**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 | CANbus, I²C, SPI, UART/USART |
| Peripherals | AC'97, Brown-out Detect/Reset, I²S, LVD, POR, PWM, WDT |
| Number of I/O | 68 |
| Program Memory Size | 144KB (48K x 24) |
| Program Memory Type | FLASH |
| EEPROM Size | 4K x 8 |
| RAM Size | 8K x 8 |
| Voltage - Supply (Vcc/Vdd) | 2.5V ~ 5.5V |
| Data Converters | A/D 16x12b |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 125°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 80-TQFP |
| Supplier Device Package | 80-TQFP (12x12) |
| Purchase URL | https://www.e-xfl.com/product-detail/microchip-technology/dspic30f6014at-20e-pt |

**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. |

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

| Bit Field | Register | Description |
|---|---|---|
| SSS<2:0> | FSS | **Secure Segment Program Memory Code Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>111 = No Secure Segment<br>110 = Standard security; Small-sized Secure Program Flash<br>[Secure Segment starts after BS and ends at 0x001FFF]<br>101 = Standard security; Medium-sized Secure Program Flash<br>[Secure Segment starts after BS and ends at 0x003FFF]<br>100 = Standard security; Large-sized Secure Program Flash<br>[Secure Segment starts after BS and ends at 0x007FFF]<br>011 = No Secure Segment<br>010 = High security; Small-sized Secure Program Flash<br>[Secure Segment starts after BS and ends at 0x001FFF]<br>001 = High security; Medium-sized Secure Program Flash<br>[Secure Segment starts after BS and ends at 0x003FFF]<br>000 = High security; Large-sized Secure Program Flash<br>[Secure Segment starts after BS and ends at 0x007FFF] |
| SWRP | FSS | **Secure Segment Program Memory Write Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>1 = Secure Segment program memory is not write-protected<br>0 = Secure program memory is write-protected |
| GSS<1:0> | FGS | **General Segment Program Memory Code Protection (only present in dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>11 = Code protection is disabled<br>10 = Standard security code protection is enabled<br>0x = High security code protection is enabled |
| GCP | FGS | **General Segment Program Memory Code Protection (present in all devices except dsPIC30F5011/5013/6010A/6011A/6012A/6013A/6014A/6015)**<br>1 = General Segment program memory is not code-protected<br>0 = General Segment program memory is code-protected |
| GWRP | FGS | **General Segment Program Memory Write Protection**<br>1 = General Segment program memory is not write-protected<br>0 = General Segment program memory is write-protected |
| BKBUG | FICD | **Debugger/Emulator Enable**<br>1 = Device will reset into Operational mode<br>0 = Device will reset into Debug/Emulation mode |
| COE | FICD | **Debugger/Emulator Enable**<br>1 = Device will reset into Operational mode<br>0 = Device will reset into Clip-on Emulation mode |
| ICS<1:0> | FICD | **ICD Communication Channel Select**<br>11 = Communicate on PGC/EMUC and PGD/EMUD<br>10 = Communicate on EMUC1 and EMUD1<br>01 = Communicate on EMUC2 and EMUD2<br>00 = Communicate on EMUC3 and EMUD3 |
| RESERVED | FBS, FSS, FGS | **Reserved (read as '1', write as '1')** |
| — | All | **Unimplemented (read as '0', write as '0')** |

### 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.

# 6.0 OTHER PROGRAMMING FEATURES

## 6.1 Erasing Memory

Memory is erased by using an ERASEB, ERASED or ERASEP command, as detailed in **Section 8.5 "Command Descriptions"**. Code memory can be erased by row using ERASEP. Data EEPROM can be erased by row using ERASED. When memory is erased, the affected memory locations are set to '1's.

ERASEB provides several Bulk Erase options. Performing a Chip Erase with the ERASEB command clears all code memory, data EEPROM and code protection registers. Alternatively, ERASEB can be used to selectively erase either all code memory or data EEPROM. Erase options are summarized in Table 6-1.

**TABLE 6-1: ERASE OPTIONS**

| Command | Affected Region |
|---|---|
| ERASEB | Entire chip(1) or all code memory or all data EEPROM, or erase by segment |
| ERASED | Specified rows of data EEPROM |
| ERASEP(2) | Specified rows of code memory |

**Note 1:** The system operation Configuration registers and device ID registers are not erasable.

**2:** ERASEP cannot be used to erase code-protect Configuration bits. These bits must be erased using ERASEB.

## 6.2 Modifying Memory

Instead of bulk-erasing the device before programming, it is possible that you may want to modify only a section of an already programmed device. In this situation, Chip Erase is not a realistic option.

Instead, you can erase selective rows of code memory and data EEPROM using ERASEP and ERASED, respectively. You can then reprogram the modified rows with the PROGP and PROGD command pairs. In these cases, when code memory is programmed, single-panel programming must be specified in the PROGP command.

For modification of Advanced Code Protection bits for a particular segment, the entire chip must first be erased with the ERASEB command. Alternatively, on devices that support Advanced Security, individual segments (code and/or data EEPROM) may be erased, by suitably changing the MS (Memory Select) field in the ERASEB command. The code-protect Configuration bits can then be reprogrammed using the PROGC command.

**Note:** If read or write code protection is enabled for a segment, no modifications can be made to that segment until code protection is disabled. Code protection can only be disabled by performing a Chip Erase or by performing a Segment Erase operation for the required segment.

## 6.3 Reading Memory

The READD command reads the data EEPROM, Configuration bits and device ID of the device. This command only returns 16-bit data and operates on 16-bit registers. READD can be used to return the entire contents of data EEPROM.

The READP command reads the code memory of the device. This command only returns 24-bit data packed as described in **Section 8.3 "Packed Data Format"**. READP can be used to read up to 32K instruction words of code memory.

**Note:** Reading an unimplemented memory location causes the programming executive to reset. All READD and READP commands **must** specify only valid memory locations.

## 6.4 Programming Executive Software Version

At times, it may be necessary to determine the version of programming executive stored in executive memory. The QVER command performs this function. See **Section 8.5.11 "QVER Command"** for more details about this command.

## 6.5 Data EEPROM Information in the Hexadecimal File

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

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

**TABLE 8-1: PROGRAMMING EXECUTIVE COMMAND SET**

| Opcode | Mnemonic | Length (16-bit words) | Time Out | Description |
|---|---|---|---|---|
| 0x0 | SCHECK | 1 | 1 ms | Sanity check. |
| 0x1 | READD | 4 | 1 ms/row | Read N 16-bit words of data EEPROM, Configuration registers or device ID starting from specified address. |
| 0x2 | READP | 4 | 1 ms/row | Read N 24-bit instruction words of code memory starting from specified address. |
| 0x3 | Reserved | N/A | N/A | This command is reserved. It will return a NACK. |
| 0x4 | PROGD(2) | 19 | 5 ms | Program one row of data EEPROM at the specified address, then verify. |
| 0x5 | PROGP(1) | 51 | 5 ms | Program one row of code memory at the specified address, then verify. |
| 0x6 | PROGC | 4 | 5 ms | Write byte or 16-bit word to specified Configuration register. |
| 0x7 | ERASEB | 2 | 5 ms | Bulk Erase (entire code memory or data EEPROM), or erase by segment. |
| 0x8 | ERASED(2) | 3 | 5 ms/row | Erase rows of data EEPROM from specified address. |
| 0x9 | ERASEP(1) | 3 | 5 ms/row | Erase rows of code memory from specified address. |
| 0xA | QBLANK | 3 | 300 ms | Query if the code memory and data EEPROM are blank. |
| 0xB | QVER | 1 | 1 ms | Query the programming executive software version. |

**Note 1:** One row of code memory consists of (32) 24-bit words. Refer to Table 5-2 for device-specific information.
**2:** One row of data EEPROM consists of (16) 16-bit words. Refer to Table 5-3 for device-specific information.

### 9.2.3 QE_Code FIELD

The QE_Code is a byte in the first word of the response. This byte is used to return data for query commands, and error codes for all other commands.

When the programming executive processes one of the two query commands (QBLANK or QVER), the returned opcode is always PASS and the QE_Code holds the query response data. The format of the QE_Code for both queries is shown in Table 9-3.

**TABLE 9-3: QE_Code FOR QUERIES**

| Query | QE_Code |
|---|---|
| QBLANK | 0x0F = Code memory and data EEPROM are NOT blank<br>0xF0 = Code memory and data EEPROM are blank |
| QVER | 0xMN, where programming executive software version = M.N<br>(i.e., 0x32 means software version 3.2) |

When the programming executive processes any command other than a Query, the QE_Code represents an error code. Supported error codes are shown in Table 9-4. If a command is successfully processed, the returned QE_Code is set to 0x0, which indicates that there was no error in the command processing. If the verify of the programming for the PROGD, PROGP or PROGC command fails, the QE_Code is set to 0x1. For all other programming executive errors, the QE_Code is 0x2.

**TABLE 9-4: QE_Code FOR NON-QUERY COMMANDS**

| QE_Code | Description |
|---|---|
| 0x0 | No error |
| 0x1 | Verify failed |
| 0x2 | Other error |

### 9.2.4 RESPONSE LENGTH

The response length indicates the length of the programming executive's response in 16-bit words. This field includes the 2 words of the response header.

With the exception of the response for the READD and READP commands, the length of each response is only 2 words.

The response to the READD command is N + 2 words, where N is the number of words specified in the READD command.

The response to the READP command uses the packed instruction word format described in **Section 8.3 "Packed Data Format"**. When reading an odd number of program memory words (N odd), the response to the READP command is (3 • (N + 1)/2 + 2) words. When reading an even number of program memory words (N even), the response to the READP command is (3 • N/2 + 2) words.

## 10.0 DEVICE ID

The device ID region is 2 x 16 bits and can be read using the `READD` command. This region of memory is read-only and can also be read when code protection is enabled.

Table 10-1 shows the device ID for each device, Table 10-2 shows the device ID registers and Table 10-3 describes the bit field of each register.

**TABLE 10-1: DEVICE IDS**

| Device | DEVID | Silicon Revision | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | A0 | A1 | A2 | A3 | A4 | B0 | B1 | B2 |
| dsPIC30F2010 | 0x0040 | 0x1000 | 0x1001 | 0x1002 | 0x1003 | 0x1004 | — | — | — |
| dsPIC30F2011 | 0x0240 | — | 0x1001 | — | — | — | — | — | — |
| dsPIC30F2012 | 0x0241 | — | 0x1001 | — | — | — | — | — | — |
| dsPIC30F3010 | 0x01C0 | 0x1000 | 0x1001 | 0x1002 | — | — | — | — | — |
| dsPIC30F3011 | 0x01C1 | 0x1000 | 0x1001 | 0x1002 | — | — | — | — | — |
| dsPIC30F3012 | 0x00C1 | — | — | — | — | — | 0x1040 | 0x1041 | — |
| dsPIC30F3013 | 0x00C3 | — | — | — | — | — | 0x1040 | 0x1041 | — |
| dsPIC30F3014 | 0x0160 | — | 0x1001 | 0x1002 | — | — | — | — | — |
| dsPIC30F4011 | 0x0101 | — | 0x1001 | 0x1002 | 0x1003 | 0x1003 | — | — | — |
| dsPIC30F4012 | 0x0100 | — | 0x1001 | 0x1002 | 0x1003 | 0x1003 | — | — | — |
| dsPIC30F4013 | 0x0141 | — | 0x1001 | 0x1002 | — | — | — | — | — |
| dsPIC30F5011 | 0x0080 | — | 0x1001 | 0x1002 | 0x1003 | 0x1003 | — | — | — |
| dsPIC30F5013 | 0x0081 | — | 0x1001 | 0x1002 | 0x1003 | 0x1003 | — | — | — |
| dsPIC30F5015 | 0x0200 | 0x1000 | — | — | — | — | — | — | — |
| dsPIC30F5016 | 0x0201 | 0x1000 | — | — | — | — | — | — | — |
| dsPIC30F6010 | 0x0188 | — | — | — | — | — | — | 0x1040 | 0x1042 |
| dsPIC30F6010A | 0x0281 | — | — | 0x1002 | 0x1003 | 0x1004 | — | — | — |
| dsPIC30F6011 | 0x0192 | — | — | — | 0x1003 | — | — | 0x1040 | 0x1042 |
| dsPIC30F6011A | 0x02C0 | — | — | 0x1002 | — | — | 0x1040 | 0x1041 | — |
| dsPIC30F6012 | 0x0193 | — | — | — | 0x1003 | — | — | 0x1040 | 0x1042 |
| dsPIC30F6012A | 0x02C2 | — | — | 0x1002 | — | — | 0x1040 | 0x1041 | — |
| dsPIC30F6013 | 0x0197 | — | — | — | 0x1003 | — | — | 0x1040 | 0x1042 |
| dsPIC30F6013A | 0x02C1 | — | — | 0x1002 | — | — | 0x1040 | 0x1041 | — |
| dsPIC30F6014 | 0x0198 | — | — | — | 0x1003 | — | — | 0x1040 | 0x1042 |
| dsPIC30F6014A | 0x02C3 | — | — | 0x1002 | — | — | 0x1040 | 0x1041 | — |
| dsPIC30F6015 | 0x0280 | — | — | 0x1002 | 0x1003 | 0x1004 | — | — | — |

**TABLE 10-2: dsPIC30F DEVICE ID REGISTERS**

| Address | Name | Bit | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0xFF0000 | DEVID | DEVID<15:0> | | | | | | | | | | | | | | | |
| 0xFF0002 | DEVREV | PROC<3:0> | | | | REV<5:0> | | | | | | DOT<5:0> | | | | | |

**TABLE 10-3: DEVICE ID BITS DESCRIPTION**

| Bit Field | Register | Description |
|---|---|---|
| DEVID<15:0> | DEVID | Encodes the device ID. |
| PROC<3:0> | DEVREV | Encodes the process of the device (always read as 0x001). |
| REV<5:0> | DEVREV | Encodes the major revision number of the device.<br>000000 = A<br>000001 = B<br>000010 = C |
| DOT<5:0> | DEVREV | Encodes the minor revision number of the device.<br>000000 = 0<br>000001 = 1<br>000010 = 2<br>000011 = 3 |
| Examples:<br>Rev A.1 = 0000 0000 0000 0001<br>Rev A.2 = 0000 0000 0000 0010<br>Rev B.0 = 0000 0000 0100 0000<br>This formula applies to all dsPIC30F devices, with the exception of the following:<br>• dsPIC30F6010<br>• dsPIC30F6011<br>• dsPIC30F6012<br>• dsPIC30F6013<br>• dsPIC30F6014<br>Refer to Table 10-1 for the actual revision IDs. | | |

## 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"**.

**TABLE 11-4: SERIAL INSTRUCTION EXECUTION FOR BULK ERASING PROGRAM MEMORY (ONLY IN NORMAL-VOLTAGE SYSTEMS) (CONTINUED)**

| Command (Binary) | Data (Hexadecimal) | Description |
|---|---|---|
| 0000 | 200558 | MOV #0x55, W8 |
| 0000 | 883B38 | MOV W8, NVMKEY |
| 0000 | 200AA9 | MOV #0xAA, W9 |
| 0000 | 883B39 | MOV W9, NVMKEY |
| **Step 11:** Initiate the erase cycle. | | |
| 0000 | A8E761 | BSET NVMCON, #WR |
| 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, #WR |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |

**Note 1:** Steps 2-8 are only required for the dsPIC30F5011/5013 devices. These steps may be skipped for all other devices in the dsPIC30F family.

## 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 |

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

| Command (Binary) | Data (Hexadecimal) | Description |
|---|---|---|
| **Step 6:** Write the Configuration register data to the write latch and increment the write pointer. | | |
| 0000 | BB1B96 | TBLWTL W6, [W7++] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **Step 7:** 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 8:** Initiate the write cycle. | | |
| 0000 | A8E761 | BSET NVMCON, #WR |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| — | — | Externally time 'P12a' ms (see **Section 13.0 "AC/DC Characteristics and Timing Requirements"**) |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| 0000 | A9E761 | BCLR NVMCON, #WR |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **Step 9:** Reset device internal PC. | | |
| 0000 | 040100 | GOTO 0x100 |
| 0000 | 000000 | NOP |
| **Step 10:** Repeat steps 3-9 until all 7 Configuration registers are cleared. | | |

## 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**

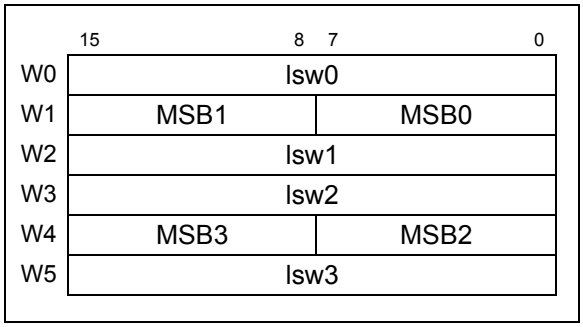


**TABLE 11-8: SERIAL INSTRUCTION EXECUTION FOR WRITING CODE MEMORY**

| 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 |

**TABLE 11-10: SERIAL INSTRUCTION EXECUTION FOR READING CODE 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 | <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 |
| 0000 | 883C24 | MOV W4, VISI |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of VISI register |
| 0000 | 000000 | NOP |
| 0000 | 883C25 | MOV W5, VISI |
| 0000 | 000000 | NOP |
| 0001 | <VISI> | Clock out contents of VISI register |
| 0000 | 000000 | NOP |
| **Step 5:** Reset the device internal PC. | | |
| 0000 | 040100 | GOTO 0x100 |
| 0000 | 000000 | NOP |
| **Step 6:** Repeat steps 3-5 until all desired code memory is read. | | |

## 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. | | |

## 11.13 Reading the Application ID Word

The application ID word is stored at address 0x8005BE in executive code memory. To read this memory location, you must use the SIX control code to move this program memory location to the VISI register. The REGOUT control code must then be used to clock the contents of the VISI register out of the device. The corresponding control and instruction codes that must be serially transmitted to the device to perform this operation are shown in Table 11-13.

Once the programmer has clocked-out the application ID word, it must be inspected. If the application ID has the value 0xBB, the programming executive is resident in memory and the device can be programmed using the mechanism described in **Section 5.0 "Device Programming"**. However, if the application ID has any other value, the programming executive is not resident in memory. It must be loaded to memory before the device can be programmed. The procedure for loading the programming executive to the memory is described in **Section 12.0 "Programming the Programming Executive to Memory"**.

## 11.14 Exiting ICSP Mode

After confirming that the programming executive is resident in memory, or loading the programming executive, ICSP mode is exited by removing power to the device or bringing $\overline{MCLR}$ to $V_{IL}$. Programming can then take place by following the procedure outlined in **Section 5.0 "Device Programming"**.

**TABLE 11-13: SERIAL INSTRUCTION EXECUTION FOR READING THE APPLICATION ID WORD**

| 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 (W0) for TBLRD instruction. | | |
| 0000 | 200800 | MOV #0x80, W0 |
| 0000 | 880190 | MOV W0, TBLPAG |
| 0000 | 205BE0 | MOV #0x5BE, W0 |
| 0000 | 207841 | MOV VISI, W1 |
| 0000 | 000000 | NOP |
| 0000 | BA0890 | TBLRDL [W0], [W1] |
| 0000 | 000000 | NOP |
| 0000 | 000000 | NOP |
| **Step 3:** Output the VISI register using the REGOUT command. | | |
| 0001 | <VISI> | Clock out contents of the VISI register |
| 0000 | 000000 | NOP |

**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 13-1: AC/DC CHARACTERISTICS (CONTINUED)**

| AC/DC CHARACTERISTICS | | | Standard Operating Conditions (unless otherwise stated) Operating Temperature: 25° C is recommended | | | |
|---|---|---|---|---|---|---|
| **Param. No.** | **Sym** | **Characteristic** | **Min** | **Max** | **Units** | **Conditions** |
| P9b | TDLY5 | Delay between PGD ↓by programming executive to PGD released by programming executive | 15 | — | µs | — |
| P10 | TDLY6 | Delay between PGD released by programming executive to first PGC ↑ of response | 5 | — | µs | — |
| P11 | TDLY7 | Delay between clocking out response words | 10 | — | µs | — |
| P12a | TPROG | Row Programming cycle time | 1 | 4 | ms | ICSP mode |
| P12b | TPROG | Row Programming cycle time | 0.8 | 2.6 | ms | Enhanced ICSP mode |
| P13a | TERA | Bulk/Row Erase cycle time | 1 | 4 | ms | ICSP mode |
| P13b | TERA | Bulk/Row Erase cycle time | 0.8 | 2.6 | ms | Enhanced ICSP mode |

# APPENDIX A: DEVICE-SPECIFIC INFORMATION

## A.1 Checksum Computation

The checksum computation is described in **Section 6.8 "Checksum Computation"**. Table A-1 shows how this 16-bit computation can be made for each dsPIC30F device. Computations for read code protection are shown both enabled and disabled. The checksum values assume that the Configuration registers are also erased. However, when code protection is enabled, the value of the FGS register is assumed to be 0x5.

## A.2 dsPIC30F5011 and dsPIC30F5013

### A.2.1 ICSP PROGRAMMING

The dsPIC30F5011 and dsPIC30F5013 processors require that the FBS and FSS registers be programmed with 0x0000 before the device is chip erased. The steps to perform this action are shown in Table 11-4.

### A.2.2 ENHANCED ICSP PROGRAMMING

The dsPIC30F5011 and dsPIC30F5013 processors require that the FBS and FSS registers be programmed with 0x0000 using the `PROGC` command before the `ERASEB` command is used to erase the chip.

**TABLE A-1: CHECKSUM COMPUTATION**

| Device | Read Code Protection | Checksum Computation | Erased Value | Value with 0xAAAAAA at 0x0 and Last Code Address |
|---|---|---|---|---|
| dsPIC30F2010 | Disabled | CFGB+SUM(0:001FFF) | 0xD406 | 0xD208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F2011 | Disabled | CFGB+SUM(0:001FFF) | 0xD406 | 0xD208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F2012 | Disabled | CFGB+SUM(0:001FFF) | 0xD406 | 0xD208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F3010 | Disabled | CFGB+SUM(0:003FFF) | 0xA406 | 0xA208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F3011 | Disabled | CFGB+SUM(0:003FFF) | 0xA406 | 0xA208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F3012 | Disabled | CFGB+SUM(0:003FFF) | 0xA406 | 0xA208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F3013 | Disabled | CFGB+SUM(0:003FFF) | 0xA406 | 0xA208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F3014 | Disabled | CFGB+SUM(0:003FFF) | 0xA406 | 0xA208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F4011 | Disabled | CFGB+SUM(0:007FFF) | 0x4406 | 0x4208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F4012 | Disabled | CFGB+SUM(0:007FFF) | 0x4406 | 0x4208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F4013 | Disabled | CFGB+SUM(0:007FFF) | 0x4406 | 0x4208 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F5011 | Disabled | CFGB+SUM(0:00AFFF) | 0xFC06 | 0xFA08 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F5013 | Disabled | CFGB+SUM(0:00AFFF) | 0xFC06 | 0xFA08 |
| | Enabled | CFGB | 0x0404 | 0x0404 |
| dsPIC30F5015 | Disabled | CFGB+SUM(0:00AFFF) | 0xFC06 | 0xFA08 |
| | 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.