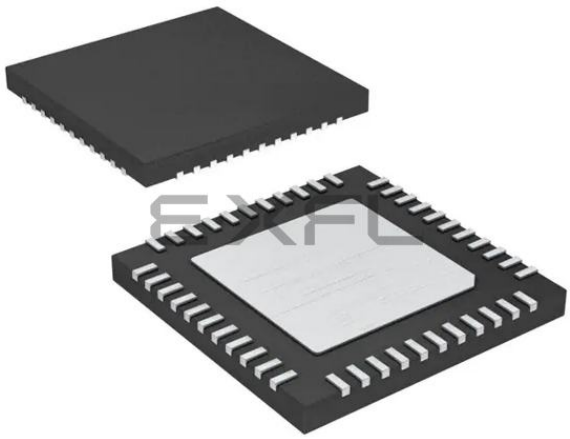
# Microchip Technology - DSPIC30F3011T-20I/ML Datasheet



**Welcome to E-XFL.COM**

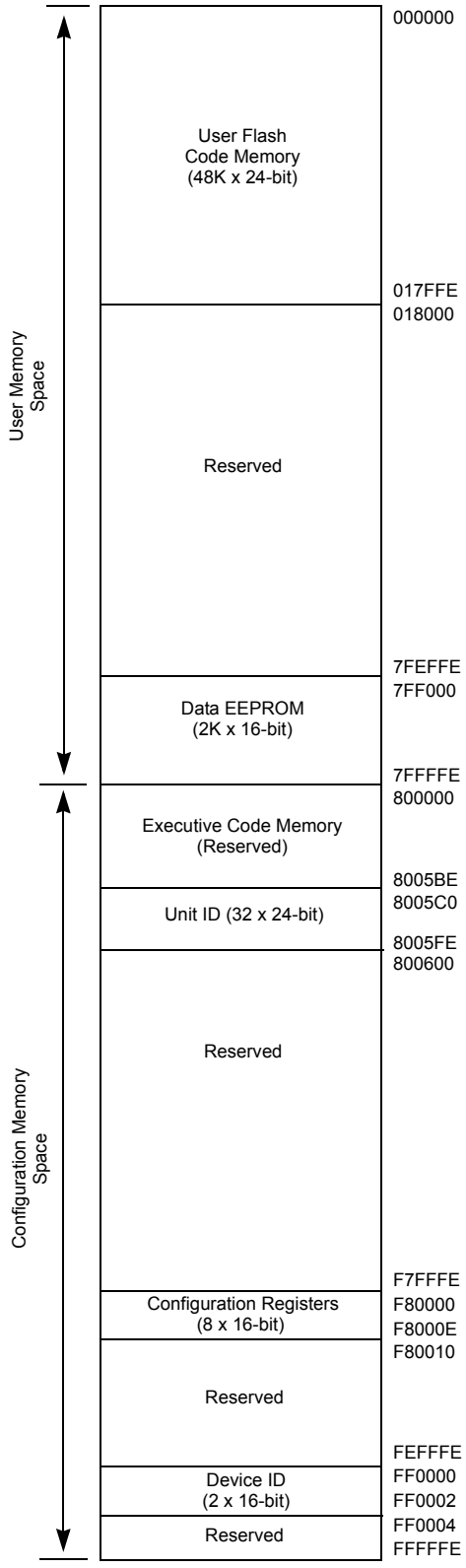
## What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

## Applications of "Embedded - Microcontrollers"

| Details | |
|---|---|
| Product Status | Obsolete |
| Core Processor | dsPIC |
| Core Size | 16-Bit |
| Speed | 20 MIPS |
| Connectivity | I²C, SPI, UART/USART |
| Peripherals | Brown-out Detect/Reset, Motor Control PWM, QEI, POR, PWM, WDT |
| Number of I/O | 30 |
| Program Memory Size | 24KB (8K x 24) |
| Program Memory Type | FLASH |
| EEPROM Size | 1K x 8 |
| RAM Size | 1K x 8 |
| Voltage - Supply (Vcc/Vdd) | 2.5V ~ 5.5V |
| Data Converters | A/D 9x10b |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 85°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 44-VQFN Exposed Pad |
| Supplier Device Package | 44-QFN (8x8) |
| Purchase URL | https://www.e-xfl.com/product-detail/microchip-technology/dspic30f3011t-20i-ml |

**FIGURE 2-2: PROGRAM MEMORY MAP**




**Note:** The address boundaries for user Flash code memory and data EEPROM are device-dependent.

## 5.5 Code Memory Programming

### 5.5.1 OVERVIEW

The Flash code memory array consists of 512 rows of thirty-two, 24-bit instructions. Each panel stores 16K instruction words, and each dsPIC30F device has either 1, 2 or 3 memory panels (see Table 5-2).

**TABLE 5-2: DEVICE CODE MEMORY SIZE**

| Device | Code Size (24-bit Words) | Number of Rows | Number of Panels |
|---|---|---|---|
| dsPIC30F2010 | 4K | 128 | 1 |
| dsPIC30F2011 | 4K | 128 | 1 |
| dsPIC30F2012 | 4K | 128 | 1 |
| dsPIC30F3010 | 8K | 256 | 1 |
| dsPIC30F3011 | 8K | 256 | 1 |
| dsPIC30F3012 | 8K | 256 | 1 |
| dsPIC30F3013 | 8K | 256 | 1 |
| dsPIC30F3014 | 8K | 256 | 1 |
| dsPIC30F4011 | 16K | 512 | 1 |
| dsPIC30F4012 | 16K | 512 | 1 |
| dsPIC30F4013 | 16K | 512 | 1 |
| dsPIC30F5011 | 22K | 704 | 2 |
| dsPIC30F5013 | 22K | 704 | 2 |
| dsPIC30F5015 | 22K | 704 | 2 |
| dsPIC30F5016 | 22K | 704 | 2 |
| dsPIC30F6010 | 48K | 1536 | 3 |
| dsPIC30F6010A | 48K | 1536 | 3 |
| dsPIC30F6011 | 44K | 1408 | 3 |
| dsPIC30F6011A | 44K | 1408 | 3 |
| dsPIC30F6012 | 48K | 1536 | 3 |
| dsPIC30F6012A | 48K | 1536 | 3 |
| dsPIC30F6013 | 44K | 1408 | 3 |
| dsPIC30F6013A | 44K | 1408 | 3 |
| dsPIC30F6014 | 48K | 1536 | 3 |
| dsPIC30F6014A | 48K | 1536 | 3 |
| dsPIC30F6015 | 48K | 1536 | 3 |

### 5.5.2 PROGRAMMING METHODOLOGY

Code memory is programmed with the PROGP command. PROGP programs one row of code memory to the memory address specified in the command. The number of PROGP commands required to program a device depends on the number of rows that must be programmed in the device.

A flowchart for programming of code memory is illustrated in Figure 5-3. In this example, all 48K instruction words of a dsPIC30F6014A device are programmed. First, the number of commands to send (called 'RemainingCmds' in the flowchart) is set to 1536 and the destination address (called 'BaseAddress') is set to '0'.

Next, one row in the device is programmed with a PROGP command. Each PROGP command contains data for one row of code memory of the dsPIC30F6014A. After the first command is processed successfully, 'RemainingCmds' is decremented by 1 and compared to 0. Since there are more PROGP commands to send, 'BaseAddress' is incremented by 0x40 to point to the next row of memory.

On the second PROGP command, the second row of each memory panel is programmed. This process is repeated until the entire device is programmed. No special handling must be performed when a panel boundary is crossed.

**FIGURE 5-3: FLOWCHART FOR PROGRAMMING dsPIC30F6014A CODE MEMORY**

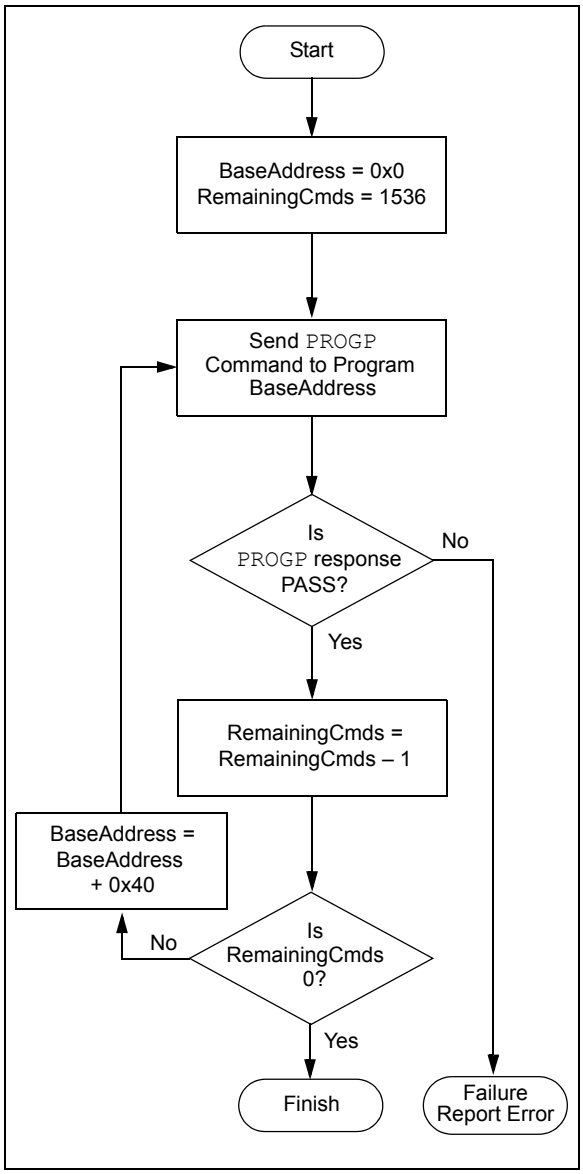


**TABLE 5-6: FOSC CONFIGURATION BITS DESCRIPTION FOR dsPIC30F2011/2012, dsPIC30F3010/3011/3012/3013/3014, dsPIC30F4013, dsPIC30F5015/5016, dsPIC30F6010A/6011A/6012A/6013A/6014A AND dsPIC30F6015**

| Bit Field | Register | Description |
|---|---|---|
| FCKSM<1:0> | FOSC | **Clock Switching Mode**<br>1x = Clock switching is disabled, Fail-Safe Clock Monitor is disabled<br>01 = Clock switching is enabled, Fail-Safe Clock Monitor is disabled<br>00 = Clock switching is enabled, Fail-Safe Clock Monitor is enabled |
| FOS<2:0> | FOSC | **Oscillator Source Selection on POR**<br>111 = Primary Oscillator<br>110 = Reserved<br>101 = Reserved<br>100 = Reserved<br>011 = Reserved<br>010 = Internal Low-Power RC Oscillator<br>001 = Internal Fast RC Oscillator (no PLL)<br>000 = Low-Power 32 kHz Oscillator (Timer1 Oscillator) |
| FPR<4:0> | FOSC | **Primary Oscillator Mode (when FOS<2:0> = 111b)**<br>11xxx = Reserved (do not use)<br>10111 = HS/3 w/PLL 16X – HS/3 crystal oscillator with 16X PLL (10 MHz-25 MHz crystal)<br>10110 = HS/3 w/PLL 8X – HS/3 crystal oscillator with 8X PLL (10 MHz-25 MHz crystal)<br>10101 = HS/3 w/PLL 4X – HS/3 crystal oscillator with 4X PLL (10 MHz-25 MHz crystal)<br>10100 = Reserved (do not use)<br>10011 = HS/2 w/PLL 16X – HS/2 crystal oscillator with 16X PLL (10 MHz-25 MHz crystal)<br>10010 = HS/2 w/PLL 8X – HS/2 crystal oscillator with 8X PLL (10 MHz-25 MHz crystal<br>10001 = HS/2 w/PLL 4X – HS/2 crystal oscillator with 4X PLL (10 MHz-25 MHz crystal)<br>10000 = Reserved (do not use)<br>01111 = ECIO w/PLL 16x – External clock with 16x PLL. OSC2 pin is I/O<br>01110 = ECIO w/PLL 8x – External clock with 8x PLL. OSC2 pin is I/O<br>01101 = ECIO w/PLL 4x – External clock with 4x PLL. OSC2 pin is I/O<br>01100 = Reserved (do not use)<br>01011 = Reserved (do not use)<br>01010 = FRC w/PLL 8x – Internal fast RC oscillator with 8x PLL. OSC2 pin is I/O<br>01001 = Reserved (do not use)<br>01000 = Reserved (do not use)<br>00111 = XT w/PLL 16X – XT crystal oscillator with 16X PLL<br>00110 = XT w/PLL 8X – XT crystal oscillator with 8X PLL<br>00101 = XT w/PLL 4X – XT crystal oscillator with 4X PLL<br>00100 = Reserved (do not use)<br>00011 = FRC w/PLL 16x – Internal fast RC oscillator with 8x PLL. OSC2 pin is I/O<br>00010 = Reserved (do not use)<br>00001 = FRC w/PLL 4x – Internal fast RC oscillator with 4x PLL. OSC2 pin is I/O<br>00000 = Reserved (do not use) |

**TABLE 5-7: CONFIGURATION BITS DESCRIPTION**

| Bit Field | Register | Description |
|---|---|---|
| FWPSA<1:0> | FWDT | **Watchdog Timer Prescaler A**<br>11 = 1:512<br>10 = 1:64<br>01 = 1:8<br>00 = 1:1 |
| FWPSB<3:0> | FWDT | **Watchdog Timer Prescaler B**<br>1111 = 1:16<br>1110 = 1:15<br>•<br>•<br>•<br>0001 = 1:2<br>0000 = 1:1 |
| FWDTEN | FWDT | **Watchdog Enable**<br>1 = Watchdog enabled (LPRC oscillator cannot be disabled. Clearing the SWDTEN bit in the RCON register will have no effect)<br>0 = Watchdog disabled (LPRC oscillator can be disabled by clearing the SWDTEN bit in the RCON register) |
| MCLREN | FBORPOR | **Master Clear Enable** $\overline{\text{MCLR}}$<br>1 = Master Clear pin ($\overline{\text{MCLR}}$) is enabled<br>0 = $\overline{\text{MCLR}}$ pin is disabled |
| PWMPIN | FBORPOR | **Motor Control PWM Module Pin Mode**<br>1 = PWM module pins controlled by PORT register at device Reset (tri-stated)<br>0 = PWM module pins controlled by PWM module at device Reset (configured as output pins) |
| HPOL | FBORPOR | **Motor Control PWM Module High-Side Polarity**<br>1 = PWM module high-side output pins have active-high output polarity<br>0 = PWM module high-side output pins have active-low output polarity |
| LPOL | FBORPOR | **Motor Control PWM Module Low-Side Polarity**<br>1 = PWM module low-side output pins have active-high output polarity<br>0 = PWM module low-side output pins have active-low output polarity |
| BOREN | FBORPOR | **PBOR Enable**<br>1 = PBOR enabled<br>0 = PBOR disabled |
| BORV<1:0> | FBORPOR | **Brown-out Voltage Select**<br>11 = 2.0V (not a valid operating selection)<br>10 = 2.7V<br>01 = 4.2V<br>00 = 4.5V |
| FPWRT<1:0> | FBORPOR | **Power-on Reset Timer Value Select**<br>11 = PWRT = 64 ms<br>10 = PWRT = 16 ms<br>01 = PWRT = 4 ms<br>00 = Power-up Timer disabled |
| RBS<1:0> | FBS | **Boot Segment Data RAM Code Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>11 = No Data RAM is reserved for Boot Segment<br>10 = Small-sized Boot RAM<br>[128 bytes of RAM are reserved for Boot Segment]<br>01 = Medium-sized Boot RAM<br>[256 bytes of RAM are reserved for Boot Segment]<br>00 = Large-sized Boot RAM<br>[512 bytes of RAM are reserved for Boot Segment in dsPIC30F5011/5013, and 1024 bytes in dsPIC30F6010A/6011A/6012A/6013A/6014A/6015] |

**TABLE 5-7: CONFIGURATION BITS DESCRIPTION (CONTINUED)**

| Bit Field | Register | Description |
|---|---|---|
| EBS | FBS | **Boot Segment Data EEPROM Code Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>1 = No Data EEPROM is reserved for Boot Segment<br>0 = 128 bytes of Data EEPROM are reserved for Boot Segment in dsPIC30F5011/5013, and 256 bytes in dsPIC30F6010A/6011A/6012A/6013A/6014A/6015 |
| BSS<2:0> | FBS | **Boot Segment Program Memory Code Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>111 = No Boot Segment<br>110 = Standard security; Small-sized Boot Program Flash<br>[Boot Segment starts after BS and ends at 0x0003FF]<br>101 = Standard security; Medium-sized Boot Program Flash<br>[Boot Segment starts after BS and ends at 0x000FFF]<br>100 = Standard security; Large-sized Boot Program Flash<br>[Boot Segment starts after BS and ends at 0x001FF]<br>011 = No Boot Segment<br>010 = High security; Small-sized Boot Program Flash<br>[Boot Segment starts after BS and ends at 0x0003FF]<br>001 = High security; Medium-sized Boot Program Flash<br>[Boot Segment starts after BS and ends at 0x000FFF]<br>000 = High security; Large-sized Boot Program Flash<br>[Boot Segment starts after BS and ends at 0x001FFF] |
| BWRP | FBS | **Boot Segment Program Memory Write Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>1 = Boot Segment program memory is not write-protected<br>0 = Boot Segment program memory is write-protected |
| RSS<1:0> | FSS | **Secure Segment Data RAM Code Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>11 = No Data RAM is reserved for Secure Segment<br>10 = Small-sized Secure RAM<br>[(256 – N) bytes of RAM are reserved for Secure Segment]<br>01 = Medium-sized Secure RAM<br>[(768 – N) bytes of RAM are reserved for Secure Segment in dsPIC30F5011/5013, and (2048 – N) bytes in dsPIC30F6010A/6011A/6012A/6013A/6014A/6015]<br>00 = Large-sized Secure RAM<br>[(1024 – N) bytes of RAM are reserved for Secure Segment in dsPIC30F5011/5013, and (4096 – N) bytes in dsPIC30F6010A/6011A/6012A/6013A/6014A/6015]<br>where N = Number of bytes of RAM reserved for Boot Sector. |
| ESS<1:0> | FSS | **Secure Segment Data EEPROM Code Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>11 = No Data EEPROM is reserved for Secure Segment<br>10 = Small-sized Secure Data EEPROM<br>[(128 – N) bytes of Data EEPROM are reserved for Secure Segment in dsPIC30F5011/5013, and (256 – N) bytes in dsPIC30F6010A/6011A/6012A/6013A/6014A/6015]<br>01 = Medium-sized Secure Data EEPROM<br>[(256 – N) bytes of Data EEPROM are reserved for Secure Segment in dsPIC30F5011/5013, and (512 – N) bytes in dsPIC30F6010A/6011A/6012A/6013A/6014A/6015]<br>00 = Large-sized Secure Data EEPROM<br>[(512 – N) bytes of Data EEPROM are reserved for Secure Segment in dsPIC30F5011/5013, (1024 – N) bytes in dsPIC30F6011A/6013A, and (2048 – N) bytes in dsPIC30F6010A/6012A/6014A/6015]<br>where N = Number of bytes of Data EEPROM reserved for Boot Sector. |

### 5.7.2 PROGRAMMING METHODOLOGY

System operation Configuration bits are inherently different than all other memory cells. Unlike code memory, data EEPROM and code-protect Configuration bits, the system operation bits cannot be erased. If the chip is erased with the ERASEB command, the system-operation bits retain their previous value. Consequently, you should make no assumption about the value of the system operation bits. They should always be programmed to their desired setting.

Configuration bits are programmed as a single word at a time using the PROGC command. The PROGC command specifies the configuration data and Configuration register address. When Configuration bits are programmed, any unimplemented bits must be programmed with a '0', and any reserved bits must be programmed with a '1'.

Four PROGC commands are required to program all the Configuration bits. Figure 5-5 illustrates the flowchart of Configuration bit programming.

**Note:** If the General Code Segment Code Protect (GCP) bit is programmed to '0', code memory is code-protected and cannot be read. Code memory must be verified before enabling read protection. See **Section 5.7.4 "Code-Protect Configuration Bits"** for more information about code-protect Configuration bits.

### 5.7.3 PROGRAMMING VERIFICATION

Once the Configuration bits are programmed, the contents of memory should be verified to ensure that the programming was successful. Verification requires the Configuration bits to be read back and compared against the copy held in the programmer's buffer. The READD command reads back the programmed Configuration bits and verifies whether the programming was successful.

Any unimplemented Configuration bits are read-only and read as '0'.

### 5.7.4 CODE-PROTECT CONFIGURATION BITS

The FBS, FSS and FGS Configuration registers are special Configuration registers that control the size and level of code protection for the Boot Segment, Secure Segment and General Segment, respectively. For each segment, two main forms of code protection are provided. One form prevents code memory from being written (write protection), while the other prevents code memory from being read (read protection).

The BWRP, SWRP and GWRP bits control write protection; and BSS<2:0>, SSS<2:0> and GSS<1:0> bits control read protection. The Chip Erase ERASEB command sets all the code protection bits to '1', which allows the device to be programmed.

When write protection is enabled, any programming operation to code memory will fail. When read protection is enabled, any read from code memory will cause a '0x0' to be read, regardless of the actual contents of code memory. Since the programming executive always verifies what it programs, attempting to program code memory with read protection enabled will also result in failure.

It is imperative that all code protection bits are '1' while the device is being programmed and verified. Only after the device is programmed and verified should any of the above bits be programmed to '0' (see **Section 5.7 "Configuration Bits Programming"**).

In addition to code memory protection, parts of data EEPROM and/or data RAM can be configured to be accessible only by code resident in the Boot Segment and/or Secure Segment. The sizes of these "reserved" sections are user-configurable, using the EBS, RBS<1:0>, ESS<1:0> and RSS<1:0> bits.

**Note 1:** All bits in the FBS, FSS and FGS Configuration registers can only be programmed to a value of '0'. ERASEB is the only way to reprogram code-protect bits from ON ('0') to OFF ('1').

**2:** If any of the code-protect bits in FBS, FSS, or FGS are clear, the entire device must be erased before it can be reprogrammed.

## 6.6 Configuration Information in the Hexadecimal File

To allow portability of code, the programmer must read the Configuration register locations from the hexadecimal file. If configuration information is not present in the hexadecimal file, a simple warning message should be issued by the programmer. Similarly, while saving a hexadecimal file, all configuration information must be included. An option to not include the configuration information can be provided.

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

## 6.7 Unit ID

The dsPIC30F devices contain 32 instructions of Unit ID. These are located at addresses 0x8005C0 through 0x8005FF. The Unit ID can be used for storing product information such as serial numbers, system manufacturing dates, manufacturing lot numbers and other such application-specific information.

A Bulk Erase does not erase the Unit ID locations. Instead, erase all executive memory using steps 1-4 as shown in Table 12-1, and program the Unit ID along with the programming executive. Alternately, use a Row Erase to erase the row containing the Unit ID locations.

## 6.8 Checksum Computation

Checksums for the dsPIC30F are 16 bits in size. The checksum is to total sum of the following:

- Contents of code memory locations
- Contents of Configuration registers

Table A-1 describes how to calculate the checksum for each device. All memory locations are summed one byte at a time, using only their native data size. More specifically, Configuration and device ID registers are summed by adding the lower two bytes of these locations (the upper byte is ignored), while code memory is summed by adding all three bytes of code memory.

| **Note:** | The checksum calculation differs depending on the code-protect setting. Table A-1 describes how to compute the checksum for an unprotected device and a read-protected device. Regardless of the code-protect setting, the Configuration registers can always be read. |
|---|---|

# 7.0 PROGRAMMER – PROGRAMMING EXECUTIVE COMMUNICATION

## 7.1 Communication Overview

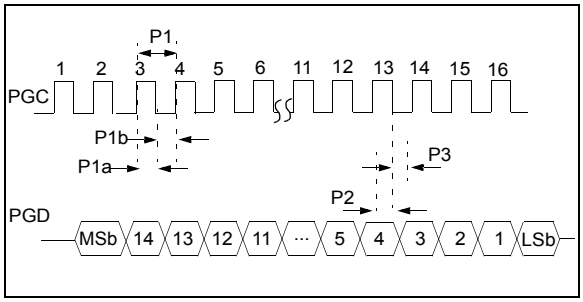
The programmer and programming executive have a master-slave relationship, where the programmer is the master programming device and the programming executive is the slave.

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. The programming executive command set is described in **Section 8.0 "Programming Executive Commands"**. The response set is described in **Section 9.0 "Programming Executive Responses"**.

## 7.2 Communication Interface and Protocol

The Enhanced ICSP interface is a 2-wire SPI interface implemented using the PGC and PGD pins. The PGC pin is used as a clock input pin, and the clock source must be provided by the programmer. The PGD pin is used for sending command data to, and receiving response data from, the programming executive. All serial data is transmitted on the falling edge of PGC and latched on the rising edge of PGC. All data transmissions are sent Most Significant bit (MSb) first, using 16-bit mode (see Figure 7-1).

**FIGURE 7-1: PROGRAMMING EXECUTIVE SERIAL TIMING**




Since a 2-wire SPI interface is used, and data transmissions are bidirectional, a simple protocol is used to control the direction of PGD. When the programmer completes a command transmission, it releases the PGD line and allows the programming executive to drive this line high. The programming executive keeps the PGD line high to indicate that it is processing the command.

After the programming executive has processed the command, it brings PGD low for 15 µsec to indicate to the programmer that the response is available to be

clocked out. The programmer can begin to clock out the response 20 µsec after PGD is brought low, and it must provide the necessary amount of clock pulses to receive the entire response from the programming executive.

Once the entire response is clocked out, the programmer should terminate the clock on PGC until it is time to send another command to the programming executive. This protocol is illustrated in Figure 7-2.

## 7.3 SPI Rate

In Enhanced ICSP mode, the dsPIC30F operates from the fast internal RC oscillator, which has a nominal frequency of 7.37 MHz. This oscillator frequency yields an effective system clock frequency of 1.84 MHz. Since the SPI module operates in Slave mode, the programmer must limit the SPI clock rate to a frequency no greater than 1 MHz.

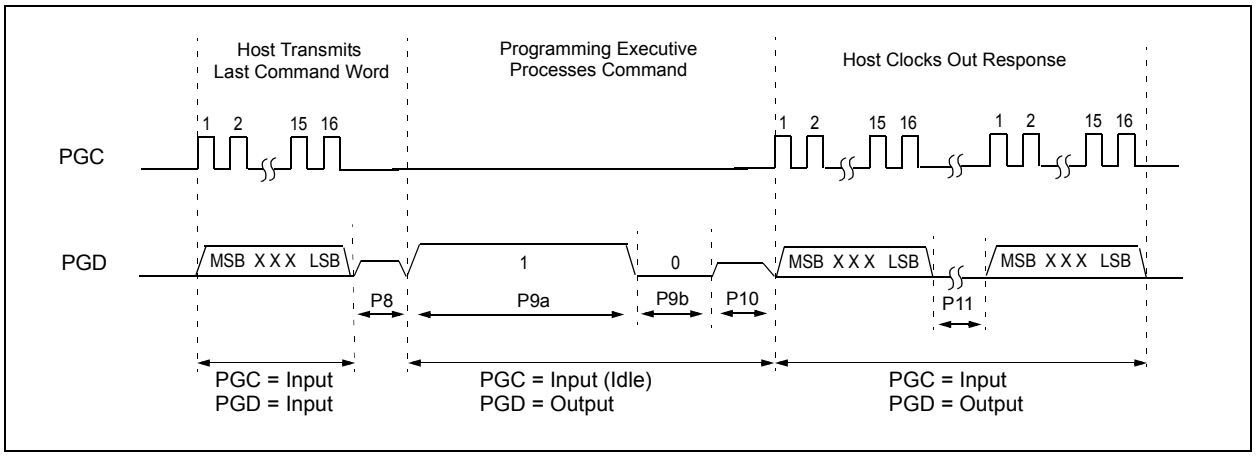
| **Note:** | If the programmer provides the SPI with a clock faster than 1 MHz, the behavior of the programming executive will be unpredictable. |
|---|---|

## 7.4 Time Outs

The programming executive uses no Watchdog Timer or time out for transmitting responses to the programmer. If the programmer does not follow the flow control mechanism using PGC, as described in **Section 7.2 "Communication Interface and Protocol"**, it is possible that the programming executive will behave unexpectedly while trying to send a response to the programmer. Since the programming executive has no time out, it is imperative that the programmer correctly follow the described communication protocol.

As a safety measure, the programmer should use the command time outs identified in Table 8-1. If the command time out expires, the programmer should reset the programming executive and start programming the device again.

**FIGURE 7-2: PROGRAMMING EXECUTIVE – PROGRAMMER COMMUNICATION PROTOCOL**



### 8.5.5 PROGP COMMAND

| 15 12 | 11 8 | 7 0 |
|---|---|---|
| Opcode | Length | |
| Reserved | Addr_MSB | |
| Addr_LS | | |
| D_1 | | |
| D_2 | | |
| ... | | |
| D_N | | |

| Field | Description |
|---|---|
| Opcode | 0x5 |
| Length | 0x33 |
| Reserved | 0x0 |
| Addr_MSB | MSB of 24-bit destination address |
| Addr_LS | LS 16 bits of 24-bit destination address |
| D_1 | 16-bit data word 1 |
| D_2 | 16-bit data word 2 |
| ... | 16-bit data word 3 through 47 |
| D_48 | 16-bit data word 48 |

The PROGP command instructs the programming executive to program one row of code memory (32 instruction words) to the specified memory address. Programming begins with the row address specified in the command. The destination address should be a multiple of 0x40.

The data to program to memory, located in command words D_1 through D_48, must be arranged using the packed instruction word format shown in Figure 8-2.

After all data has been programmed to code memory, the programming executive verifies the programmed data against the data in the command.

**Expected Response (2 words):**
0x1500
0x0002

**Note:** Refer to Table 5-2 for code memory size information.

### 8.5.6 PROGC COMMAND

| 15 12 | 11 8 | 7 0 |
|---|---|---|
| Opcode | Length | |
| Reserved | Addr_MSB | |
| Addr_LS | | |
| Data | | |

| Field | Description |
|---|---|
| Opcode | 0x6 |
| Length | 0x4 |
| Reserved | 0x0 |
| Addr_MSB | MSB of 24-bit destination address |
| Addr_LS | LS 16 bits of 24-bit destination address |
| Data | Data to program |

The PROGC command programs data to the specified Configuration register and verifies the programming. Configuration registers are 16 bits wide, and this command allows one Configuration register to be programmed.

**Expected Response (2 words):**
0x1600
0x0002

**Note:** This command can only be used for programming Configuration registers.

### 8.5.9 ERASEP COMMAND

| 15 12 | 11 8 | 7 0 |
|---|---|---|
| Opcode | Length | |
| Num_Rows | | Addr_MSB |
| Addr_LS | | |

| Field | Description |
|---|---|
| Opcode | 0x9 |
| Length | 0x3 |
| Num_Rows | Number of rows to erase |
| Addr_MSB | MSB of 24-bit base address |
| Addr_LS | LS 16 bits of 24-bit base address |

The ERASEP command erases the specified number of rows of code memory from the specified base address. The specified base address must be a multiple of 0x40.

Once the erase is performed, all targeted words of code memory contain 0xFFFFFF.

**Expected Response (2 words):**
0x1900
0x0002

**Note:** The ERASEP command cannot be used to erase the Configuration registers or device ID. Code-protect Configuration registers can only be erased with the ERASEB command, while the device ID is read-only.

### 8.5.10 QBLANK COMMAND

| 15 12 | 11 0 |
|---|---|
| Opcode | Length |
| PSize | |
| Reserved | DSize |

| Field | Description |
|---|---|
| Opcode | 0xA |
| Length | 0x3 |
| PSize | Length of program memory to check (in 24-bit words), max of 49152 |
| Reserved | 0x0 |
| DSize | Length of data memory to check (in 16-bit words), max of 2048 |

The QBLANK command queries the programming executive to determine if the contents of code memory and data EEPROM are blank (contains all '1's). The size of code memory and data EEPROM to check must be specified in the command.

The Blank Check for code memory begins at 0x0 and advances toward larger addresses for the specified number of instruction words. The Blank Check for data EEPROM begins at 0x7FFFFE and advances toward smaller addresses for the specified number of data words.

QBLANK returns a QE_Code of 0xF0 if the specified code memory and data EEPROM are blank. Otherwise, QBLANK returns a QE_Code of 0x0F.

**Expected Response (2 words for blank device):**
0x1AF0
0x0002

**Expected Response (2 words for non-blank device):**
0x1A0F
0x0002

**Note:** The QBLANK command does not check the system Configuration registers. The READD command must be used to determine the state of the Configuration registers.

### 8.5.11 QVER COMMAND

| 15 12 | 11 0 |
|---|---|
| Opcode | Length |

| Field | Description |
|---|---|
| Opcode | 0xB |
| Length | 0x1 |

The QVER command queries the version of the programming executive software stored in test memory. The “version.revision” information is returned in the response’s QE_Code using a single byte with the following format: main version in upper nibble and revision in the lower nibble (i.e., 0x23 is version 2.3 of programming executive software).

**Expected Response (2 words):**
0x1BMN (where “MN” stands for version M.N)
0x0002

# 9.0 PROGRAMMING EXECUTIVE RESPONSES

## 9.1 Overview

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

The programming executive response set is shown in Table 9-1. This table contains the opcode, mnemonic and description for each response. The response format is described in **Section 9.2 “Response Format”**.

**TABLE 9-1: PROGRAMMING EXECUTIVE RESPONSE SET**

| Opcode | Mnemonic | Description |
|---|---|---|
| 0x1 | PASS | Command successfully processed. |
| 0x2 | FAIL | Command unsuccessfully processed. |
| 0x3 | NACK | Command not known. |

## 9.2 Response Format

As shown in Example 9-1, all programming executive responses have a general format consisting of a two word header and any required data for the command. Table 9-2 lists the fields and their descriptions.

**EXAMPLE 9-1: FORMAT**

| 15 12 | 11 8 | 7 0 |
|---|---|---|
| Opcode | Last_Cmd | QE_Code |
| Length | | |
| D_1 (if applicable) | | |
| ... | | |
| D_N (if applicable) | | |

**TABLE 9-2: FIELDS AND DESCRIPTIONS**

| Field | Description |
|---|---|
| Opcode | Response opcode. |
| Last_Cmd | Programmer command that generated the response. |
| QE_Code | Query code or Error code. |
| Length | Response length in 16-bit words (includes 2 header words.) |
| D_1 | First 16-bit data word (if applicable). |
| D_N | Last 16-bit data word (if applicable). |

### 9.2.1 Opcode FIELD

The Opcode is a 4-bit field in the first word of the response. The Opcode indicates how the command was processed (see Table 9-1). If the command is processed successfully, the response opcode is PASS. If there is an error in processing the command, the response opcode is FAIL, and the QE_Code indicates the reason for the failure. If the command sent to the programming executive is not identified, the programming executive returns a NACK response.

### 9.2.2 Last_Cmd FIELD

The Last_Cmd is a 4-bit field in the first word of the response and indicates the command that the programming executive processed. Since the programming executive can only process one command at a time, this field is technically not required. However, it can be used to verify whether the programming executive correctly received the command that the programmer transmitted.

# 11.0 ICSP™ MODE

## 11.1 ICSP Mode

ICSP mode is a special programming protocol that allows you to read and write to the dsPIC30F programming executive. The ICSP mode is the second (and slower) method used to program the device. This mode also has the ability to read the contents of executive memory to determine whether the programming executive is present. This capability is accomplished by applying control codes and instructions serially to the device using pins PGC and PGD.

In ICSP mode, the system clock is taken from the PGC pin, regardless of the device's oscillator Configuration bits. All instructions are first shifted serially into an internal buffer, then loaded into the Instruction register and executed. No program fetching occurs from internal memory. Instructions are fed in 24 bits at a time. PGD is used to shift data in and PGC is used as both the serial shift clock and the CPU execution clock.

Data is transmitted on the rising edge and latched on the falling edge of PGC. For all data transmissions, the Least Significant bit (LSb) is transmitted first.

**Note 1:** During ICSP operation, the operating frequency of PGC must not exceed 5 MHz.

**2:** Because ICSP is slower, it is recommended that only Enhanced ICSP (E-ICSP) mode be used for device programming, as described in **Section 5.1 "Overview of the Programming Process"**.

## 11.2 ICSP Operation

Upon entry into ICSP mode, the CPU is idle. Execution of the CPU is governed by an internal state machine. A 4-bit control code is clocked in using PGC and PGD, and this control code is used to command the CPU (see Table 11-1).

The SIX control code is used to send instructions to the CPU for execution, while the REGOUT control code is used to read data out of the device via the VISI register. The operation details of ICSP mode are provided in **Section 11.2.1 "SIX Serial Instruction Execution"** and **Section 11.2.2 "REGOUT Serial Instruction Execution"**.

**TABLE 11-1: CPU CONTROL CODES IN ICSP™ MODE**

| 4-bit Control Code | Mnemonic | Description |
|---|---|---|
| 0000b | SIX | Shift in 24-bit instruction and execute. |
| 0001b | REGOUT | Shift out the VISI register. |
| 0010b-1111b | N/A | Reserved. |

### 11.2.1 SIX SERIAL INSTRUCTION EXECUTION

The SIX control code allows execution of dsPIC30F assembly instructions. When the SIX code is received, the CPU is suspended for 24 clock cycles as the instruction is then clocked into the internal buffer. Once the instruction is shifted in, the state machine allows it to be executed over the next four clock cycles. While the received instruction is executed, the state machine simultaneously shifts in the next 4-bit command (see Figure 11-2).

**Note 1:** Coming out of the ICSP entry sequence, the first 4-bit control code is always forced to SIX and a forced NOP instruction is executed by the CPU. Five additional PGC clocks are needed on start-up, thereby resulting in a 9-bit SIX command instead of the normal 4-bit SIX command. After the forced SIX is clocked in, ICSP operation resumes as normal (the next 24 clock cycles load the first instruction word to the CPU). See Figure 11-1 for details.

**2:** TBLRDH, TBLRDL, TBLWTH and TBLWTL instructions must be followed by a NOP instruction.

## 11.4 Flash Memory Programming in ICSP Mode

Programming in ICSP mode is described in **Section 11.4.1 "Programming Operations"** through **Section 11.4.3 "Starting and Stopping a Programming Cycle"**. Step-by-step procedures are described in **Section 11.5 "Erasing Program Memory in Normal-Voltage Systems"** through **Section 11.13 "Reading the Application ID Word"**. All programming operations must use serial execution, as described in **Section 11.2 "ICSP Operation"**.

### 11.4.1 PROGRAMMING OPERATIONS

Flash memory write and erase operations are controlled by the NVMCON register. Programming is performed by setting NVMCON to select the type of erase operation (Table 11-2) or write operation (Table 11-3), writing a key sequence to enable the programming and initiating the programming by setting the WR control bit, NVMCON<15>.

In ICSP mode, all programming operations are externally timed. An external 2 ms delay must be used between setting the WR control bit and clearing the WR control bit to complete the programming operation.

**TABLE 11-2: NVMCON ERASE OPERATIONS**

| NVMCON Value | Erase Operation |
|---|---|
| 0x407F | Erase all code memory, data memory (does not erase UNIT ID). |
| 0x4075 | Erase 1 row (16 words) of data EEPROM. |
| 0x4074 | Erase 1 word of data EEPROM. |
| 0x4072 | Erase all executive memory. |
| 0x4071 | Erase 1 row (32 instruction words) from 1 panel of code memory. |
| 0x406E | Erase Boot Secure and General Segments, then erase FBS, FSS and FGS configuration registers. |
| 0x4066 | Erase all Data EEPROM allocated to Boot Segment. |
| 0x405E | Erase Secure and General Segments, then erase FSS and FGS configuration registers. |
| 0x4056 | Erase all Data EEPROM allocated to Secure Segment. |
| 0x404E | Erase General Segment, then erase FGS configuration register. |
| 0x4046 | Erase all Data EEPROM allocated to General Segment. |

**TABLE 11-3: NVMCON WRITE OPERATIONS**

| NVMCON Value | Write Operation |
|---|---|
| 0x4008 | Write 1 word to configuration memory. |
| 0x4005 | Write 1 row (16 words) to data memory. |
| 0x4004 | Write 1 word to data memory. |
| 0x4001 | Write 1 row (32 instruction words) into 1 panel of program memory. |

### 11.4.2 UNLOCKING NVMCON FOR PROGRAMMING

Writes to the WR bit (NVMCON<15>) are locked to prevent accidental programming from taking place. Writing a key sequence to the NVMKEY register unlocks the WR bit and allows it to be written to. The unlock sequence is performed as follows:

```
MOV     #0x55, W8
MOV     W8, NVMKEY
MOV     #0xAA, W9
MOV     W9, NVMKEY
```

**Note:** Any working register, or working register pair, can be used to write the unlock sequence.

### 11.4.3 STARTING AND STOPPING A PROGRAMMING CYCLE

Once the unlock key sequence has been written to the NVMKEY register, the WR bit (NVMCON<15>) is used to start and stop an erase or write cycle. Setting the WR bit initiates the programming cycle. Clearing the WR bit terminates the programming cycle.

All erase and write cycles must be externally timed. An external delay must be used between setting and clearing the WR bit. Starting and stopping a programming cycle is performed as follows:

```
BSET    NVMCON, #WR
<Wait 2 ms>
BCLR    NVMCON, #WR
```

## 11.5 Erasing Program Memory in Normal-Voltage Systems

The procedure for erasing program memory (all code memory, data memory, executive memory and code-protect bits) consists of setting NVMCON to 0x407F, unlocking NVMCON for erasing and then executing the programming cycle. This method of bulk erasing program memory only works for systems where VDD is between 4.5 volts and 5.5 volts. The method for erasing program memory for systems with a lower VDD (3.0 volts-4.5 volts) is described in **Section 6.1 "Erasing Memory"**.

## 11.7 Writing Configuration Memory

The FOSC, FWDT, FBORPOR and FICD registers are not erasable. It is recommended that all Configuration registers be set to a default value after erasing program memory. The FWDT, FBORPOR and FICD registers can be set to a default all '1's value by programming 0xFFFF to each register. Since these registers contain unimplemented bits that read as '0' the default values shown in Table 11-6 will be read instead of 0xFFFF. The recommended default FOSC value is 0xC100, which selects the FRC clock oscillator setting.

The FGS, FBS and FSS Configuration registers are special since they enable code protection for the device. For security purposes, once any bit in these registers is programmed to '0' (to enable some code protection feature), it can only be set back to '1' by performing a Bulk Erase or Segment Erase as described in **Section 11.5 "Erasing Program Memory in Normal-Voltage Systems"**. Programming these bits from a '0' to '1' is not possible, but they may be programmed from a '1' to a '0' to enable code protection.

Table 11-7 shows the ICSP programming details for clearing the Configuration registers. In Step 1, the Reset vector is exited. In Step 2, the write pointer (W7) is loaded with 0x0000, which is the original destination address (in TBLPAG 0xF8 of program memory). In Step 3, the NVMCON is set to program one Configuration register. In Step 4, the TBLPAG register is initialized, to 0xF8, for writing to the Configuration registers. In Step 5, the value to write to the each Configuration register (0xFFFF) is loaded to W6. In Step 6, the Configuration register data is written to the write latch using the TBLWTL instruction. In Steps 7 and 8, the NVMCON is unlocked for programming and the programming cycle is initiated, as described in **Section 11.4 "Flash Memory Programming in ICSP Mode"**. In Step 9, the internal PC is set to 0x100 as a safety measure to prevent the PC from incrementing into unimplemented memory. Lastly, Steps 3-9 are repeated six times until all seven Configuration registers are cleared.

**TABLE 11-6: DEFAULT CONFIGURATION REGISTER VALUES**

| Address | Register | Default Value |
|---|---|---|
| 0xF80000 | FOSC | 0xC100 |
| 0xF80002 | FWDT | 0x803F |
| 0xF80004 | FBORPOR | 0x87B3 |
| 0xF80006 | FBS | 0x310F |
| 0xF80008 | FSS | 0x330F |
| 0xF8000A | FGS | 0x0007 |
| 0xF8000C | FICD | 0xC003 |

**TABLE 11-7: SERIAL INSTRUCTION EXECUTION FOR WRITING CONFIGURATION REGISTERS**

| Command (Binary) | Data (Hexadecimal) | Description |
|---|---|---|
| **Step 1:** Exit the Reset vector. | | |
| 0000<br>0000<br>0000 | 040100<br>040100<br>000000 | GOTO 0x100<br>GOTO 0x100<br>NOP |
| **Step 2:** Initialize the write pointer (W7) for the TBLWT instruction. | | |
| 0000 | 200007 | MOV #0x0000, W7 |
| **Step 3:** Set the NVMCON to program 1 Configuration register. | | |
| 0000<br>0000 | 24008A<br>883B0A | MOV #0x4008, W10<br>MOV W10, NVMCON |
| **Step 4:** Initialize the TBLPAG register. | | |
| 0000<br>0000 | 200F80<br>880190 | MOV #0xF8, W0<br>MOV W0, TBLPAG |
| **Step 5:** Load the Configuration register data to W6. | | |
| 0000<br>0000 | 2xxxx0<br>000000 | MOV #<CONFIG_VALUE>, W0<br>NOP |

## 11.8 Writing Code Memory

The procedure for writing code memory is similar to the procedure for clearing the Configuration registers, except that 32 instruction words are programmed at a time. To facilitate this operation, working registers W0:W5 are used as temporary holding registers for the data to be programmed.

Table 11-8 shows the ICSP programming details, including the serial pattern with the ICSP command code, which must be transmitted Least Significant bit first using the PGC and PGD pins (see Figure 11-2). In Step 1, the Reset vector is exited. In Step 2, the NVMCON register is initialized for single-panel programming of code memory. In Step 3, the 24-bit starting destination address for programming is loaded into the TBLPAG register and W7 register. The upper byte of the starting destination address is stored to TBLPAG, while the lower 16 bits of the destination address are stored to W7.

To minimize the programming time, the same packed instruction format that the programming executive uses is utilized (Figure 8-2). In Step 4, four packed instruction words are stored to working registers W0:W5 using the `MOV` instruction and the read pointer W6 is initialized. The contents of W0:W5 holding the packed instruction word data is shown in Figure 11-4.

In Step 5, eight `TBLWT` instructions are used to copy the data from W0:W5 to the write latches of code memory. Since code memory is programmed 32 instruction words at a time, Steps 4 and 5 are repeated eight times to load all the write latches (Step 6).

After the write latches are loaded, programming is initiated by writing to the NVMKEY and NVMCON registers in Steps 7 and 8. In Step 9, the internal PC is reset to 0x100. This is a precautionary measure to prevent the PC from incrementing into unimplemented memory when large devices are being programmed. Lastly, in Step 10, Steps 2-9 are repeated until all of code memory is programmed.

**FIGURE 11-5: PACKED INSTRUCTION WORDS IN W0:W5**

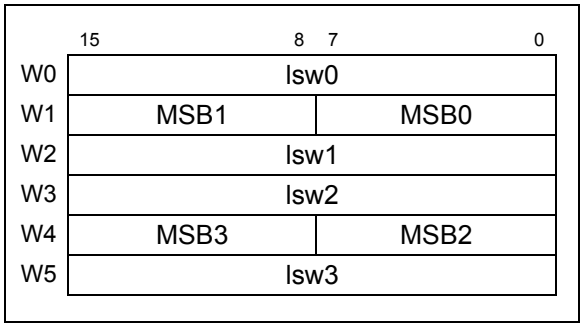



**TABLE 11-8: SERIAL INSTRUCTION EXECUTION FOR WRITING CODE MEMORY**

| Command (Binary) | Data (Hexadecimal) | Description |
|---|---|---|
| **Step 1:** Exit the Reset vector. | | |
| 0000<br>0000<br>0000 | 040100<br>040100<br>000000 | GOTO 0x100<br>GOTO 0x100<br>NOP |
| **Step 2:** Set the NVMCON to program 32 instruction words. | | |
| 0000<br>0000 | 24001A<br>883B0A | MOV #0x4001, W10<br>MOV W10, NVMCON |
| **Step 3:** Initialize the write pointer (W7) for `TBLWT` instruction. | | |
| 0000<br>0000<br>0000 | 200xx0<br>880190<br>2xxxx7 | MOV #<DestinationAddress23:16>, W0<br>MOV W0, TBLPAG<br>MOV #<DestinationAddress15:0>, W7 |
| **Step 4:** Initialize the read pointer (W6) and load W0:W5 with the next 4 instruction words to program. | | |
| 0000<br>0000<br>0000<br>0000<br>0000<br>0000 | 2xxxx0<br>2xxxx1<br>2xxxx2<br>2xxxx3<br>2xxxx4<br>2xxxx5 | MOV #<LSW0>, W0<br>MOV #<MSB1:MSB0>, W1<br>MOV #<LSW1>, W2<br>MOV #<LSW2>, W3<br>MOV #<MSB3:MSB2>, W4<br>MOV #<LSW3>, W5 |

## 11.12 Reading Data Memory

The procedure for reading data memory is similar to that of reading code memory, except that 16-bit data words are read instead of 24-bit words. Since less data is read in each operation, only working registers W0:W3 are used as temporary holding registers for the data to be read.

Table 11-12 shows the ICSP programming details for reading data memory. Note that the TBLPAG register is hard-coded to 0x7F (the upper byte address of all locations of data memory).

**TABLE 11-12: SERIAL INSTRUCTION EXECUTION FOR READING DATA MEMORY**

| Command (Binary) | Data (Hexadecimal) | Description |
|---|---|---|
| **Step 1:** Exit the Reset vector. | | |
| 0000 | 040100 | GOTO 0x100 |
| 0000 | 040100 | GOTO 0x100 |
| 0000 | 000000 | NOP |
| **Step 2:** Initialize TBLPAG and the read pointer (W6) for TBLRD instruction. | | |
| 0000 | 2007F0 | MOV #0x7F, W0 |
| 0000 | 880190 | MOV W0, TBLPAG |
| 0000 | 2xxxx6 | MOV #<SourceAddress15:0>, W6 |
| **Step 3:** Initialize the write pointer (W7) and store the next four locations of code memory to W0:W5. | | |
| 0000 | EB0380 | CLR W7 |
| 0000 | 000000 | NOP |
| 0000 | BA1BB6 | TBLRDL [W6++], [W7++] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0000 | BA1BB6 | TBLRDL [W6++], [W7++] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0000 | BA1BB6 | TBLRDL [W6++], [W7++] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0000 | BA1BB6 | TBLRDL [W6++], [W7++] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **Step 4:** Output W0:W5 using the VISI register and REGOUT command. | | |
| 0000 | 883C20 | MOV W0, VISI |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of VISI register |
| 0000 | 000000 | NOP |
| 0000 | 883C21 | MOV W1, VISI |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of VISI register |
| 0000 | 000000 | NOP |
| 0000 | 883C22 | MOV W2, VISI |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of VISI register |
| 0000 | 000000 | NOP |
| 0000 | 883C23 | MOV W3, VISI |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of VISI register |
| 0000 | 000000 | NOP |
| **Step 5:** Reset device internal PC. | | |
| 0000 | 040100 | GOTO 0x100 |
| 0000 | 000000 | NOP |
| **Step 6:** Repeat steps 3-5 until all desired data memory is read. | | |

# 12.0 PROGRAMMING THE PROGRAMMING EXECUTIVE TO MEMORY

## 12.1 Overview

If it is determined that the programming executive does not reside in executive memory (as described in **Section 4.0 "Confirming the Contents of Executive Memory"**), it must be programmed into executive memory using ICSP and the techniques described in **Section 11.0 "ICSP™ Mode"**.

Storing the programming executive to executive memory is similar to normal programming of code memory. The executive memory must first be erased, and then the programming executive must be programmed 32 words at a time. This control flow is summarized in Table 12-1.

**TABLE 12-1: PROGRAMMING THE PROGRAMMING EXECUTIVE**

| Command (Binary) | Data (Hexadecimal) | Description |
|---|---|---|
| **Step 1:** Exit the Reset vector and erase executive memory. | | |
| 0000 | 040100 | GOTO 0x100 |
| 0000 | 040100 | GOTO 0x100 |
| 0000 | 000000 | NOP |
| **Step 2:** Initialize the NVMCON to erase executive memory. | | |
| 0000 | 24072A | MOV #0x4072, W10 |
| 0000 | 883B0A | MOV W10, NVMCON |
| **Step 3:** Unlock the NVMCON for programming. | | |
| 0000 | 200558 | MOV #0x55, W8 |
| 0000 | 883B38 | MOV W8, NVMKEY |
| 0000 | 200AA9 | MOV #0xAA, W9 |
| 0000 | 883B39 | MOV W9, NVMKEY |
| **Step 4:** Initiate the erase cycle. | | |
| 0000 | A8E761 | BSET NVMCON, #15 |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| — | — | Externally time 'P13a' ms (see **Section 13.0 "AC/DC Characteristics and Timing Requirements"**) |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0000 | A9E761 | BCLR NVMCON, #15 |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **Step 5:** Initialize the TBLPAG and the write pointer (W7). | | |
| 0000 | 200800 | MOV #0x80, W0 |
| 0000 | 880190 | MOV W0, TBLPAG |
| 0000 | EB0380 | CLR W7 |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **Step 6:** Initialize the NVMCON to program 32 instruction words. | | |
| 0000 | 24001A | MOV #0x4001, W10 |
| 0000 | 883B0A | MOV W10, NVMCON |
| **Step 7:** Load W0:W5 with the next 4 words of packed programming executive code and initialize W6 for programming. Programming starts from the base of executive memory (0x800000) using W6 as a read pointer and W7 as a write pointer. | | |
| 0000 | 2<LSW0>0 | MOV #<LSW0>, W0 |
| 0000 | 2<MSB1:MSB0>1 | MOV #<MSB1:MSB0>, W1 |
| 0000 | 2<LSW1>2 | MOV #<LSW1>, W2 |
| 0000 | 2<LSW2>3 | MOV #<LSW2>, W3 |
| 0000 | 2<MSB3:MSB2>4 | MOV #<MSB3:MSB2>, W4 |
| 0000 | 2<LSW3>5 | MOV #<LSW3>, W5 |

**TABLE 12-2: READING EXECUTIVE MEMORY (CONTINUED)**

| Command (Binary) | Data (Hexadecimal) | Description |
|---|---|---|
| **Step 4:** Output W0:W5 using the VISI register and REGOUT command. | | |
| 0000 | 883C20 | MOV W0, VISI |
| 0000 | 000000 | NOP |
| 0001 | — | Clock out contents of VISI register |
| 0000 | 883C21 | MOV W1, VISI |
| 0000 | 000000 | NOP |
| 0001 | — | Clock out contents of VISI register |
| 0000 | 883C22 | MOV W2, VISI |
| 0000 | 000000 | NOP |
| 0001 | — | Clock out contents of VISI register |
| 0000 | 883C23 | MOV W3, VISI |
| 0000 | 000000 | NOP |
| 0001 | — | Clock out contents of VISI register |
| 0000 | 883C24 | MOV W4, VISI |
| 0000 | 000000 | NOP |
| 0001 | — | Clock out contents of VISI register |
| 0000 | 883C25 | MOV W5, VISI |
| 0000 | 000000 | NOP |
| 0001 | — | Clock out contents of VISI register |
| **Step 5:** Reset the device internal PC. | | |
| 0000 | 040100 | GOTO 0x100 |
| 0000 | 000000 | NOP |
| **Step 6:** Repeat Steps 3-5 until all 736 instruction words of executive memory are read. | | |

**TABLE A-1: CHECKSUM COMPUTATION (CONTINUED)**

| Device | Read Code Protection | Checksum Computation | Erased Value | Value with 0xAAAAAA at 0x0 and Last Code Address |
|---|---|---|---|---|
| dsPIC30F5016 | Disabled | CFGB+SUM(0:00AFFF) | 0xFC06 | 0xFA08 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6010 | Disabled | CFGB+SUM(0:017FFF) | 0xC406 | 0xC208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6010A | Disabled | CFGB+SUM(0:017FFF) | 0xC406 | 0xC208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6011 | Disabled | CFGB+SUM(0:015FFF) | 0xF406 | 0xF208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6011A | Disabled | CFGB+SUM(0:015FFF) | 0xF406 | 0xF208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6012 | Disabled | CFGB+SUM(0:017FFF) | 0xC406 | 0xC208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6012A | Disabled | CFGB+SUM(0:017FFF) | 0xC406 | 0xC208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6013 | Disabled | CFGB+SUM(0:015FFF) | 0xF406 | 0xF208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6013A | Disabled | CFGB+SUM(0:015FFF) | 0xF406 | 0xF208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6014 | Disabled | CFGB+SUM(0:017FFF) | 0xC406 | 0xC208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6014A | Disabled | CFGB+SUM(0:017FFF) | 0xC406 | 0xC208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F6015 | Disabled | CFGB+SUM(0:017FFF) | 0xC406 | 0xC208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |

**Item Description:**

**SUM(a:b)** = Byte sum of locations a to b inclusive (all 3 bytes of code memory)

**CFGB** = **Configuration Block (masked)** = Byte sum of ((FOSC&0xC10F) + (FWDT&0x803F) + (FBORPOR&0x87B3) + (FBS&0x310F) + (FSS&0x330F) + (FGS&0x0007) + (FICD&0xC003))

## APPENDIX B: HEX FILE FORMAT

Flash programmers process the standard HEX format used by the Microchip development tools. The format supported is the Intel® HEX 32 Format (INHX32). Please refer to Appendix A in the "*MPASM User's Guide"* (DS33014) for more information about hex file formats.

The basic format of the hex file is:

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

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

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

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

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

The hex file will be produced with the following contents:

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

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