

PIC32MX Flash Programming Specification

1.0 DEVICE OVERVIEW

This document defines the programming specification for the PIC32MX family of 32-bit microcontroller devices. This programming specification is designed to guide External Programmer Tools developers. End customers developing applications for PIC32MX devices should use development tools that already provide support for device programming.

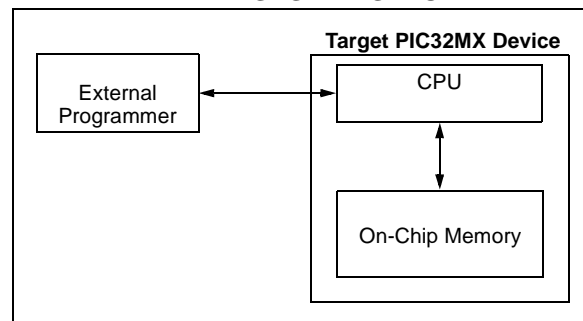
This document includes programming specifications for the PIC32MX family of devices.

2.0 PROGRAMMING OVERVIEW

All PIC32MX devices can be programmed via two primary methods – Self-programming and external tool programming. The self-programming method requires that the target device already contain executable code with necessary logic to complete the programming sequence. The external tool programming method, on the other hand, does not require any code in the target device – it can program all target devices with or without any executable code. This document describes the external tool programming method only. Refer to the PIC32MX Family Reference Manual (DS61132) and the PIC32MX Family Data Sheet (DS61143) to learn more about the self-programming method.

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

FIGURE 2-1: A PROGRAMMING SYSTEM SETUP



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

- 2-wire In-Circuit Serial Programming™ (ICSP™)
- 4-wire JTAG

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

Each of these methods may or may not use a downloadable Program 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 detailed discussion of each step.

See Appendices for more specific details on EJTAG, programming commands and DC specs.

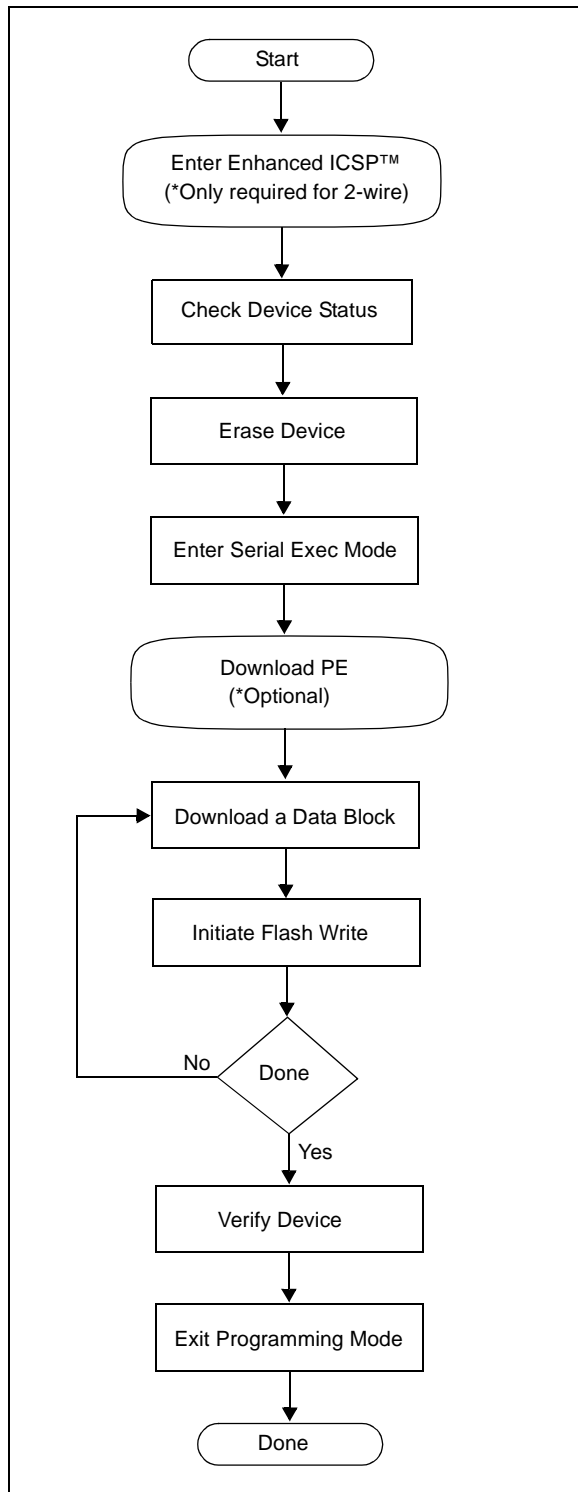
2.1 Assumptions

Both 2 and 4-wire interfaces use the EJTAG protocol to exchange data with the programmer. While this document provides a working description of this protocol as needed, advanced users are advised to refer to the EJTAG Specification by MIPS Technology, “MD00047”.

3.0 PROGRAMMING STEPS

All programmers must perform a common set of steps regardless of the actual method they are using. Figure 3-1 shows the steps required to program PIC32MX devices.

FIGURE 3-1: PROGRAMMING FLOW



The following sequence lists each step and provides a brief explanation. Detailed discussion of each step is provided in 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”** for more information.
2. Place the target device in Programming mode.
This step is not required if the 4-wire programming method is used. For 2-wire programming methods, the target device must be placed in a special Programming mode before executing further steps. See **Section 7.0 “Entering Programming Mode”** for more information.
3. Check the status of the device.
This step 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 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.
This step verifies that the device is not code-protected and boots the TAP controller in order to start sending and receiving data to the PIC32MX CPU. See **10.0 “Entering Serial Execution Mode”** for more information.
6. Download the Programming Executive (PE).
This step is not required if a method that does not require PE is used. The PE is a small block of executable code that is first downloaded into the RAM of the target device, which in turn receives and programs the actual data. See **11.0 “Downloading the Programming Executive (PE)”** for more information.
7. Download the block of data to program.
All methods with or without PE must download the desired programming data into a block of memory in RAM. See **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 **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 **14.0 “Verify Device Memory”** for more information.

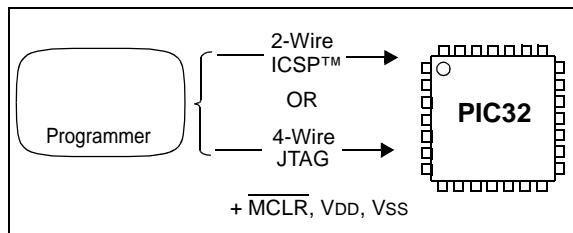
11. Exit the Programming mode.

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 **15.0 “Exiting Programming Mode”** for more information.

4.0 CONNECTING TO THE DEVICE

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

FIGURE 4-1: PROGRAMMING INTERFACES



4.1 4-Wire Interface

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

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

These signals are described in detail below. Refer to the data sheet of the particular device for the connection of the signals to chip pins.

4.1.1 TEST CLOCK INPUT (TCK)

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

4.1.2 TEST MODE SELECT INPUT (TMS)

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

4.1.3 TEST DATA INPUT (TDI)

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

4.1.4 TEST DATA OUTPUT (TDO)

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

TABLE 4-1: 4-WIRE INTERFACE PINS

Pin Name			
	Pin Name	Pin Type	Pin Description
MCLR	MCLR	P	Programming Enable
ENVREG	ENVREG	I	Enable for On-Chip Voltage Regulator
VDD and AVDD ⁽¹⁾	VDD	P	Power Supply
VSS and AVSS ⁽¹⁾	VSS	P	Ground
VDDCORE	VDDCORE	P	Regulated Power Supply for Core
TDI	TDI	I	Test Data In
TDO	TDO	O	Test Data Out
TCK	TCK	I	Test Clock
TMS	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).

PIC32MX

4.2 2-Wire Interface

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

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

These signals are described in detail below. Refer to the data sheet of the particular device for the connection of the signals to chip pins.

4.2.1 SERIAL PROGRAM CLOCK (PGCx)

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

4.2.2 SERIAL PROGRAM DATA (PGDx)

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

TABLE 4-2: 2-WIRE INTERFACE PINS

Pin Name			
	Pin Name	Pin Type	Pin Description
MCLR	MCLR	P	Programming Enable
ENVREG	ENVREG	I	Enable for On-Chip Voltage Regulator
VDD and AVDD ⁽¹⁾	VDD	P	Power Supply
VSS and AVSS ⁽¹⁾	VSS	P	Ground
VDDCORE	VDDCORE	P	Regulated Power Supply for Core
PGC1	PGC	I	Primary Programming Pin Pair: Serial Clock
PGD1	PGD	I/O	Primary Programming Pin Pair: Serial Data
PGC2	PGC	I	Secondary Programming Pin Pair: Serial Clock
PGD2	PGD	I/O	Secondary Programming Pin Pair: Serial Data

Legend: I = Input, O = Output, P = Power

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

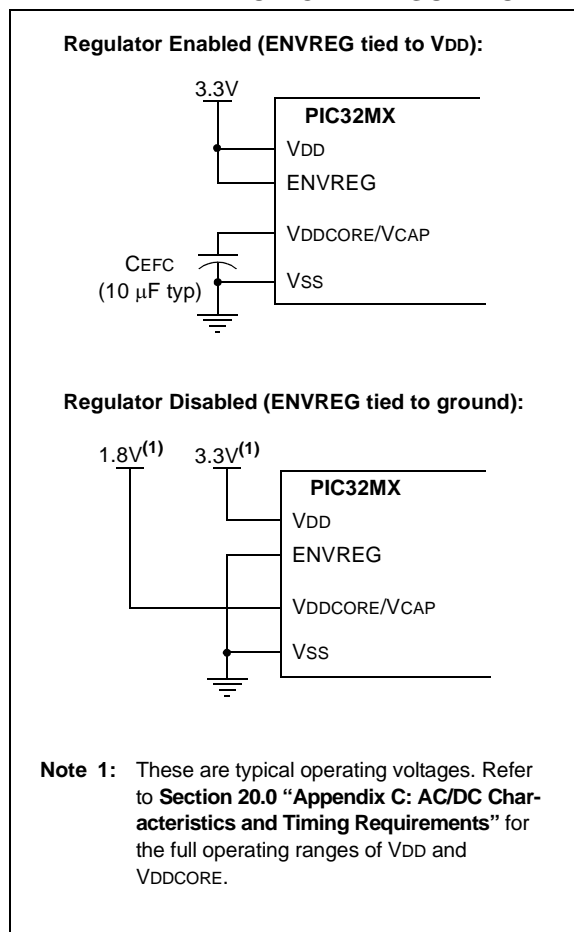
4.3 Power Requirements

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

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

The regulator provides power to the core from the other VDD pins. A low ESR capacitor (such as tantalum) must be connected to the VDDCORE pin (Figure 4-2). This helps to maintain the stability of the regulator. The specifications for core voltage and capacitance are listed in **Section 20.0 “Appendix C: AC/DC Characteristics and Timing Requirements”**.

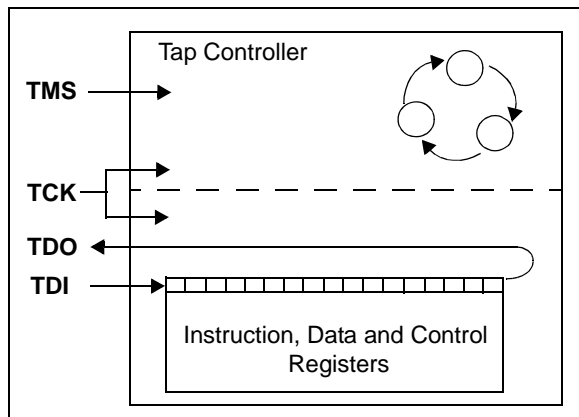
FIGURE 4-2: CONNECTIONS FOR THE ON-CHIP REGULATOR



5.0 EJTAG vs. ICSP

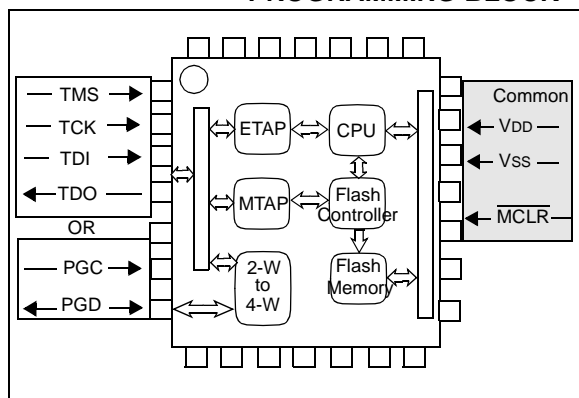
Programming is accomplished via the EJTAG module in the CPU core. EJTAG is connected to either the full set of JTAG pins, or a reduced 2-wire to 4-wire EJTAG interface. In both modes, programming of the PIC32MX Flash memory is accomplished through the ETAP controller. The TAP Controller uses the TMS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

FIGURE 5-1: TAP CONTROLLER



The basic concept of EJTAG that is used for programming is the use of a special memory area called DMSEG (FF200000h-FF2FFFFFh), which is only available when the processor is running in DEBUG mode. All instructions are serially shifted into an internal buffer, then loaded into the instruction register and executed by the CPU. Instructions are fed through the ETAP State machine 32 bits at a time.

FIGURE 5-2: BASIC PIC32MX PROGRAMMING BLOCK



2-Wire to 4-Wire:

Converts 2-wire ICSP interface to 4-wire JTAG.

- **ETAP**
 - Serially feeds instructions and data into CPU.
- **MTAP**
 - In addition to the EJTAG TAP (ETAP) controller, the PIC32MX device uses a second proprietary TAP controller for additional operations. The Microchip TAP (MTAP) controller supports two instructions relevant to programming. These are the MTAP_COMMAND and MTAP_SWTAP instructions, see Table 19-1 for a complete list of commands. The MTAP_COMMAND instruction provides a mechanism for a JTAG probe to send commands to the device via its data register.
 - The programmer sends commands by shifting in the MTAP_COMMAND instruction via the SendCommand pseudo op, and then sending MTAP_COMMAND DR commands via XferData pseudo op (see Table 19-2 for specific commands). The probe does not need to issue an MTAP_COMMAND instruction for every command shifted into the data register.
- **CPU**
 - The CPU executes instructions at 8 MHz via the internal oscillator.
- **Flash Controller**
 - The Flash Controller controls erasing and programming of the Flash memory on the device.
- **Flash Memory**
 - The PIC32MX device Flash memory is divided into two logical Flash partitions consisting of the Boot Flash Memory (BFM) and Program Flash Memory (PFM). The Boot Flash Memory (BFM) map extends from 1FC00000h to 1FC02FFFFh, and the Program Flash Memory (PFM) map extends from 1D000000h to 1D07FFFFh. Code storage begins with the BFM and supports up to 12K bytes, then continues with the PFM which supports up to 512K bytes. Table 5-1 shows the program memory size of each device variant. Each erase block, or page, contains 1K instructions (4 KBytes), and each program block, or row, contains 128 instructions (512 Bytes).
 - The last four implemented program memory locations in BFM are reserved for the Device Configuration registers.

TABLE 5-1: CODE MEMORY SIZE

PIC32MX Device	Boot Flash Memory Address (Bytes)	Program Flash Memory Address (Bytes)
PIC32MX300F032H	1FC00000h-1FC02FFFh (12KB)	1D000000h-1D007FFFh (32KB)
PIC32MX320F064H	1FC00000h-1FC02FFFh (12KB)	1D000000h-1D00FFFFh (64KB)
PIC32MX320F128H	1FC00000h-1FC02FFFh (12KB)	1D000000h-1D01FFFFh (128KB)
PIC32MX340F256H	1FC00000h-1FC02FFFh (12KB)	1D000000h-1D03FFFFh (256KB)
PIC32MX320F128L	1FC00000h-1FC02FFFh (12KB)	1D000000h-1D01FFFFh (128KB)
PIC32MX360F256L	1FC00000h-1FC02FFFh (12KB)	1D000000h-1D03FFFFh (256KB)
PIC32MX360F512L	1FC00000h-1FC02FFFh (12KB)	1D000000h-1D07FFFFh (512KB)

5.1 4-Wire JTAG Details

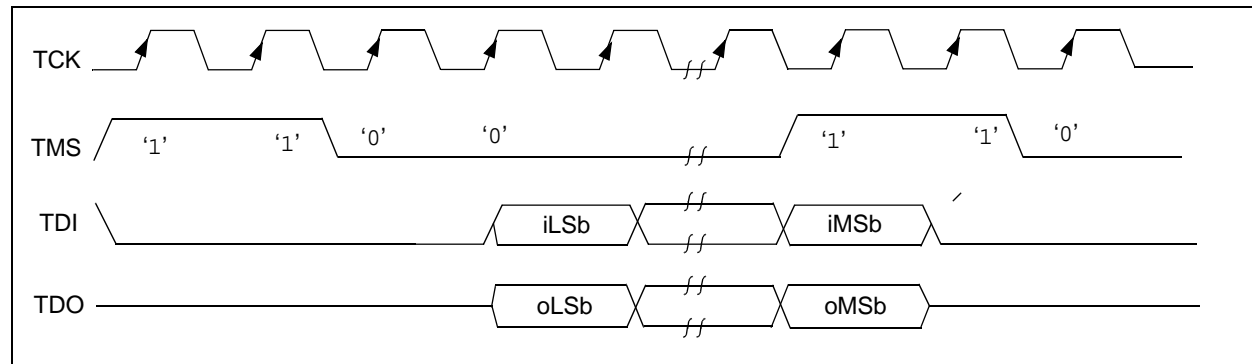
The 4-wire interface uses standard JTAG (IEEE 1149.1) interface signals.

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

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

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

FIGURE 5-3: 4-WIRE JTAG INTERFACE



5.2 2-wire ICSP Details

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

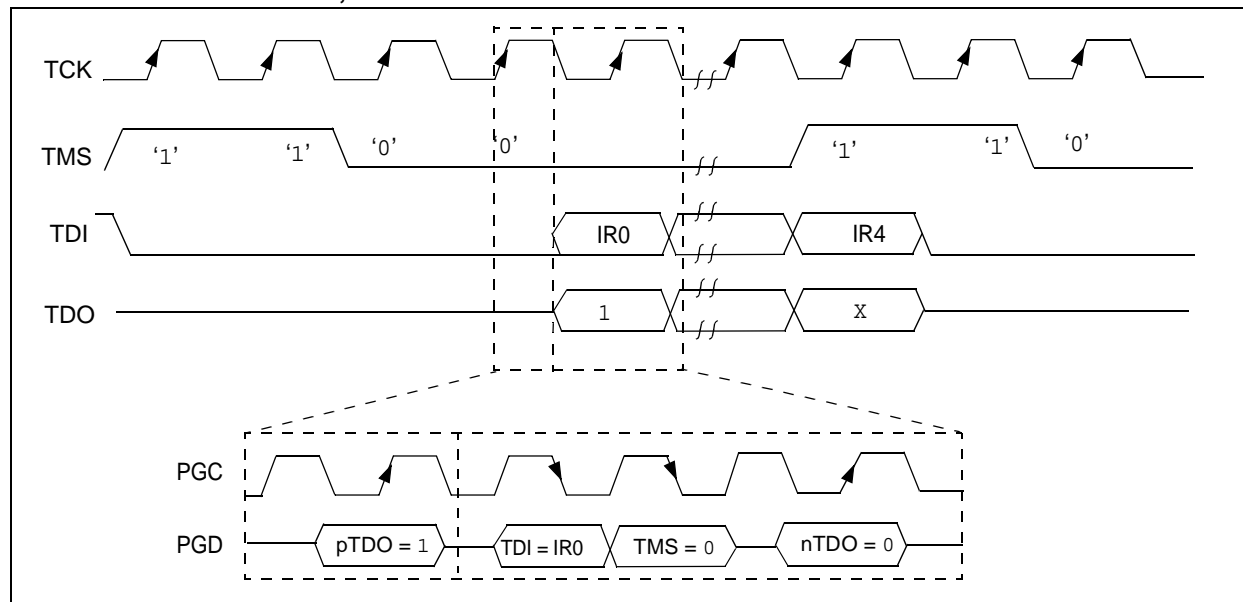
There are two possible modes of operation:

- 4-Phase ICSP
- 2-Phase ICSP

5.2.1 4-PHASE ICSP

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

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

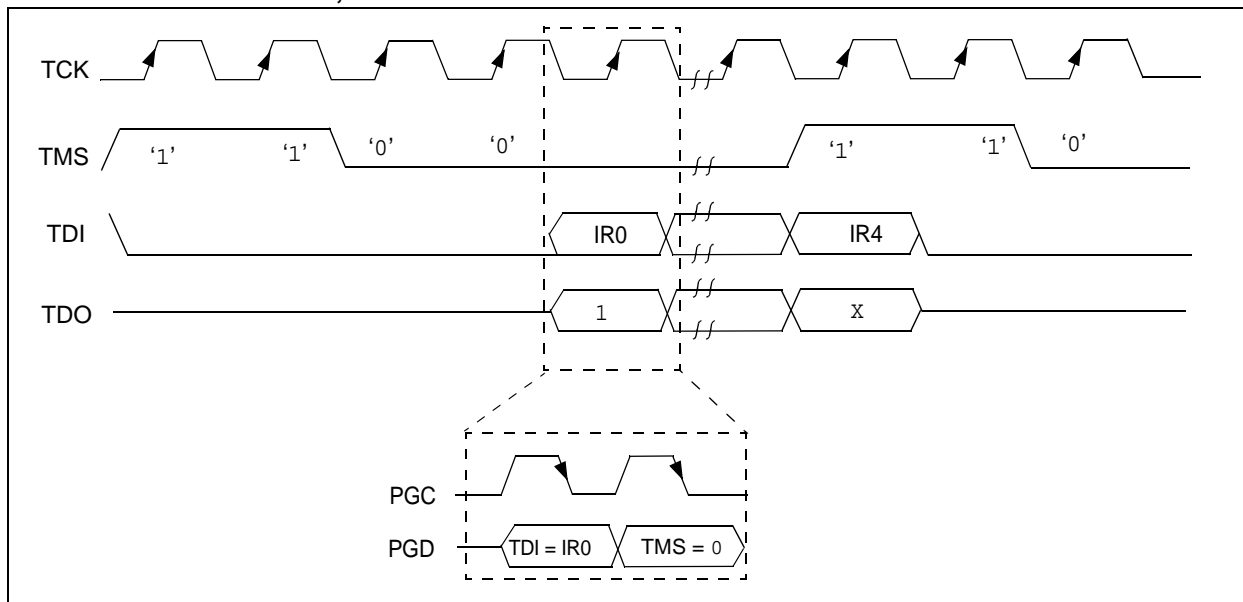


5.2.2 2-PHASE ICSP

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

Note: The packet is NOT actually executed until the first clock of the next packet.

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



6.0 PSEUDO OPERATIONS

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

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

6.1 SetMode Pseudo Operation

Format: SetMode (mode)

Purpose:

To set the EJTAG state machine to a specific state.

Description:

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

Restrictions:

None.

Example:

SetMode(6'b011111)

FIGURE 6-1: SetMode 4-WIRE

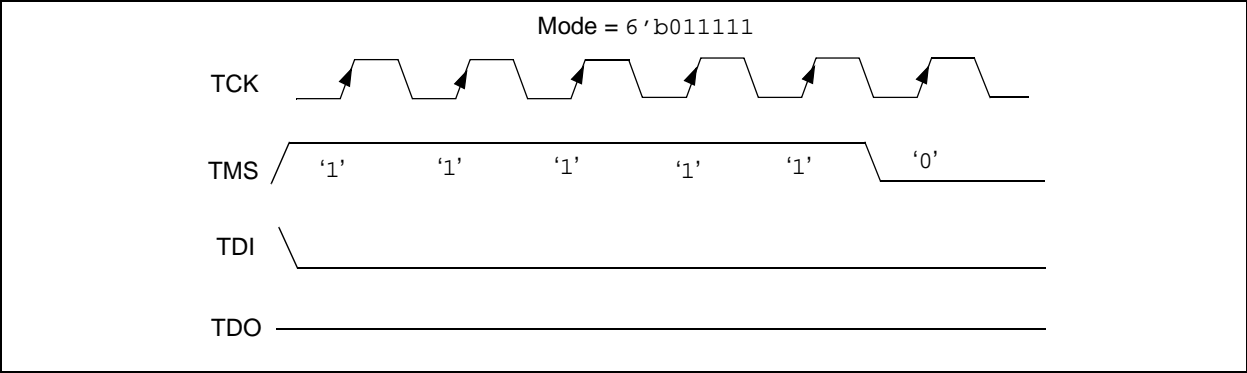
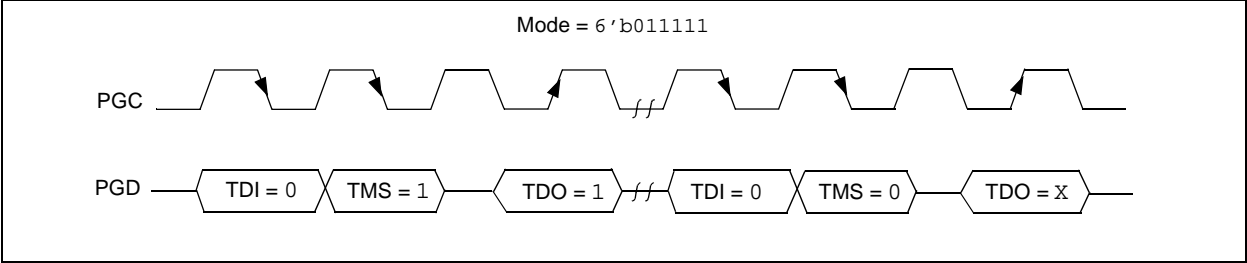


FIGURE 6-2: SetMode 2-WIRE



6.2 SendCommand Pseudo Operation

Format: SendCommand (command)

Purpose:

To send a command to select a specific TAP register.

Description:

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

Restrictions:

None.

Example:

SendCommand (5'h07)

FIGURE 6-3: SendCommand 4-WIRE

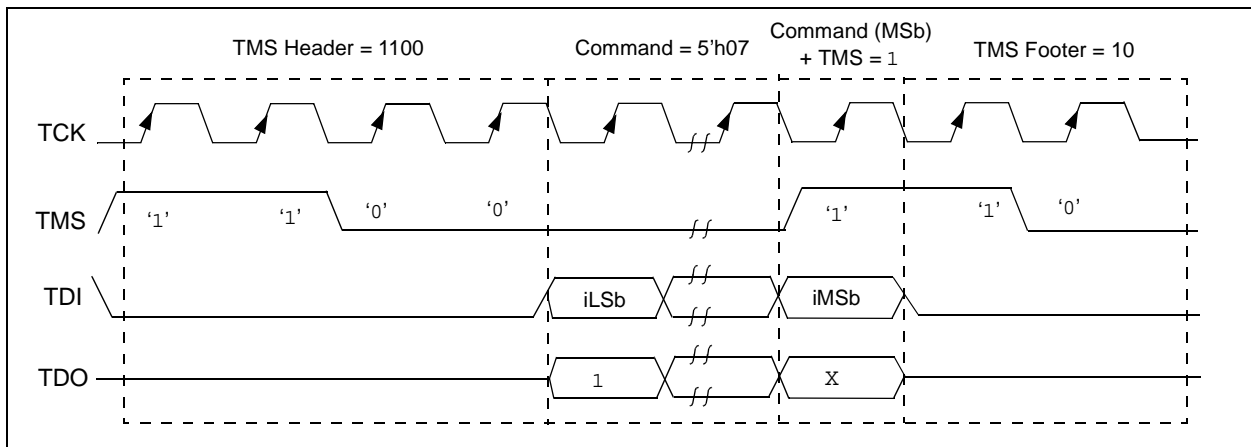
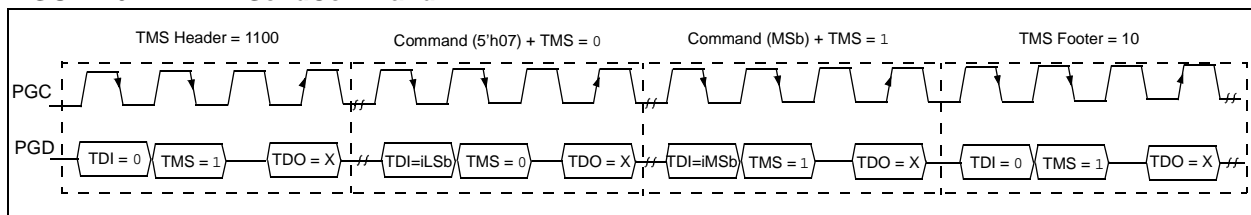


FIGURE 6-4: SendCommand 2-WIRE



6.3 XferData Pseudo Operation

Format: oData = XferData (iData)

Purpose:

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

Description:

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

Restrictions:

None.

Example:

oData = XferData (32'h12)

FIGURE 6-5: XferData 4-WIRE

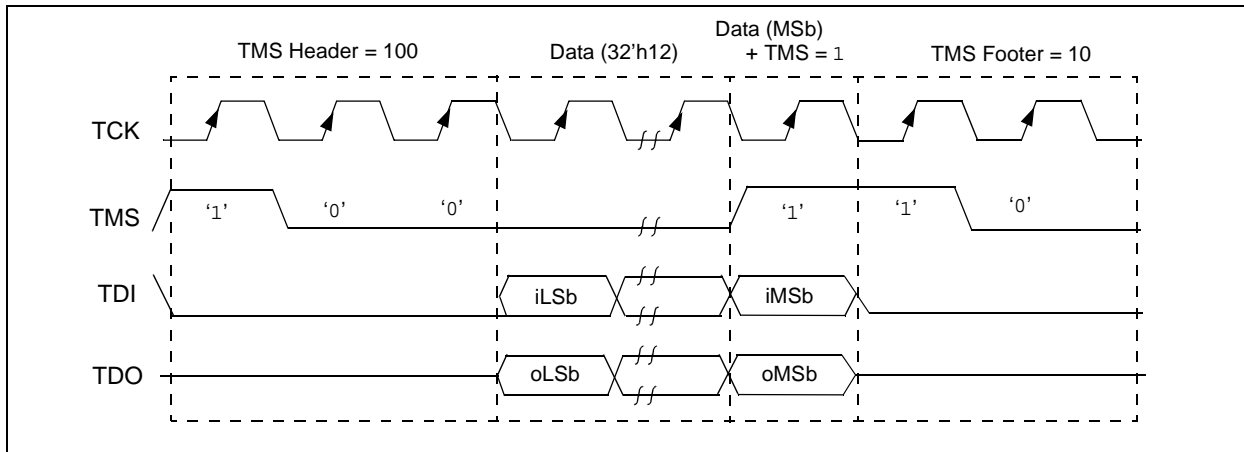
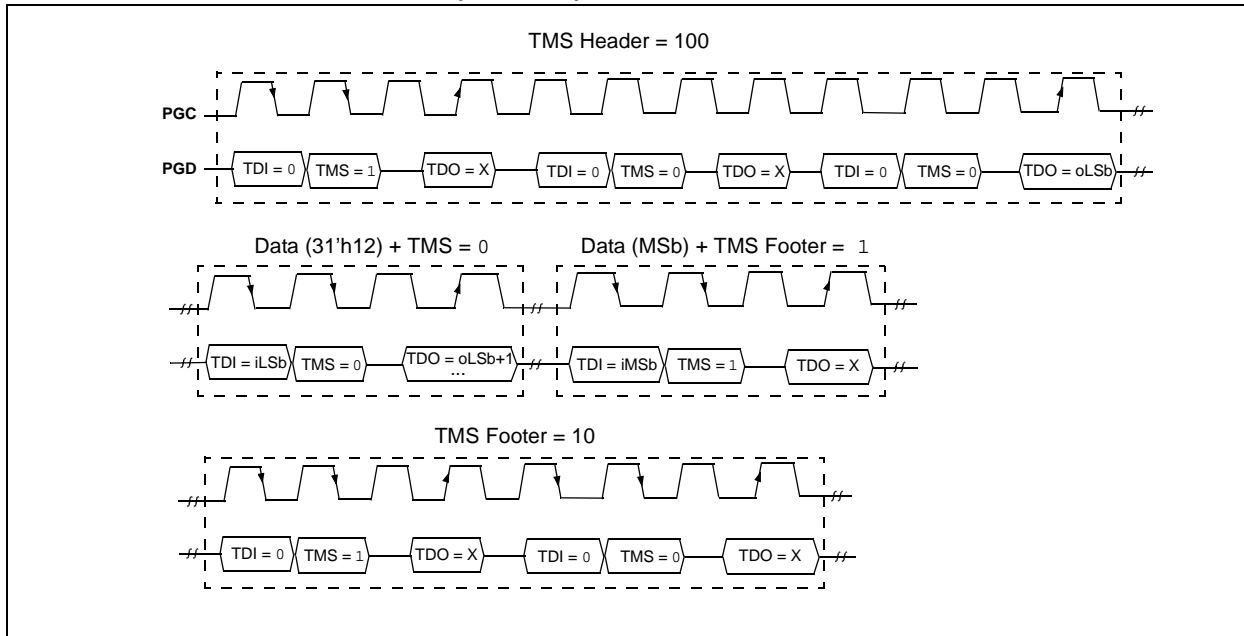


FIGURE 6-6: XferData 2-WIRE (4-PHASE)



6.4 XferFastData Pseudo Operation

Format: oData = XferFastData (iData)

Purpose:

To send 32 bits of data in/out of the device fast.

Description:

First the TMS Header is clocked into the device to select the Shift DR state. For 2-wire (4-phase), note that on the last clock, oPrAcc bit is shifted out on TDO while clocking in the TMS Header. If the value of oPrAcc is not '1', then the whole operation must be repeated. Next, the input value of PrAcc bit, which is '0', is clocked in. For 2-wire (4-phase), the TDO during this operation will be the LSb of out output. The rest of the 31 bits of the input data is clocked in and the 31 bits of output data is clocked out. For the last bit of the input data, the TMS Footer = 1 is set. Finally, 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. See Table 19-4 for complete description of commands.

The 2-Phase XferData is only used when talking to the PE. See **Section 16.0 “The Programming Executive”** for more details.

Example:

```
// Select the Fast Data Register
SendCommand(ETAP_FASTDATA)

// Send/Receive 32-bit Data
oData = XferFastData(32'h12)
```

FIGURE 6-7: XferFastdata 4-WIRE

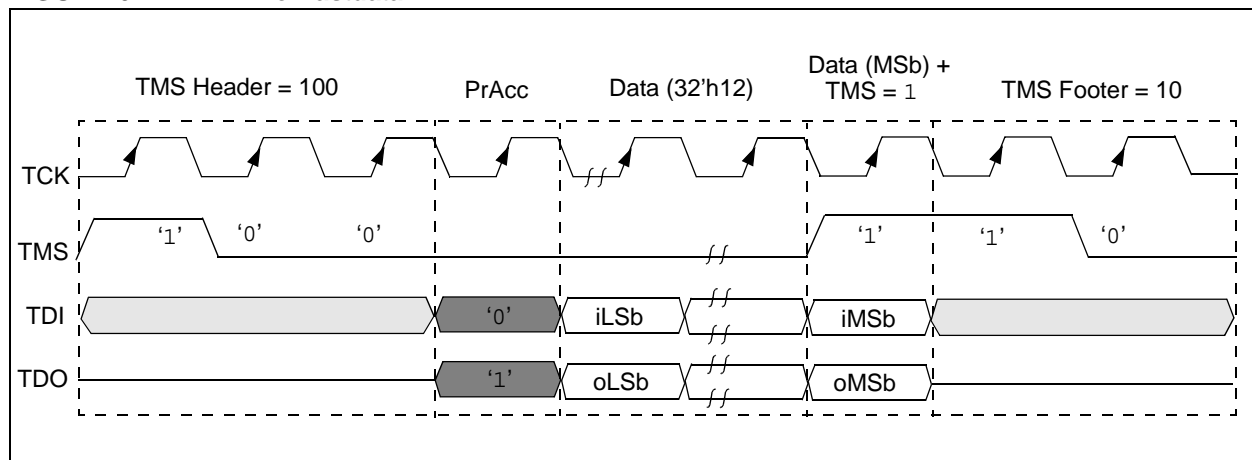


FIGURE 6-8: XferFastdata 2-WIRE (2-PHASE)

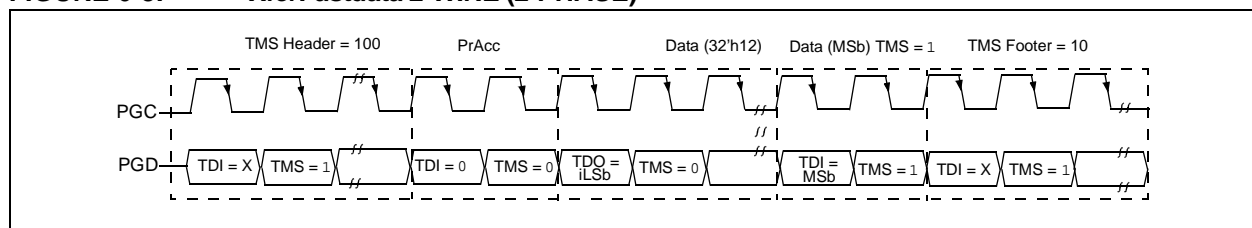
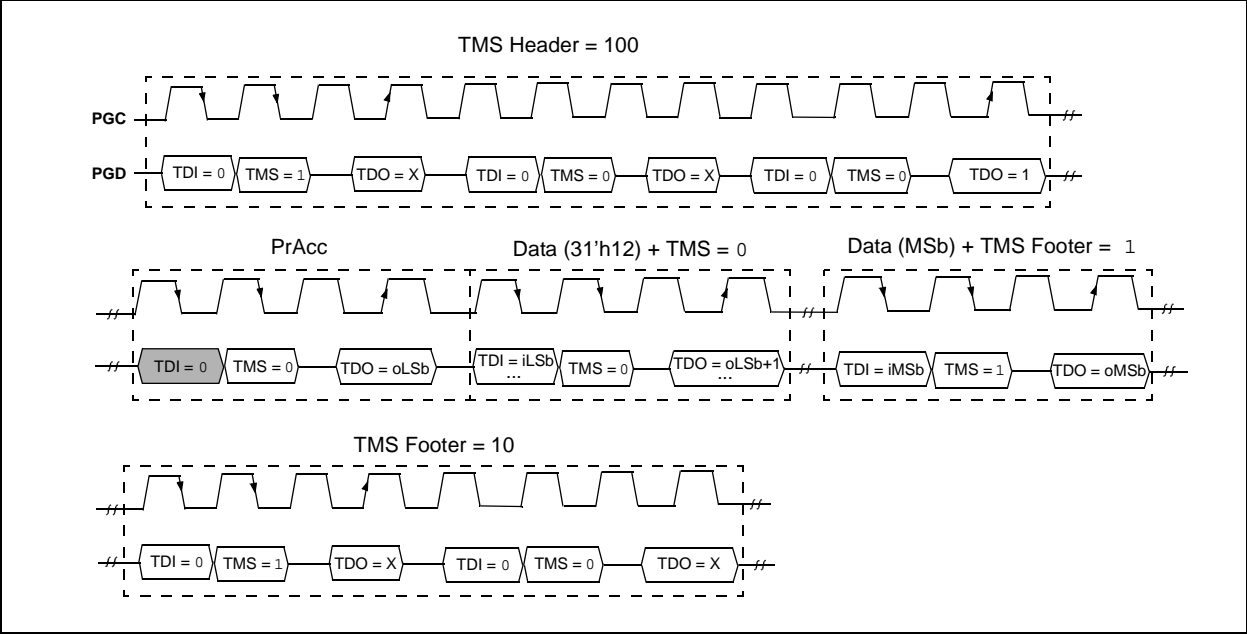


FIGURE 6-9: XferFastData 2-WIRE (4-PHASE)



6.5 XferInstruction Pseudo Operation

Format: XferInstruction (instruction)

Purpose:

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

Description:

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

Restrictions:

The device must be in Debug mode.

EXAMPLE 6-1: XferInstruction EXAMPLE

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

    // Select Data Register
    SendCommand(ETAP_DATA);

    // Send the instruction
    XferData(instruction);

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

7.0 ENTERING PROGRAMMING MODE

This step is not required if a 4-wire programming method is used. For 2-wire programming methods, the target device must be placed in a special Programming mode before executing further steps.

The following steps are required to enter Programming mode:

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

Please refer to **20.0 “Appendix C: AC/DC Characteristics and Timing Requirements”** for timing requirements.

The programming voltage applied to $\overline{\text{MCLR}}$ is V_{IH} , which is essentially V_{DD} in the case of PIC32MX devices. There is no minimum time requirement for

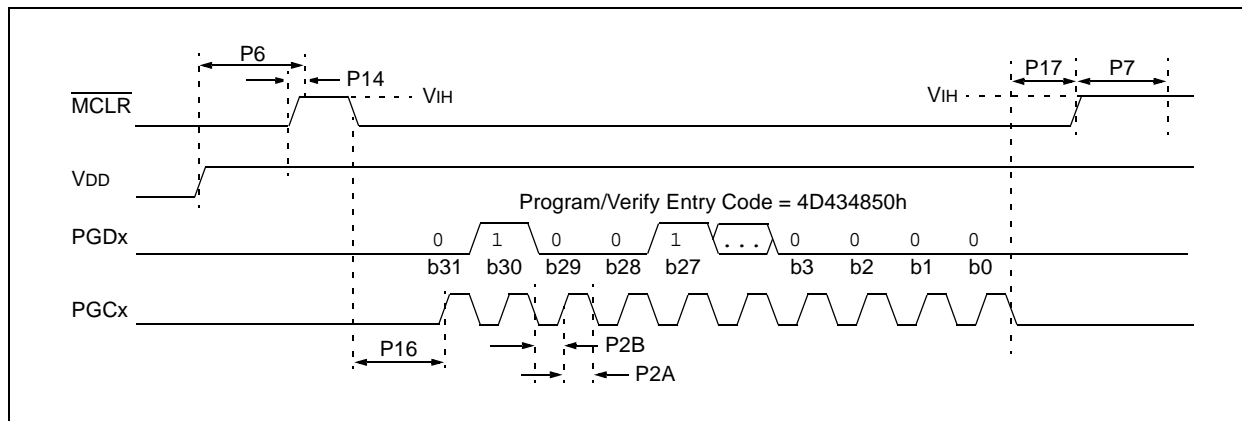
holding at V_{IH} . After V_{IH} is removed, an interval of at least P16 must elapse before presenting the key sequence on PGDx.

The key sequence is a specific 32-bit pattern: ‘0100 1101 0100 0011 0100 1000 0101 0000’ (more easily remembered as ascii ‘MCHP’). The device will enter Program/Verify mode only if the key sequence is valid. The Most Significant bit (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 Programming mode is to be maintained. An interval of at least time P17 and P7 must elapse before presenting data on PGDx. Signals appearing on PGDx before P7 has elapsed will not be interpreted as valid.

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

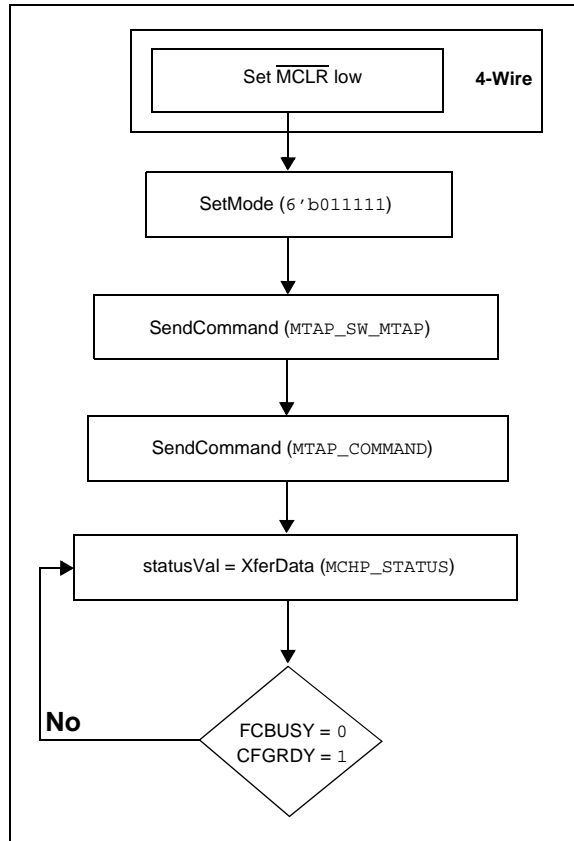
FIGURE 7-1: ENTERING ENHANCED ICSP™ MODE



8.0 CHECK DEVICE STATUS

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

FIGURE 8-1: CHECK DEVICE STATUS



8.1 4-Wire Interface

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

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

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

8.2 2-Wire Interface

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

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

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

9.0 ERASING THE DEVICE

Before a device can be programmed, it must be erased. The erase operation writes all '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 memory (Program, Boot, and Configuration memory) consists of selecting the MTAP and sending the MCHP_ERASE command. The programmer then must wait for the erase operation to complete by reading and verifying bits in the MCHP_STATUS value. Figure 9-1 shows the process for performing a Chip Erase.

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

The following steps are required to erase a target device:

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

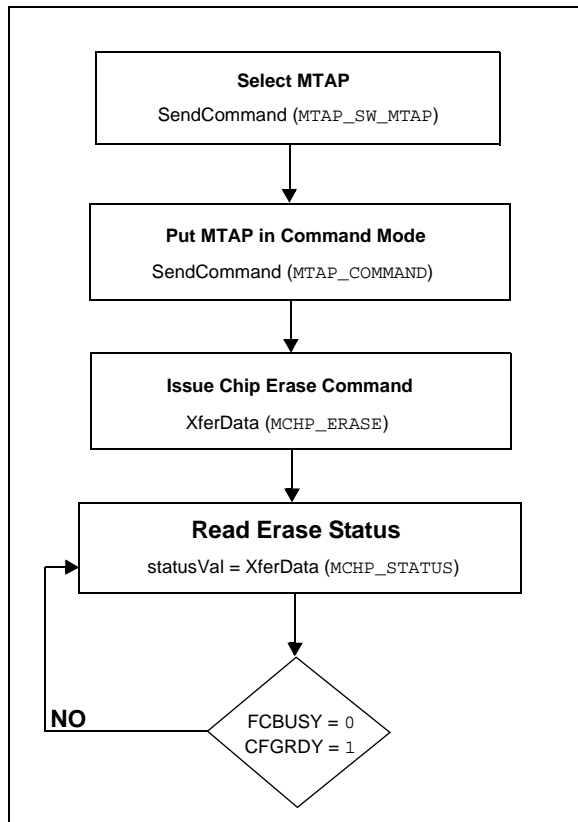
Note: The Chip Erase operation is a self-timed operation. If the FCBUSY and CFGRDY bits do not become properly set within the specified Chip Erase time, the sequence may have been executed wrong or the device is damaged.

9.1 Blank Check

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

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

FIGURE 9-1: ERASE DEVICE

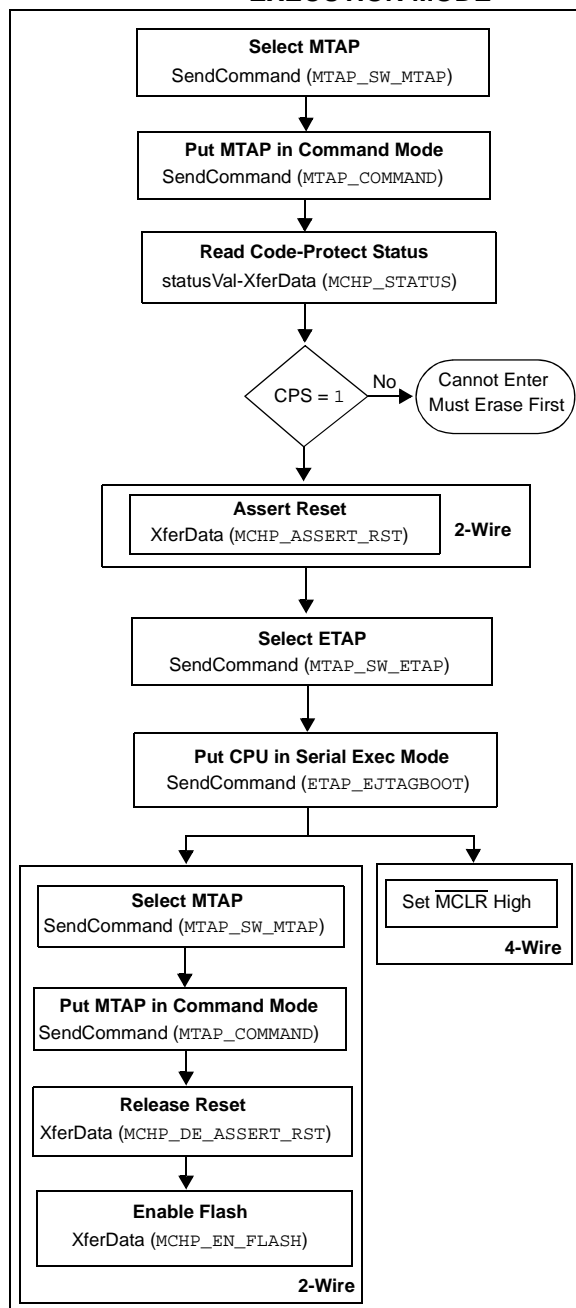


10.0 ENTERING SERIAL EXECUTION MODE

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

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

FIGURE 10-1: ENTERING SERIAL EXECUTION MODE



10.1 4-Wire Interface

The following steps are required to enter Serial Execution mode:

1. SendCommand (MTAP_SW_MTAP).
2. SendCommand (MTAP_COMMAND).
3. statusVal = XferData (MCHP_STATUS).
4. If CPS (statusVal<7>) is not '1' the device must be erased first.
5. SendCommand (MTAP_SW_ETAP).
6. SendCommand (ETAP_EJTAGBOOT).
7. Set MCLR 'High'.

10.2 2-wire Interface

The following steps are required to enter Serial Execution mode:

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

11.0 DOWNLOADING THE PROGRAMMING EXECUTIVE (PE)

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

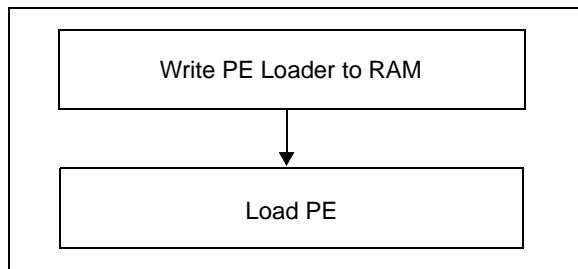
- Read Memory
- Erase Memory
- Program Memory
- Blank Check
- Read Executive Firmware Revision
- Get CRC of Flash Memory Locations

The programming executive 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 “Programming Executive Command Set”**.

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

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

FIGURE 11-1: DOWNLOAD PE



Loading the programming executive in the memory is a two step process:

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

The following steps are required to download the programming executive:

TABLE 11-1: DOWNLOAD PROGRAM EXEC

Operation	Operand
Step 1: Initialize BMXCON to 0x1f0040. The instruction sequence executed by the PIC32MX core is as follows:	
lui	a0,0xbf88
ori	a0,a0,0x2000 /* address of BMXCON */
lui	a1,0x1f
ori	a1,a1,0x40 /* \$a1 has 0x1f0040 */
sw	a1,0(a0) /* BMXCON initialized */
XferInstruction	0x3c04bf88
XferInstruction	0x34842000
XferInstruction	0x3c05001f
XferInstruction	0x34a50040
XferInstruction	0xac850000
Step 2: Initialize BMXDKPBA to 0x800. The instruction sequence executed by the PIC32MX core is as follows:	
li	a1,0x800
sw	a1,16(a0)
XferInstruction	0x34050800
XferInstruction	0xac850010
Step 3: Initialize BMXDUDBA and BMXDUPBA to 0x8000. The instruction sequence executed by the PIC32MX core is as follows:	
li	a1,0x8000
sw	a1,32(a0)
sw	a1,48(a0)
XferInstruction	0x34058000
XferInstruction	0xac850020
XferInstruction	0xac850030
Step 4: Setup PIC32MX RAM address for PE. The instruction sequence executed by the PIC32MX core is as follows:	
lui	a0,0xa000
ori	a0,a0,0x800
XferInstruction	0x3c04a000
XferInstruction	0x34840800
Step 5: Load PE_Loader. Repeat this step (step 5) until the entire PE_Loader is loaded in the PIC32MX memory. In the operands field, "<PE_loader hi++>" represents the MSBs 31-to-16 of the PE loader opcodes shown in Table 11.2. Likewise, "<PE_loader lo++>" represents the LSbs 15-to-0 of the PE loader opcodes shown in Table 11.2. The "++" sign indicates that when these operations are performed in succession, the new word is to be transferred from the list of opcodes of the LPE Loader shown in Table 11.2. The instruction sequence executed by the PIC32MX core is as follows:	
lui	a2, <PE_loader hi++>
ori	a0,a0, <PE_loader lo++>
sw	a2,0(a0)
addiu	a0,a0,4

TABLE 11-1: DOWNLOAD PROGRAM EXEC (CONTINUED)

XferInstruction	(0x3c06 <PE_loader hi++>)
XferInstruction	(0x34c6 <PE_loader lo++>)
XferInstruction	0xac860000
XferInstruction	0x24840004
Step 6: Jump to PE_Loader. The instruction sequence executed by the PIC32MX core is as follows: <pre>lui t9,0xa000 ori t9,t9,0x800 jr t9 nop</pre>	
XferInstruction	0x3c19a000
XferInstruction	0x37390800
XferInstruction	0x03200008
XferInstruction	0x00000000
Step 7: Load PE using the PE_Loader. Repeat this step (step 7) until the entire PE is loaded in the PIC32MX memory. In this step, you are given a Intel(R) Hex format file of Program Executive that you will parse and transfer number of 32-bit words at a time to the PIC32MX memory. The instruction sequence executed by the PIC32MX is shown in the "Instruction" column of Table 11.2, PE Loader Opcodes.	
SendCommand	ETAP_FASTDATA
XferFastData	PE_ADDRESS (Address of PE program block from PE Hex file)
XferFastData	PE_SIZE (Number of 32-bit words of the program block from PE Hex file)
XferFastData	PE software opcode from PE Hex file (PE Instructions)
Step 8: Jump to PE. Magic number (0xDEAD0000) instructs the PE_Loader that PE is completely loaded in the memory. When the PE_Loader sees the magic number, it jumps to PE.	
XferFastData	0x00000000
XferFastData	0xDEAD0000

TABLE 11-2: PE LOADER OPCODES

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

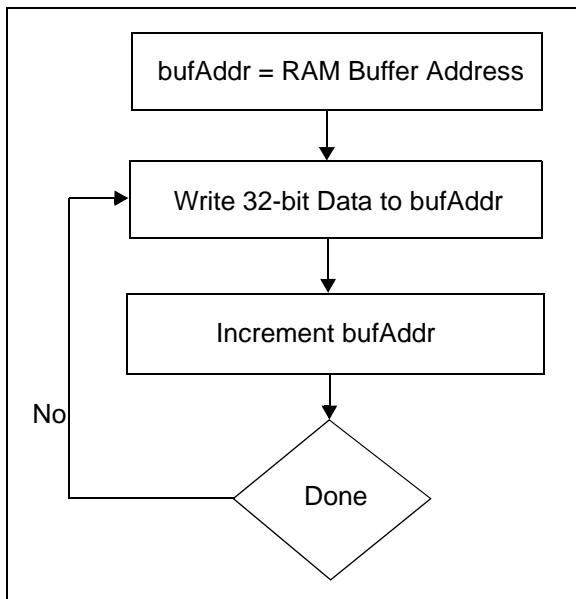
12.0 DOWNLOADING A DATA BLOCK

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

12.1 Without PE

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

FIGURE 12-1: DOWNLOADING DATA WITHOUT PE



The following steps are required to download a block data:

1. XferInstruction (opcode).
2. Repeat step 1 until last instruction is transferred to CPU.

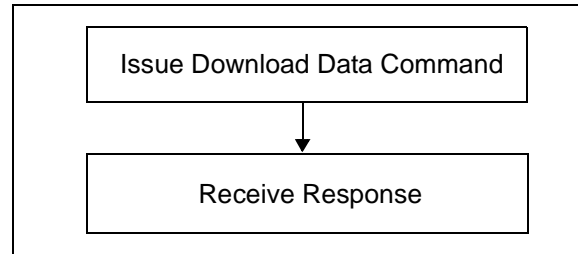
TABLE 12-1: DOWNLOAD DATA OPCODES

Opcode	Instruction
Step 1: Initialize SRAM Base Address to 0xA000_0000	
3c10a000	lui \$s0, 0xA000;
Step 2: Write the entire row of data to be programmed into system SRAM.	
3c08<DATA> 3508<DATA> ae08<OFFSET>	lui \$t0, <DATA(31:16)>; ori \$t0, <DATA(15:0)>; sw \$t0, <OFFSET>(\$s0); // OFFSET increments by 4
Step3: Repeat Step 2 until 1 row of data has been loaded.	

12.2 With PE

When using the PE, the code memory is programmed with the Program command (see Table 16-3). The program can program up to one row of code memory starting from the memory address specified in the command. The number of Program commands required to program a device depends on the number of write blocks that must be programmed in the device.

FIGURE 12-2: DOWNLOADING DATA WITH PE



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

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

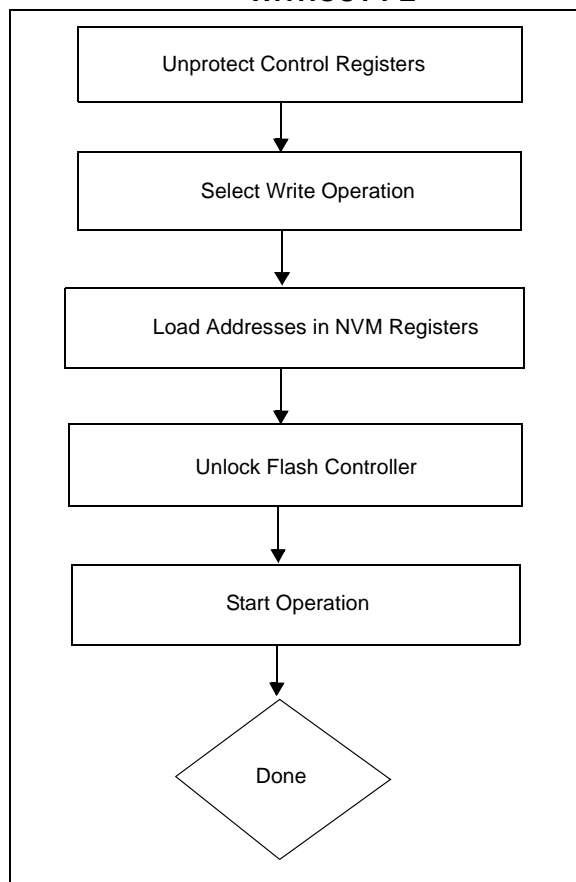
13.0 INITIATING A FLASH ROW WRITE

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

13.1 Without 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 NVMWR control bit NVMCON<15>.

FIGURE 13-1: INITIATING FLASH WRITE WITHOUT PE



The following steps are required to initiate a Flash write:

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

TABLE 13-1: INITIATE FLASH ROW WRITE OPCODES

Opcode	Instruction
Step 1: Initialize some constants.	
3c04bf80	lui a0,0xbf80
3484f400	ori a0,a0,0xf400
34054003	ori a1,\$0,0x4003
34068000	ori a2,\$0,0x8000
34074000	ori a3,\$0,0x4000
3c11aa99	lui s1,0xaa99
34116655	ori s1,\$0,0x6655
3c125566	lui s2,0x5566
341299aa	ori s2,\$0,0x99aa
3c13ff20	lui s3,0xff20
3c10a000	lui s0,0xa000
Step 2: Set NVMADDR with the address of the Flash row to be programmed.	
3c08<ADDR> 3408<ADDR> ac880020	lui t0, <FLASH_ROW_ADDR(31:16)> ori t0,\$0, <FLASH_ROW_ADDR(15:0)> sw t0,32(a0)
Step 3: Set NVMSRCADDR with the physical source SRAM address.	
ac900040	sw s0,64(a0)
Step 4: Write the operation in NVMCON. Run the unlock sequence using Row Program command.	
ac850000	sw a1,0(a0)
ac910010	sw s1,16(a0)
ac910010	sw s1,16(a0)
ac860008	sw a2,8(a0)
Step 5: Repeatedly read the NVMCON register and poll for NVMWR bit to get cleared.	
8c880000 01064024 1500fffd 00000000	here1: lw t0,0(a0) and t0,t0,a2 bne t0,\$0,<here1> nop
Step 6: Delay needed to address B3 ES SI Errata. Wait at least 500 nS after seeing a '0' in NVMCON<15> before writing to any NVM registers. This requires inserting NOP is the execution. Example: The following example assumes that the core is executing at 8 MHz, hence 4 NOP instructions equate to 500 nS.	
00000000 00000000 00000000 00000000	nop nop nop nop

PIC32MX

Step 7: Clear NVMCON.NVMWREN bit.

ac870004	sw	a3,4(a0)
----------	----	----------

Step 8: Check the NVMCON.NVMERR bit to ensure that the program sequence completed successfully. If error, jump to error-processing routine.

8c880000	lw	t0,0(a0)
30082000	andi	t0,zero,0x2000
1500<ERR_PR	bne	t0,\$0,
OC>		<err_proc_offset>
00000000	nop	

13.2 With PE

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

14.0 VERIFY DEVICE MEMORY

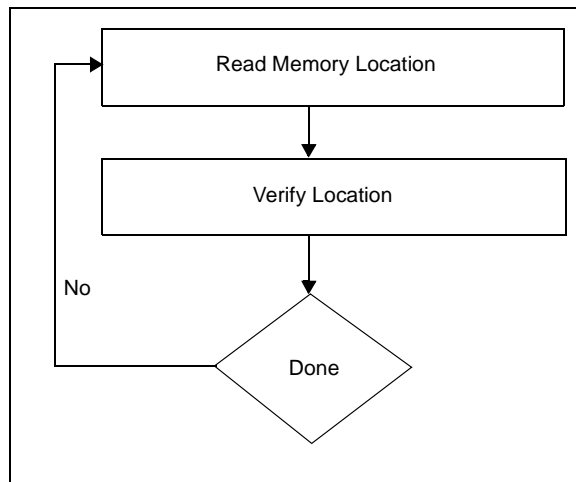
The verify step involves reading back the code memory space and comparing it against the copy held in the programmer's buffer. The Configuration registers are verified with the rest of the code.

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

14.1 Without PE

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

FIGURE 14-1: VERIFYING MEMORY WITHOUT PE



The following steps are required to verify memory:

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

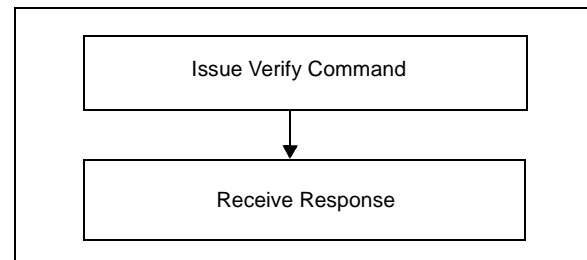
TABLE 14-1: VERIFY DEVICE OPCODES

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

14.2 With PE

Memory verify is performed using the GET_CRC (see Table 16-3) command as shown below.

FIGURE 14-2: VERIFYING MEMORY WITH PE



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

1. XferFastData (GET_CRC)
2. XferFastData (start_Address)
3. XferFastData (length)
4. valCkSum = XferFastData (32'h0x0)
5. Verify valCkSum matches the checksum of the copy held in the programmer's buffer.

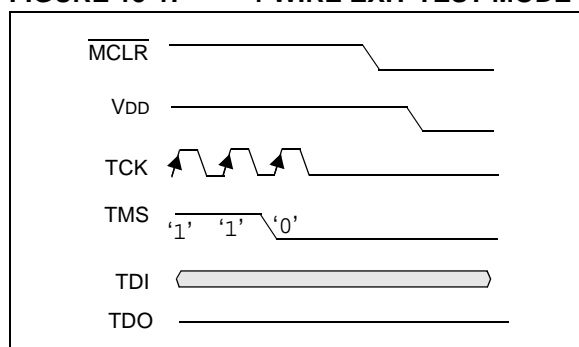
15.0 EXITING PROGRAMMING MODE

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

15.1 4-Wire Interface

Exiting Test mode is done by removing V_{IH} from \overline{MCLR} , as shown in Figure 15-1. The only requirement for exit is that an interval, P9B, should elapse between the last clock and program signals on PGCx and PGDx before removing V_{IH} .

FIGURE 15-1: 4-WIRE EXIT TEST MODE



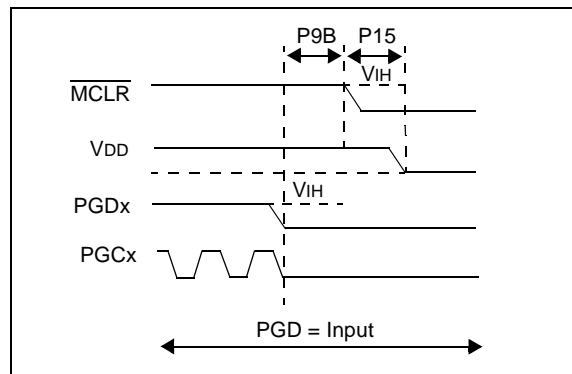
The following steps are required to exit Test mode:

1. SetMode (5'b11111).
2. Assert \overline{MCLR}
3. Remove Power (* if powering device)

15.2 2-Wire Interface

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

FIGURE 15-2: 2-WIRE EXIT TEST MODE



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

1. SetMode (5'b11111)
2. Assert \overline{MCLR}
3. Issue a clock pulse on PGCx
4. Remove Power (* if powering device)

16.0 THE PROGRAMMING EXECUTIVE

16.1 Programming Executive Communication

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

All communication is initiated by the programmer in the form of a command. Only one command at a time can be sent to the programming executive. In turn, the programming executive only sends one response to the programmer after receiving and processing a command.

16.1.1 2-WIRE ICSP EJTAG RATE

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

16.1.2 COMMUNICATION OVERVIEW

The programmer and the programming executive communicate using the EJTAG address, data and fastdata registers. In particular, the programmer transfers the command and data to the programming executive using the fastdata register. The programmer receives response from the programming executive using the address and data register. The pseudo operation of receiving response is shown in `GetPEResponse` pseudo operation below:

Format: `response = GetPEResponse()`

Purpose: The programmer receives 32-bit response value from programming executive.

EXAMPLE 16-1: GET PE RESPONSE

```
WORD GetPEResponse()
{
    WORD response;

    // Wait until CPU is ready
    SendCommand(ETAP_CONTROL);
    // Check if Processor Access bit
    (bit 18) is set
    do {
        controlVal=Xfer-
        Data(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 Table 16-1 shows the typical communication sequence between the programmer and the programming executive. In step 1, the programmer sends the command and optional additional data to the programming executive. Next, the programming executive carries out the command. Once the programming executive finishes execution of command, in step 2, 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 PE

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

16.2 Programming Executive Command Set

The programming executive command set is shown in Table 16-3. This table contains the opcode, mnemonic, length, time-out and short description for each command. Functional details on each command are provided in **Section 16.2.3 “ROW_PROGRAM”** through **Section 16.2.14 “CHANGE_CFG”**.

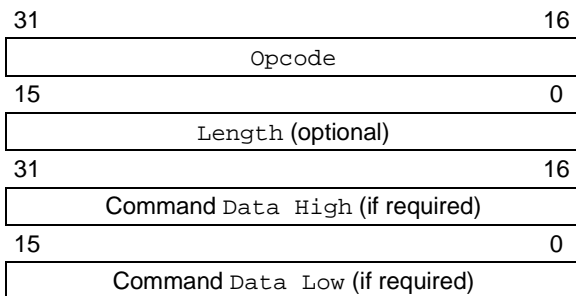
The programming executive sends a response to the programmer for each command that it receives. The response indicates if the command was processed correctly. It includes any required response data or error data.

16.2.1 COMMAND FORMAT

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

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

FIGURE 16-1: COMMAND FORMAT



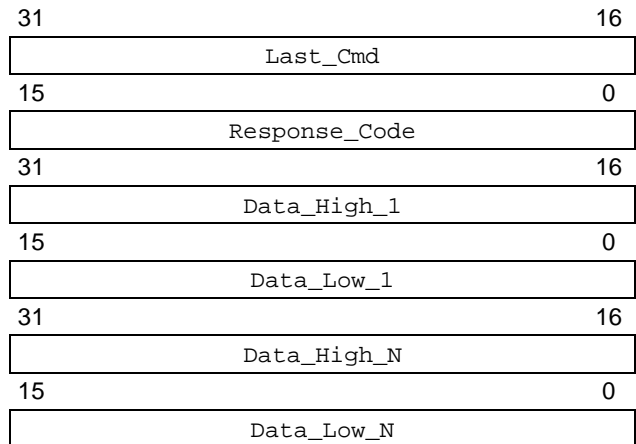
The command `Opcode` must match one of those in the command set. Any command that is received which does not match the list in Table 16-3 will return a “NACK” response (see **Section 16.2.2 “Response Format”**).

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

16.2.2 RESPONSE FORMAT

The programming executive response set is shown in Table 16-4. All programming executive 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

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

16.2.2.2 Response Code

The response code indicates whether the last command succeeded, failed, or if the command is an un-recognized value. The response code values are shown in Table 16-2.

TABLE 16-2: RESPONSE VALUES

Opcode	Mnemonic	Description
0h	PASS	Command successfully processed.
2h	FAIL	Command unsuccessfully processed.
3h	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.

TABLE 16-3: PROGRAMMING EXECUTIVE COMMAND SET

Opcode	Mnemonic	Length* (32-bit words)	Description
0h	ROW_PROGRAM	2	Program one row of Flash memory at the specified address.
1h	READ	2	Read N 32-bit words of memory starting from the specified address. (N < 65536)
2h	PROGRAM	130	Program Flash memory starting at the specified address.
3h	WORD_PROGRAM	3	Program one word of Flash memory at the specified address.
4h	CHIP_ERASE	1	Chip Erase of entire chip.
5h	PAGE_ERASE	2	Erase pages of code memory from the specified address.
6h	BLANK_CHECK	1	Blank Check code.
7h	EXEC_VERSION	1	Read the programming executive software version.
8h	GET_CRC	2	Get CRC of Flash memory.

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

Note: One row of code memory consists of (128) 32-bit words. Refer to Table 16-1 for device-specific information.

16.2.3 ROW_PROGRAM

FIGURE 16-3: ROW_PROGRAM COMMAND

31	16
Opcode	
15	0
Length	
31	16
Addr_High	
15	0
Addr_Low	
31	16
Data_High_1	
15	0
Data_Low_1	
31	16
Data_High_N	
15	0
Data_Low_N	

TABLE 16-4: ROW_PROGRAM FORMAT

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

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

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

Expected Response (1 word):

FIGURE 16-4: ROW_PROGRAM RESPONSE

31	16
Last Command	
15	0
Return Code	

16.2.4 READ

FIGURE 16-5: READ COMMAND

31	16
Opcode	
15	0
Length	
31	16
Addr_High	
15	0
Addr_Low	

TABLE 16-5: READ FORMAT

Field	Description
Opcode	1h
Length	Number of 32-bit words to read (max. of 65535)
Addr_Low	Low 16-bits of 32-bit source address
Addr_High	High 16-bits of 32-bit source address

The Read command instructs the programming executive to read Length 32-bit words of Flash memory, including Configuration Words, starting from the 32-bit address specified by Addr_Low and Addr_High. This command can only be used to read 32-bit data. All data returned in response to this command uses the packed data format as shown in Table 16-5.

Expected Response:

FIGURE 16-6: READ RESPONSE

31	16
Last Command	
15	0
Return Code	
31	16
Data High	
15	0
Data Low	

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

16.2.5 PROGRAM

FIGURE 16-7: PROGRAM COMMAND

31	16
Opcode	
15	0
Not Used	
31	16
Addr_High	
15	0
Addr_Low	
31	16
Length_High	
15	0
Length_Low	
31	16
Data_High_N	
15	0
Data_Low_N	

TABLE 16-6: PROGRAM FORMAT

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

The Program command instructs the programming executive to program Flash memory, including Configuration Words, starting from the 32-bit address specified by `Addr_Low` and `Addr_High`. The address must be aligned to 512 byte boundary (aligned to flash row size). Also, the length must be multiple of 512 bytes (multiple of Flash row size).

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

There are three programming scenarios:

1. The data to be programmed is 512 bytes long.
2. The data to be programmed is 1024 bytes.
3. The data to be programmed is larger than 1024 bytes.

When the data length is equal to 512 bytes, the PE will send the response of this command immediately after receiving the 512 bytes.

When the length is 1024 bytes, the PE will receive the first two blocks of 512-byte data continuously. Next, the PE responds with the status of the write operation of the first 512-byte block, followed immediately by the status of the write operation of the second 512-byte block.

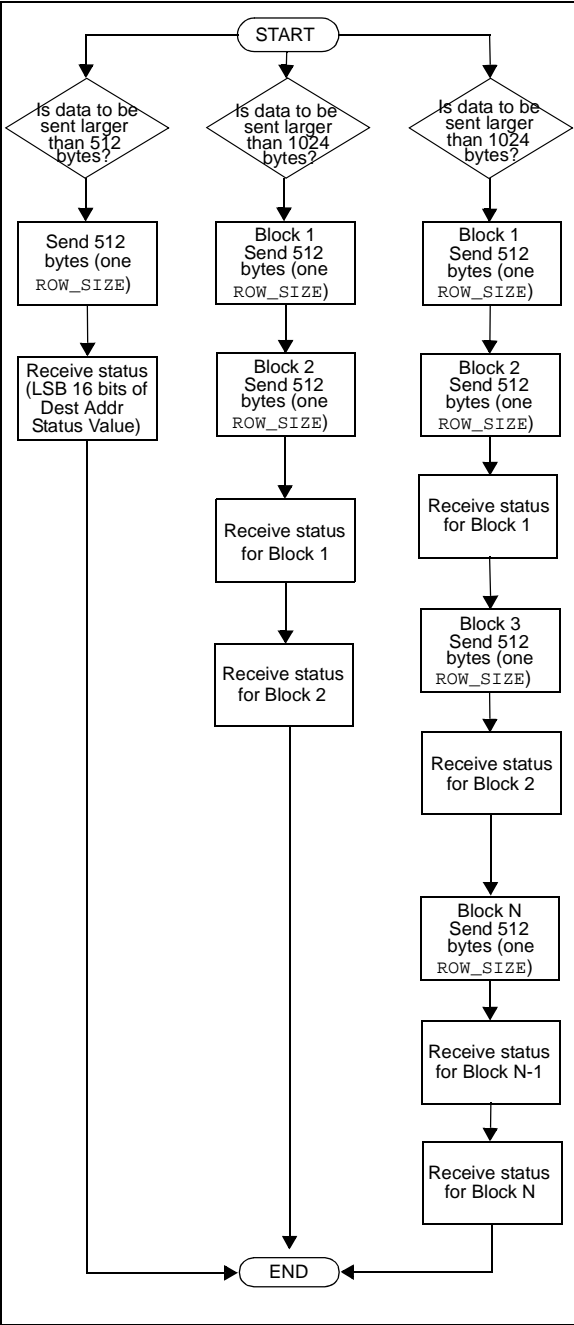
If the data to be programmed is larger than 1024 bytes, the first two 512-byte blocks are sent consecutively followed by the response for block 1. Next, the 3rd 512-byte block is sent, followed by the response for block 2 is received. The successive blocks and responses are exchanged in similar fashion. After sending the last 512-byte block, the probe shall receive the response for the second to last block, followed by the response for the last block.

If the PE encounters an error in programming any of the blocks, it sends a failure status to the probe. Upon receiving the failure status, the probe must stop sending any further data. The PE will not receive any further data for this command from the probe.

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

The Figure 16-8 shows the programming concept.

FIGURE 16-8: PROGRAM ALGORITHM



Expected Response (1 word):

FIGURE 16-9: PROGRAM RESPONSE

31	16
LSB 16 bits of the destination address of last block	
15	0
Return Code	

16.2.6 WORD_PROGRAM

FIGURE 16-10: WORD_PROGRAM COMMAND

31	16
Opcode	
15	0
Length	
31	16
Addr_High	
15	0
Addr_Low	
31	16
Data_High	
15	0
Data_Low	

TABLE 16-7: WORD-PROGRAM FORMAT

Field	Description
Opcode	3h
Length	2
Addr_High	High 16 bits of 32-bit destination address
Addr_Low	Low 16 bits of 32-bit destination address
Data_High	High 16 bits data word
Data_Low	Low 16 bits data word

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

Expected Response (1 word):

FIGURE 16-11: WORD_PROGRAM RESPONSE

31	16
Last Command	
15	0
Return Code	

16.2.7 CHIP_ERASE

FIGURE 16-12: CHIP_ERASE COMMAND

31	16
Opcode	
15	0
Length	

TABLE 16-8: CHIP_ERASE FORMAT

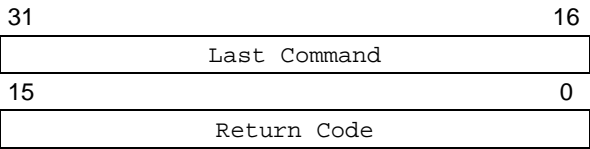
Field	Description
Opcode	4h
Length	Ignored
Addr_Low	Low 16 bits of 32-bit destination address
Addr_High	High 16 bits of 32-bit destination address

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

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

Expected Response (1 word):

FIGURE 16-13: CHIP_ERASE RESPONSE



16.2.8 PAGE_ERASE

FIGURE 16-14: PAGE_ERASE COMMAND

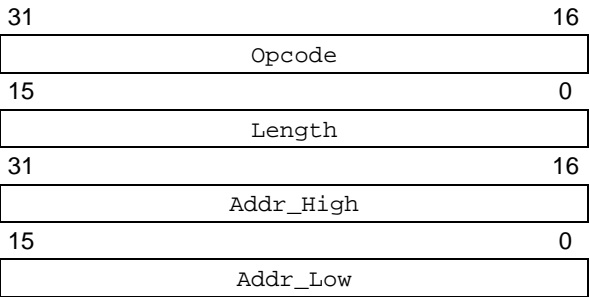


TABLE 16-9: PAGE_ERASE FORMAT

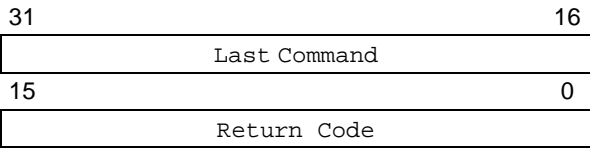
Field	Description
Opcode	5h
Length	Number of pages to erase
Addr_Low	Low 16 bits of 32-bit destination address
Addr_High	High 16 bits of 32-bit destination address

The PAGE_ERASE command erases the specified number of pages of code memory from the specified base address. The specified base address must be a multiple of 0x400.

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

Expected Response (1 word):

FIGURE 16-15: PAGE_ERASE RESPONSE



16.2.9 BLANK_CHECK

FIGURE 16-16: BLANK_CHECK COMMAND

31	16
Opcode	
15	0
Not Used	
31	16
Addr_High	
15	0
Addr_Low	
31	16
Length_High	
15	0
Length_Low	

TABLE 16-10: BLANK_CHECK FORMAT

Field	Description
Opcode	6h
Length	Number of program memory locations to check in terms of bytes
Address	Address where to start the Blank Check

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

Expected Response (1 word for blank device):

FIGURE 16-17: BLANK_CHECK RESPONSE

31	16
Last Command	
15	0
Return Code	

16.2.10 EXEC_VERSION

FIGURE 16-18: EXEC_VERSION COMMAND

31	16
Opcode	
15	0
Length	

TABLE 16-11: EXEC_VERSION FORMAT

Field	Description
Opcode	7h
Length	Ignored

The EXEC_VERSION command queries the version of the programming executive software stored in RAM.

The version value of the current Programming Executive is 0x0105.

Expected Response (1 word):

FIGURE 16-19: EXEC_VERSION RESPONSE

31	16
Last Command	
15	0
Version Number	

16.2.11 GET_CRC

FIGURE 16-20: GET_CRC COMMAND

31	16
Opcode	
15	0
Not Used	
31	16
Addr_High	
15	0
Addr_Low	
31	16
Length_High	
15	0
Length_Low	

TABLE 16-12: GET_CRC FORMAT

Field	Description
Opcode	8h
Address	Address where to start calculating CRC
Length	Length of buffer on which to calculate CRC in number of bytes

The GET_CRC command 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

MSB shifted in first.

Note that in the response, only the CRC LSB 16-bit are valid.

Expected Response (2 words):

FIGURE 16-21: GET_CRC RESPONSE

31	16
Last Command	
15	0
Return Code	
31	16
CRC_High	
15	0
CRC_Low	

16.2.12 PROGRAM_CLUSTER

FIGURE 16-22: PROGRAM_CLUSTER COMMAND

31	16
Opcode	
15	0
Not Used	
31	16
Addr_High	
15	0
Addr_Low	
31	16
Length_High	
15	0
Length_Low	

TABLE 16-13: PROGRAM_CLUSTER FORMAT

Field	Description
Opcode	9h
Address	Address where to start calculating CRC
Length	Length of buffer on which to calculate CRC in number of bytes

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

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

Expected Response (1 word):

FIGURE 16-23: PROGRAM_CLUSTER RESPONSE

31	16
Last Command	
15	0
Return Code	

16.2.13 GET_DEVICEID

FIGURE 16-24: GET_DEVICEID COMMAND

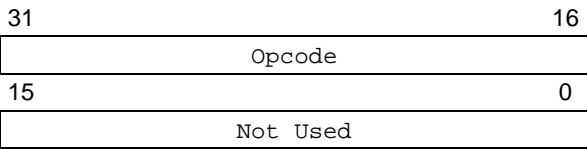


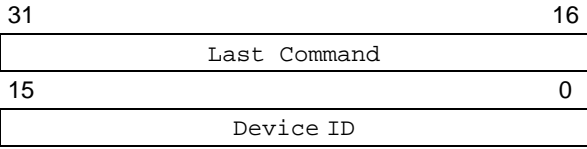
TABLE 1: GET_DEVICEID FORMAT

Field	Description
Opcode	Ah

The GET_DEVICEID command returns the hardware ID of the device.

Expected Response (1 word):

FIGURE 16-25: GET_DEVICEID RESPONSE



16.2.14 CHANGE_CFG

FIGURE 16-26: CHANGE_CFG COMMAND

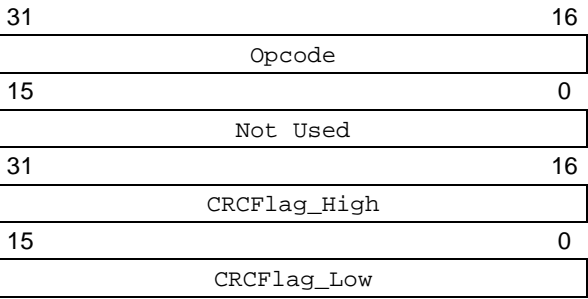


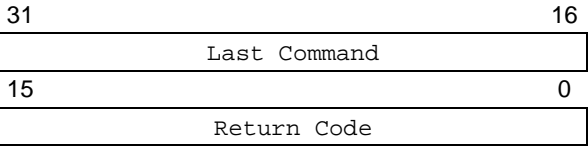
TABLE 16-14: CHANGE_CFG FORMAT

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

The CHANGE_CFG command is used by the probe to set various configuration settings for the PE. Currently, the only configuration setting available is whether PE should use the hardware or software CRC calculation method.

Expected Response (1 word):

FIGURE 16-27: CHANGE_CFG RESPONSE



17.0 CHECKSUM

17.1 Theory

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

17.2 Mask Values

The mask value of a device Configuration is calculated by setting all the unimplemented bits to '0' and all the implemented bits to '1'. For example, Figure 17-1 shows the DEVCFG0 register of the PIC32MX360F512L device. The mask value for this register is:

mask_value_devcfg0 = 0x110FF00B

FIGURE 17-1: DEVCFG0 REGISTER OF PIC32MX360F512L

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
r-0	r-1	r-1	R/P-1	r-1	r-1	r-1	R/P-1
—	—	—	CP	—	—	—	BWP
bit 31							bit 24

r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	—	PWP19	PWP18	PWP17	PWP16
bit 23							bit 16

R/P-1	R/P-1	R/P-1	R/P-1	r-1	r-1	r-1	r-1
PWP15	PWP14	PWP13	PWP12	—	—	—	—
bit 15							bit 8

r-1	r-1	r-1	r-1	R/P-1	r-1	R/P-1	R/P-1
—	—	—	—	ICESEL	—	DEBUG<1:0>	
bit 7							bit 0

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

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

Device	DEVCFG0	DEVCFG1	DEVCFG2	DEVCFG3	DEVID
All PIC32MX3XX devices	0x110FF00B	0x009FF7A7	0x00070077	0x0000FFFF	0x000FF000
All PIC32MX4XX devices	0x110FF00B	0x009FF7A7	0x00078777	0x0000FFFF	0x000FF000

PIC32MX

For quick reference, Table 17-2 shows the addresses of DEVCFG and DEVID registers for currently supported devices.

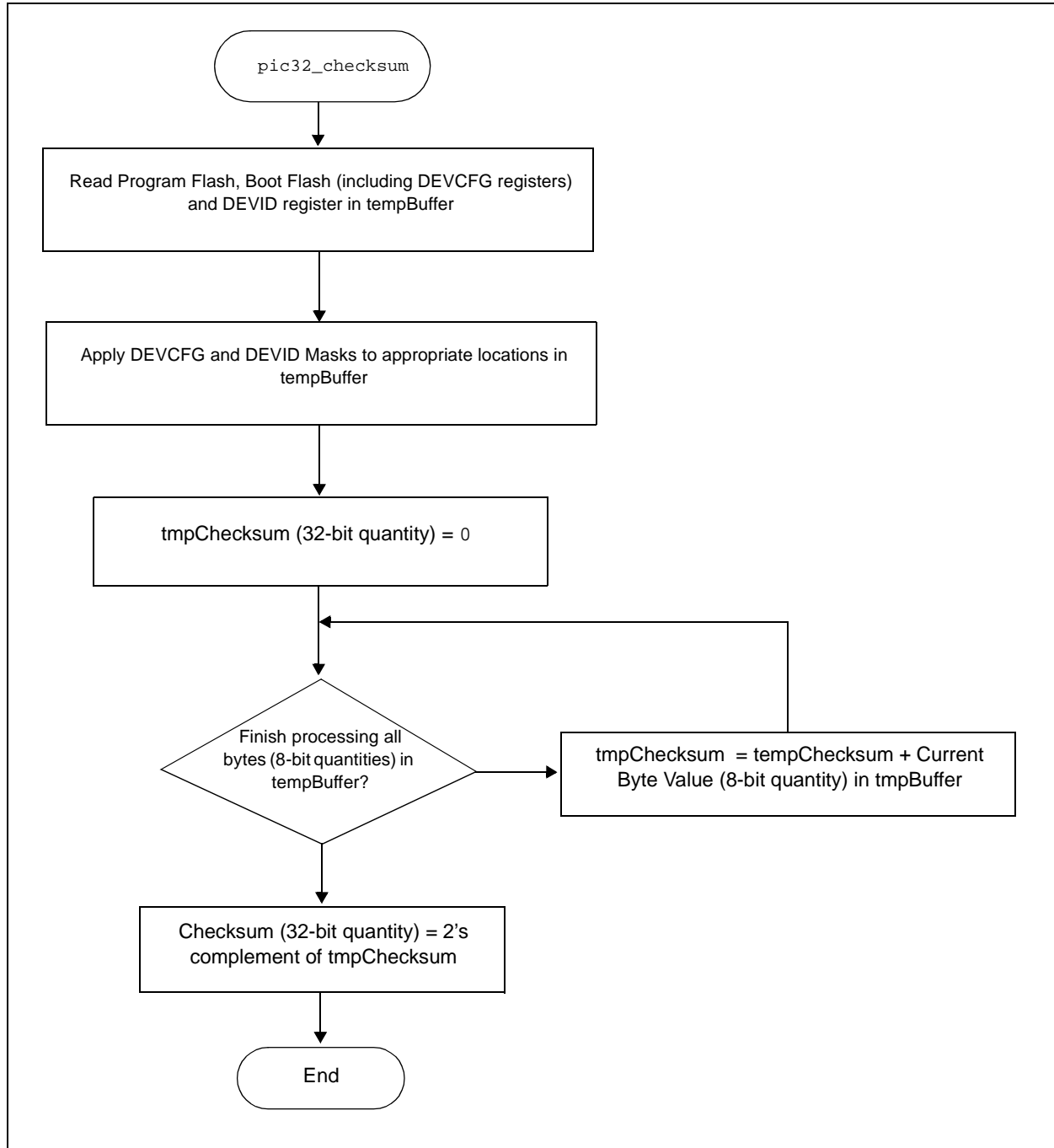
**TABLE 17-2: DEVICE CONFIGURATION
 AND DEVICE ID REGISTER
 ADDRESSES**

Register	Address
DEVCFG0	0xBFC02FF0
DEVCFG1	0xBFC02FF4
DEVCFG2	0xBFC02FF8
DEVCFG3	0xBFC02FFC
DEVID	0xBF80F220

17.3 Algorithm

Figure 17-2 shows high level algorithm of checksum calculation for PIC32 devices. This is only an example of how the actual calculations can be carried out. There may be more efficient implementation methods to perform the same task and is left up to the individual software implementer to decide.

FIGURE 17-2: HIGH-LEVEL ALGORITHM FOR CHECKSUM CALCULATION



PIC32MX

As stated earlier, the PIC32 checksum is calculated as the 32-bit summation of all bytes (8-bit quantities) in program Flash, boot Flash (except device Configuration Words), the Device ID register with applicable mask, and the device Configuration Words with applicable masks. Next, the 2's complement of the summation is calculated. This final 32-bit number is presented as the checksum.

The mask values of the device Configuration and Device ID registers are derived as described in **Section 17.2 “Mask Values”**. Another noteworthy point is that the last four 32-bit quantities in boot Flash are the device Configuration registers. An arithmetic AND operation of these device Configuration register values is performed with the appropriate mask value before adding their bytes to the checksum. Similarly, arithmetic AND operation of Device ID register is performed with appropriate mask value before addition its bytes to the checksum.

The formula for the PIC32 checksum can be shown as

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

Where,

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

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

$$\text{DCR} = \sum_{X=0}^3 \text{32-bit summation of bytes (MASK}_{\text{devcfg}_X} \& \text{DEVCFG_X})$$

Where, MASK_{devcfg_X} is mask value from Table 17-1

$$\text{DIR} = 32\text{-bit summation of bytes (MASK}_{\text{dev}_\text{devid}} \text{ and DEVID)}$$

Where, MASK_{dev_{devid}} is mask value from Table 17-1

17.4 Example

The following example shows the checksum calculation for PIC32MX360F512L. The following assumptions are made for the example:

- The program Flash and boot Flash are in erased state (all bytes are 0xFF)
- The device Configuration is in part's default state (no configuration changes are made)

We will use the formula for checksum as shown in the **Section 17.3 "Algorithm"**. We will individually calculate each item on the right-hand side of the equation before deriving the final value of the checksum.

17.4.1 CALCULATE PF

The size of program Flash is 512KB, which equals 524288 bytes. Since the program Flash is assumed to be in erased state, we calculate the value of PF as under:

$$\text{PF} = 0\text{xFF} + 0\text{xFF} + \dots 524288 \text{ times}$$

$$\text{PF} = 0\text{x7F80000} \text{ (note it is 32-bit number)}$$

17.4.2 CALCULATE BF

The size of the boot Flash is 12KB, which equals 12288 bytes. However, the last 16 bytes are device Configuration registers, which are treated separately. Hence, the number of bytes in boot Flash that we consider in this step is 12272. Since the boot Flash is assumed to be in erased state, we calculate the value of BF as under:

$$\text{BF} = 0\text{xFF} + 0\text{xFF} + \dots 12272 \text{ times}$$

$$\text{BF} = 0\text{x002FC010} \text{ (note it is 32-bit number)}$$

17.4.3 CALCULATE DCR

Since the device Configuration registers are left in their default state, the values of the DEVCFG registers as read by PIC32 core, their respective mask values, the result of applying the mask and the 32-bit summation of bytes is as shown in Table 17-3.

TABLE 17-3: DCR CALCULATION EXAMPLE

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

From Table 17-3, the value of DCR is:

DCR = 0x000005D4 (note it is 32-bit number)

17.4.4 CALCULATE DIR

The value of Device ID register, its mask, the result of applying the mask and the 32-bit summation of bytes is shown in Table 17-4.

TABLE 17-4: DIR CALCULATION EXAMPLE

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

From Table 17-4, the value of DIR is:

DIR = 0x00000083 (note it is 32-bit number)

17.4.5 CALCULATE CHECKSUM

The values derived in previous sections allows us to calculate the checksum value. First, perform the 32-bit summation of the PF, BF, DCR and DIR as derived in previous sections and store it in a variable, called temp.

$\text{temp} = \text{PF} + \text{BF} + \text{DCR} + \text{DIR}$

$\text{temp} = 0x7F80000 + 0x002FC010 + 0x000005D4 + 0x00000083$

$\text{temp} = 0x0827C667$

Next, the 1's complement of temp, called temp1, is:

$\text{temp1} = 1\text{'s complement}(\text{temp})$

$\text{temp1} = 0xF7D83998$

Finally, the 2's complement of temp is the checksum:

$\text{Checksum} = 2\text{'s complement}(\text{temp})$

$\text{Checksum} = \text{temp1} + 1$

Checksum = 0xF7D83999

17.5 Checksum for PIC32 Devices

17.5.1 CHECKSUM VALUES FOR ERASED DEVICES

This section lists the checksums of the currently supported devices. The checksums are provided when the program Flash and boot Flash are both in erased state. Also, the device Configuration Words are assumed to be in Power-on Reset default values.

TABLE 17-5: CHECKSUMS FOR PIC32 DEVICES

Device	Checksum
PIC32MX300F032H	0xFF50BA1C
PIC32MX320064H	0xFED139BC
PIC32MX320F128H	0xFDD2397C
PIC32MX340F128H	0xFDD2394C
PIC32MX340F256H	0xFBD439FB
PIC32MX340F512H	0xF7D839BB
PIC32MX420F032H	0xFF50B971
PIC32MX440F128H	0xFDD238C1
PIC32MX440F256H	0xFBD43970
PIC32MX440F512H	0xF7D83930
PIC32MX320F128L	0xFDD2397A
PIC32MX340F128L	0xFDD2394A
PIC32MX360F256L	0xFBD439D9
PIC32MX360F512L	0xF7D83999
PIC32MX440F128L	0xFDD238BF
PIC32MX460F256L	0xFBD4394E
PIC32MX460F512L	0xF7D8390E

17.5.2 CHECKSUM VALUES WHILE DEVICE IS CODE-PROTECTED

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

PIC32MX

18.0 APPENDIX A: CONFIGURATION MEMORY AND DEVICE ID

PIC32MX devices include several features intended to maximize application flexibility and reliability, and minimize cost through elimination of external components. These are:

- Flexible Device Configuration
- Code Protection
- Internal Voltage Regulator

TABLE 18-1: DEVCFG – DEVICE CONFIGURATION WORD SUMMARY

Virtual Address	Name	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit
		31/23/15/7	30/22/14/6	29/21/13/5	28/20/12/4	27/19/11/3	26/18/10/2	25/17/9/1	24/16/8/0
BFC0_2FF0	DEVCFG3	31:24	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—
		15:8	USERID15	USERID14	USERID13	USERID12	USERID11	USERID10	USERID9
		7:0	USERID7	USERID6	USERID5	USERID4	USERID3	USERID2	USERID1
BFC0_2FF4	DEVCFG2	31:24	—	—	—	—	—	—	—
		23:16	—	—	—	—	FPLLODIV<2:0>		
		15:8	FUPLLEN	—	—	—	FUPLLDIV<2:0>		
		7:0	—	FPLLMULT<2:0>			—	FPLLDIV<2:0>	
BFC0_2FF8	DEVCFG1	31:24	—	—	—	—	—	—	—
		23:16	FWDTEN	—	—	WDTPS<4:0>			
		15:8	FCKSM<1:0>		—	—	—	—	—
		7:0	IESO	—	FSOSCEN	—	—	FNOSC<2:0>	
BFC0_2FFC	DEVCFG0	31:24	—	—	—	CP	—	—	—
		23:16	—	—	—	—	PWP19	PWP18	PWP17
		15:8	PWP15	PWP14	PWP13	PWP12	—	—	—
		7:0	—	—	—	—	ICESEL	—	DEBUG<1:0>

TABLE 18-2: DEVID SUMMARY

Virtual Address	Name	Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit
		31/23/15/7	30/22/14/6	29/21/13/5	28/20/12/4	27/19/11/3	26/18/10/2	25/17/9/1	24/16/8/0
BF80_F220	DEVID	31:24	VER11	VER10	VER9	VER8	VER7	VER6	VER5
		23:16	VER3	VER2	VER1	VER0	DEVID7	DEVID6	DEVID5
		15:8	DEVID3	DEVID2	DEVID1	DEVID0	MANID11	MANID10	MANID9
		7:0	MANID7	MANID6	MANID5	MANID4	MANID3	MANID2	MANID1

REGISTER 18-1: DEVCFG0: DEVICE CONFIGURATION WORD 0

r-0	r-1	r-1	R/P-1	r-1	r-1	r-1	R/P-1
—	—	—	CP	—	—	—	BWP
bit 31				bit 24			
r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	—	PWP19	PWP18	PWP17	PWP16
bit 23				bit 16			
R/P-1	R/P-1	R/P-1	R/P-1	r-1	r-1	r-1	r-1
PWP15	PWP14	PWP13	PWP12	—	—	—	—
bit 15				bit 8			
r-1	r-1	r-1	r-1	R/P-1	r-1	R/P-1	R/P-1
—	—	—	—	ICESEL	—	DEBUG1	DEBUG0
bit 7				bit 0			

Legend:

R = readable bit W = writable bit P = programmable bit r = reserved bit
 U = unimplemented bit, read as '0' -n = bit value at POR: ('0', '1', x = unknown)

bit 31 **Reserved:** Maintain as '0'.

bit 30-29 **Reserved:** Maintain as '1'

bit 28 **CP:** Code Protect bit

Prevents boot and program Flash memory from being read or modified by an external programming device.

1 = Protection Disabled

0 = Protection Enabled

bit 27-25 **Reserved:** Maintain as '1'

bit 24 **BWP:** Boot Flash Write Protect bit

Prevents boot Flash memory from being modified during code execution.

1 = Boot Flash is writable

0 = Boot Flash is not writable

bit 23-20 **Reserved:** Maintain as '1'

bit 19-12	<p>PWP<19:12>: Program Flash Write Protect bits</p> <p>Prevents selected program Flash memory pages from being modified during code execution. The PWP bits represent the one's compliment of the number of write protected program Flash memory pages.</p> <p>11111111 = Disabled 11111110 = 0xBD00_0FFF 11111101 = 0xBD00_1FFF 11111100 = 0xBD00_2FFF 11111011 = 0xBD00_3FFF 11111010 = 0xBD00_4FFF 11111001 = 0xBD00_5FFF 11111000 = 0xBD00_6FFF 11110111 = 0xBD00_7FFF 11110110 = 0xBD00_8FFF 11110101 = 0xBD00_9FFF 11110100 = 0xBD00_AFFF 11110011 = 0xBD00_BFFF 11110010 = 0xBD00_CFFF 11110001 = 0xBD00_DFFF 11110000 = 0xBD00_EFFF 11101111 = 0xBD00_FFFF ... 01111111 = 0xBD07_FFFF</p>
bit 11-4	<p>Reserved: Maintain as '1'</p>
bit 3	<p>ICESEL: ICE/ICD Comm Channel Select bit</p> <p>1 = ICE uses PGC2/PGD2 pins 0 = ICE uses PGC1/PGD1 pins</p>
bit 2	<p>Reserved: Maintain as '1'</p>
bit 1-0	<p>DEBUG<1:0>: Background Debugger Enable bits</p> <p>11 = Debugger Disabled (forced if device is code protected) 10 = ICE Debugger Enabled 01 = Reserved 00 = Reserved</p>

REGISTER 18-2: DEVCFG1: DEVICE CONFIGURATION WORD 1

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 31							bit 24
R/P-1	r-1	r-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
FWDTEN	—	—	WDTPS<4:0>				
bit 23							bit 16
R/P-1	R/P-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1
FCKSM<1:0>		—	—	—	OSCIOFNC	POSCMD<1:0>	
bit 15							bit 8
R/P-1	r-1	R/P-1	r-1	r-1	R/P-1	R/P-1	R/P-1
IESO	—	FSOSCEN	—	—	FNOSC<2:0>		
bit 7							bit 0

Legend:

R = readable bit W = writable bit P = programmable bit r = reserved bit
 U = unimplemented bit, read as '0' -n = bit value at POR: ('0', '1', x = unknown)

- bit 31-24 **Reserved:** Maintain as '1'
- bit 23 **FWDTEN:** Watchdog Timer Enable bit
 1 = The WDT is enabled and cannot be disabled by software
 0 = The WDT is not enabled. It can be enabled in software
- bit 22 **WINDIS:** Windowed Watchdog Timer Disable bit
 1 = Standard WDT selected; windowed WDT disabled
 0 = Windowed WDT enabled
- bit 21 **Reserved:** Maintain as '1'
- bit 20-16 **WDTPS<4:0>:** Watchdog Timer Postscale Select bits
 10100 = 1:1048576
 10011 = 1:524288
 10010 = 1:262144
 10001 = 1:131072
 10000 = 1:65536
 01111 = 1:32768
 01110 = 1:16384
 01101 = 1:8192
 01100 = 1:4096
 01011 = 1:2048
 01010 = 1:1024
 01001 = 1:512
 01000 = 1:256
 00111 = 1:128
 00110 = 1:64
 00101 = 1:32
 00100 = 1:16
 00011 = 1:8
 00010 = 1:4
 00001 = 1:2
 00000 = 1:1

All other combinations not shown result in operation = 10100

PIC32MX

bit 15-14	FCKSM<1:0> : Clock Switching and Monitor Selection Configuration bits 1x = Clock switching is disabled, fail-safe clock monitor is disabled 01 = Clock switching is enabled, fail-safe clock monitor is disabled 00 = Clock switching is enabled, fail-safe clock monitor is enabled
bit 13-12	FPBDIV<1:0> : Peripheral Bus Clock Divisor Default Value bits 11 = PBCLK is SYSCLK divided by 8 10 = PBCLK is SYSCLK divided by 4 01 = PBCLK is SYSCLK divided by 2 00 = PBCLK is SYSCLK divided by 1
bit 11	Reserved : Maintain as '1'
bit 10	OSCIOFNC : CLKO Enable Configuration bit 1 = CLKO output signal active on the OSCO pin; primary oscillator must be disabled or configured for the External Clock mode (EC) for the CLKO to be active (POSCMD<1:0> = 11 OR 00) 0 = CLKO output disabled
bit 9-8	POSCMD<1:0> : Primary Oscillator Configuration bits 11 = Primary oscillator disabled 10 = HS oscillator mode selected 01 = XT oscillator mode selected 00 = External clock mode selected
bit 7	IESO : Internal External Switch Over bit 1 = Internal External Switch Over mode enabled (Two-Speed Start-up enabled) 0 = Internal External Switch Over mode disabled (Two-Speed Start-up disabled)
bit 6	Reserved : Maintain as '1'
bit 5	FSOSCEN : Secondary Oscillator Enable bit 1 = Enable Secondary Oscillator 0 = Disable Secondary Oscillator
bit 4-3	Reserved : Maintain as '1'
bit 2-0	FNOSC<2:0> : Oscillator Selection bits 000 = Fast RC Oscillator (FRC) 001 = Fast RC Oscillator with divide-by-N with PLL Module (FRCDIV+PLL) 010 = Primary Oscillator (XT, HS, EC) 011 = Primary Oscillator with PLL Module (XT+PLL, HS+PLL, EC+PLL) 100 = Secondary Oscillator (SOSC) 101 = Low-Power RC Oscillator (LPRC) 110 = Reserved; do not use 111 = Fast RC Oscillator with divide-by-N (FRCDIV)

REGISTER 18-3: DEVCFG2: DEVICE CONFIGURATION WORD 2

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 31						bit 24	
r-1	r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1
—	—	—	—	—	FPLLODIV<2:0>		
bit 23						bit 16	
R/P-1	r-1	r-1	r-1	r-1	R/P-1	R/P-1	R/P-1
FUPLLEN	—	—	—	—	FUPLLDIV<2:0>		
bit 15						bit 8	
r-1	R/P-1	R/P-1	R/P-1	r-1	R/P-1	R/P-1	R/P-1
—	FPLLMULT<2:0>			—	FPLLDIV<2:0>		
bit 7						bit 0	

Legend:

R = readable bit W = writable bit P = programmable bit r = reserved bit
U = unimplemented bit, read as '0' -n = bit value at POR: ('0', '1', x = unknown)

- bit 31-19 **Reserved:** Maintain as '1'
- bit 18-16 **FPLLODIV<2:0>:** Default Postscaler for PLL bits
111 = PLL output divided by 256
110 = PLL output divided by 64
101 = PLL output divided by 32
100 = PLL output divided by 16
011 = PLL output divided by 8
010 = PLL output divided by 4
001 = PLL output divided by 2
000 = PLL output divided by 1
- bit 15 **FUPLLEN:** USB PLL Enable bit
00 = Enable USB PLL
00 = Disable and bypass USB PLL
- bit 14-11 **Reserved:** Maintain as '1'
- bit 10-8 **FUPLLDIV<2:0>:** PLL Input Divider bits
000 = 1x divider
001 = 2x divider
010 = 3x divider
011 = 4x divider
100 = 5x divider
101 = 6x divider
110 = 10x divider
111 = 12x divider
- bit 7 **Reserved:** Maintain as '1'

(Continued)

bit 6-4	FPLLMULT<2:0> : PLL Multiplier bits
	111 = 24x multiplier
	110 = 21x multiplier
	101 = 20x multiplier
	100 = 19x multiplier
	011 = 18x multiplier
	010 = 17x multiplier
	001 = 16x multiplier
	000 = 15x multiplier
bit 3	Reserved: Maintain as '1'
bit 2-0	FPLLIDIV<2:0> : PLL Input Divider bits
	111 = 12x divider
	110 = 10x divider
	101 = 6x divider
	100 = 5x divider
	011 = 4x divider
	010 = 3x divider
	001 = 2x divider
	000 = 1x divider

REGISTER 18-4: DEVCFG3: DEVICE CONFIGURATION WORD 3

r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 31							bit 24
r-1	r-1	r-1	r-1	r-1	r-1	r-1	r-1
—	—	—	—	—	—	—	—
bit 23							bit 16
R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x
USERID15	USERID14	USERID13	USERID12	USERID11	USERID10	USERID9	USERID8
bit 15							bit 8
R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x	R/P-x
USERID7	USERID6	USERID5	USERID4	USERID3	USERID2	USERID1	USERID0
bit 7							bit 0

Legend:

R = readable bit W = writable bit P = programmable bit r = reserved bit
U = unimplemented bit, read as '0' -n = bit value at POR: ('0', '1', x = unknown)

bit 31-16 **Reserved:** Maintain as '1'
bit 15-0 **USERID:** This is a 16-bit value that is user defined and is readable via ICSP™ and JTAG.

PIC32MX

REGISTER 18-5: DEVID: DEVICE ID REGISTER

R	R	R	R	r	r	r	r
VER3	VER2	VER1	VER0	—	—	—	—
bit 31				bit 24			
r	r	r	r	R	R	R	R
—	—	—	—	DEVID7	DEVID6	DEVID5	DEVID4
bit 23				bit 16			
R	R	R	R	r	r	r	r
DEVID3	DEVID2	DEVID1	DEVID0	—	—	—	—
bit 15				bit 8			
r	r	r	r	r	r	r	R-1
—	—	—	—	—	—	—	1
bit 7				bit 0			

Legend:

R = readable bit

W = writable bit

P = programmable bit

r = reserved bit

U = unimplemented bit, read as '0'

-n = bit value at POR: ('0', '1', x = unknown)

bit 31-20 **VER<11:0>**: Revision Identifier bits

bit 19-12 **DEVID<7:0>**: Device ID

78h = PIC32MX460F512L

74h = PIC32MX460F256L

6Dh = PIC32MX440F128L

56h = PIC32MX440F512H

52h = PIC32MX440F256H

4Dh = PIC32MX440F128H

42h = PIC32MX420F032H

38h = PIC32MX360F512L

34h = PIC32MX360F256L

2Dh = PIC32MX340F128L

2Ah = PIC32MX320F128L

16h = PIC32MX340F512H

12h = PIC32MX340F256H

0Dh = PIC32MX340F128H

0Ah = PIC32MX320F128H

06h = PIC32MX320F064H

02h = PIC32MX320F032H

bit 11-1 **Reserved:** For Factory use only

bit 0 **Fixed Value:** Read as '1'

18.1 Device Configuration

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

TABLE 18-3: DEVCFG LOCATIONS

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

On Power-on Reset (POR) or any Reset, the Configuration Words are copied from boot Flash memory to their corresponding Configuration registers. A Configuration bit can only be programmed = 0 (un-programmed state = 1). During programming, a Configuration Word can be programmed a maximum of two times before a page erase must be performed.

After programming the Configuration Words, the user should reset the device to ensure the Configuration registers are reloaded with the new programmed data.

18.1.1 CONFIGURATION REGISTER PROTECTION

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

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

18.2 Device Code Protection

The PIC32MX features a single device code protection bit, CP that when programmed = 0, protects boot Flash and program Flash from being read or modified by an external programming device. When code protection is enabled, only the Device ID and User ID registers are available to be read by an external programmer. Boot Flash and program Flash memory are not protected from self-programming during program execution when code protection is enabled. See **Section 18.3 “Program Write Protection (PWP)”**

18.3 Program Write Protection (PWP)

In addition to a device code protection bit, the PIC32MX also features write protection bits to prevent boot Flash and program Flash memory regions from being written during code execution.

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

Program Flash memory can be write-protected entirely or in selectable page sizes using Configuration bits PWP<7:0> (BCFG0<19:12>). A page of program Flash memory is 4096 bytes (1024 words). The PWP bits represent the one's complement of the number of protected pages. For example, programming PWP bits = 0xFF selects 0 pages to be write-protected, effectively disabling the program Flash write protection. Programming PWP bits = 0xFE selects the first page to be write-protected. When enabled, the write-protected memory range is inclusive from the beginning of program Flash memory (0xBD00_0000) up through the selected page. Refer to Table 18-4.

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

TABLE 18-4: FLASH PROGRAM MEMORY WRITE PROTECT RANGES

PWP Bit Value	Range Size (K-bytes)	Write Protected Memory Ranges ⁽¹⁾
0xFF	0	disabled
0xFE	4	0xBD00_0FFF
0xFD	8	0xBD00_1FFF
0xFC	12	0xBD00_2FFF
0xFB	16	0xBD00_3FFF
0xFA	20	0xBD00_4FFF
0xF9	24	0xBD00_5FFF
0xF8	28	0xBD00_6FFF
0xF7	32	0xBD00_7FFF
0xF6	36	0xBD00_8FFF
0xF5	40	0xBD00_9FFF
0xF4	44	0xBD00_AFFF
0xF3	48	0xBD00_BFFF
0xF2	52	0xBD00_CFFF
0xF1	56	0xBD00_DFFF
0xF0	60	0xBD00_EFFF
0xEF	64	0xBD00_FFFF
...		
0x7F	512	0xBD07_FFFF

Note 1: Write-protected memory range is inclusive from 0xBD00_0000.

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

19.0 APPENDIX B: TAP CONTROLLERS

TABLE 19-1: MCHP TAP INSTRUCTIONS

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

19.1 Microchip TAP Controllers (MTAP)

19.1.1 MTAP_COMMAND

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

19.1.1.1 MCHP_STATUS

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

19.1.1.2 MCHP_ASERT_RST

This command performs a persistent device Reset. It is similar to asserting and holding $\overline{\text{MCLR}}$ with the exception that test modes are not detected. Its associated Status bit is DEVRST.

19.1.1.3 MCHP_DE_ASERT_RST

This commands removes the persistent device Reset. It is similar to de-asserting $\overline{\text{MCLR}}$. Its associated Status bit is DEVRST.

19.1.1.4 MCHP_ERASE

This command 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), this internal bit is cleared.

19.1.1.5 MCHP_FLASH_ENABLE

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

19.1.1.6 MCHP_FLASH_DISABLE

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

19.1.2 MTAP_SW_MTAP

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

19.1.3 MTAP_SW_ETAP

The MTAP_SW_ETAP instruction 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_SWTAP instruction is decoded by the MCHP TAP controller.

TABLE 19-2: MTAP_COMMAND DR COMMANDS

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

TABLE 19-3: MCHP STATUS VALUE

CPS	0	0	0	CFGRDY	FCBUSY	FAEN	DEVIRST
bit 7							bit 0

bit 7	CPS: Code-Protect State bit 0 = Device is code-protected 1 = Device is NOT code-protected
bit 6	Unimplemented: Read as '0'
bit 5-4	Unimplemented: Read as '0'
bit 3	CFGRDY: Code-Protect State bit 0 = Configuration has not been read 1 = Configuration has been read and CP is valid
bit 2	FCBUSY: Flash Controller Busy bit 0 = Flash Controller is Not Busy (Either erase has not started or it has finished) 1 = Flash Controller is Busy (Erase is in progress)
bit 1	FAEN: Flash Access Enable bit This bit reflects the state of CFGCON.FAEN. 0 = Flash access is disabled (i.e., processor accesses are blocked) 1 = Flash access is enabled
bit 0	DEVIRST: Device Reset State bit 0 = Device Reset is NOT active 1 = Device Reset is active

TABLE 19-4: EJTAG TAP INSTRUCTIONS

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

19.2 EJTAG TAP Controller

19.2.1 ETAP_ADDRESS

This command 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 LSBs of the register are used with the Psz field from the EJTAG Control register to indicate the size and data position of the pending processor access transfer. These bits are not taken directly from the address referenced by the load/store.

19.2.2 ETAP_DATA

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

19.2.3 ETAP_CONTROL

This command selects the control register. The EJTAG Control Register (ECR) handles processor Reset and soft Reset indication, Debug mode indication, access start, finish, and size and read/write indication. The ECR also:

- controls debug vector location and indication of serviced processor accesses,
- allows a debug interrupt request,
- indicates processor Low-Power mode, and
- allows implementation-dependent processor and peripheral Resets.

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

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

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

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

19.2.4 ETAP_EJTAGBOOT

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

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

The results of setting these bits are:

- Setting the EjtagBrk causes a Debug interrupt exception to be requested right after the processor Reset from the EJTAGBOOT instruction
- The debug handler is executed from the EJTAG memory because ProbTrap is set to indicate debug vector in EJTAG memory at 0xFF20_0200
- Service of the processor access is indicated because ProbEn is set

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

19.2.5 ETAP_FASTDATA

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

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

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

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

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

20.0 APPENDIX C: AC/DC CHARACTERISTICS AND TIMING REQUIREMENTS

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

Standard Operating Conditions						
Operating Temperature: 0°C to +70°C. Programming at +25°C is recommended.						
Param No.	Symbol	Characteristic	Min.	Max.	Units	Conditions
D111	VDD	Supply Voltage During Programming	3.0V	3.60	V	Normal programming ^(1,2)
D112	I _{PP}	Programming Current on $\overline{\text{MCLR}}$	—	5	μA	
D113	I _{DDP}	Supply Current During Programming	—	40	mA	
D031	V _{IL}	Input Low Voltage	V _{SS}	0.2 V _{DD}	V	
D041	V _{IH}	Input High Voltage	0.8 V _{DD}	V _{DD}	V	
D080	V _{OL}	Output Low Voltage	—	0.4	V	I _{OL} = 8.5 mA @ 3.6V
D090	V _{OH}	Output High Voltage	1.4	—	V	I _{OH} = -3.0 mA @ 3.6V
D012	C _{IO}	Capacitive Loading on I/O pin (PGDx)	—	50	pF	To meet AC specifications
D013	C _F	Filter Capacitor Value on V _{CAP}	1	10	μF	Required for controller core
P1	T _{PGC}	Serial Clock (PGCx) Period	100	—	ns	
P1A	T _{PGCL}	Serial Clock (PGCx) Low Time	40	—	ns	
P1B	T _{PGCH}	Serial Clock (PGCx) High Time	40	—	ns	
P2	T _{SET1}	Input Data Setup Time to Serial Clock ↓	15	—	ns	
P3	T _{HL1}	Input Data Hold Time from PGCx ↓	15	—	ns	
P4	T _{DLY1}	Delay between 4-bit Command and Command Operand	40	—	ns	
P4A	T _{DLY1A}	Delay between 4-bit Command Operand and Next 4-bit Command	40	—	ns	
P5	T _{DLY2}	Delay between Last PGCx ↓ of Command Byte to First PGCx ↑ of Read of Data Word	20	—	ns	
P6	T _{SET2}	V _{DD} ↑ Setup Time to $\overline{\text{MCLR}}$ ↑	100	—	ns	
P7	T _{HL2}	Input Data Hold Time from $\overline{\text{MCLR}}$ ↑	500	—	ns	
P8	T _{DLY3}	Delay between Last PGCx ↓ of Command Byte to PGDx ↑ by Programming Executive	20	—	μs	
P9A	T _{DLY4}	Programming Executive Command Processing Time	40	—	μs	
P9B	T _{DLY5}	Delay between PGDx ↓ by Programming Executive to PGDx Released by Programming Executive	15	—	μs	
P10	T _{DLY6}	PGCx Low Time After Programming	400	—	ns	
P11	T _{DLY7}	Chip Erase Time	400	—	ms	
P12	T _{DLY8}	Page Erase Time	40	—	ms	
P13	T _{DLY9}	Row Programming Time	2	—	ms	
P14	T _R	$\overline{\text{MCLR}}$ Rise Time to Enter ICSP™ mode	—	1.0	μs	
P15	T _{VALID}	Data Out Valid from PGCx ↑	10	—	ns	
P16	T _{DLY8}	Delay between Last PGCx ↓ and $\overline{\text{MCLR}}$ ↓	0	—	s	
P17	T _{HL3}	$\overline{\text{MCLR}}$ ↓ to V _{DD} ↓	—	100	ns	
P18	T _{KEY1}	Delay from First $\overline{\text{MCLR}}$ ↓ to First PGCx ↑ for Key Sequence on PGDx	40	—	ns	

Note 1: VDDCORE must be supplied to the VDDCORE/VCAP pin if the on-chip voltage regulator is disabled. See **Section 4.3 “Power Requirements”** for more information.

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

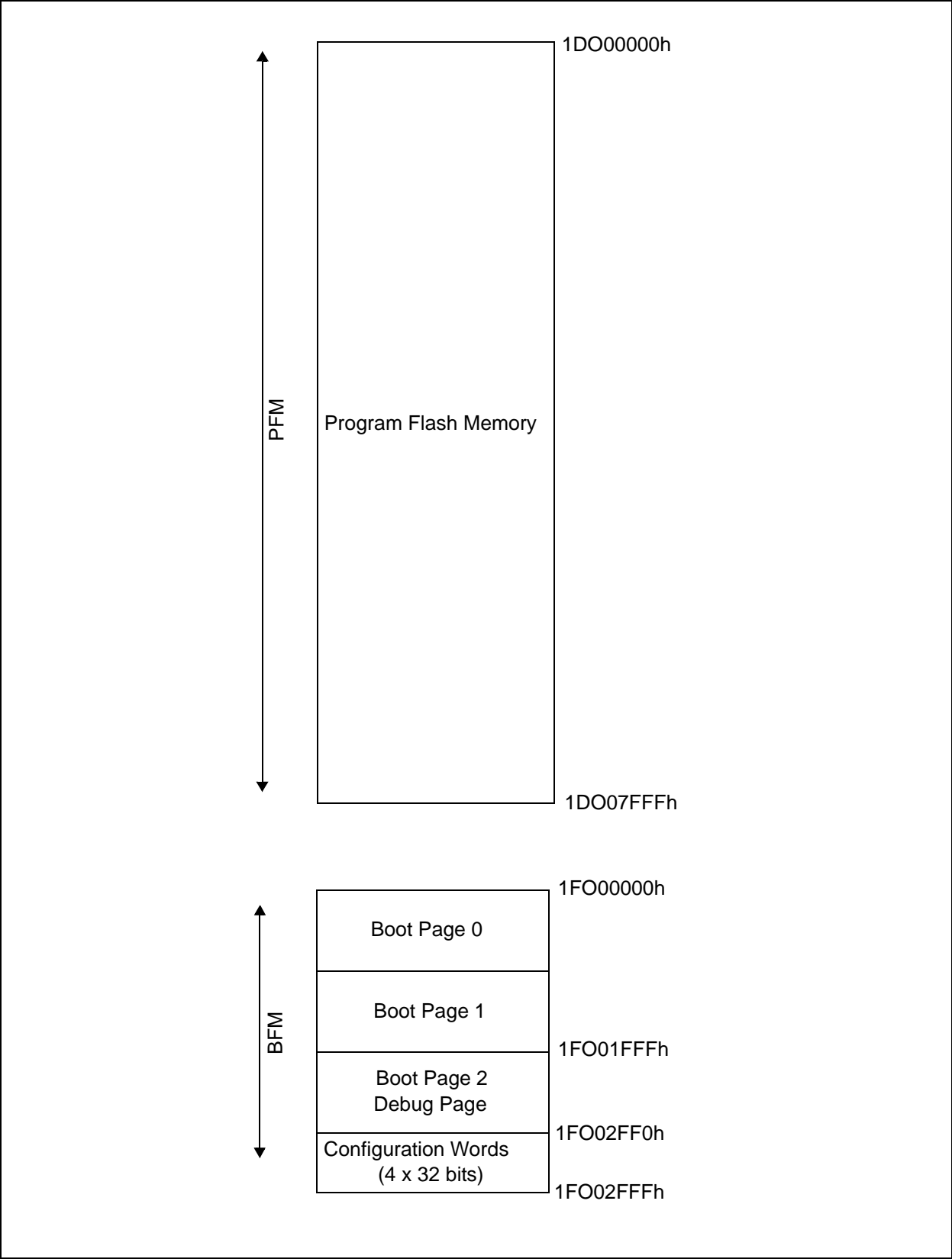
TABLE 20-1: AC/DC CHARACTERISTICS AND TIMING REQUIREMENTS (CONTINUED)

Standard Operating Conditions Operating Temperature: 0°C to +70°C. Programming at +25°C is recommended.						
Param No.	Symbol	Characteristic	Min.	Max.	Units	Conditions
P19	TKEY2	Delay from Last PGCx ↓ for Key Sequence on PGDx to Second MCLR ↑	40	—	ns	

- Note 1:** VDDCORE must be supplied to the VDDCORE/VCAP pin if the on-chip voltage regulator is disabled. See **Section 4.3 “Power Requirements”** for more information.
- 2:** VDD must also be supplied to the AVDD pins during programming. AVDD and AVSS should always be within ±0.3V of VDD and VSS, respectively.

21.0 APPENDIX D: PIC32MX FLASH MEMORY MAP

FIGURE 21-1: FLASH MEMORY MAP



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

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

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

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


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

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC³² logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rLAB, Select Mode, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

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

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



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta

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

Boston

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

Chicago

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

Dallas

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

Detroit

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

Kokomo

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

Los Angeles

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

Santa Clara

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

Toronto

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

ASIA/PACIFIC

Asia Pacific Office

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

Australia - Sydney

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

China - Beijing

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

China - Chengdu

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

China - Hong Kong SAR

Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing

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

China - Qingdao

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

China - Shanghai

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

China - Shenyang

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

China - Shenzhen

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

China - Wuhan

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

China - Xiamen

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

China - Xian

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

China - Zhuhai

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

ASIA/PACIFIC

India - Bangalore

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

India - New Delhi

Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune

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

Japan - Yokohama

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

Korea - Daegu

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

Korea - Seoul

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

Malaysia - Kuala Lumpur

Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang

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

Philippines - Manila

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

Singapore

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

Taiwan - Hsin Chu

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

Taiwan - Kaohsiung

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

Taiwan - Taipei

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

Thailand - Bangkok

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

EUROPE

Austria - Wels

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

Denmark - Copenhagen

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

France - Paris

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

Germany - Munich

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

Italy - Milan

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

Netherlands - Drunen

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

Spain - Madrid

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

UK - Wokingham

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