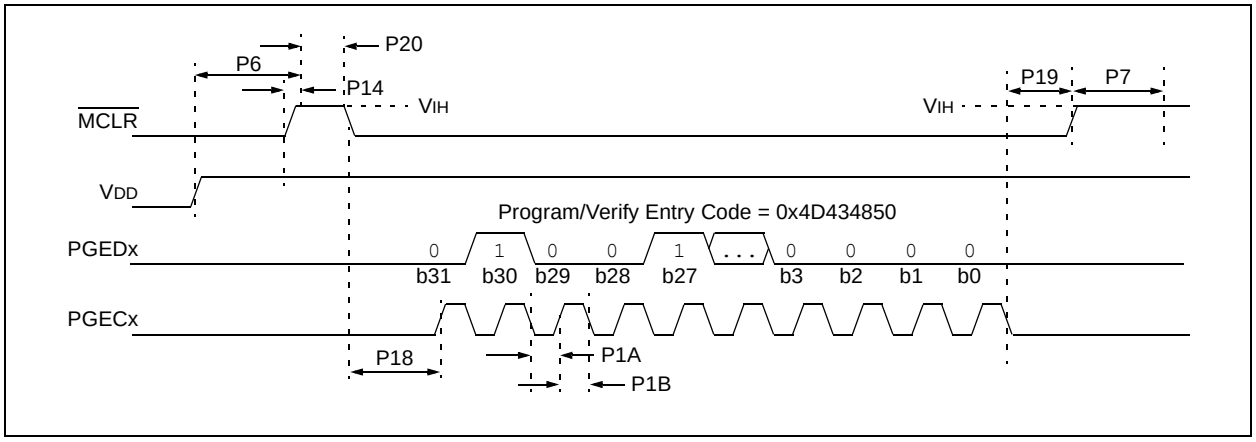
The key sequence is a specific 32-bit pattern: '0100 1101 0100 0011 0100 1000 0101 0000' (the acronym, 'MCHP', in ASCII). The device will enter Program/Verify mode only if the key sequence is valid. The MSb of the Most Significant nibble must be shifted in first.

Once the key sequence is complete, V_{IH} must be applied to $\overline{\text{MCLR}}$ and held at that level for as long as the 2-wire ICSP interface is to be maintained. An interval of at least time, P19 and P7, must elapse before presenting data on PGEDx. Signals appearing on PGEDx before P7 has elapsed will not be interpreted as valid.

Upon successful entry, the programming operations documented in subsequent sections can be performed. While in 2-Wire Enhanced ICSP mode, all unused I/Os are placed in the high-impedance state.

Note:	Entry into ICSP mode places the device in Reset to prevent instructions from executing. To release the Reset, the MCHP_DE_ASSERT_RST command must be issued.

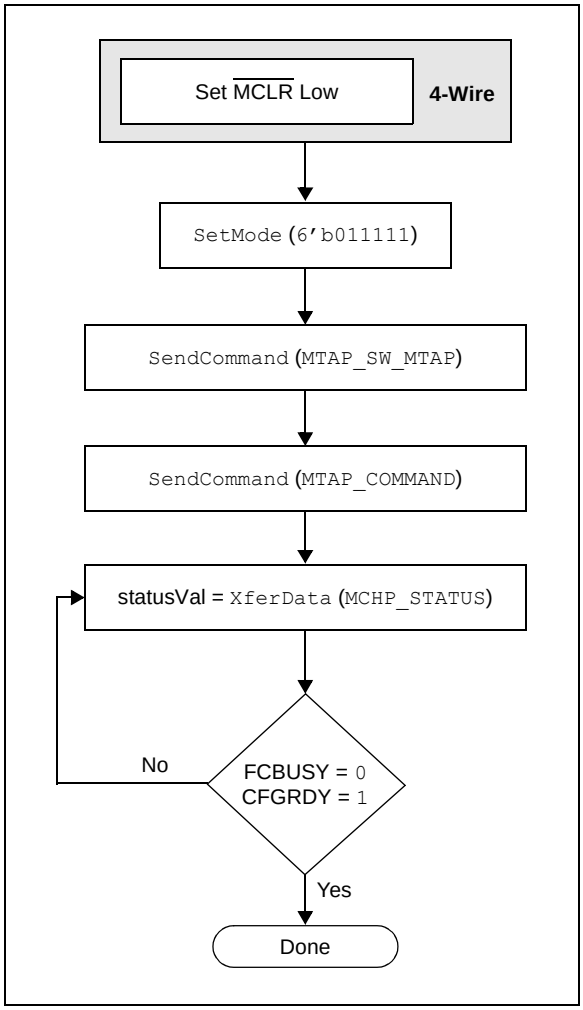
FIGURE 7-1: ENTERING ENHANCED ICSP™ MODE



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

The following steps are required to check the device status using the 4-wire JTAG interface. MCLR is asserted to hold the device in Reset to prevent code execution.

1. Set MCLR pin low.
2. SetMode (6'b011111) to force the chip TAP controller into the 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', go to Step 5.

Note: If using the 4-wire interface, the oscillator source, as selected by the Configuration Words, must be present to access Flash memory. In an unprogrammed device, the oscillator source is the internal FRC, allowing for Flash memory access. If the Configuration Words have been reprogrammed to select an external oscillator source, then it must be present for Flash memory access. See the **"Special Features"** chapter in the specific device data sheet for details regarding oscillator selection using the Configuration Word settings.

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 the 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', go to Step 4.

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

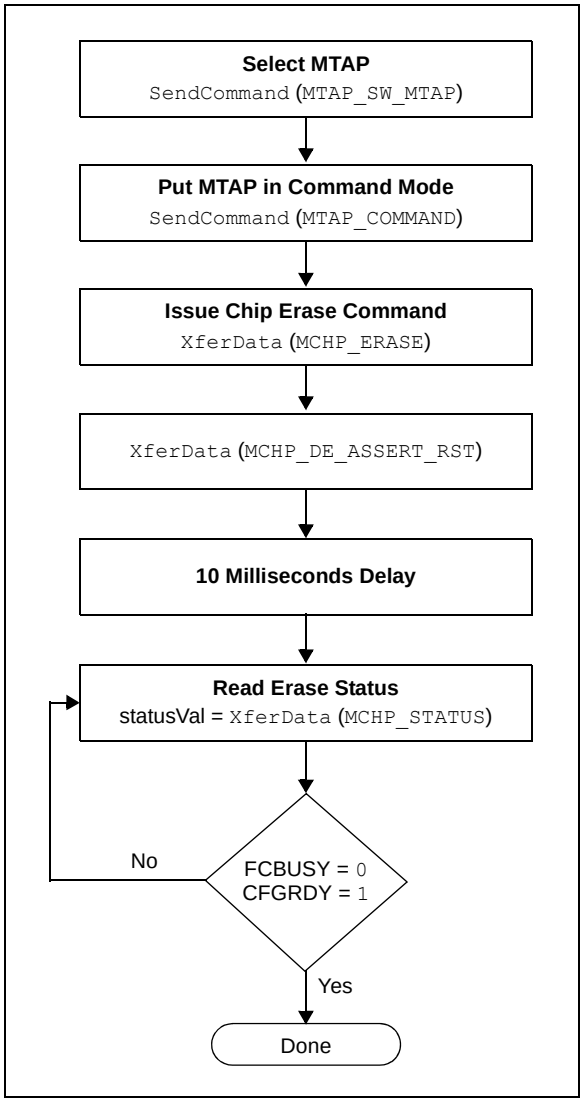
9.0 ERASING THE DEVICE

Before a device can be programmed, it must be erased. The erase operation writes all '1's 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 Flash Memory (Program Flash, Boot Flash 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. XferData (MCHP_DE_ASSERT_RST).
5. Delay 10 ms.
6. statusVal = XferData (MCHP_STATUS).
7. If CFGRDY (statusVal<3>) is not '1' and FCBUSY (statusVal<2>) is not '0', go to Step 5.

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 incorrectly or the device is damaged.

9.1 Blank Check

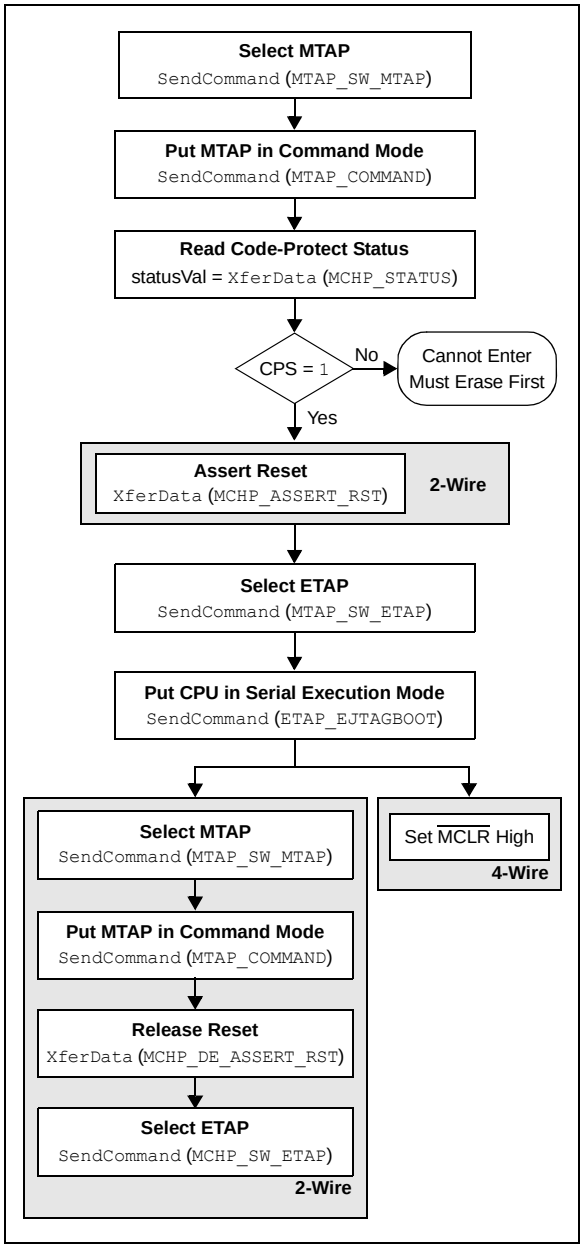
The term, "Blank Check", implies verifying that the device has been successfully erased and has no programmed memory locations. A blank or erased memory location always reads as '1'.

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

10.0 ENTERING SERIAL EXECUTION MODE

Serial Execution mode allows instructions to be fed to the CPU via the JTAG interface. Before a device can be programmed, it must be placed in Serial Execution mode. If the device is code-protected, a Chip Erase must be performed prior to entering Test Serial Execution mode. 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 4-Wire Serial Execution mode:

Note:	It is assumed that $\overline{\text{MCLR}}$ has been driven low from the previous Check Device Status step (see Figure 8-1).

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 2-Wire 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. SendCommand (MTAP_SW_ETAP).

11.0 DOWNLOADING THE PROGRAMMING EXECUTIVE (PE)

The PE provides the mechanism for the programmer to program and verify PIC32MM devices using a simple command set, and communication protocol. The PE resides in RAM memory and is executed by the CPU to program the device. There are several basic functions provided by the PE:

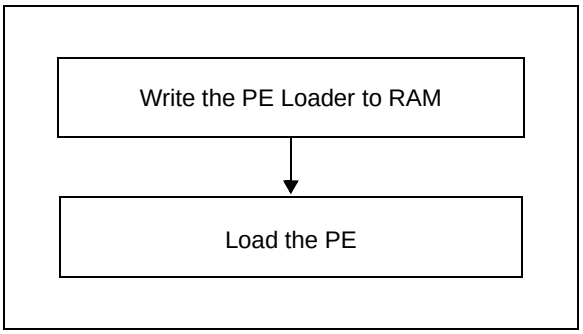
- Read Memory
- Erase Memory
- Program Memory
- Blank Check
- Read Executive Firmware Revision
- Get the Cyclic Redundancy Check (CRC) of Flash Memory Locations

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 PIC32MM family can be programmed using the command set shown in Table 16-2.

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 Executive 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: Set up the PIC32MM RAM address for the PE. The instruction sequence executed by the PIC32MM core is:
`lui a0, 0xa000`
`ori a0, a0, 0x800`	

XferInstruction	0xA00041A4
XferInstruction	0x08005084

Step 2: Load the PE_loader. Repeat this step (Step 3) until the entire PE_loader is loaded in the PIC32MM memory. In the operands field, "<PE_loader hi++>", represents the MSbs 31 through 16 of the PE_loader op codes shown in Table 11-2. Likewise, "<PE_loader lo++>" represents the LSbs 15 through 0 of the PE_loader op codes 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 op codes of the PE_loader shown in Table 11-2. The instruction sequence executed by the PIC32MM core is:
`lui a2, <PE_loader hi++>`
`ori a2, a2, <PE_loader lo++>`
`sw a2, 0(a0)`
`addiu a0, a0, 4`	

XferInstruction	0x<PE_loader hi++>41A6
XferInstruction	0x<PE_loader lo++>50C6
XferInstruction	0x6E42EB40

Step 3: Jump to the PE_loader. The instruction sequence executed by the PIC32MM core is:
`lui t9, 0xa000`
`ori t9, t9,0x800`
`jr t9`
`nop`	

XferInstruction	0xA00041B9
XferInstruction	0x08005339
XferInstruction	0x0C004599
Step 4: Load the PE using the PE_loader. Repeat the last instruction of this step (Step 5) until the entire PE is loaded into the PIC32MM 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 PIC32MM memory (refer to **Appendix B: "Hex File Format"**). The instruction sequence executed by the PIC32MM is shown in Table 11-2.	
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)
Step 5: 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 OP CODES

Opcode	Instruction
0xDEAD41A7	lui a3, 0xdead
0xFF2041A6	lui a2, 0xff20
0xFF2041A5	lui a1, 0xff20
	here1
0x69E06A60	lw a0, 0(a2)
	lw v1, 0(a2)
0x000C94E3	beq v1, a3, <here3>
0x8DFA0C00	nop
	beqz v1, <here1>
	here2
	lw v0, 0(a1)
0xE9406DBE	addiu v1, v1,-1
	sw v0, 0(a0)
0xADFB6E42	addiu a0, a0, 4
	bnez v1, <here2>
0xCFF20C00	nop
	b <here1>
0x0C000C00	nop; nop
	here3
0xA00041A2	lui v0, 0xa000
0x09015042	ori v0, v0, 0x901
0x0C004582	jr v0
	nop

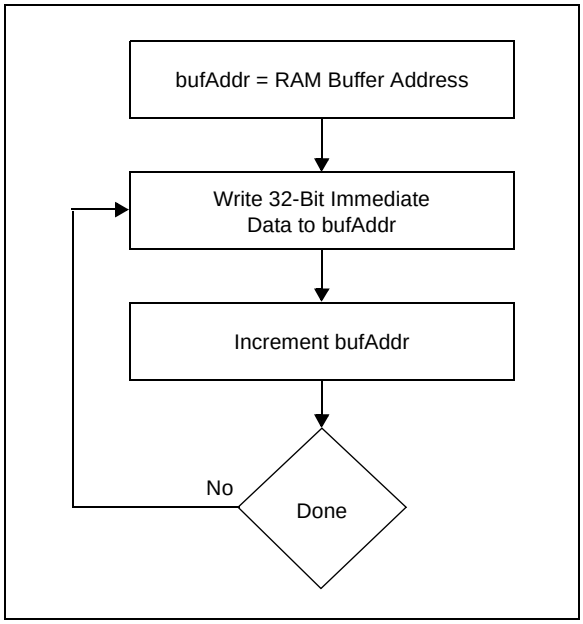
12.0 DOWNLOADING A DATA BLOCK

To program a block of data to the PIC32MM device, it must 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 (immediate) data for writing the block of data to the device's 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 the CPU.

TABLE 12-1: DOWNLOAD DATA OP CODES

Opcode	Instruction
Step 1: Initialize SRAM base address to 0xA0000000.	
`0xA00041A4`	`lui a0, 0xA000`
Step 2: Write the entire row of data to be programmed into system SRAM.	

`0x<DATA(31:16)>41A5`
`0x<DATA(15:0)>50A5`
`0x<OFFSET>F8A4`
 	
`0x0C000C00`
	`lui a1, <DATA(31:16)>;`
	`ori a1, <DATA(15:0)>;`
	`sw a1, <OFFSET>(a0);`
	`//OFFSET increments by 4`
	`Two NOPs`

Step 3: Repeat Step 2 until one row of data has been loaded.	

12.2 With the PE

When using the PE, the steps in **Section 12.0 "Downloading a Data Block"** and **Section 13.0 "Initiating a Flash Row Write"** are handled in two single commands: `ROW_PROGRAM` and `PROGRAM`.

The `ROW_PROGRAM` command programs a single row of Flash data, while the `PROGRAM` command programs multiple rows of Flash data. Both of these commands are documented in **Section 16.0 "The Programming Executive"**.

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.

See Table 13-1 for the opcode and instructions for initiating a Flash row write.

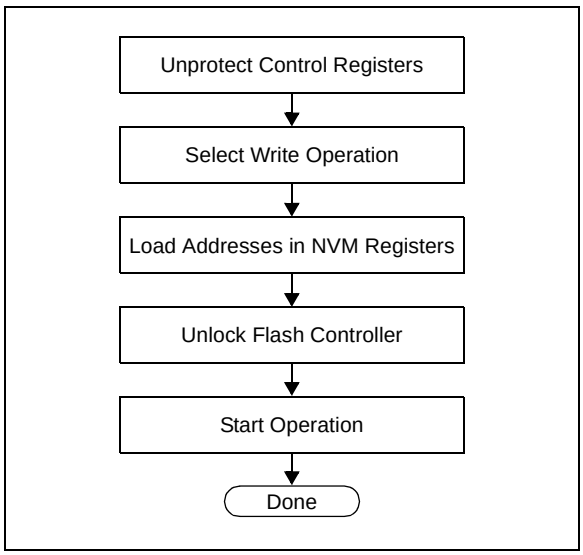
13.1 With the PE

When using the 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



In the Flash write procedure (see Table 13-1), the Row Programming method is used to program Flash memory, as it is typically the most expedient. The double-word programming method is also available and may be used or required depending on your application. Refer to the **"Flash Program Memory"** chapter in the specific device data sheet and the related section in the *"PIC32 Family Reference Manual"* for more information.

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 OP CODES

Operation	Operand

Step 1: Initialize constants. Registers a1 and a2 are set for WREN = `1` or NVMOP<3:0> = `0011`, and WR = `1`, respectively. Registers s1 and s2 are set for the unlock data values and s0 is initialized to '`0`'.
`ori a1, $0, 0x4003`
`ori a2, $0, 0x8000`
`lui s1, 0xaa99`
`ori s1, s1, 0x6655`
`lui s2, 0x5566`
`ori s2, s2, 0x99aa`
`lui s0, 0x0000`	

`XferInstruction`	`0x400350A0`
`XferInstruction`	`0x800050C0`
`XferInstruction`	`0xAA9941B1`
`XferInstruction`	`0x66555231`
`XferInstruction`	`0x556641B2`
`XferInstruction`	`0x99AA5252`
`XferInstruction`	`0x000041B0`

Step 2: Set Register a0 to the base address of the NVM controller (0xBF80_2380). Register s3 is set for the value used to disable write protection in NVMBPB.
`lui a0, 0xbf80`
`ori a0, a0, 0x2380`
`ori s3, $0, 0x8000`
`/* BWPAUNLK bit mask */`	

`XferInstruction`	`0xBF8041A4`
`XferInstruction`	`0x23805084`
`XferInstruction`	`0x80005260`

Step 3: Unlock and disable Boot Flash write protection.
`sw s1, 16(a0)`
`/* NVMKEY = 0xaa996655 */`
`sw s2, 16(a0)`
`/* NVMKEY = 0x556699aa */`
`sw s3, 112(a0)`
`/*BWPAUNLK bit (NVMBPB register) = 1*/`
`nop`	

`XferInstruction`	`0xBF8041A4`
`XferInstruction`	`0x23805084`
`XferInstruction`	`0x80005260`

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

Operation	Operand

Step 4: Set the NVMADDR register with the address of the Flash row to be programmed.
`lui t0, <FLASH_ROW_ADDR(31:16)>`
`ori t0, t0, <FLASH_ROW_ADDR(15:0)>`
`sw t0, 32(a0)`	

`XferInstruction`	`0x<FLASH_ROW_ADDR(31:16)>41A8`
`XferInstruction`	`0x<FLASH_ROW_ADDR(15:0)>5108`
`XferInstruction`	`0x0020F904`

Step 5: Set the NVMSRCADDR register with the physical source SRAM address.
`ori s0, $0, <RAM_ADDR(15:0)>`
`sw s0, 80(a0)`	

`XferInstruction`	`0x<RAM_ADDR(15:0)>5200`
`XferInstruction`	`0x0050FA04`

Step 6: Set up the NVMCON register for write operation.
`sw a1, 0(a0)`
`/* NVMCON = 0x4003 */`
`delay (6 µs)`	

`XferInstruction`	`0x0C00EAC0`

Step 7: Unlock the NVMCON register and start the write operation (WR bit = `1`).
`sw s1, 16(a0)`
`/* NVMKEY = 0xaa996655 */`
`sw s2, 16(a0)`
`/* NVMKEY = 0x556699aa */`
`sw a2, 8(a0)`
`/* NVMCONSET = 0x8000 */`	

`XferInstruction`	`0xFA44E8C4`
`XferInstruction`	`0xEB420010`
Step 8: Read NVMCON register until the WR bit (NVMCON<15>) is clear.	
`ReadFromAddress`	`0xBF802380`
Step 9: Wait at least 500 ns after seeing a '`0`' in the WR bit (NVMCON<15>) before writing to any of the NVM registers. This requires inserting a `NOP` in the execution. `nop` `nop` `nop` `nop`	
`XferInstruction`	`0x0C000C00`
`XferInstruction`	`0x0C000C00`

Step 10: Clear the WREN bit (NVMCONM<14>).
`ori a3, $0, 0x4000`
`sw a3, 4(a0)`
`nop`	

`XferInstruction`	`0x400050E0`
`XferInstruction`	`0x0C00EBC1`

14.0 VERIFY DEVICE MEMORY

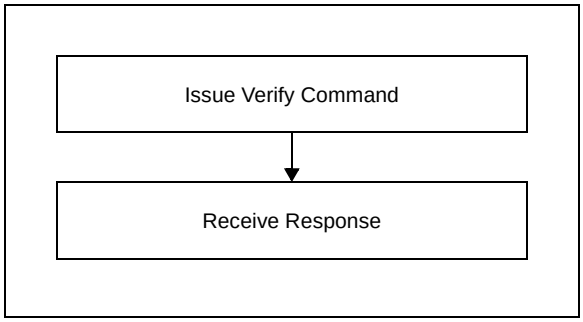
The verify step involves reading back the code memory space and comparing it with the copy held in the programmer's buffer. The Configuration registers are verified with the rest of the code.

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. Table 16-2 lists the op codes and instructions.

FIGURE 14-1: VERIFYING MEMORY WITH THE PE



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

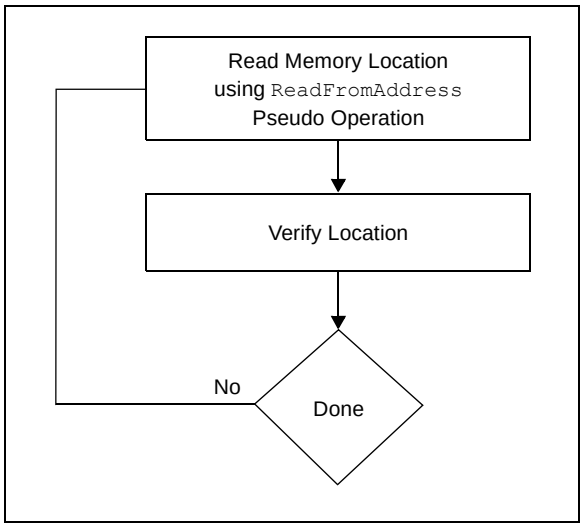
1. `XferFastData` (`GET_CRC`).
2. `XferFastData` (`start_Address`).
3. `XferFastData` (length).
4. valCkSum = `XferFastData` (`32'h0x00`).

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 OP CODES

Opcode	Instruction
Step 1: Initialize some constants.	
3c13ff20	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 register 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.	

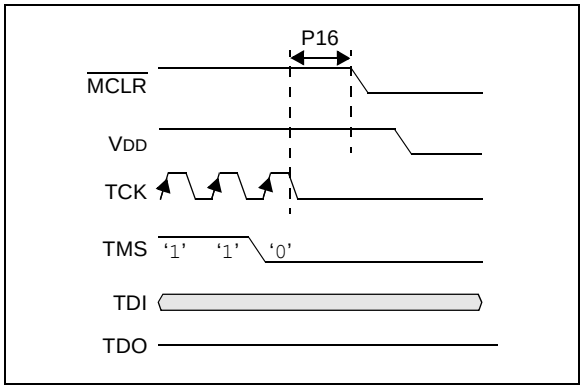
15.0 EXITING PROGRAMMING MODE

Once a device has been 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 Programming mode is done by removing VIH from MCLR, as illustrated in Figure 15-1. The only requirement for exit is that an interval, P16, should elapse between the last clock and program signals before removing VIH.

FIGURE 15-1: 4-WIRE EXIT PROGRAMMING MODE



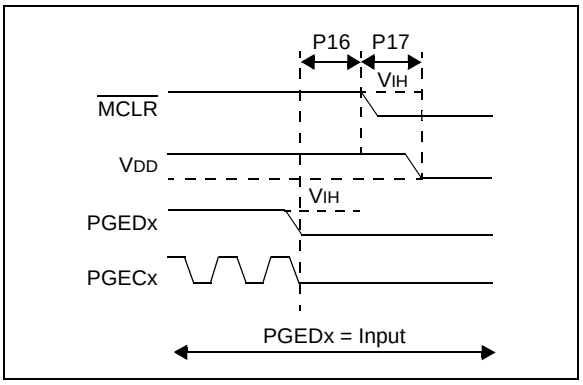
The following steps are required to exit Programming mode:

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

15.2 2-Wire Interface

Exiting Programming mode is done by removing VIH from MCLR, as illustrated in Figure 15-2. The only requirement for exit is that an interval, P16, should elapse between the last clock and program signals on PGECx and PGEDx before removing VIH.

FIGURE 15-2: 2-WIRE EXIT PROGRAMMING MODE



Use the following steps to exit Programming mode:

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

16.0 THE PROGRAMMING EXECUTIVE

Note: The Programming Executive (PE) is included with your installation of MPLAB® X IDE. To download the appropriate PE file for your device, please visit the related product page on the Microchip web site (www.microchip.com).

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 ICSP mode, the PIC32MM family devices operate from the internal Fast RC (FRC) oscillator, which has a nominal frequency of 8 MHz.

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

Table 16-2 provides PE command set details, such as opcode, mnemonic, and a 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 "DOUBLE_WORD_PROGRAM Command"**.

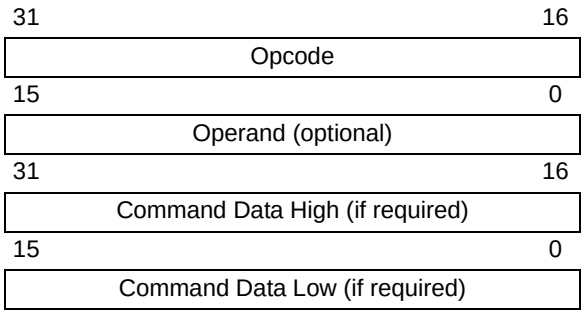
The PE sends a response to the programmer for each command that it receives. The response indicates if the command was processed correctly. It includes any required response data or error data.

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, and a 16-bit command operand field. Use of the operand field varies by command.

Note:	Some commands have no operand information; however, the operand 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 in the list returns a NACK response, as shown in Table 16-3.

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

TABLE 16-2: PE COMMAND SET

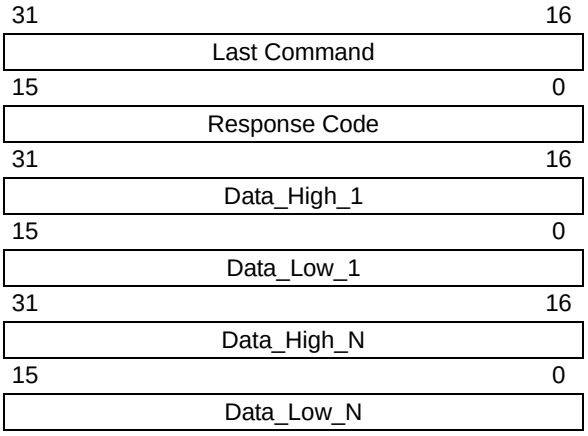
Opcode	Mnemonic	Description
0x0	ROW_PROGRAM(1)	Programs one row of Flash memory at the specified address.
0x1	READ	Reads N 32-bit words of memory starting from the specified address (N < 65,536).
0x2	PROGRAM	Programs Flash memory starting at the specified address.
0x4	CHIP_ERASE	Chip Erase of entire chip.
0x5	PAGE_ERASE	Erases pages of code memory from the specified address.
0x6	BLANK_CHECK	Blank Check code.
0x7	EXEC_VERSION	Reads the PE software version.
0x8	GET_CRC	Gets the CRC of Flash memory.
0x9	PROGRAM_CLUSTER	Programs the specified number of bytes to the specified address.
0xA	GET_DEVICEID	Returns the hardware ID of the device.
0xC	GET_CHECKSUM	Gets the checksum of Flash memory.
0xE	DOUBLE_WORD_PGRM	Programs two words of Flash memory at the specified address.

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

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_N, must be arranged using the packed instruction word format provided in Table 16-4 (this command expects an entire row of data).

FIGURE 16-3: ROW_PROGRAM COMMAND

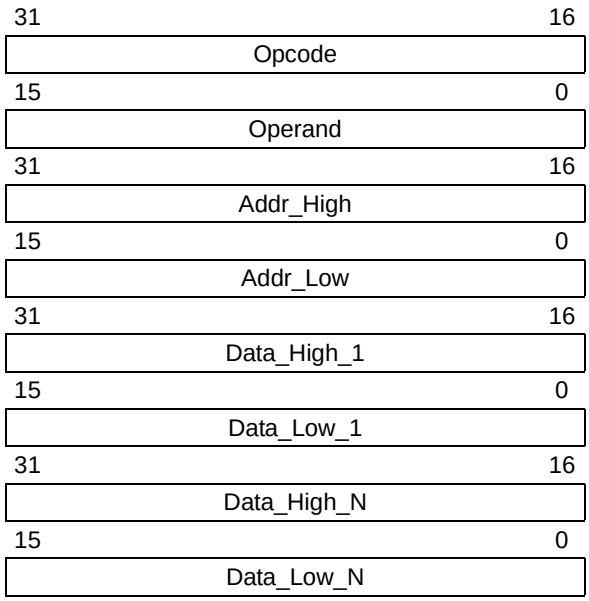
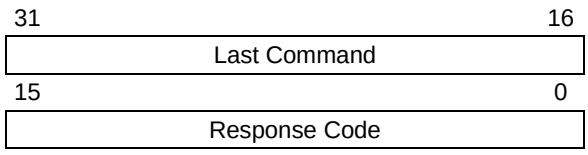


TABLE 16-4: ROW_PROGRAM FORMAT

Field	Description
Opcode	0x0
Operand	Not used
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 of Data Word 1
Data_Low_1	Low 16 bits of Data Word 1
Data_High_N	High 16 bits of Data Word 2 through N
Data_Low_N	Low 16 bits of Data Word 2 through N

Expected Response (1 word):

FIGURE 16-4: ROW_PROGRAM RESPONSE



16.2.4 READ COMMAND

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

FIGURE 16-5: READ COMMAND

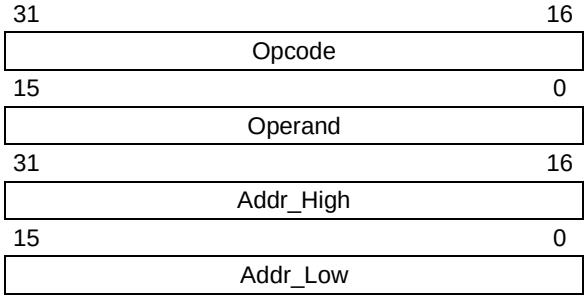
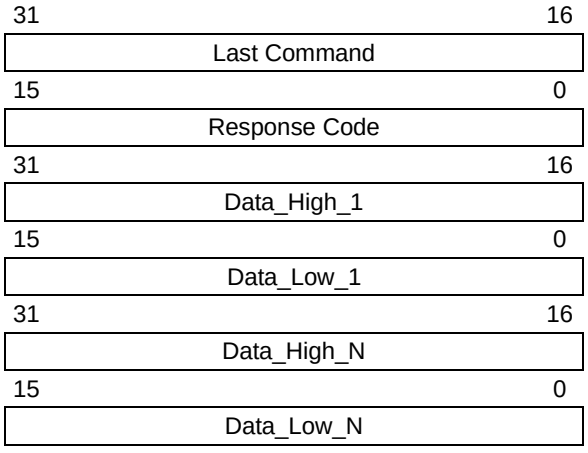


TABLE 16-5: READ FORMAT

Field	Description
Opcode	0x1
Operand	N number of 32-bit words to read (maximum of 65,535)
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. 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. A 32-bit length field specifies the number of bytes to program.

The address must be aligned to a Flash row size boundary and the length must be a multiple of a Flash row size. Refer to Table 5-1 for device row sizes.

FIGURE 16-7: PROGRAM COMMAND

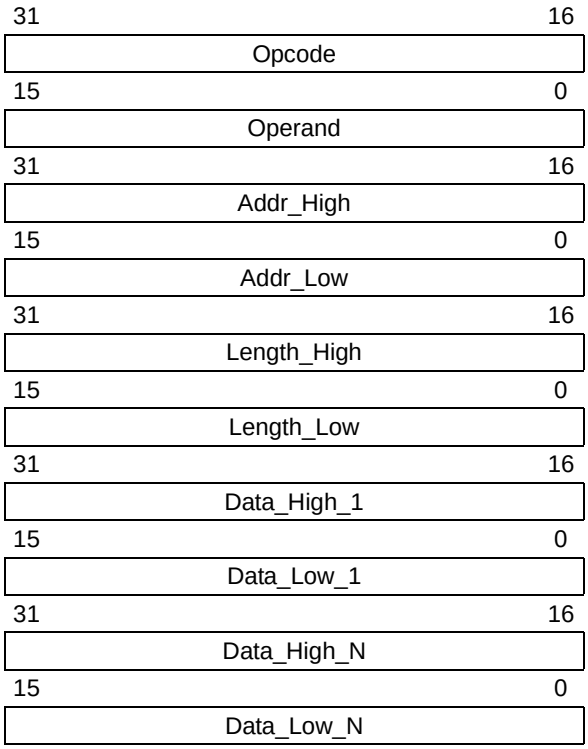


TABLE 16-6: PROGRAM FORMAT

Field	Description
Opcode	0x2
Operand	Not used
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 of Data Word 2 through N
Data_High_N	High 16 bits of Data Word 2 through N

The following are three programming scenarios:

- The length of the data to be programmed is the size of a single Flash row
- The length of the data to be programmed is the size of two Flash rows
- The length of the data to be programmed is larger than the size of two Flash rows

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

The PE will respond for each row of data that it receives. If the data length of the command is equal to a single row, a single PE response is generated. If the data length is equal to two rows, the PE waits to receive both rows of data and then sends back-to-back responses for each data row. If the data length is greater than two rows of data, the PE will send the response for the first row after receiving the first two rows of data. Subsequent responses are sent after receiving subsequent data row packets. The responses will lag the data by one row. When the last row of data is received, the PE will respond with back-to-back responses for the second to last data row, followed by the last row.

If the PE encounters an error in programming any of the blocks, it sends a failure status to the probe and aborts the PROGRAM command. On receiving the failure status, the probe must stop sending data. The PE will not process 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. Then, 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

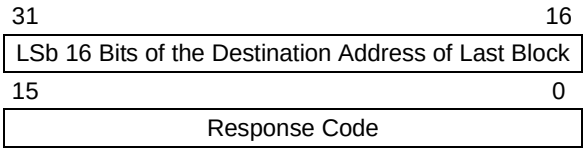
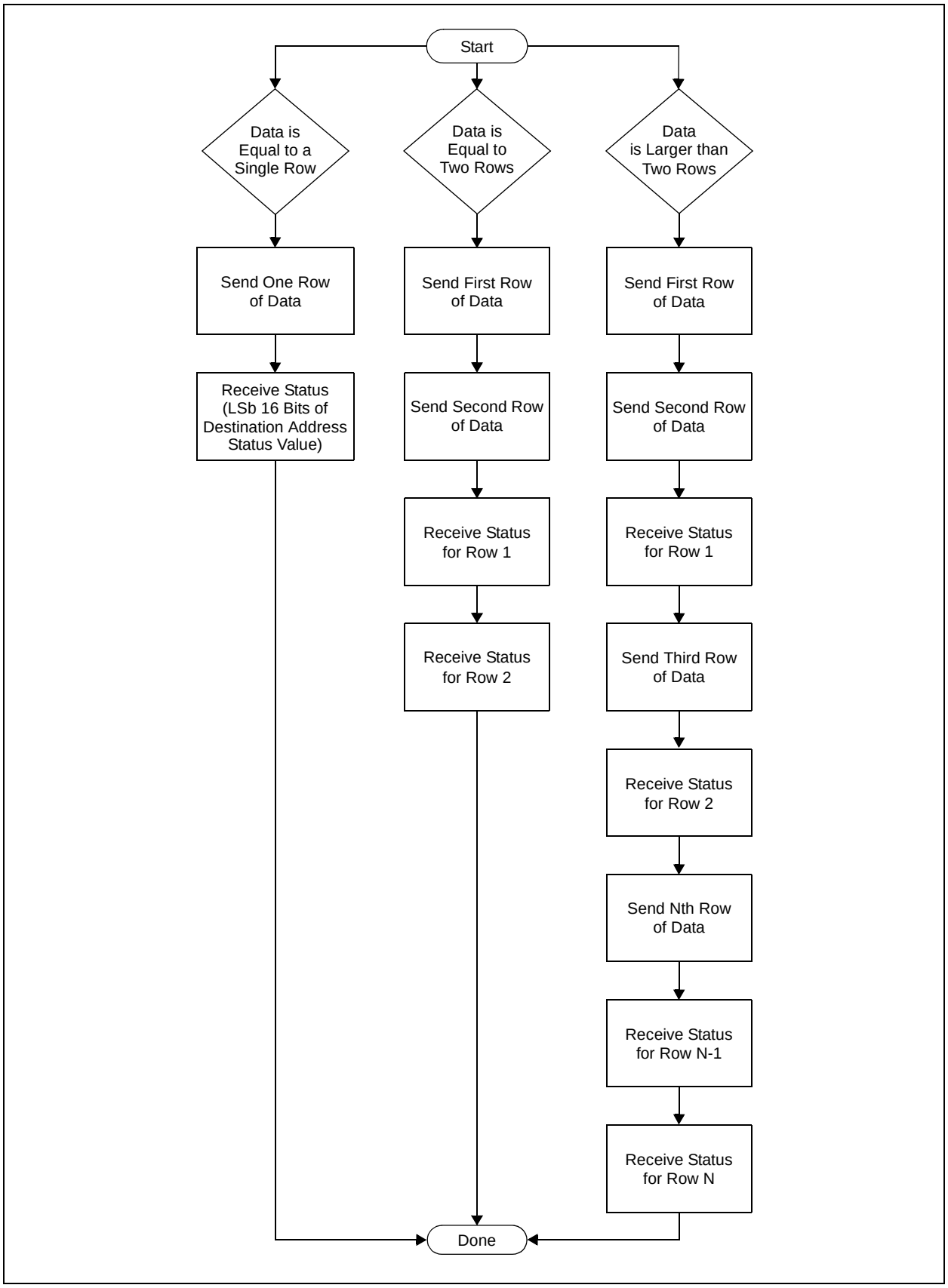


FIGURE 16-9: PROGRAM COMMAND ALGORITHM



16.2.6 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 0xFFFFFFFF.

FIGURE 16-10: CHIP_ERASE COMMAND

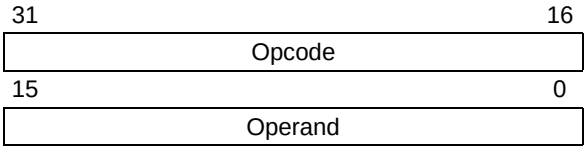
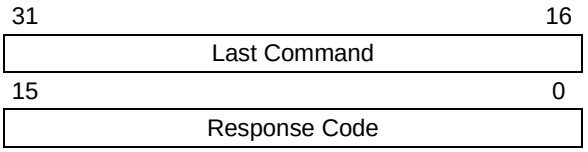


TABLE 16-7: CHIP_ERASE FORMAT

Field	Description
Opcode	0x4
Operand	Not used
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 16-11: CHIP_ERASE RESPONSE



16.2.7 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 0xFFFFFFFF.

FIGURE 16-12: PAGE_ERASE COMMAND

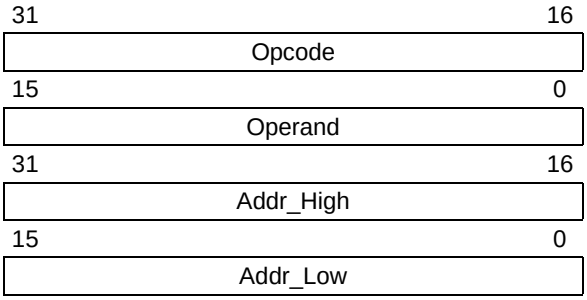
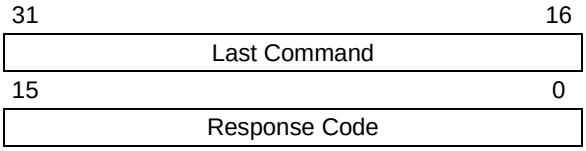


TABLE 16-8: PAGE_ERASE FORMAT

Field	Description
Opcode	0x5
Operand	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 16-13: PAGE_ERASE RESPONSE



16.2.8 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 16-14: BLANK_CHECK COMMAND

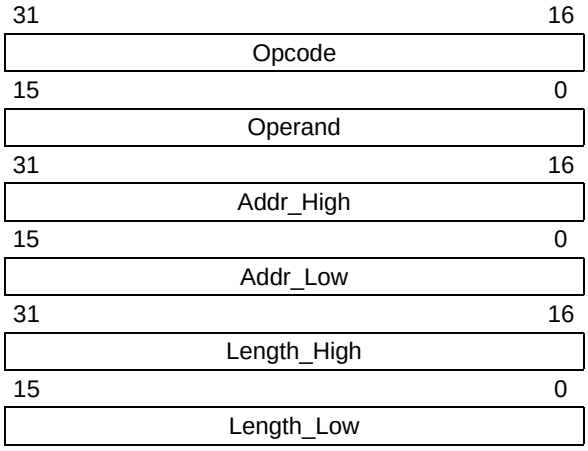
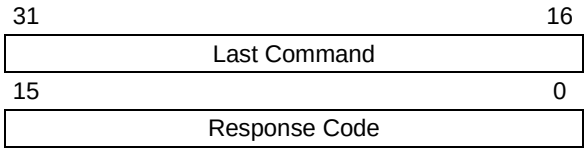


TABLE 16-9: BLANK_CHECK FORMAT

Field	Description
Opcode	0x6
Operand	Not used
Address	Address where to start the Blank Check
Length	Number of program memory locations to check in terms of bytes

Expected Response (1 word for blank device):

FIGURE 16-15: BLANK_CHECK RESPONSE



16.2.9 EXEC_VERSION COMMAND

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

FIGURE 16-16: EXEC_VERSION COMMAND

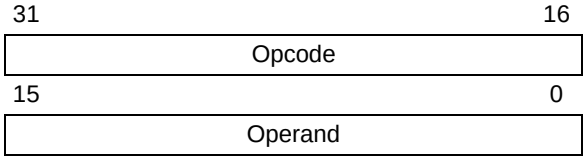
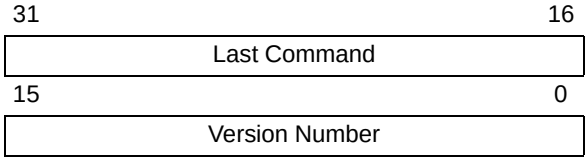


TABLE 16-10: EXEC_VERSION FORMAT

Field	Description
Opcode	0x7
Operand	Not used

Expected Response (1 word):

FIGURE 16-17: EXEC_VERSION RESPONSE



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

FIGURE 16-18: GET_CRC COMMAND

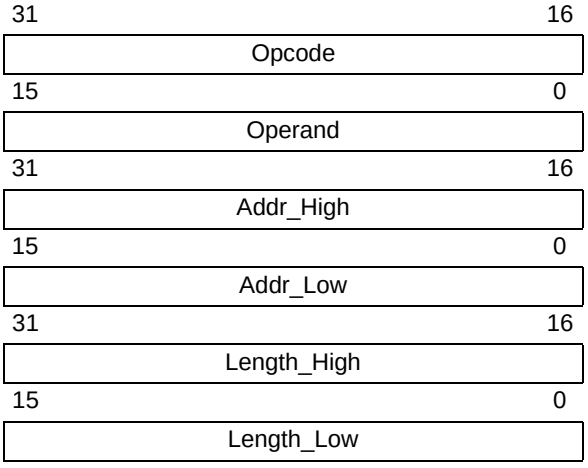
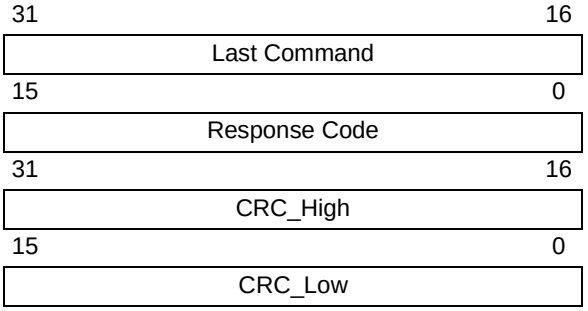


TABLE 16-11: GET_CRC FORMAT

Field	Description
Opcode	0x8
Operand	Not used
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 16-19: GET_CRC RESPONSE



16.2.11 PROGRAM_CLUSTER COMMAND

PROGRAM_CLUSTER programs the specified number of bytes to the specified address. The address must be 64-bit aligned and the number of bytes must be a multiple of a 64-bit word.

FIGURE 16-20: PROGRAM_CLUSTER COMMAND

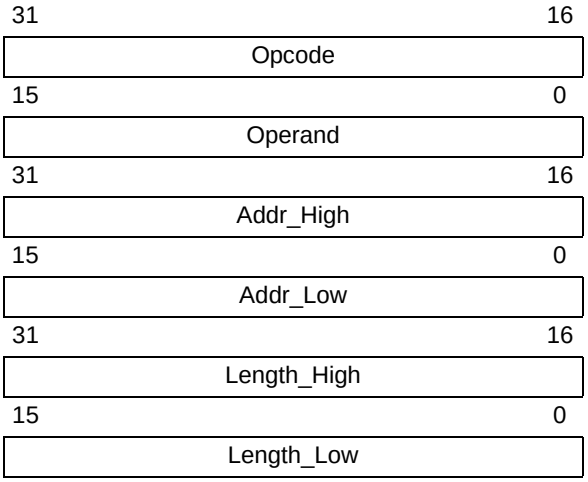


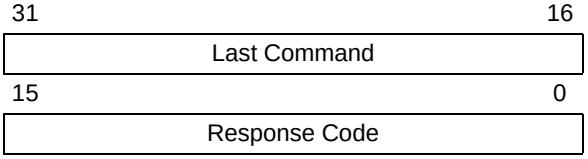
TABLE 16-12: PROGRAM_CLUSTER FORMAT

Field	Description
Opcode	0x9
Operand	Not used
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. Then, 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 16-21: PROGRAM_CLUSTER RESPONSE



16.2.12 GET_DEVICEID COMMAND

The GET_DEVICEID command returns the hardware ID of the device.

FIGURE 16-22: GET_DEVICEID COMMAND

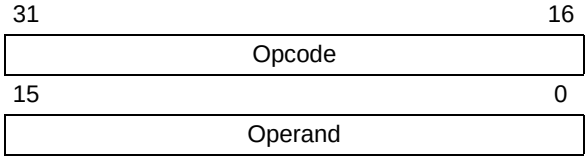
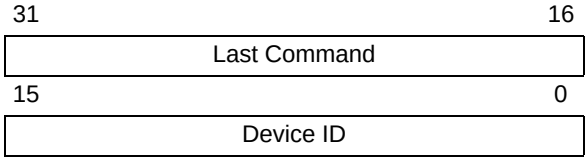


TABLE 16-13: GET_DEVICEID FORMAT

Field	Description
Opcode	0xA
Operand	Not used

Expected Response (1 word):

FIGURE 16-23: GET_DEVICEID RESPONSE



16.2.13 GET_CHECKSUM COMMAND

GET_CHECKSUM returns the sum of all the bytes, starting at the address argument up to the length argument. The result is a 32-bit word.

FIGURE 16-24: GET_CHECKSUM COMMAND

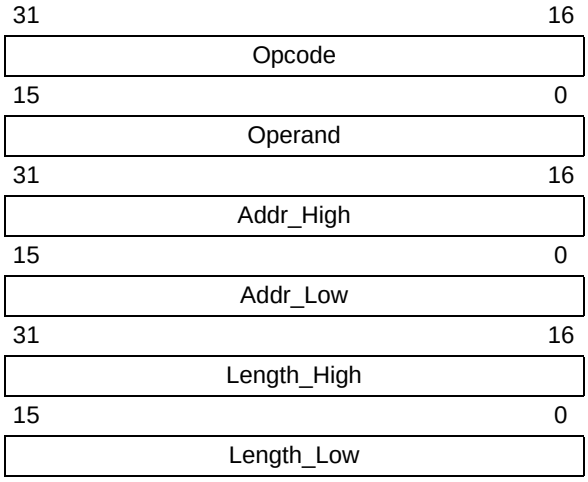
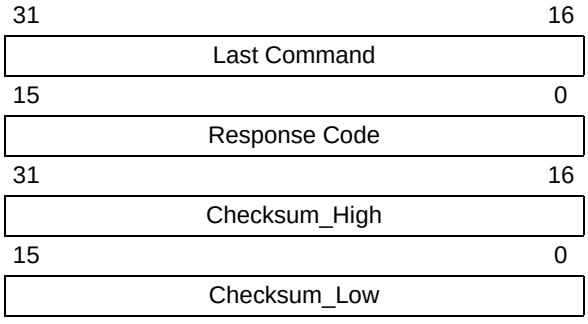


TABLE 16-14: GET_CHECKSUM FORMAT

Field	Description
Opcode	0x0C
Operand	Not used
Addr_High	High-order 16 bits of the 32-bit starting address of the data to calculate the checksum for
Addr_Low	Low-order 16 bits of the 32-bit starting address of the data to calculate the checksum for
Length_High	High-order 16 bits of the 32-bit length of data to calculate the checksum for, in bytes
Length_Low	Low-order 16 bits of the 32-bit length of data to calculate the checksum for, in bytes

Expected Response (1 word):

FIGURE 16-25: GET_CHECKSUM RESPONSE



16.2.14 DOUBLE_WORD_PROGRAM COMMAND

DOUBLE_WORD_PROGRAM instructs the PE to program two, 32-bit words at the specified address. The address must be an aligned two-word boundary (bit 0 must be '0'). If not, the command will return a FAIL response value and no data will be programmed.

FIGURE 16-26: DOUBLE_WORD_PROGRAM COMMAND

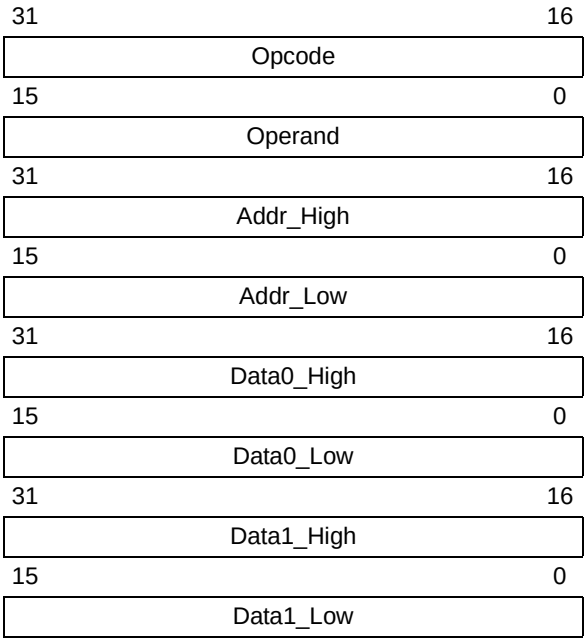
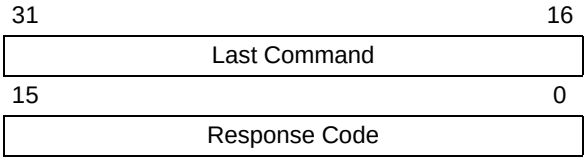


TABLE 16-15: DOUBLE_WORD_PROGRAM FORMAT

Field	Description
Opcode	0x0E
Operand	Not used
Addr_High	High-order 16 bits of the 32-bit starting address
Addr_Low	Low-order 16 bits of the 32-bit starting address
Data0_High	High-order 16 bits of Data Word 0
Data0_Low	Low-order 16 bits of Data Word 0
Data1_High	High-order 16 bits of Data Word 1
Data1_Low	Low-order 16 bits of Data Word 1

Expected Response (1 word):

FIGURE 16-27: DOUBLE_WORD_PROGRAM RESPONSE



16.3 JTAGEN Configuration Bit Programming

For the PIC32MM00XXGPL0XX family of devices, if the JTAGEN Configuration bit in the FICD or AFICD Configuration Word is programmed, the MCHP TAP controller must be selected immediately after any PROGRAM command (such as DOUBLE_WORD_PROGRAM or ROW_PROGRAM). This can be done by using a SendCommand(MTAP_SW_MTAP) pseudo operation. After 400 µS, the EJTAG TAP controller can be selected again by calling the SendCommand(MTAP_SW_ETAP) pseudo operation.

17.0 CHECKSUM

17.1 Theory

The checksum is calculated as the 32-bit summation of all bytes (8-bit quantities) in Program Flash Memory, Boot Flash Memory (except device Configuration Words), the Device ID register with applicable mask and the device Configuration Words with applicable masks. Then, 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 FICD register of the PIC32MM0064GPL036 device. The mask value for this register is:

`mask_value_FCID` = 0xFFFFFFF4

REGISTER 17-1: FICD REGISTER OF PIC32MM0064GPL036

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-1	r-1	r-1	r-1	R/P-1
	—	—	—	—	—	—	—	BWP
23:16	r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
	—	—	—	—	—	—	—	—
15:8	r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
	—	—	—	—	—	—	—	—
7:0	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
	—	—	—	ICESEL<1:0>		FJTAGEN	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 lists the mask values of the device Configuration registers and Device ID register to be used in the checksum calculations for PIC32MM devices.

TABLE 17-1: DEVICE CONFIGURATION REGISTER MASK VALUES OF PIC32MM DEVICES

	Configuration Words						DEVID
	FDEVOPT	FICD	FPOR	FWDT	FOSCEL	FSEC	
Register Mask	0xFFFFFFFF	0xFFFFFFFC	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0x3FFFFFFF	0x0FFFF000

17.3 Algorithm

Figure 17-1 illustrates an example of a high-level algorithm for calculating the checksum for a PIC32MM device which demonstrates one method of deriving 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 PIC32MM checksum is calculated as the 32-bit summation of all bytes (8-bit quantities) in Program Flash Memory, Boot Flash Memory (except device Configuration Words), the Device ID register with applicable mask and the device Configuration Words with applicable masks.

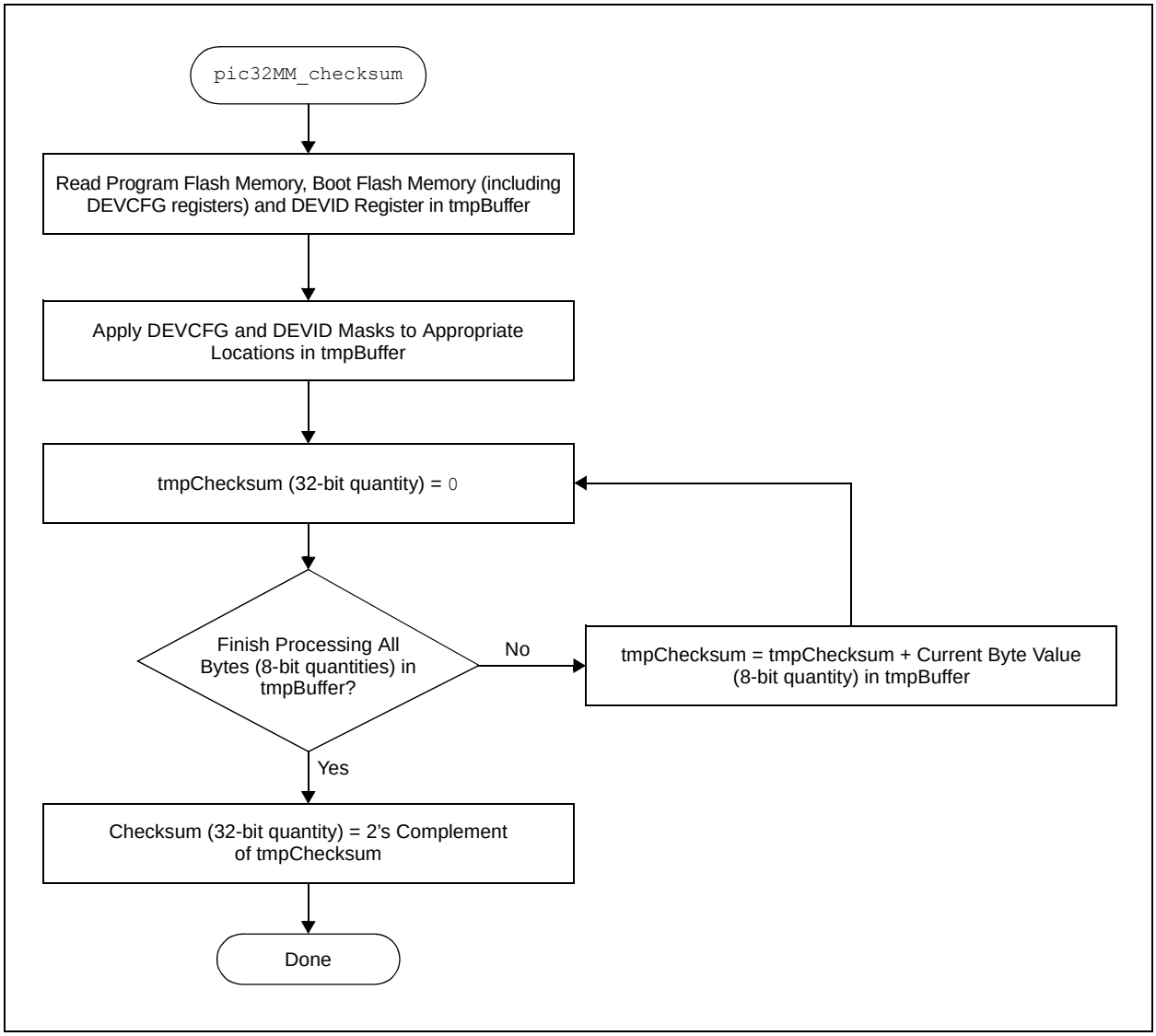
Then. the 2's complement of the summation is calculated. This final 32-bit number is presented as the checksum.

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

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 (see **Section 18.0 "Configuration Memory Device ID and Unique Device Identifier"** for more information).

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



The formula to calculate the checksum for a PIC32MM device is provided in Equation 17-1.

EQUATION 17-1: CHECKSUM FORMULA

$$Checksum = 2's\ Complement\ (PF + BF + DCR + DIR)$$

Where:

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

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

DCR = 32-bit summation of bytes (FDEVOPT + FICD + FPOR + FWDT + FOSCSEL + FSEC)

DIR = 32-bit summation of bytes

MASKID = Mask value from Table 17-1

17.4 Example of Checksum Calculation

The following five sections demonstrate a checksum calculation for the PIC32MM0064GPL036 device using Equation 17-1.

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

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

Each item on the right side of the equation (PF, BF, DCR, DIR) is individually calculated. After deriving the values, the final value of the checksum can be calculated.

17.4.1 CALCULATING FOR “PF” IN THE CHECKSUM FORMULA

The size of Program Flash Memory is 64 Kbytes, which equals 65536 bytes. Since the Program Flash Memory is assumed to be in an erased state, the value of “PF” is resolved through the following calculation:

PF = 0xFF + 0xFF + … 65536 times

PF = 0x00FF0000 (32-bit number)

17.4.2 CALCULATING FOR “BF” IN THE CHECKSUM FORMULA

The size of the Boot Flash Memory is 6 Kbytes, which equals 6144 bytes. However, the last 256 bytes are device Configuration registers and reserved locations, which are treated separately. Therefore, the number of bytes in Boot Flash that we consider in this step is 5888. Since the Boot Flash is assumed to be in an erased state, the value of “BF” is resolved through the following calculation:

BF = 0xFF + 0xFF + … 5888 times

BF = 0x0016E900 (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 Configuration register (as read by the PIC32MM 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 = 0x000003D6 (32-bit number)

TABLE 17-2: DCR CALCULATION EXAMPLE

Register	POR Default Value	Mask	POR Default Value & Mask	32-Bit Summation of Bytes
FDEVOPT	FFFFFFFF	FFFFFFFF	FFFFFFFF	0x000003FC
FICD	FFFFFFFF	FFFFFFFC	FFFFFFFC	0x000003F9
FPOR	FFFFFFFF	FFFFFFFF	FFFFFFFF	0x000003FC
FWDT	FFFFFFFF	FFFFFFFF	FFFFFFFF	0x000003FC
FOSCEL	FFFFFFFF	FFFFFFFF	FFFFFFFF	0x000003FC
FSEC	FFFFFFFF	3FFFFFFF	3FFFFFFF	0x0000033C
Total of the 32-Bit Summation of Bytes =				**0x00001725**

17.4.4 CALCULATING FOR “DIR” IN THE CHECKSUM FORMULA

The value of the 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	0x07708053	0x0FFFF000	0x07708000	0x000000F7

17.4.5 COMPLETING THE CHECKSUM CALCULATION

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

17.4.6 CHECKSUM VALUES WHILE DEVICE IS CODE-PROTECTED

Since the device Configuration Words are not readable while the PIC32MM devices are in the code-protected state, the checksum values are zeros for all devices.

EXAMPLE 17-1: CHECKSUM CALCULATION PROCESS

1. First, *temp = PF + BF + DCR + DIR*, which translates to:

   temp = 0x00FF0000 + 0x0016E900 + 0x00001725 + 0x000000F7
2. Adding all four values results in *temp* being equal to 0x0116011C.
3. Next, the 1’s complement of *temp*, called *temp1*, is calculated:

   temp1 = 2’s complement (temp), which is now equal to 0xFEE9FEE3

   Checksum = 2’s complement (temp), which is *Checksum = temp1 + 1*, resulting in 0xFEE9FEE4

18.0 CONFIGURATION MEMORY DEVICE ID AND UNIQUE DEVICE IDENTIFIER

PIC32MM 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 **"Special Features"** chapter in the specific device data sheet for a full list of available features, Configuration bits and the Device ID register.

For the current Silicon Revision and Revision ID for a particular device, please refer to the related Family Silicon Errata and Data Sheet Clarification. These documents are available from the Microchip web site by visiting, http://www.microchip.com/PIC32 and navigating to: *Documentation > Errata*.

18.1 Device ID

The Device ID region of memory can be used to determine variant and manufacturing information about the chip. This region of memory is read-only and can be read when code protection is enabled. The DEVID register identifies the specific part number of the device, the device silicon revision and the Manufacture ID.

Table 18-1 lists the identification information for each device. Table 18-2 shows the Device ID register and Table 18-3 describes the bit field of each register.

TABLE 18-1: DEVICE IDs

Device	DEVID
PIC32MM0016GPL020	0x6B04
PIC32MM0032GPL020	0x6B0C
PIC32MM0064GPL020	0x6B14
PIC32MM0016GPL028	0x6B02
PIC32MM0032GPL028	0x6B0A
PIC32MM0064GPL028	0x6B12
PIC32MM0016GPL036	0x6B06
PIC32MM0032GPL036	0x6B0B
PIC32MM0064GPL036	0x6B16
PIC32MM0064GPM028	0x7708
PIC32MM0128GPM028	0x7710
PIC32MM0256GPM028	0x7718
PIC32MM0064GPM036	0x770A
PIC32MM0128GPM036	0x7712
PIC32MM0256GPM036	0x771A
PIC32MM0064GPM048	0x772C
PIC32MM0128GPM048	0x7734
PIC32MM0256GPM048	0x773C
PIC32MM0064GPM064	0x770E
PIC32MM0128GPM064	0x7716
PIC32MM0256GPM064	0x771E

TABLE 18-2: PIC32MM DEVICE ID REGISTERS

Address	Name	Bit															
		31/15	30/14	29/13	28/12	27/11	26/10	25/9	24/8	23/7	22/6	21/5	20/4	19/3	18/2	17/1	16/0
0xBF803B20	ID	VER<3:0>				DEVID<15:4>											
		DEVID<3:0>				Manufacturer ID<11:0>											

TABLE 18-3: DEVICE ID BIT FIELD DESCRIPTIONS

Bit Field	Description
VER<3:0>	The device silicon revision
DEVID<15:0>	Encodes the individual ID of the device
Manufacturing ID<11:0>	0x053

18.2 Unique Device Identifier (UDID)

All PIC32MM devices are individually encoded during final manufacturing with a Unique Device Identifier or UDID. The UDID cannot be erased by the Chip Erase command or any other user-accessible means. This feature allows for manufacturing traceability of Microchip devices in applications where this is a requirement. It may also be used by the application manufacturer for any number of things that may require unique identification, such as:

- Tracking the device
- Unique serial number
- Unique security key

The UDID comprises 5 consecutive, 32-bit read-only locations, located between 0x1FC41840 and 0x1FC41850 in the device configuration space. When concatenated, this data forms a 160-bit identifier. Table 18-4 lists the addresses of the Identifier Words.

TABLE 18-4: UDID ADDRESSES

UDID	Address	Description
UDID1	0x1FC41840	UDID Word 1
UDID2	0x1FC41844	UDID Word 2
UDID3	0x1FC41848	UDID Word 3
UDID4	0x1FC4184C	UDID Word 4
UDID1	0x1FC41850	UDID Word 5

Note: The UDID value for a given part is only unique within a device family. Refer to the Family Devices table **(Table 1)** in the device data sheet for the list of devices in the family.

18.3 Device Configuration

In PIC32MM devices, the Configuration Words select various device configurations that are set at device Reset prior to execution of any code. These values are located at the highest locations of Boot Flash Memory (BFM), and since they are part of the program memory, are included in the programming file along with executable code and program constants. The names and locations of these Configuration Words are listed in Table 18-1 through Table 18-5. Table 18-5 lists the Configuration Word locations for PIC32MM family devices.

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).

Note: Writing a Flash location more than once without an erase violates the Flash specifications and may reduce the Flash panel life. Due to the ECC implementation, rewriting a Flash location with different data will cause an ECC error when that location is read

After programming the Configuration Words, a device Reset will cause the new values to be loaded into the Configuration registers. Because of this, the programmer should program the Configuration Words prior to verification of the device. The final step is programming the code protection Configuration Word.

These Configuration Words determine the oscillator source. If using the 2-Wire Enhanced ICSP mode, the Configuration Words are ignored and the device will always use the FRC; however, in 4-Wire mode, this is not the case. If an oscillator source is selected by the Configuration Words that is not present on the device after Reset, the programmer will not be able to perform Flash operations on the device after it is reset. See the **"Special Features"** chapter in the specific device data sheet for details regarding oscillator selection using the Configuration Words.

TABLE 18-5: PRIMARY CONFIGURATION WORDS

Configuration Word	Physical Address	
__(1)	0x1FC017C0	BCFG4(2)
FDEVOPT	0x1FC017C4	
FICD	0x1FC017C8	BCFG3(2)
FPOR	0x1FC017CC	
FWDT	0x1FC017D0	BCFG2(2)
FOSCEL	0x1FC017D4	
FSEC	0x1FC017D8	BCFG1(2)
__(1)	0x1FC017DC	
FSIGN	0x1FC017E0	BCFG0(2)
__(1)	0x1FC017E4	

Note 1: These locations do not contain device configuration information, but must be programmed as they are part of a double word shared with configuration information. It is recommended they be programmed with a value of 0xFFFFFFFF.

2: Each BCFGx is comprised of 2 Flash locations forming a double word. Both locations in the double word must be programmed together with a double word or a row program operation.

TABLE 18-6: ALTERNATE CONFIGURATION WORDS

Configuration Word	Physical Address	
—(1)	0x1FC01740	ABCFG4(2)
AFDEVOPT	0x1FC01744	
AFICD	0x1FC01748	ABCFG3(2)
AFPOR	0x1FC0174C	
AFWDT	0x1FC01750	ABCFG2(2)
AFOSCEL	0x1FC01754	
AFSEC	0x1FC01758	ABCFG1(2)
—(1)	0x1FC0175C	
AFSIGN	0x1FC01760	ABCFG0(2)
—(1)	0x1FC01764	

Note 1: These locations do not contain device configuration information, but must be programmed as they are part of a double word shared with configuration information. It is recommended they be programmed with a value of 0xFFFFFFFF.

2: Each BCFGx is comprised of 2 Flash locations forming a double word. Both locations in the double word must be programmed together with a double word or a row program operation.

18.3.1 CONFIGURATION REGISTER REPROGRAMMING

Configuration registers must not be rewritten without being erased first. Reprogramming without a preceding erase operation will violate the Flash memory specification, degrading the Flash endurance and generating an ECC error.

18.3.2 ALTERNATE CONFIGURATION WORDS

PIC32MM devices contain two sets of Configuration Words: Primary and Alternate. The Primary Configuration Word is read from Flash following a device Reset. If an uncorrectable ECC error occurs during this read, the Alternate Configuration Word is read. In typical applications, the Primary and Alternate Configuration Words should be programmed to the same values.

For the purpose of checksum calculations, only the Primary Configuration Word is used.

18.4 Device Code Protection Bit (CP)

The PIC32MM families of devices feature code protection, which when enabled, prevents reading of Flash memory by an external programming device. Once code protection is enabled, it can only be disabled by erasing the device with the Chip Erase command (MCHP_ERASE).

When programming a device that has opted to utilize code protection, the programming device must perform verification prior to enabling code protection. Enabling code protection should be the last step of the programming process. Location of the code protection enable bits vary by device. Refer to the **"Special Features"** chapter in the specific device data sheet for details.

Note: Once code protection is enabled, the Flash memory can no longer be read and can only be disabled by an external programmer using the Chip Erase command (MCHP_ERASE).

18.5 Program Write Protection Bits (PWP)

The PIC32MM families of devices include write protection features, which prevent designated Boot and Program Flash Memory regions from being erased or written during program execution. Write protection is implemented through the SFRs in the Flash controller.

Certain steps may be required during initialization of the device by the external programmer prior to programming Flash regions. Refer to the **"Flash Program Memory"** chapter in the specific device data sheet for details.

19.0 TAP CONTROLLERS

TABLE 19-1: MCHP TAP INSTRUCTIONS

Command	Value	Description
MTAP_COMMAND	5'h0x07	TDI and TDO are connected to MCHP Command Shift register (see Table 19-2)
MTAP_SW_MTAP	5'h0x04	Switch TAP controller to Microchip TAP controller
MTAP_SW_ETAP	5'h0x05	Switch TAP controller to EJTAG TAP controller
MTAP_IDCODE	5'h0x01	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 provides the format of the status value returned.

19.1.1.2 MCHP_ASSERT_RST INSTRUCTION

MCHP_ASSERT_RST performs a persistent device Reset. It is similar to asserting and holding MCLR. Its associated status bit is DEVRST.

19.1.1.3 MCHP_DE_ASSERT_RST INSTRUCTION

MCHP_DE_ASSERT_RST removes the persistent device Reset. It is similar to deasserting 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.

19.1.2 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'h0x00	NOP and return status.
MCHP_ASSERT_RST	8'h0xD1	Requests the Reset controller to assert a device Reset.
MCHP_DE_ASSERT_RST	8'h0xD0	Removes the request for a device Reset, which causes the Reset controller to deassert the device Reset if there is no other source requesting a Reset (i.e., MCLR).
MCHP_ERASE	8'h0xFC	Causes the Flash controller to perform a Chip Erase.

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	0	CFGRDY	FCBUSY	0	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 = An error occurred during an NVM operation
0 = An error did not occur during an 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 **Unimplemented:** Read as '0'

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

TABLE 19-4: EJTAG TAP INSTRUCTIONS

Command	Value	Description
ETAP_ADDRESS	5'h0x08	Selects the Address register.
ETAP_DATA	5'h0x09	Selects the Data register.
ETAP_CONTROL	5'h0x0A	Selects the EJTAG Control register.
ETAP_EJTAGBOOT	5'h0x0C	Sets EjtagBrk, ProbEn and ProbTrap to '1' as the Reset value.
ETAP_FASTDATA	5'h0x0E	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<1:0> bits 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 stored 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 EJTAG 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 a Processor Low-Power mode
- Allows implementation-dependent processor and peripheral Resets

19.2.3.1 EJTAG Control Register (ECR)

The EJTAG Control register (see Register 19-1) is not updated/written in the Update-DR state unless a 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 Read/Write (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 the Capture-DR state.

REGISTER 19-1: ECR: EJTAG CONTROL REGISTER

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/W-0	R-0	R-0	U-0	U-0	U-0	U-0	U-0
	Rocc	Psz<1:0>		—	—	—	—	—
23:16	R-0	R-0	R-0	R/W-0	R-0	R/W-0	U-0	R/W-0
	VPED	Doze	Halt	PerRst	PrnW	PrACC	—	PrRst
15:8	R/W-0	R/W-0	U-0	R/W-0	U-0	U-0	U-0	U-0
	ProbEn	ProbTrap	—	EjtagBrk	—	—	—	—
7:0	U-0	U-0	U-0	U-0	R-0	U-0	U-0	U-0
	—	—	—	—	DM	—	—	—

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

bit 31-29 See **Note 1**

bit 28-24 **Unimplemented:** Read as '0'

bit 23-19 See **Note 1**

bit 18 **PrACC:** Pending Processor Access and Control bit

This bit indicates a pending processor access and controls finishing of a pending processor access. A write of '0' finishes processor access if pending. A write of '1' is ignored. A successful Fastdata access will clear this bit.

1 = Pending processor access

0 = No pending preprocessor access

bit 17 **Unimplemented:** Read as '0'

bit 16 See **Note 1**

bit 15 **ProbEn:** Processor Access Service Control bit

This bit controls where the probe handles accesses to the DMSEG segment through servicing of processor accesses.

1 = Probe services processor accesses

0 = Probe does not service processor access

bit 14 **ProbTrap:** Debug Exception Vector Control Location bit

This bit controls the location of the debug exception vector.

1 = 0xFF200200

0 = 0xBFC00480

bit 13 **Unimplemented:** Read as '0'

bit 12 **EjtagBrk:** Debug Interrupt Exception Request bit

This bit requests a debug interrupt exception to the processor when this bit is written as '1'. A write of '0' is ignored.

1 = A debug interrupt exception request is pending

0 = A debug interrupt exception request is not pending

bit 11-4 **Unimplemented:** Read as '0'

bit 3 See **Note 1**

bit 2-0 **Unimplemented:** Read as '0'

Note 1: For descriptions of these bits, please refer to the Imagination Technologies Limited web site (www.imgtec.com).

19.2.4 ETAP_EJTAGBOOT COMMAND

The ETAP_EJTAGBOOT command causes the processor to fetch code from the debug exception vector after a Reset. This allows the programmer to send instructions to the processor to execute, instead of the processor fetching them from the normal Reset vector. 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 the debug vector in EJTAG memory at 0xFF200200
- Service of the processor access is indicated because ProbEn is set

With this configuration in place, an interrupt exception will occur and the processor will fetch the handler from the DMSEG at 0xFF200200. Since ProbEn is set, the processor will wait for the instruction to be provided by the probe.

19.2.5 ETAP_FASTDATA COMMAND

The ETAP_FASTDATA command provides a mechanism for quickly transferring data between the processor and the probe. The width of the Fastdata register is one 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 fast data 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 "fast data area" specifies the legal range of DMSEG segment addresses (0xFF200000 to 0xFF20000F) that can be used for uploads and downloads. The Data and Fastdata registers (selected with the FASTDATA instruction) allow efficient completion of pending fast data area accesses.

During fast data uploads and downloads, the processor will stall on accesses to the fast data 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 fast data area address was used).

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

As noted above, the following two conditions must be true for the fast data access to succeed:

- PrACC must be '1' (i.e., there must be a pending processor access)
- The fast data operation must use a valid fast data area address in the DMSEG segment (0xFF200000 to 0xFF20000F)

20.0 AC/DC CHARACTERISTICS AND TIMING REQUIREMENTS

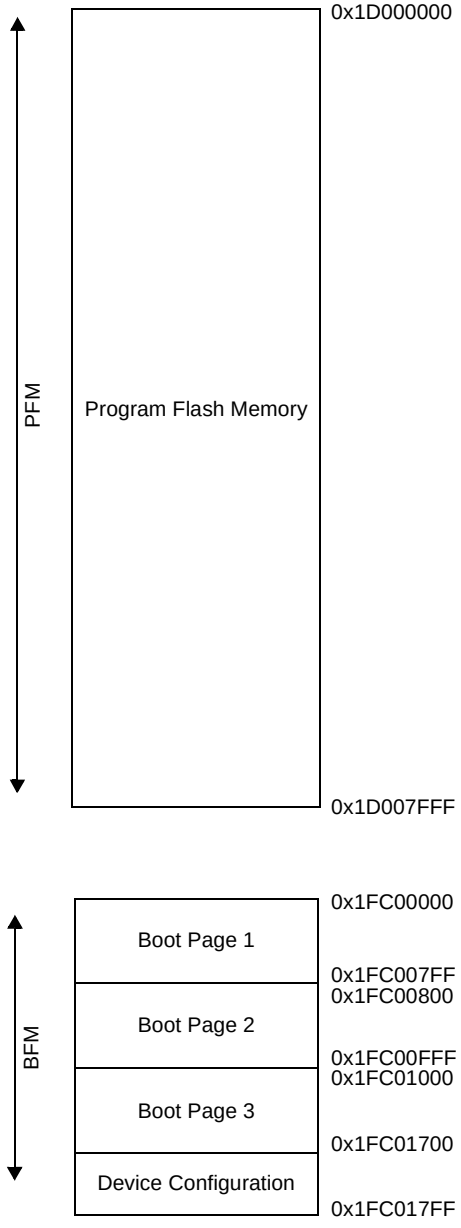
TABLE 20-1: AC/DC CHARACTERISTICS AND TIMING REQUIREMENTS

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	—	—	V	See **Note 1**
D113	IDDP	Supply Current During Programming	—	—	mA	See **Note 1**
D114	IPEAK	Instantaneous Peak Current During Start-up	—	—	mA	See **Note 1**
D031	VIL	Input Low Voltage	—	—	V	See **Note 1**
D041	VIH	Input High Voltage	—	—	V	See **Note 1**
D080	VOL	Output Low Voltage	—	—	V	See **Note 1**
D090	VOH	Output High Voltage	—	—	V	See **Note 1**
D012	CIO	Capacitive Loading on I/O Pin (PGEDx)	—	—	pF	See **Note 1**
D013	CF	Filter Capacitor Value on VCAP	—	—	µF	See **Note 1**
P1	TPGC	Serial Clock (PGECx) Period	100	—	ns	
P1A	TPGCL	Serial Clock (PGECx) Low Time	40	—	ns	
P1B	TPGCH	Serial Clock (PGECx) High Time	40	—	ns	
P6	TSET2	VDD ↑ Setup Time to $\overline{\text{MCLR}}$ ↑	100	—	ns	
P7	THLD2	Input Data Hold Time from $\overline{\text{MCLR}}$ ↑	500	—	ns	
P9A	TDLY4	PE Command Processing Time	40	—	µs	
P9B	TDLY5	Delay between PGEDx ↓ by the PE to PGEDx Released by the PE	15	—	µs	
P11	TDLY7	Chip Erase Time	—	—	ms	See **Note 1**
P12	TDLY8	Page Erase Time	—	—	ms	See **Note 1**
P13	TDLY9	Row Programming Time	—	—	ms	See **Note 1**
P14	TR	$\overline{\text{MCLR}}$ Rise Time to Enter ICSP™ mode	—	1.0	µs	
P15	TVALID	Data Out Valid from PGECx ↑	10	—	ns	
P16	TDLY8	Delay between Last PGECx ↓ and $\overline{\text{MCLR}}$ ↓	0	—	s	
P17	THLD3	$\overline{\text{MCLR}}$ ↓ to VDD ↓	—	100	ns	
P18	TKEY1	Delay from First $\overline{\text{MCLR}}$ ↓ to First PGECx ↑ for Key Sequence on PGEDx	40	—	ns	
P19	TKEY2	Delay from Last PGECx ↓ for Key Sequence on PGEDx to Second $\overline{\text{MCLR}}$ ↑	40	—	ns	
P20	TMCLRH	$\overline{\text{MCLR}}$ High Time	—	500	µs	

Note 1: Refer to the **"Electrical Characteristics"** chapter in the specific device data sheet for the Minimum and Maximum values for this parameter.

APPENDIX A: PIC32MM FLASH MEMORY MAP

FIGURE A-1: FLASH MEMORY MAP



Note: The memory map shown is for reference only. Refer to the **"Memory Organization"** chapter in the specific device data sheet for the memory map for your device.

APPENDIX B: HEX FILE FORMAT

Flash programmers process the standard hexadecimal (Hex) format used by the Microchip development tools. The format supported is the Intel® HEX32 Format (INHX32). Please refer to **Section 1.7.5 "Hex File Formats"** in the *"MPASM™ Assembler, MPLINK™ Object Linker, MPLIB™ Object Librarian User's 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 (EOF) 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.

APPENDIX C: REVISION HISTORY

Revision A (September 2015)

This is the initial version of the document supporting PIC32MMXXXXGPLXXX devices.

Revision B (October 2015)

PIC32MMXXXXGPMXXX devices have been added.

Revision C (May 2016)

Added **Section 16.3 "JTAGEN Configuration Bit Programming"** and updated Table 18.1.

Removed PIC32MM0XXXGPM0XX devices from document.

Revision D (March 2017)

Added devices to the list in **Section 1.0 "Device Overview"**.

Added devices to Table 18-1.

Updated **Section 5.3 "2-Wire ICSP Details"**.

Removed **Section 5.3.2 "2-Phase ICSP"**.

Removed Figure 5-5 and Figure 6-8.

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.

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.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Trademarks

ISBN: 978-1-5224-1461-2

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

Austin, TX
Tel: 512-257-3370

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

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

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

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
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 - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
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: 86-21-3326-8000
Fax: 86-21-3326-8021

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

China - Shenzhen
Tel: 86-755-8864-2200
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

ASIA/PACIFIC

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

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

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: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

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: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
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

Finland - Espoo
Tel: 358-9-4520-820

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

France - Saint Cloud
Tel: 33-1-30-60-70-00

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

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

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

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

Italy - Padova
Tel: 39-049-7625286

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

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

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

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820