Section 5. Flash Programming

HIGHLIGHTS

This section of the manual contains the following topics:

5.1 INTRODUCTION

This section describes the programming technique for Flash program memory. The PIC24H device family has an internal programmable Flash memory for execution of user code. There are two methods to program this memory:

- Run-Time Self Programming (RTSP)
- In-Circuit Serial Programming (ICSP)

This section describes RTSP programming, which is performed by the user's software.

ICSP is performed using a serial data connection to the device and allows for faster programming time than RTSP. The ICSP protocol is defined in the PIC24H Flash Programming Specification (DS70152), which can be downloaded from the Microchip web site (www.microchip.com).

5.2 TABLE INSTRUCTION OPERATION

The table instructions provide one method of transferring data between the program memory space and the data memory space of PIC24H devices. This section provides a summary of the table instructions used during programming of the Flash program memory. There are four basic table instructions:

- TBLRDL: Table Read Low
- TBLRDH: Table Read High
- TBLWTL: Table Write Low
- TBLWTH: Table Write High

The TBLRDL instruction reads from bits <15:0> of program memory space. The TBLWTL instruction writes to bits <15:0> of program memory space. TBLRDL and TBLWTL can access program memory in Word or Byte mode.

The TBLRDH and TBLWTH instructions reads or writes to bits <23:16> of program memory space. TBLRDH and TBLWTH can access program memory in Word or Byte mode. Since the program memory is only 24-bits wide, the TBLRDH and TBLWTH instructions can address an upper byte of program memory that does not exist. This byte is called the "phantom byte." Any read of the phantom byte return 0x00 and a write to the phantom byte has no effect.

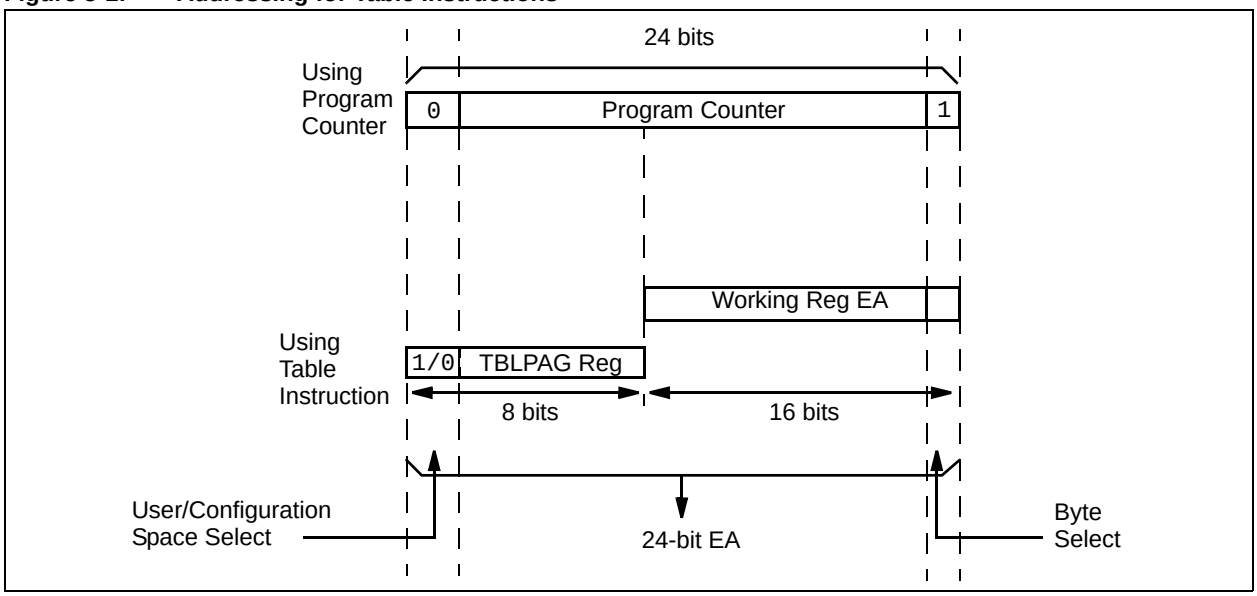
The 24-bit program memory can be regarded as two side-by-side 16-bit spaces, with each space sharing the same address range. Therefore, the TBLRDL and TBLWTL instructions access the "low" program memory space (PM<15:0>). The TBLRDH and TBLWTH instructions access the "high" program memory space (PM<31:16>). Any reads or writes to PM<31:24> will access the phantom (unimplemented) byte. When any of the table instructions are used in Byte mode, the Least Significant bit (LSb) of the table address is used as the byte select bit. The LSb determines which byte in the high or low program memory space is accessed.

Figure 5-1 shows how the program memory is addressed using the table instructions. A 24-bit program memory address is formed using bits <7:0> of the TBLPAG register and the effective address (EA) from a W register, specified in the table instruction. Figure 5-1 shows the 24-bit program counter for reference. The upper 23 bits of the EA are used to select the program memory location.

For the Byte mode table instructions, the LSb of the W register EA selects which byte of the 16-bit program memory word is addressed. A '1' selects bits <15:8>, a '0' selects bits <7:0>. The LSb of the W register EA is ignored for a table instruction in Word mode.

In addition to the program memory address, the table instruction also specifies a W register (or a W register pointer to a memory location) that is the source of the program memory data to be written, or the destination for a program memory read. For a table write operation in Byte mode, bits <15:8> of the source working register are ignored.

Figure 5-1: Addressing for Table Instructions



5.2.1 Using Table Read Instructions

Table reads require two steps. First, an address pointer is set up using the TBLPAG register and one of the W registers. Then, the program memory contents at the address location may be read.

5.2.1.1 READ WORD MODE

The code in Example 5-1 shows how to read a word of program memory using the table instructions in Word mode:

Example 5-1: Read Word Mode

```
; Set up the address pointer to program space
    MOV         #tblpage(PROG_ADDR),W0        ; get table page value
    MOV         W0,TBLPAG                     ; load TBLPAG register
    MOV         #tbloffset(PROG_ADDR),W0      ; load address LS word
; Read the program memory location
    TBLRDH      [W0],W3                       ; Read high byte to W3
    TBLRDL      [W0],W4                       ; Read low word to W4
```

5.2.1.2 READ BYTE MODE

The code in Example 5-2 shows the post-increment operator on the read of the low byte causes the address in the working register to increment by one. This sets EA<0> to a '1' for access to the middle byte in the third write instruction. The last post-increment sets W0 back to an even address, pointing to the next program memory location.

Note:	The `tblpage()` and `tbloffset()` directives are provided by the Microchip assembler for the PIC24H. These directives select the appropriate TBLPAG and W register values for the table instruction from a program memory address value. For further details, refer to the MPLAB ASM 30, MPLAB LINK30 and Utilities User's Guide (DS51317).

Example 5-2: Read Byte Mode

```
; Set up the address pointer to program space
    MOV         #tblpage(PROG_ADDR),W0        ; get table page value
    MOV         W0,TBLPAG                     ; load TBLPAG register
    MOV         #tbloffset(PROG_ADDR),W0      ; load address LS word
; Read the program memory location
    TBLRDH.B    [W0],W3                       ; Read high byte to W3
    TBLRDL.B    [W0++],W4                     ; Read low byte to W4
    TBLRDL.B    [W0++],W5                     ; Read middle byte to W5
```

5.2.1.3 TABLE WRITE HOLDING LATCHES

Table write instructions do not write directly to the nonvolatile program. Instead, the table write instructions load holding latches that store the write data. The holding latches are not memory mapped and can only be accessed using table write instructions. When all of the holding latches have been loaded, the actual memory programming operation is started by executing a special sequence of instructions.

The PIC24H Flash program memory is segmented into pages (512 instruction words) and rows (64 instruction words).

The PIC24H family of devices support 64 holding registers to program one row of memory at a time. The memory logic automatically decides which set of write latches to load based on the address value in the table write instruction.

Refer to the specific device data sheet for further details.

5.2.1.4 WRITE WORD MODE

The code sequence in Example 5-3 can be used to write a single program memory latch location in Word mode:

Example 5-3: Writing a Single Program Memory Latch Location in Word Mode

```
; Setup the address pointer to program space
    MOV        #tblpage(PROG_ADDR),W0    ; get table page value
    MOV        W0,TBLPAG                 ; load TBLPAG register
    MOV        #tbloffset(PROG_ADDR),W0  ; load address LS word
; Load write data into W registers
    MOV        #PROG_LOW_WORD,W2
    MOV        #PROG_HI_BYTE,W3
; Perform the table writes to load the latch
    TBLWTL     W2,[W0]
    TBLWTH     W3,[W0++]
```

In Example 5-3, the contents of the upper byte of W3 does not matter because this data is written to the phantom byte location. W0 is post-incremented by 2, after the second `TBLWTH` instruction, to prepare for the write to the next program memory location.

5.2.1.5 WRITE BYTE MODE

The code sequence in Example 5-4 can be used to write a single program memory latch location in Byte mode:

Example 5-4: Writing a Single Program Memory Latch Location in Byte Mode

```
; Setup the address pointer to program space
    MOV        #tblpage(PROG_ADDR),W0    ; get table page value
    MOV        W0,TBLPAG                 ; load TBLPAG register
    MOV        #tbloffset(PROG_ADDR),W0  ; load address LS word
; Load data into working registers
    MOV        #LOW_BYTE,W2
    MOV        #MID_BYTE,W3
    MOV        #HIGH_BYTE,W4
; Write data to the latch
    TBLWTH.B   W4,[W0]                   ; write high byte
    TBLWTL.B   W2,[W0++]                 ; write low byte
    TBLWTL.B   W3,[W0++]                 ; write middle byte
```

In Example 5-4, the post-increment operator on the write to the low byte causes the address in W0 to increment by one. This sets EA<0> = `1` for access to the middle byte in the third write instruction. The last post-increment sets W0 back to an even address pointing to the next program memory location.

5.3 CONTROL REGISTERS

Two SFRs are used to program Flash memory erase and write operations: NVMCON and NVMKEY.

The Flash Memory Control (NVMCON) register controls various Flash programming and erase operations, such as which blocks are to be erased, which memory type is to be programmed and the start of the programming cycle.

The Nonvolatile Memory Key (NVMKEY) register is a write-only register that is used to protect the NVMCON register from accidental writes. To start a programming or erase sequence, the user must consecutively write 0x55 and 0xAA to the NVMKEY register.

5.3.1 NVMCON Register

The NVMCON register is the primary control register for Flash and program/erase operations. This register selects whether an erase or program operation is performed, and can start the program or erase cycle.

Register 5-1 shows the NVMCON register. The lower byte of NVMCON configures the type of NVM operation to perform.

5.3.2 NVMKEY Register

The NVMKEY register (Register 5-2) is a write-only register used to prevent accidental writes or erasures of Flash memory. Use the following steps in the exact order to start a programming or erase sequence:

1. Write 0x55 to NVMKEY.
2. Write 0xAA to NVMKEY.
3. Execute two NOP instructions.

After this sequence, a write is allowed to the NVMCON register for one instruction cycle. In most cases, the user application needs to set the WR bit in the NVMCON register to start the program or erase cycle. Interrupts should be disabled during the unlock sequence. Example 5-5 shows how the unlock sequence is performed.

Example 5-5: NVMKEY Unlock Sequence

```
;   PUSH    SR              ; Disable interrupts, if enabled
    MOV     #0x00E0,W0
    IOR     SR

    MOV     #0x55,W0
    MOV     W0, NVMKEY
    MOV     #0xAA,W1
    MOV     W1, NVMKEY      ; NOP not required
    BSET    NVMCON,#WR      ; Start the program/erase cycle
    NOP
    NOP
    POP     SR              ; Re-enable interrupts
```

For further programming examples, refer to **Section 5.4.2 “Flash Programming Operations”**.

Register 5-1: NVMCON: Flash Memory Control Register

R/SO-0(1)	R/W-0(1)	R/W-0(1)	U-0	U-0	U-0	U-0	U-0
WR	WREN	WRERR	—	—	—	—	—
bit 15							bit 8

U-0	R/W-0(1)	U-0	U-0	R/W-0(1)	R/W-0(1)	R/W-0(1)	R/W-0(1)
—	ERASE	—	—	NVMOP<3:0>(2)			
bit 7							bit 0

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

bit 15 **WR:** Write Control bit
- 1 = Initiates a Flash memory program or erase operation. The operation is self-timed and the bit is cleared by hardware once operation is complete
- 0 = Program or erase operation is complete and inactive

bit 14 **WREN:** Write Enable bit
- 1 = Enable Flash program/erase operations
- 0 = Inhibit Flash program/erase operations

bit 13 **WRERR:** Write Sequence Error Flag bit
- 1 = An improper program or erase sequence attempt or termination has occurred (bit is set) automatically on any set attempt of the WR bit
- 0 = The program or erase operation completed normally

bit 12-7 **Unimplemented:** Read as '0'

bit 6 **ERASE:** Erase/Program Enable bit
- 1 = Perform the erase operation specified by NVMOP<3:0> on the next WR command
- 0 = Perform the program operation specified by NVMOP<3:0> on the next WR command

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **NVMOP<3:0>:** NVM Operation Select bits(2)

If ERASE = 1,
- 1111 = Memory bulk erase operation
- 1110 = Reserved
- 1101 = Erase General Segment and FGS Configuration register
- 1100 = Erase Secure Segment and FSS Configuration register
- 1011 = Reserved
- 0011 = No operation
- 0010 = Memory page erase operation
- 0001 = No operation
- 0000 = Erase a single Configuration register byte

If ERASE = 0,
- 1111 = No operation
- 1110 = Reserved
- 1101 = No operation
- 1100 = No operation
- 1011 = Reserved
- 0011 = Memory word program operation
- 0010 = No operation
- 0001 = Memory row program operation
- 0000 = Program a single Configuration register byte

Note 1: These bits can only be reset on POR.

2: All other combinations of NVMOP<3:0> are unimplemented.

Register 5-2: NVMKEY: Nonvolatile Memory Key Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
NVMKEY<7:0>							
bit 7							bit 0

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

bit 15-8 **Unimplemented:** Read as ‘0’

bit 7-0 **NVMKEY<7:0>:** Key Register (write-only) bits

5.4 RUN-TIME SELF-PROGRAMMING (RTSP)

RTSP allows the user application to modify Flash program memory contents. RTSP is accomplished using TBLRD (table read) and TBLWT (table write) instructions, and the NVM Control registers. With RTSP, the user application can erase program memory, eight rows (64x8 = 512 instructions) at a time, and can write program memory data, single row (64 instructions) at a time.

5.4.1 RTSP Operation

The PIC24H Flash program memory array is organized into rows of 64 instructions or 192 bytes. RTSP allows the user application to erase blocks of eight rows (512 instructions) at a time, and to program 64 instructions at a time. The 8-row erase blocks and single-row write blocks are edge-aligned, from the beginning of program memory, on boundaries of 1536 bytes and 192 bytes, respectively.

The program memory implements holding buffers that can contain 64 instructions of programming data. Prior to the actual programming operation, the write data must be loaded into the buffers in sequential order. The instruction words loaded must always be from a group of 64 boundaries.

The basic sequence for RTSP programming is to set up a Table Pointer, and then do a series of TBLWT instructions to load the buffers. Programming is performed by setting the control bits in the NVMCON register. A total of 64 TBLWTL and TBLWTH instructions are required to load the instructions.

All of the table write operations are single-word writes (two instruction cycles), because only the buffers are written. A programming cycle is required for programming each row.

5.4.2 Flash Programming Operations

A program or erase operation is necessary for programming or erasing the internal Flash program memory in RTSP mode. The program or erase operation is automatically timed by the device (refer to the device data sheet for timing information). Setting the WR bit (NVMCON<15>) starts the operation and the WR bit is automatically cleared when the operation is finished.

The CPU stalls (waits) until the programming operation is finished. The CPU will not execute any instructions or respond to interrupts during this time. If any interrupts do occur during the programming cycle, they remain pending until the cycle completes.

5.4.2.1 FLASH ROW PROGRAMMING ALGORITHM

The user application can program one row of Flash program memory (64 instruction words). To do this, it is necessary to erase a page (512 instruction words) containing the desired row.

The general process is as follows:

1. Read one page (512 instruction words) of Flash program memory and store into data RAM as a data “image”. The RAM image must be read from an even 512-word program memory address boundary.
2. Update the RAM data image with the new program memory data.
3. Erase the Flash program memory page.
   a) Set up the NVMCON register to erase one page of Flash program memory.
   b) Select the page for erase operation by performing a dummy table write to any location in the page.
   c) Disable interrupts.
   d) Write the key sequence to the NVMKEY register to enable the erase.
   e) Set the WR bit. This starts the erase cycle.
   f) The CPU will stall for the duration of the erase cycle.
   g) The WR bit is cleared when erase cycle ends.
   h) Re-enable interrupts.
4. Load the row (64 instruction words) of instructions words from RAM into the write latches using a table write operation.

5. Program the row (64 instruction words) in Flash program memory.
   a) Set up the NVMCON register to program one row of Flash program memory.
   b) Disable interrupts.
   c) Write the key sequence to the NVMKEY register to enable the program cycle.
   d) Set the WR bit. This starts the program cycle.
   e) The CPU will stall for the duration of the program cycle.
   f) The WR bit is cleared by the hardware when program cycle ends.
   g) Re-enable interrupts.
6. Repeat Steps 4 through 6 to program all eight rows in the program memory page.
7. Repeat Steps 1 through 7, as needed, to program the desired amount of Flash program memory.

Note 1: The user should remember that the minimum amount of program memory that can be erased using RTSP is 512 instruction word locations. Therefore, it is important that an image of these locations be stored in general purpose RAM before an erase cycle is initiated.

2: A row or word in Flash Program memory should not be programmed more than twice before being erased.

5.4.2.2 ERASING ONE PAGE OF FLASH

The code sequence in Example 5-6 can be used to erase a page (512 instructions) of program memory. The NVMCON register is configured to erase one page of program memory. A dummy table write to any location in the page selects the address of the row to be erased. The program memory must be erased at an “even” 512 instruction word address boundary. Therefore, the 10 Least Significant bits of table write program memory address have no effect when a page is erased.

The erase operation is initiated by writing a special unlock, or key sequence to the NVMKEY register before setting the WR control bit (NVMCON<15>). The unlock sequence needs to be executed in the exact order shown without interruption. Therefore, interrupts should be disabled prior to writing the sequence.

Two NOP instructions should be inserted in the code at the point where the CPU resumes operation. Finally, interrupts can be enabled (if required).

Example 5-6: Erasing a Page of Flash Program Memory

```
; Perform dummy table write to the Page to be erased.
       MOV     #tblpage(PROG_ADDR),W0
       MOV     W0,TBLPAG
       MOV     #tbloffset(PROG_ADDR),W0
       TBLWTL  w0,[w0]
; Setup NVMCON to erase one row of Flash
       MOV     #0x4042,W0
       MOV     W0,NVMCON
; Disable interrupts while the KEY sequence is written
       PUSH    SR
       MOV     #0x00E0,W0
       IOR     SR
; Write the KEY Sequence
       MOV     #0x55,W0
       MOV     W0,NVMKEY
       MOV     #0xAA,W0
       MOV     W0,NVMKEY
; Start the erase operation
       BSET    NVMCON,#WR
; Insert two NOPs after the erase cycle (required)
       NOP
       NOP
;Re-enable interrupts, if needed
       POP     SR
```


5.4.2.3 LOADING WRITE LATCHES

The write latches are used as a storage mechanism between the user application table writes and the actual row programming sequence. Example 5-7 shows the sequence of instructions that can be used to load 64 write latches (64 instruction words). 64 TBLWTL and 64 TBLWTH instructions are needed to load the write latches for programming a row of Flash memory.

The row of 64 instruction words does not necessarily have to be written in sequential order. The 7 LSbs of the table write address determine which of the latches are written. However, all 64 instruction words should be written for each programming cycle to overwrite old data.

Table write program memory address are automatically captured to select the address of the row to be programmed. The program memory must be programmed at an “even” 64 instruction word address boundary. In effect, the 7 LSbs of table write operation selects the write latches to program the instruction word within the row, and they have no effect when a row is programmed.

Note: The code shown in Example 5-7 is the `Load_Write_Latch` code referred to in subsequent examples.

Example 5-7: Loading Write Latches

```
; Set up a pointer to the first program memory location to be written.
       MOV     #tblpage(PROG_ADDR),W0
       MOV     W0,TBLPAG
       MOV     #tbloffset(PROG_ADDR),W0

; Perform the TBLWT instructions to write the latches
; W0 is incremented in the TBLWTH instruction to point to the
; next instruction location.
       MOV     #64,W3
       MOV     #ram_image,W1
loop:
       TBLWTL  [W1++],[W0]
       TBLWTH  [W1++],[W0++]
       DEC     W3,W3
       BRA     NZ,loop
```

5.4.2.4 SINGLE ROW PROGRAMMING EXAMPLE

The NVMCON register is configured to program one row of Flash memory. The program operation is initiated by writing a special unlock, or key sequence to the NVMKEY register before setting the WR control bit (NVMCON<15>). Example 5-8 shows the unlock sequence that needs to be executed in the exact order without interruption. Therefore, interrupts should be disabled prior to writing the sequence.

Two NOP instructions should be inserted in the code at the point where the CPU resumes operation. Finally, interrupts can be enabled (if required).

Example 5-8: Single Row Programming

```
; Setup NVMCON to write 1 row of program memory
       MOV     #0x4001,W0
       MOV     W0,NVMCON
; Load the 32 program memory write latches
       CALL    Load_Write_Latch(1)
; Disable interrupts, if enabled
       PUSH    SR
       MOV     #0x00E0,W0
       IOR     SR
; Write the KEY sequence
       MOV     #0x55,W0
       MOV     W0,NVMKEY
       MOV     #0xAA,W0
       MOV     W0,NVMKEY
; Start the programming sequence
       BSET    NVMCON,#WR
; Insert two NOPs after programming
       NOP
       NOP
; Re-enable interrupts, if required
       POP     SR
```

Note: For additional information, refer to **Section 5.4.2.3 "Loading Write Latches"**.

5.4.2.5 PROGRAMMING ONE WORD OF FLASH MEMORY

Assuming the user application has erased the Flash location to be programmed, use table write instructions to write an instruction word (24-bit) into the write latch. The 7 LSbs of table write operation selects the write latches to program the instruction word within the row.

The TBLPAG register is loaded with the 8 Most Significant Bytes of the Flash address. The 16 Least Significant Bytes (LSB) of the Flash address are automatically captured internally when the table write is executed to program the word during program operation. The LSB of the NVMADR register has no effect on the programming operation.

The NVMCON register is configured to program one word of Flash memory. The program operation is initiated by writing a special unlock, or key sequence to the NVMKEY register before setting the WR control bit (NVMCON<15>). The unlock sequence needs to be executed in the exact order shown in Example 5-9 without interruption. Therefore, interrupts should be disabled prior to writing the sequence.

Two NOP instructions should be inserted in the code at the point where the CPU resume operation. Finally, interrupts can be enabled (if required).

Example 5-9: Programming One Word of Flash Memory

```
; Setup a pointer to data Flash
       MOV     #tblpage(PROG_ADDR),W0
       MOV     W0,TBLPAG
       MOV     #tbloffset(PROG_ADDR),W0
; Perform the TBLWT instructions to write the latches
       MOV     #LOW_WORD_N,W2
       MOV     #HIGH_BYTE_N,W3
       TBLWTL  W2,[W0]
       TBLWTH  W3,[W0++]
; Setup NVMCON for programming one word to data Flash
       MOV     #0x4003,W0
       MOV     W0,NVMCON
; Disable interrupts while the KEY sequence is written
       PUSH    SR
       MOV     #0x00E0,W0
       IOR     SR
; Write the key sequence
       MOV     #0x55,W0
       MOV     W0,NVMKEY
       MOV     #0xAA,W0
       MOV     W0,NVMKEY
; Start the write cycle
       BSET    NVMCON,#WR
;Re-enable interrupts, if needed
       POP     SR
```

5.4.3 Writing to Device Configuration Registers

Run-Time Self-Programming (RTSP) can be used to write to the Device Configuration registers. RTSP allows each Configuration register, to be individually rewritten without first performing an erase cycle. Caution must be exercised when writing the Configuration registers since they control critical device operating parameters, such as the system clock source, PLL multiplication ratio and WDT enable.

The procedure for programming a Device Configuration register is similar to the procedure for Flash program memory, except that only `TBLWTL` instructions are required. This is because the upper eight bits are unused in each Device Configuration register. Furthermore, bit 23 of the table write address must be set to access the Configuration registers. For a full description of the Device Configuration registers, refer to **Section 25. "Device Configuration"** and the device data sheet.

5.4.3.1 CONFIGURATION REGISTER WRITE ALGORITHM

1. Write the new configuration value to the table write latch using a `TBLWTL` instruction.
2. Configure NVMCON for a Configuration register write (NVMCON = 0x4000).
3. Disable interrupts, if enabled.
4. Write the key sequence to NVMKEY.
5. Start the write sequence by setting WR (NVMCON<15>).
6. CPU execution resumes when the write is finished.
7. Re-enable interrupts, if needed.

Example 5-10 shows the code sequence that can be used to modify a Device Configuration register.

Example 5-10: Configuration Register Write Code Example

```
; Set up a pointer to the location to be written.
       MOV     #tblpage(CONFIG_ADDR),W0
       MOV     W0,TBLPAG
       MOV     #tbloffset(CONFIG_ADDR),W0
; Get the new data to write to the configuration register
       MOV     #ConfigValue,W1
; Perform the table write to load the write latch
       TBLWTL  W1,[W0]
; Configure NVMCON for a configuration register write
       MOV     #0x4000,W0
       MOV     W0,NVMCON
; Disable interrupts, if enabled
       PUSH    SR
       MOV     #0x00E0,W0
       IOR     SR
; Write the KEY sequence
       MOV     #0x55,W0
       MOV     W0,NVMKEY
       MOV     #0xAA,W0
       MOV     W0,NVMKEY
; Start the programming sequence
       BSET    NVMCON,#WR
; Insert two NOPs after programming
       NOP
       NOP
; Re-enable interrupts, if required
       POP     SR
```

5.5 DESIGN TIPS

Question 1: ***I cannot get the device to program or erase properly. My code appears to be correct. What could be the cause?***

Answer: Interrupts should be disabled when a program or erase cycle is initiated to ensure that the key sequence executes without interruption. Interrupts can be disabled by raising the current CPU priority to level 7.

The code examples in this chapter disable interrupts by saving the current SR register value on the stack, then ORing the value 0x00E0 with SR to force IPL<2:0> = 111. If no priority level 7 interrupts are enabled, the `DISI` instruction provides another method to temporarily disable interrupts, while the key sequence is executed.

5.6 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC24H, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Flash Programming module are:

Title	Application Note #
No related application notes at this time.	

Note: For additional application notes and code examples of the PIC24H device family, visit the Microchip website (www.microchip.com).

5.7 REVISION HISTORY

Revision A (February 2007)

This is the initial released revision of this document.

Revision B (May 2007)

Minor updates were made to this document.

Revision C (June 2007)

Minor updates were made to this document.