



---

## Dual Partition Flash Program Memory

---

### HIGHLIGHTS

This section of the manual contains the following major topics:

1.0	Introduction .....	2
2.0	Program Memory Architecture .....	2
3.0	Program Memory Partition Flash Operation .....	7
4.0	Flash Memory Programming.....	13
5.0	Program Space Visibility and Extended Data Space (PSV and EDS).....	25
6.0	Register Map.....	26
7.0	Related Application Notes.....	27
8.0	Revision History .....	28

## 1.0 INTRODUCTION

All PIC24 and dsPIC33 devices have an internal programmable Flash array for the execution of user code. The high-endurance Flash array provides great flexibility in code development and storage, combining a long retention life with a high number of read/write cycles.

This version of Flash program memory adds these new features:

- Dual Partition Flash operation, allowing the support of robust bootloader systems and fail-safe storage of application code, with options designed to enhance code security
- Live Update operation, allowing the inactive Code Segment to be modified or completely erased while the main application continues to execute
- Direct Run-Time Programming of the Flash array from the data RAM space, with optional compression of the data RAM image

## 2.0 PROGRAM MEMORY ARCHITECTURE

PIC24 and dsPIC33 devices address a 4M x 24-bit program memory address space, as shown in [Figure 2-1](#). The program memory map is equally divided into the user program space (000000h to 7FFFFFFh) and the configuration (or test) memory space (800000h to FFFFFFFh).

The user program space contains the Reset vector, Interrupt Vector Tables (IVTs) and program memory. There are three methods for accessing program space.

1. The 23-bit Program Counter (PC).
2. Table Read (TBLRD) and Table Write (TBLWT) instructions.
3. By mapping any 32-Kbyte segment of program memory into the data memory address space.

Implemented program memory can be further divided into the vector area, which includes the Reset and interrupt vectors, and the code area, which also includes the Flash configuration data. Accessing unimplemented areas of the user program space (i.e., above the upper implemented boundary of program memory) will cause an address error trap.

### 2.1 Vector Area

The vector area starts at the beginning of program memory space, at 000000h. It contains the master Reset vector, the hardware trap vectors and the Interrupt Vector Table (IVT) for all implemented hardware interrupts.

Because of architectural differences and the size of the IVT, the vector area occupies a different amount of memory in different device families. For PIC24 devices, the vector area extends to 0000FEh. For dsPIC33 devices, the vector area extends to 0001FEh. [Figure 2-2](#) shows the difference between the IVTs for different devices. Regardless of device family, hardware interrupt vectors always start at 000014h with Interrupt Vector 0.

The vector area roughly corresponds to the Vector Segment (VS) in CodeGuard™ security implementations. Depending on the security configuration, the vector area may be treated as part of the Boot Segment (BS) or the General Segment (GS).

#### 2.1.1 ALTERNATE VECTOR INTERRUPT TABLE

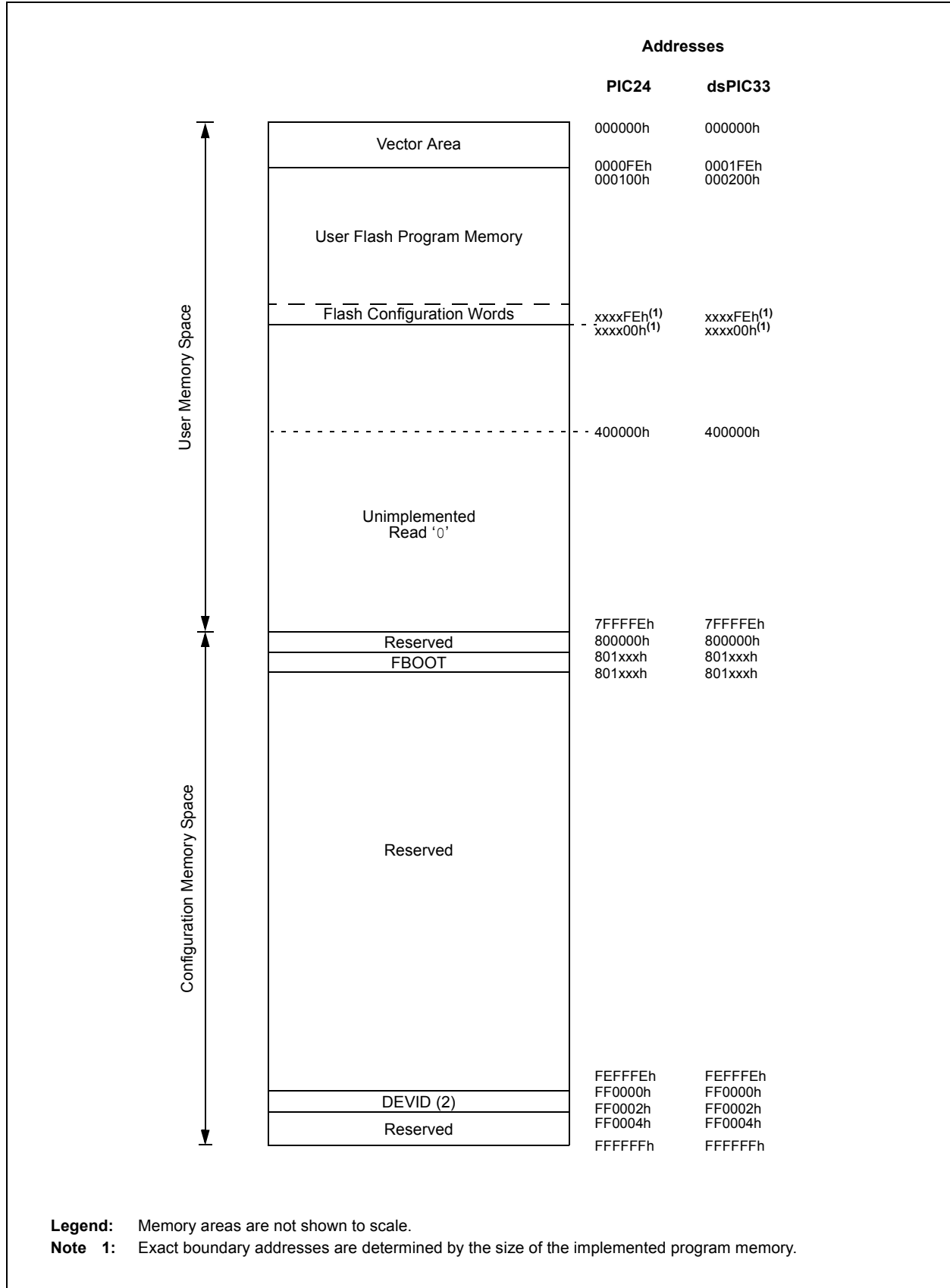
All dsPIC33 and PIC24 devices provide for the implementation of an Alternate IVT (AIVT), which can be used in high-security code applications and for alternate exception handling. Unlike earlier devices in these families, the AIVT is not permanently allocated in program memory at a fixed address range. Instead, AIVT is only present when:

- CodeGuard security is configured for a Boot Segment with a size of at least two pages (set by the FBSLIM Configuration register), and
- AIVT is enabled by programming the AIVTDIS Configuration bit to '0'.

When the AIVT is enabled, it is located at an address range starting at the beginning of the last page of the BS; each vector is located at a fixed offset from the page boundary. The total size and content (i.e., vector order) of the AIVT mirrors those of the IVT.

# Dual Partition Flash Program Memory

**Figure 2-1: Default Program Space Memory Map for dsPIC33 and PIC24 Devices**



# dsPIC33/PIC24 Family Reference Manual

Figure 2-2: Vector Area Detail

	PIC24 Devices	dsPIC33 Devices
Interrupt Vector Table	Reset – GOTO Instruction 000000	Reset – GOTO Instruction 000000
	Reset – GOTO Address 000002	Reset – GOTO Address 000002
	Oscillator Fail Trap Vector 000004	Oscillator Fail Trap Vector 000004
	Address Error Trap Vector	Address Error Trap Vector
	Stack Error Trap Vector	Hard Trap Vector
	Math Error Trap Vector	Stack Error Trap Vector
	...	Math Error Trap Vector
	Interrupt Vector 0 000014	...
	Interrupt Vector 1 000016	Interrupt Vector 0 000014
	...	Interrupt Vector 1 000016
	Interrupt Vector 52 00007C	...
	Interrupt Vector 53 00007E	Interrupt Vector 116 0000FC
	Interrupt Vector 54 000080	Interrupt Vector 117 0000FE
	...	Interrupt Vector 118 000100
	Interrupt Vector 116 0000FC	...
	Interrupt Vector 117 0000FE	Interrupt Vector 244 0001FC
		Interrupt Vector 245 0001FE
Alternate Interrupt Vector Table (optional)	Reserved BOA + 00	Reserved BOA + 00
	Reserved BOA + 02	Reserved BOA + 02
	Reserved BOA + 04	Oscillator Fail Trap Vector BOA + 04
	Oscillator Fail Trap Vector	Address Error Trap Vector
	Address Error Trap Vector	Hard Trap Vector
	Stack Error Trap Vector	Stack Error Trap Vector
	Math Error Trap Vector	Math Error Trap Vector
	...	...
	Interrupt Vector 0 BOA + 14	Interrupt Vector 0 BOA + 14
	Interrupt Vector 1 BOA + 16	Interrupt Vector 1 BOA + 16
	...	...
	Interrupt Vector 52 BOA + 7C	Interrupt Vector 116 BOA + 0FC
	Interrupt Vector 53 BOA + 7E	Interrupt Vector 117 BOA + 0FE
	Interrupt Vector 54 BOA + 80	Interrupt Vector 118 BOA + 100
	...	...
	Interrupt Vector 116 BOA + FC	Interrupt Vector 244 BOA + 1FC
	Interrupt Vector 117 BOA + FE	Interrupt Vector 245 BOA + 1FE

**Legend:** BOA = Base Offset Address, the starting address of the last page of the Boot Segment. Addresses are shown in hexadecimal.

**Note:** Vector area organization shown is the default organization for the given architectures. Specific devices may differ. Refer to the device data sheet for device-specific details.

## 2.2 Code Area

The code area is the area of user program memory that contains the user's application code. It extends from the end of the vector area to the beginning of the Flash Configuration Words. If a Boot Segment is implemented, it starts at the end of the vector area and extends for a predetermined range. The part of the code area that is not in the Boot Segment corresponds to the General Segment (GS) in CodeGuard security systems.

With the exception of the Flash Configuration Words at the end of implemented memory, as described below, the entire area is available for application code.

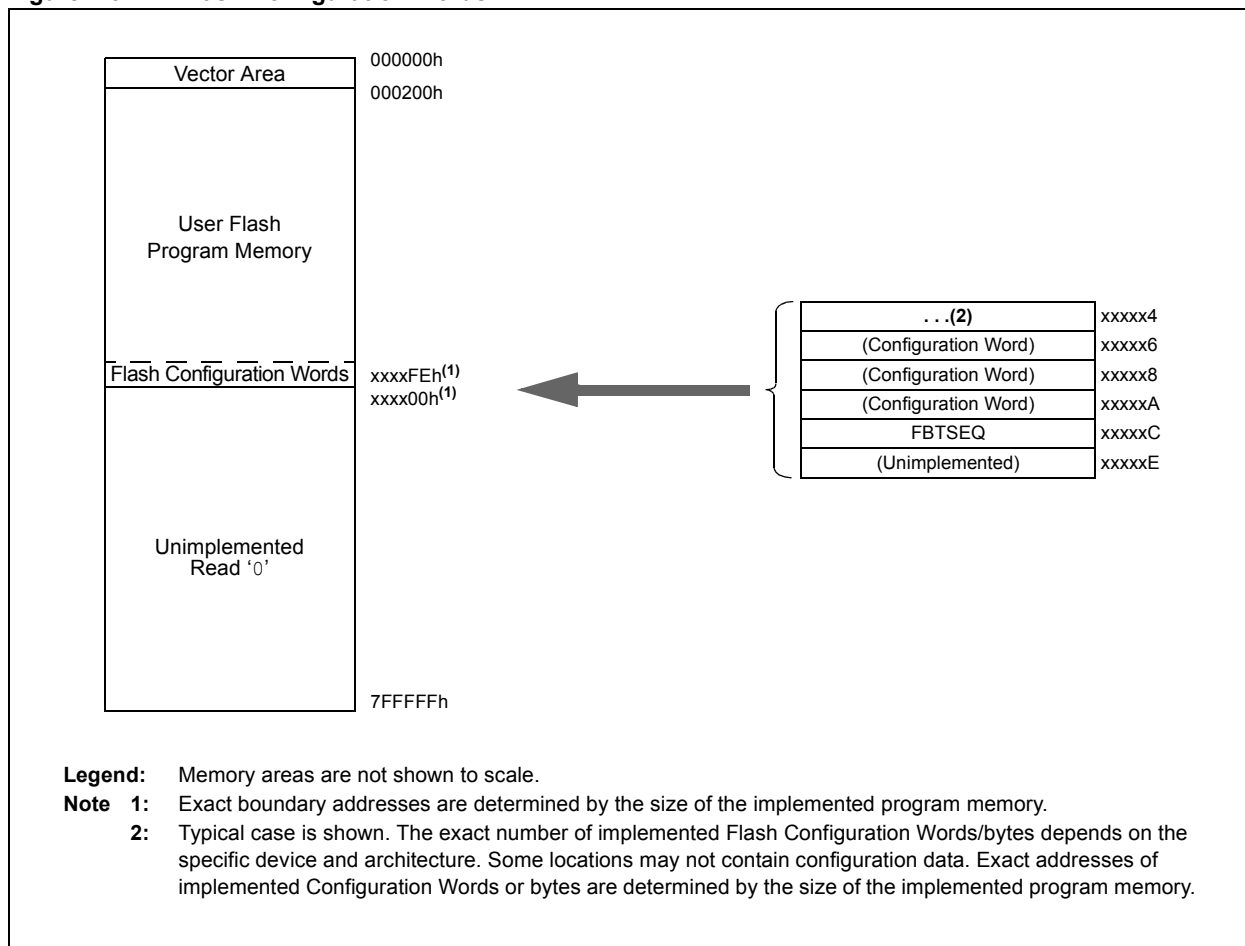
### 2.2.1 FLASH CONFIGURATION DATA

The area at the end of implemented Flash program memory (typically the last row) is reserved for Flash configuration data. On device Reset, this configuration information is copied into the appropriate device Configuration registers, which are not accessible to the user. Device configuration data can only be programmed by programming the desired values in the Flash Configuration Words.

The number, order and organization of the Configuration bits varies between device architectures, and among device families within the same architecture. Some devices organize Configuration bits as 16-bit Configuration Words, which are generally grouped in functional terms. Other devices organize Configuration bits in terms of individually addressable Configuration bytes. Figure 2-3 shows the area as organized for Configuration Words. Refer to the device data sheet for family-specific information.

For devices with Dual Partition capability, the FBTSEQ Configuration Word is usually the next-to-last Configuration Word, located at the end of implemented program memory.

**Figure 2-3: Flash Configuration Words**

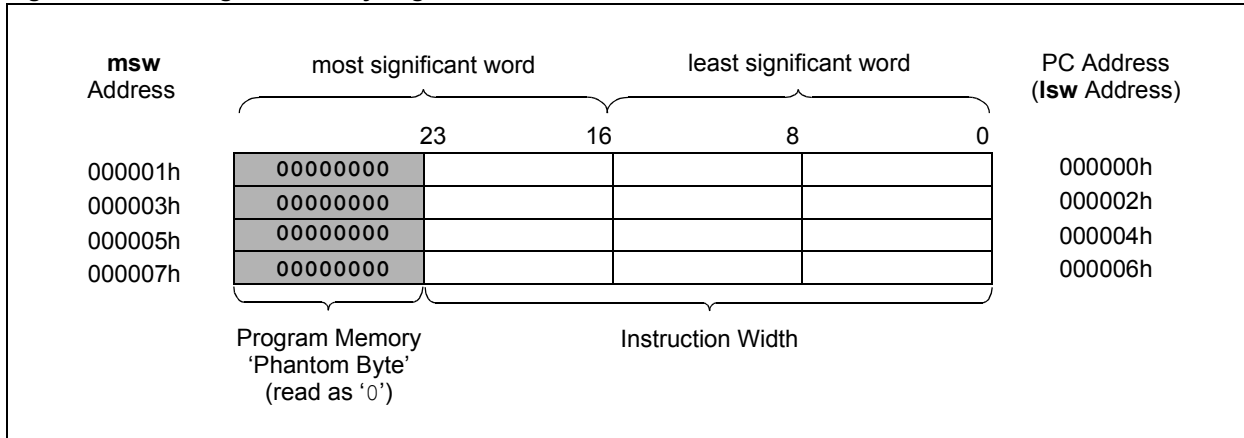


## 2.3 Memory Organization

The program memory space is organized as word-addressable blocks. Although it is treated as 24 bits wide, it is more appropriate to think of each address of the program memory as a lower and upper word, with the upper byte of the upper word being unimplemented. The lower word always has an even address, while the upper word has an odd address (Figure 2-4).

Program memory addresses and the PC are always word-aligned on the lower word (i.e., the Least Significant bit (LSb) is always '0'). Addresses are incremented or decremented by two during code execution.

Figure 2-4: Program Memory Organization



### 2.3.1 ADDRESSING PROGRAM MEMORY

For normal code execution, the Effective Address (EA) for execution is provided by the Program Counter (PC). The PC is 23 bits wide, allowing direct access to any location in the user program space. PC<0> is fixed as '0' in order to maintain program instruction alignment. The PC is incremented to the next sequential address by incrementing PC<1>, thus increasing the value of the PC by two.

For Table Read and Table Write operations, the EA is created by concatenating the 16-bit address from one of the W registers with the 8-bit address from the TBLPAG register. This permits table operations access to both the user and configuration spaces. Address generation for table operations is discussed in more detail in [Section 4.2.1 "Address Generation for Table Operations"](#).

For Extended Data Space (EDS) and Program Space Visibility (PSV) operations, the EA is created by concatenating the lower 15 bits of a W register with the 8-bit address from either the DSRPAG/DSWPAG (dsPIC33) or PSVPAG (PIC24F) registers. Extended Data Space and Program Space Visibility operations are discussed in the *dsPIC33/PIC24 Family Reference Manual*, **"Data Memory"** (dsPIC33, DS70595) and/or **"Data Memory"** (PIC24, DS39717).

## 3.0 PROGRAM MEMORY PARTITION FLASH OPERATION

For devices with Dual Partition Flash capability, the Dual Partition Program Memory mode is selected by programming the BTMODE<1:0> bits in the FBOOT Configuration Word. Unlike other Configuration Words, FBOOT is located in configuration memory space, apart from all other Flash Configuration registers. The exact address is architecture-specific (i.e., PIC24 or dsPIC33) and may vary between device families. [Table 3-1](#) lists the possible Flash Partition options, which are discussed in the following sections.

When a device is first programmed via In-Circuit Serial Programming™ (ICSP™), the programmer should program FBOOT to correctly set the device Flash Partition mode. Note that it is not possible to reprogram FBOOT at run time using Run-Time Self-Programming (RTSP). The FBOOT bits must be configured in ICSP mode by a programmer. This is because the location of the Flash Configuration Words changes from Standard Partition mode to Dual Partition mode, which could cause unexpected device operation.

**Table 3-1: Flash Partition Options**

BTMODE<1:0>	Partition Option
11	Standard Mode (Single Partition, default)
10	Dual Partition Mode
01	Protected Dual Partition Mode
00	Privileged Dual Partition Mode <sup>(1)</sup>

**Note 1:** Not implemented on all Dual Partition devices.

### 3.1 Standard (Single Partition) Mode

Standard mode, also referred to as Single Partition or Standard Partition mode, is the default operating mode for program memory. It is selected when the BTMODE<sub>x</sub> Configuration bits are '11' (their unprogrammed configuration). This is also the single program memory operating mode available to all previous dsPIC33 and PIC24 devices.

In Standard mode, the entire user program memory is mapped as a flat, continuous memory space, ranging from 000000h to the upper boundary of implemented Flash memory. For example, a device with 256 Kbytes of Flash memory has a program memory address range of 000000h to 02AFFh, with addresses above this range being unimplemented. The entire implemented memory range (excluding reserved spaces for Reset vectors, IVTs and Flash Configuration Words) is available for the user's application. In devices with segmented code security, a Boot Segment may also be implemented.

### 3.2 Dual Partition Modes

When the BTMODE<sub>x</sub> Configuration bits are programmed to a value other than '11', the device operates in one of three Dual Partition modes. In all of these modes, the implemented Flash memory is symmetrically split into two regions: an Active Partition, beginning at 000000h, and an Inactive Partition, beginning at 400000h. For the device in the previous example, the 256-Kbyte Flash memory would be implemented as two areas of 128 Kbytes each, ranging from addresses 000000h to 0157FFh and 400000h to 4157FFh. Addresses between the two areas are unimplemented (see [Figure 3-1](#)).

In the Dual Partition modes, two independent applications may be programmed into the device, one to each of two Flash memory partitions, known as Partition 1 and Partition 2. When the device is initialized, one of these is dynamically mapped to the Active Partition and executed. The other is mapped to the Inactive Partition, where it remains available to program memory operations. The assignment of a partition to the Active or Inactive Partition is determined automatically by a code signature, known as the Boot Sequence Number. The code partitions may also be swapped between Active and Inactive Partitions, during run time, under software control.

Dual Partition modes allow for the Active Partition's application to access (but not execute) program data in the Inactive Partition or to reprogram the Inactive Partition. Writing to Flash memory in the Inactive Partition does not require the CPU to stall while Flash writes occur. This allows for Live Update functionality, where execution of critical control functions or timing-sensitive communications can happen simultaneously with application updates. Certain Dual Partition modes place additional limitations on the process to help ensure code security and robustness of operation. Code cannot be executed when it is mapped to the Inactive Partition. The partitions may be swapped, but only code in the Active Partition can be executed.

## 3.2.1 DUAL PARTITION MODE

The simplest Dual Partition mode places no restrictions on operations from the Active Partition to the code in either Partitions 1 or 2. Any limitations on the interactions between Code Segments in different partitions are determined by the configuration of enhanced security features.

## 3.2.2 PROTECTED DUAL PARTITION MODE

Protected Dual Partition mode protects the default Code Segment (Partition 1) from any Flash write or erase operations. This allows for the implementation of a "Factory Default" mode by allowing a fail-safe backup image to be stored in Partition 1.

When Protected Dual Partition mode is used, Partition 1 cannot be written or erased by Flash memory operations while it is in the Inactive Partition. If Partition 1 is also write-protected via Configuration bit settings, it cannot be erased or written at any time. In contrast, Partition 2 can be erased or written by operations from either partition.

This allows for a fail-safe bootloader to be placed in Partition 1, along with a fail-safe backup code image. This code image can then be executed by default and used to rewrite Partition 2 in the event that a Flash update should fail.

## 3.2.3 PRIVILEGED DUAL PARTITION MODE

Privileged Dual Partition mode implements additional security protections in those cases where an application may have Code Segments written by different authors and a higher level of security is required to protect intellectual property for one of those segments. An example would be a system where the bulk of the code is written by the hardware's application developer, but includes a proprietary, third-party library. This mode is designed to work with the enhanced security features in select devices, which can selectively protect different Code Segments in the program memory space.

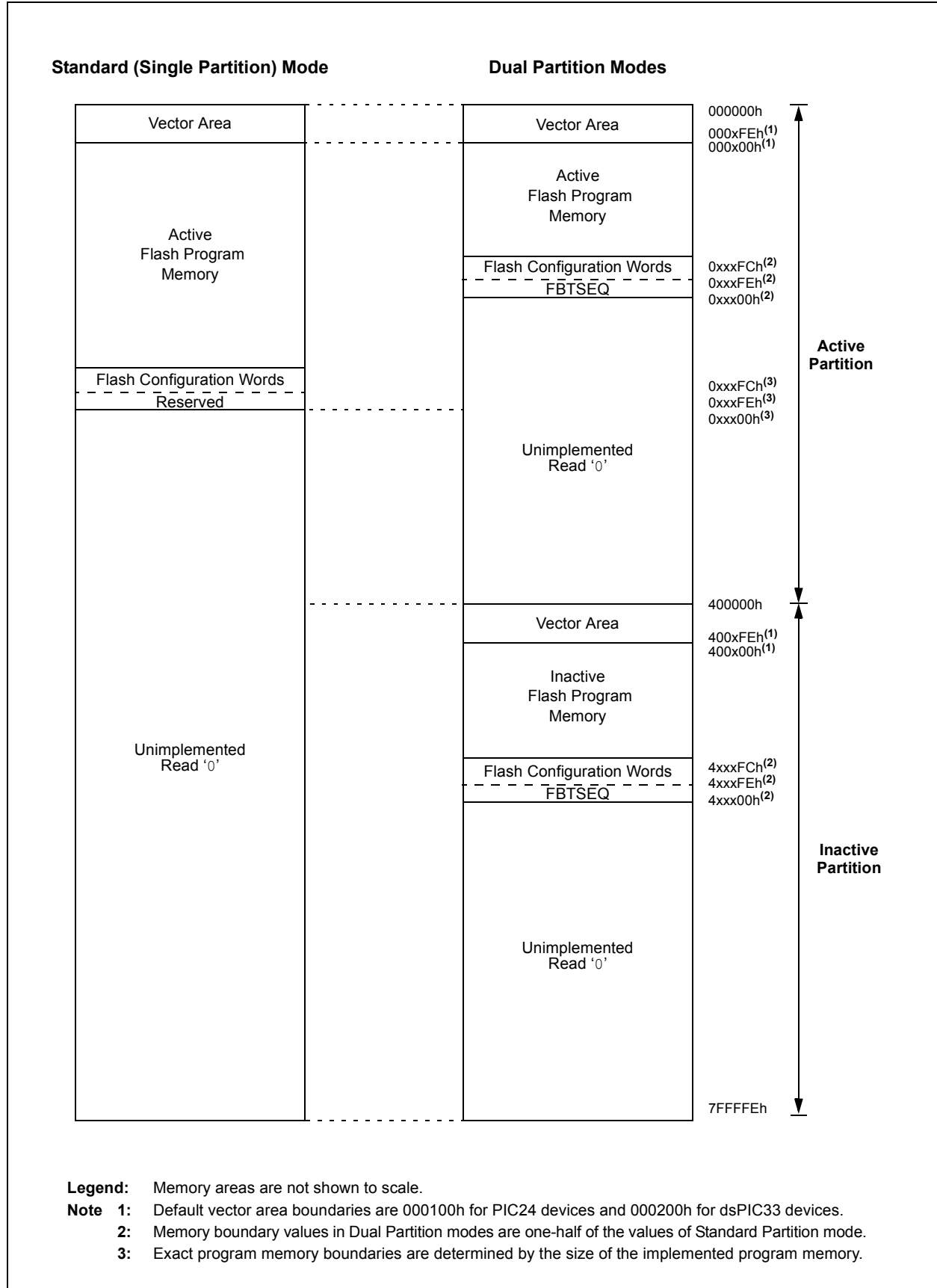
Privileged Dual Partition mode differs from Standard Dual Partition mode by adding special protection to the BSLIMx Configuration bits of both partitions. This protection effectively locks the bits, and prevents changes to the size of the Boot Segment and the General Segment. With the proper security settings, this ensures that neither segment will be altered or unexpectedly read at run time.

Privileged Dual Partition mode is not implemented on all devices with Dual Partition capability. Refer to the specific device data sheet for details.



# Dual Partition Flash Program Memory

**Figure 3-1: Standard and Dual Partition User Memory Space Map**



3.2.4 SELECTING A CODE PARTITION

In Dual Partition modes, there are two methods of determining which partition will be mapped to the Active Partition and executed: the Boot Sequence Number and the `BOOTSWP` instruction. The `P2ACTIV` bit (`NVMCON<10>`) can be used to determine which physical partition is the Active Partition. If `P2ACTIV = 1`, Partition 2 is active; if `P2ACTIV = 0`, Partition 1 is active.

The Boot Sequence Number is a 12-bit value that is used for automatically determining the Active Partition upon device Reset. Each partition should have a unique Boot Sequence Number, which is stored in the `FBTSEQ` Flash Configuration Word.

The `BOOTSWP` instruction is used to swap Active and Inactive Partitions without a device Reset.

3.2.4.1 Boot Sequence Number

The 12-bit Boot Sequence Number is stored in the `FBTSEQ` Flash Configuration Word, which is always located at the last location of user program memory, above the other Flash Configuration Words (see [Figure 3-2](#)). Unlike other Configuration registers, which only use the lower 16 bits of the program memory word, `FBTSEQ` is a full 24 bits wide. Each partition should, under normal operating conditions, have a different value for `FBTSEQ`. When Dual Partition modes are not used, the value of `FBTSEQ` is ignored.

The Boot Sequence Number is stored in two parts: the actual value in the bit field, `BSEQx` (`FBTSEQ<11:0>`), and the one's complement of the value in the `IBSEQx` bits field (`FBTSEQ<23:12>`). When the Boot Sequence Number is read upon a device Reset, the values of `BSEQx` and `IBSEQx` are automatically compared. If these two values are not mutual complements, the Boot Sequence Number is considered invalid. The complement value is not automatically created by hardware, nor is it verified by hardware upon programming. The application must calculate and program the appropriate value.

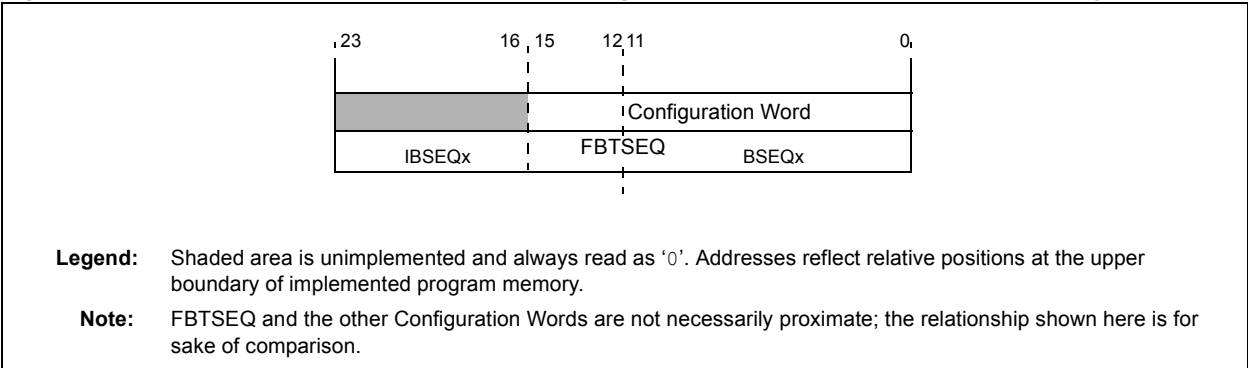
On device Reset, the Boot Sequence Numbers in both partitions are compared. The partition with the lower `BSEQx` value is the one that is mapped to the Active Partition and its code is executed. If one of the Boot Sequence Numbers is invalid, the device will select the partition with the valid Boot Sequence Number as the Active Partition, regardless of which Boot Sequence Number is lower. If both Boot Sequence Numbers are invalid, Partition 1 will be selected by default as the Active Partition.

The partitions can be prepared to be swapped during run time by reprogramming the Boot Sequence Number of the Inactive Partition to have a lower value. When a Reset is executed, the partition that has the lower value now becomes active. This method is used when the Inactive Partition has been updated and is then mapped to the Active Partition after a Reset.

The location of `FBTSEQ` allows it to be easily excluded from a checksum or other verification of the Flash program memory. Because the `FBTSEQ` value is likely to be determined at run time (based on the `BSEQx` of the other partition), it often cannot be included in a checksum, such as a CRC.

The sequence at the top of [Figure 3-3](#) shows the relationship between the code partitions when the Boot Sequence Number is altered and a device Reset is executed.

Figure 3-2: FBTSEQ in Relationship to Other Configuration Words (Dual Partition Modes Only)



## 3.2.4.2 BOOTSWP Instruction

The `BOOTSWP` instruction is an extension to the PIC24 and dsPIC33 instruction set. It supports the code, Live Update, by allowing Code Segments to be swapped between the Active and Inactive Partitions without the need for a device Reset. A partition swap using the `BOOTSWP` instruction is referred to as a “soft swap”. To execute a `BOOTSWP` instruction, the Configuration bit, `BTSWP` (`FIDC<25>`), must be cleared. If a `BOOTSWP` instruction is attempted with `BTSWP` set, a `NOF` instruction will result.

The `BOOTSWP` instruction must always be followed by a single-word instruction that writes the PC (e.g., `GOTO W`, `CALL W` or `BRA W`); the target of the instruction must be at an address within 32 Kbytes of the current address. Upon execution, the Active and Inactive Partitions trade places, and the PC vectors to the location specified by the `GOTO` instruction in the newly Active Partition.

<b>Note:</b> If the <code>BOOTSWP</code> instruction is executed from within a function that has created a new stack frame using the <code>LNK</code> instruction, a <code>CALL</code> must be used following <code>BOOTSWP</code> rather than a <code>GOTO</code> ; otherwise, the device will generate a stack error trap.
--

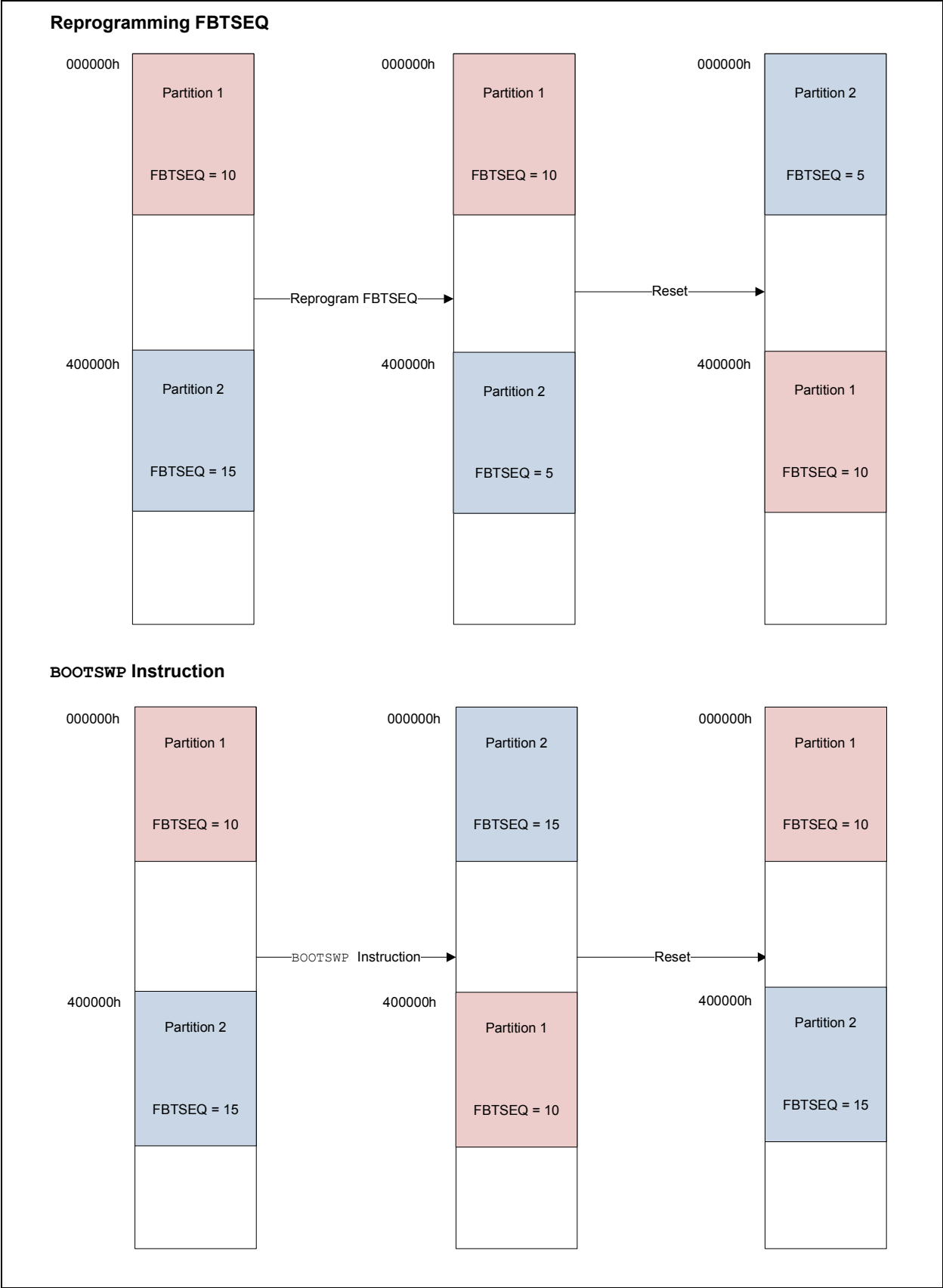
After the execution of the `BOOTSWP` instruction, the `SFTSWP` bit (`NVMCON<11>`) is set. This bit indicates to the firmware that the `BOOTSWP` instruction occurred correctly and that the currently Active Partition was entered via `BOOTSWP` rather than via a device Reset. Status bit, `P2ACTIV` (`NVMCON<10>`), can also be read to verify which partition is active.

It is important to note that, after the partition swap, all peripherals and interrupts which were previously enabled remain enabled. Additionally, the RAM and stack maintain their states after the swap. It is highly recommended that applications using soft swaps jump to a routine that re-initializes the device in order to ensure the application continues to run as expected.

For robustness of operation, it is necessary to execute the standard NVM unlocking sequence prior to executing the `BOOTSWP` instruction (writing 55h and AAh to the `NVMKEY` register in two sequential steps; see [Section 4.1 “Registers”](#) for more information). It is important to also disable interrupts before executing the unlock sequence. If the unlocking sequence is not performed, `BOOTSWP` will be executed as a forced `NOF`. The `GOTO` instruction following `BOOTSWP` is still executed, causing the PC to jump to that location in the current operating partition. Similarly, `BOOTSWP` has no effect in Standard Partition mode.

The sequence at the bottom of [Figure 3-3](#) shows the relationship between the partitions when a `BOOTSWP` instruction is executed. Note that a `BOOTSWP` partition change is temporary; after a subsequent device Reset, the partition with the lower Boot Sequence Number is reassigned to the Active Partition.

Figure 3-3: Comparing Partition Swap Methods



## 4.0 FLASH MEMORY PROGRAMMING

PIC24 and dsPIC33 devices can be programmed by any one of three methods:

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

RTSP is performed by the application software during execution, while ICSP and EICSP are performed from an external programmer using a serial data connection to the device. ICSP and EICSP allow much faster programming time than RTSP.

RTSP techniques are described in this section. The ICSP and EICSP protocols are defined in the programming specification documents for the respective devices, which can be downloaded from the Microchip web site ([www.microchip.com](http://www.microchip.com)).

### 4.1 Registers

Programming operations are controlled through six registers. The NVMCON and NVMKEY registers are used to enable and select all operations. The remaining four registers define Data and Address Pointers.

<b>Note:</b> Not all devices implement data RAM buffer programming. Refer to the specific device data sheet for more information.
---

#### 4.1.1 CONTROL REGISTERS

The NVMCON register ([Register 4-1](#)) controls all Flash programming operations. The NVMOP<3:0> bits (NVMCOM<3:0>) select the particular write or erase operation to be performed. The WR bit (NVMCOM<15>) triggers the appropriate operation; it remains set until the operation has been completed and is then cleared by hardware. The WREN bit (NVMCOM<14>) enables or disables write and erase operations. The WR bit cannot be set to trigger operations when WREN is clear.

NVMKEY is a write-only register that is used to prevent accidental writes or erasures of Flash memory; only the lower byte is implemented. To start a program or erase sequence, an “unlock” sequence of two writes is performed on the register:

1. Write 55h to NVMKEY<7:0>.
2. Write AAh to NVMKEY<7:0>.

After this sequence, a write will be allowed to the NVMCON register for one instruction cycle.

#### 4.1.2 ADDRESS REGISTERS

The NVMADRL and NVMADRH registers define the Start Address Pointer for write operations. Both types of program memory writes (latch-based and RAM buffered) use these registers to set the destination address.

The NVMSRCADRL and NVMSRCADRH registers define the starting address in data RAM of the source data when using RAM buffered programming. The NVMSRCADRH register is used on devices with Extended Data Space (EDS) to point to addresses in the Extended Data Space memory.

# dsPIC33/PIC24 Family Reference Manual

**Register 4-1: NVMCON: Flash Programming Control Register**

R/S-0 <sup>(1)</sup>	R/C-0	R/C-0	R/W-0	R/C-0	R-0	R/W-0	R/C-0
WR	WREN	WRERR	NVMPIDL <sup>(5)</sup>	SFTSWP	P2ACTIV	RPDF <sup>(2)</sup>	URERR <sup>(2)</sup>
bit 15							bit 8

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	NVMOP3	NVMOP2	NVMOP1	NVMOP0
bit 7							bit 0

<b>Legend:</b>	S = Settable Only bit	C = Clearable 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<sup>(1)</sup>  
1 = Initiates a Flash program/erase operation  
0 = Program/erase operation is complete and inactive
- bit 14      **WREN:** Program/Erase Enable bit  
1 = Allows program/erase cycles  
0 = Inhibits programming/erasing of memory or fuse elements
- bit 13      **WRERR:** Sequence Error Flag bit  
1 = An improper program/erase termination has occurred or an unimplemented programming operation has been selected  
0 = A program or erase operation is under way, has completed normally or has yet to start
- bit 12      **NVMPIDL:** NVM Power-Down in Idle Enable bit<sup>(5)</sup>  
1 = Removes power from Flash arrays when device enters Idle mode  
0 = Keeps Flash arrays powered in Standby mode when device enters Idle mode
- bit 11      **SFTSWP:** Soft Swap Status bit  
When BTMODE<1:0> = 10 or 0x:  
1 = Partitions have been successfully swapped using the BOOTSWP instruction  
0 = Awaiting successful partition swap using the BOOTSWP instruction  
When BTMODE<1:0> = 11:  
Unimplemented, read as '0'.
- bit 10      **P2ACTIV:** Dual Partition Active Status bit  
When BTMODE<1:0> = 10 or 0x:  
1 = Partition 2 Flash is the Active Partition  
0 = Partition 1 Flash is the Active Partition  
When BTMODE<1:0> = 11:  
Unimplemented, read as '0'.
- bit 9      **RPDF:** RAM Buffer Row Programming Data Format Control bit<sup>(2)</sup>  
1 = Row data is stored in RAM in compressed format  
0 = Row data is stored in RAM in uncompressed format

- Note 1:** This bit is also reset on a Brown-out Reset (BOR).
- Note 2:** RAM buffer row operations are not available on all devices; in those cases, these bits are unimplemented and read as '0'.
- Note 3:** Selecting these options will set the WRERR bit and clear the WR bit.
- Note 4:** Double-word program operations require two adjacent instruction words (24 bits each), aligned on a 4-instruction word boundary.
- Note 5:** Implemented in select devices only; refer to the specific device data sheet for details.

# Dual Partition Flash Program Memory

---

## Register 4-1: NVMCON: Flash Programming Control Register (Continued)

bit 8	<b>URERR:</b> RAM Buffer Row Programming Data Underrun Error Flag bit <sup>(2)</sup> 1 = Programming operation has terminated due to a data underrun error 0 = No data underrun error is detected.
bit 7-4	<b>Unimplemented:</b> Read as '0'
bit 3-0	<b>NVMOP&lt;3:0&gt;:</b> NVM Operation Select bits (initiated by the next setting of WR) 1xxx = Reserved <sup>(3)</sup> 011x = Reserved <sup>(3)</sup> 0101 = Reserved <sup>(3)</sup> 0100 = Inactive Partition erase operation (reserved option in Standard Partition mode) 0011 = Page erase operation 0010 = Row program operation 0001 = Double-word program operation <sup>(4)</sup> 0000 = Reserved <sup>(3)</sup>

- Note 1:** This bit is also reset on a Brown-out Reset (BOR).
- 2:** RAM buffer row operations are not available on all devices; in those cases, these bits are unimplemented and read as '0'.
- 3:** Selecting these options will set the WRERR bit and clear the WR bit.
- 4:** Double-word program operations require two adjacent instruction words (24 bits each), aligned on a 4-instruction word boundary.
- 5:** Implemented in select devices only; refer to the specific device data sheet for details.

## 4.2 Table Operation Instructions

The table instructions provide one method of transferring data between the program memory space and the data memory space of the PIC24 and dsPIC33 devices. A summary of the table instructions used during programming of the Flash program memory is provided in this section. There are four basic table instructions:

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

The TBLRDL and the TBLWTL instructions are used to read and write 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 are used to read or write 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 have the ability to address an upper byte of program memory that does not exist. This byte is called the 'phantom byte'. Any read of the phantom byte returns 00h; a write to the phantom byte has no effect.

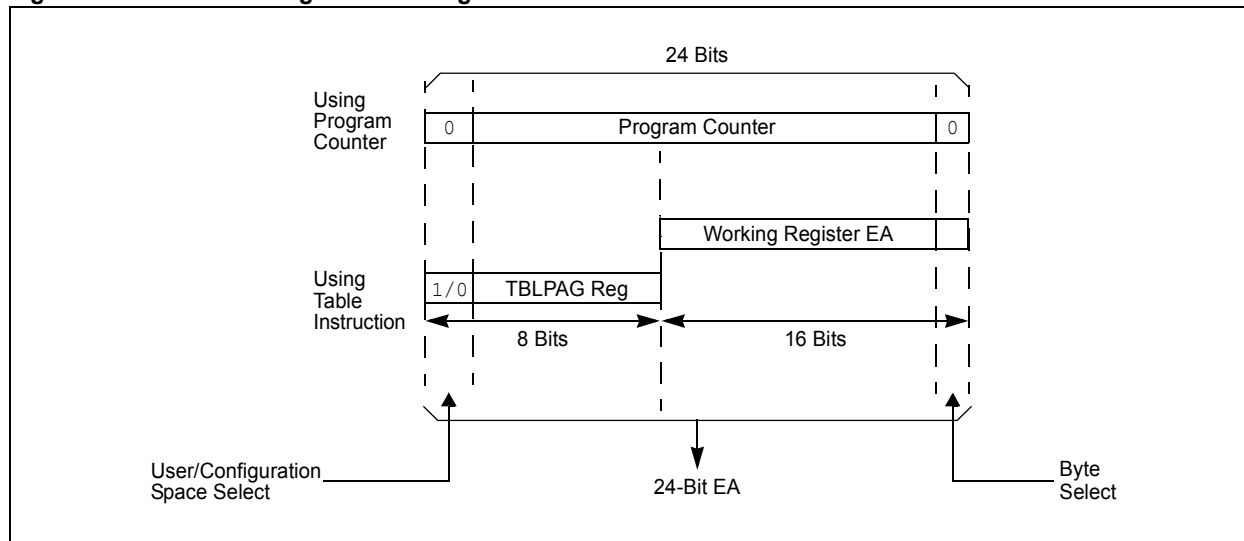
### 4.2.1 ADDRESS GENERATION FOR TABLE OPERATIONS

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 LSb of the table address will be used as the byte select bit. The LSb determines which byte in the high or low program memory space is accessed.

Figure 4-1 illustrates how the program memory is addressed using the table instructions. A 24-bit program memory address is formed using the TBLPAG<7:0> bits and the Effective Address (EA) from a W register, specified in the table instruction (the 24-bit Program Counter is shown 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 is used to pick 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 instructions also specify a W register (or a W 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 4-1: Addressing for Table Registers**



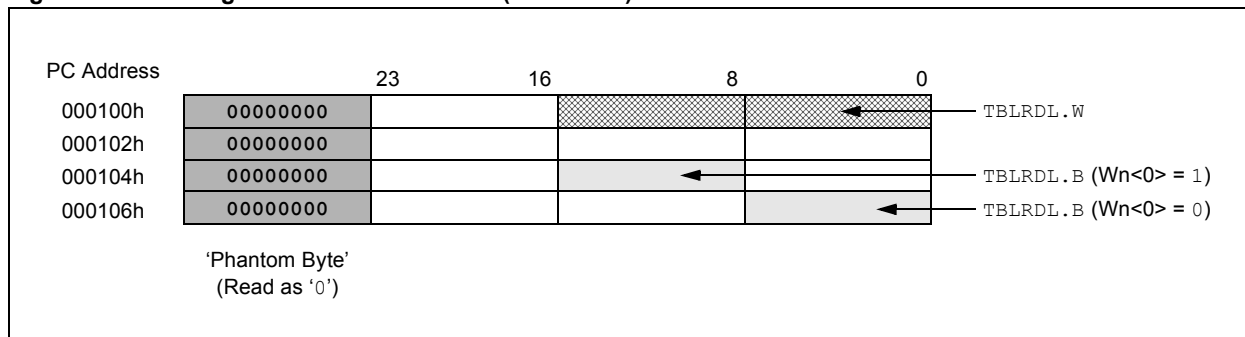


# Dual Partition Flash Program Memory

## 4.2.2 LOW WORD ACCESS

The `TBLRDL` and `TBLWTL` instructions are used to access the lower 16 bits of program memory data. The LSb of the W register address is ignored for word-wide table accesses. For byte-wide accesses, the LSb of the W register address determines which byte is read. Figure 4-2 illustrates the program memory data regions accessed by the `TBLRDL` and `TBLWTL` instructions.

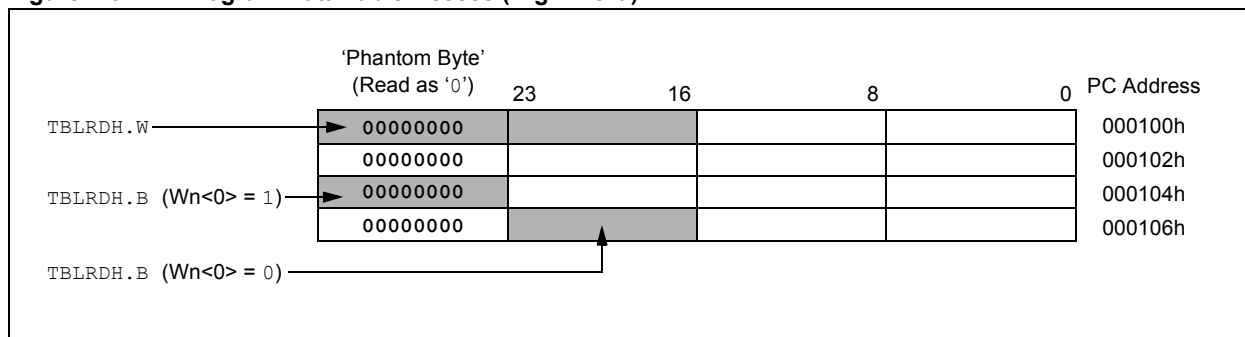
Figure 4-2: Program Data Table Access (Low Word)



## 4.2.3 HIGH WORD ACCESS

The `TBLRDH` and `TBLWTH` instructions are used to access the upper 8 bits of the program memory data. These instructions also support Word or Byte Access modes for orthogonality, but the high byte of the program memory data will always return '0', as shown in Figure 4-3.

Figure 4-3: Program Data Table Access (High Word)



## 4.2.4 DATA STORAGE IN PROGRAM MEMORY

It is assumed that for most applications, the high byte (PM<23:16>) will not be used for data, making the program memory appear 16 bits wide for data storage. It is recommended that the upper byte of program data be programmed either as a `NOP` (00h or FFh), or as an illegal opcode (3Fh) value, to protect the device from accidental execution of stored data. The `TBLRDH` and `TBLWTH` instructions are primarily provided for array program/verification purposes, and for those applications that require compressed data storage.

## 4.2.5 PROGRAM MEMORY BIT BEHAVIOR

Bits in Flash program memory can only be programmed from '1' to '0' and can be subsequently erased to '1'. Attempting to set a bit with a programming sequence will have no effect.

## 4.2.6 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.

The code examples in [Example 4-1](#) and [Example 4-2](#) demonstrate how to read a word of program memory using the table instructions in Word mode.

### Example 4-1: Read Word Mode (in Assembly)

```
; 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
; Perform the table writes to load the latch
TBLRDL [W0], W2
TBLRDH [W0], W3
```

### Example 4-2: Read Word Mode (in C)

```
int addrOffset;
int varWord1;
int varWord2;

TBLPAG = ((PROG_ADDR & 0x7F0000)>>16);
addrOffset = (PROG_ADDR & 0x00FFFE);

varWord1 = __builtin_tblrdl(addrOffset);
varWord2 = __builtin_tblrdh(addrOffset);
```

**Note:** The `tblpage()` and `tbloffset()` directives are provided by the Microchip assembler for dsPIC33 and PIC24 devices. These directives select the appropriate TBLPAG and W register values for the table instruction from a program memory address value. Refer to the “*MPLAB® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User's Guide*” (DS51317) for more information.

## 4.2.7 TABLE WRITE HOLDING LATCHES

Table Write instructions do not write directly to the Flash program array. Instead, the instructions cause the data to be programmed to be loaded first into holding latches. These latches are memory-mapped in configuration memory space, typically starting at FA0000h, and can only be accessed using the 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.

Different devices implement different numbers of holding latches, based on a specific program array design (i.e., the row programming size and row programming algorithm). Please refer to the specific device data sheet and/or programming specification for further details.

### 4.2.7.1 Performing a Two-Word Write

Word writes are performed for two words at a time using a pair of TBLWTH and TBLWTL instructions. The code sequences in either [Example 4-3](#) or [Example 4-4](#) (C equivalent) can be used to write two program memory latch locations to be programmed to Flash using Word Write mode.

#### Example 4-3: Two-Word Write Example (in Assembly)

```
; Set up the address pointer to 1st write latch
MOV    0xFA, W0                      ; get table page value
MOV    W0,TBLPAG                     ; load TBLPAG register
MOV    0x0, W0                       ; load address LS word
; Load write data into W registers
MOV    #PROG_LOW_WORD_1, W2
MOV    #PROG_HI_BYTE_1, W3
MOV    #PROG_LOW_WORD_2, W4
MOV    #PROG_HI_BYTE_2, W5
; Perform the table writes to load the latch
TBLWTL W2, [W0]
TBLWTH W3, [W0++]
TBLWTL W4, [W0]
TBLWTH W5, [W0++]
```

#### Example 4-4: Two-Word Write Example (in C)

```
int varWord1L = 0xFFFF;
int varWord1H = 0x00XX;
int varWord2L = 0xFFFF;
int varWord2H = 0x00XX;
int addrOffset;
int TargetWriteAddressL; // bits<15:0>
int TargetWriteAddressH; // bits<22:16>

NVMCON = 0x4001; // Set WREN and word program mode
TBLPAG = 0xFA; // write latch upper address
addrOffset = (PROG_ADDR & 0x00FFFE); // ensure address is properly aligned
NVMADRL = TargetWriteAddressL; // set target write address
NVMADRH = TargetWriteAddressH;

__builtin_tblwtl(0,varWord1L); // load write latches
__builtin_tblwth(0,varWord1H);
__builtin_tblwtl(0x2,varWord2L);
__builtin_tblwth(0x2,varWord2H);
__builtin_disi(5); // Disable interrupts for NVM unlock sequence
__builtin_write_NVM(); // initiate write
```

## 4.3 Run-Time Self-Programming (RTSP)

RTSP allows the user code to modify Flash program memory contents. RTSP is accomplished using **TBLRD** (Table Read) and **TBLWT** (Table Write) instructions, and the NVM Control registers. PIC24 and dsPIC33 devices support the following Flash programming operations:

- Flash page erases
- Row programming (either latch-based or RAM-based)
- Word programming

Flash programming via RTSP is performed, either with blocks of memory called rows, or with two words of Flash memory. Prior to programming, a memory location must be erased. Erase operations are performed on blocks of memory called pages which consist of multiple rows. The size of a row will vary by device; refer to the device data sheet for details. Typically, for dsPIC33 and PIC24 devices, a page is defined as eight (8) rows. This document uses examples with 64 instructions per row (512 instructions per page).

### 4.3.1 ROW PROGRAMMING USING WRITE HOLDING LATCHES

As discussed in [Section 4.2.7 “Table Write Holding Latches”](#), devices which implement latch-based row programming have holding latches which contain the programming data. Prior to the actual programming operation, the write data must be loaded into the latches via **TBLWT** instructions in sequential order. When performing a row write, the instruction words must be loaded into the latches as a full row.

The basic sequence for RTSP programming is to set up a Table Pointer, then do a series of **TBLWT** instructions to load the buffers. Programming is performed by setting the control bits in the NVMCON register. For example, on a device with 64 instruction rows, a programming cycle would consist of 64 **TBLWTL** and 64 **TBLWTH** instructions to load the write latches, followed by a programming sequence unlocking NVMCON and setting the WR bit. [Example 4-5](#) shows an example of the process.

#### Example 4-5: Row Programming with Write Latches (in C)

```
int varWordL[64];
int varWordH[64];
int targetWriteAddressL;           // bits<15:0>
int targetWriteAddressH;           // bits<22:16>
int i;

NVMCON = 0x4002;                   // Set WREN and row program mode
TBLPAG = 0xFA;
NVMADRL = targetWriteAddressH;     // set target write address
NVMADRH = targetWriteAddressL;

for(i=0; i<=63; i++)               // load write latches with data
{                                   // to be written
    __builtin_tblwtl(i, varWordL[i]);
    __builtin_tblwth((i * 2), varWordH[i]);
}
__builtin_disi(5);                  // Disable interrupts for NVM unlock sequence
__builtin_write_NVM();              // initiate write
```

## 4.3.2 ROW PROGRAMMING USING THE RAM BUFFER

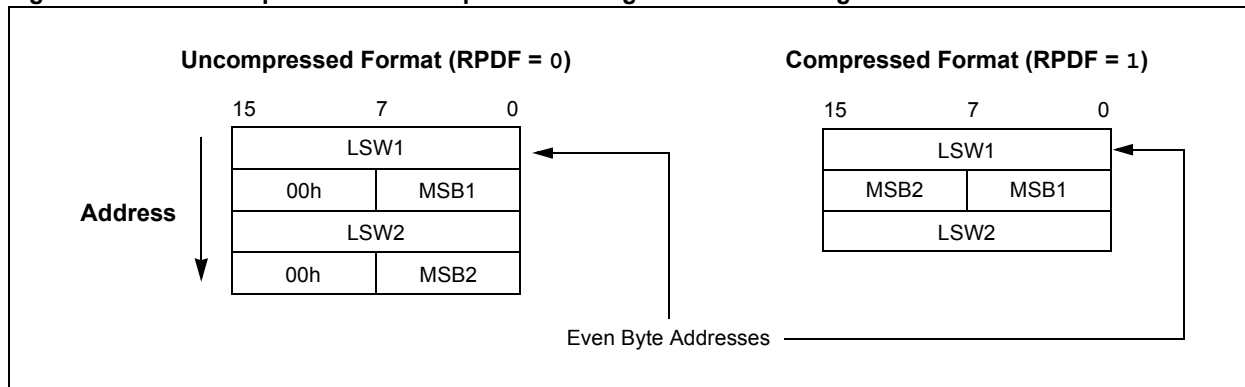
Select dsPIC33 and PIC24 devices permit row programming to be performed directly from a buffer space in data RAM, rather than going through the holding latches to transfer data with `TBLWT` instructions. The location of the RAM buffer is determined by the `NVMSRCADR` register(s), which are loaded with the data RAM address containing the first word of program data to be written.

Prior to performing the program operation, the buffer space in RAM must be loaded with the row of data to be programmed. The RAM can be loaded in either a compressed (packed) or uncompressed format. Compressed storage uses one data word to store the Most Significant Bytes (MSBs) of two adjacent program data words. The uncompressed format uses two data words for each program data word, with the upper byte of every other word being 00h. Compressed format uses about 3/4 of the space in data RAM as compared to uncompressed format. Uncompressed format, on the other hand, mimics the structure of the 24-bit program data word, complete with the upper phantom byte. The data format is selected by the `RPDF` bit (`NVMCON<9>`). These two formats are shown in [Figure 4-4](#).

Once the RAM buffer is loaded, the Flash Address Pointers, `NVMADRL` and `NVMADRH`, are loaded with the 24-bit start address of the Flash row to be written. As with programming the write latches, the process is initiated by writing the NVM unlock sequence, followed by setting the `WR` bit. Once initiated, the device automatically loads the right latches and increments the NVM Address registers until all bytes have been programmed. [Example 4-6](#) shows an example of the process. If `NVMSRCADR` is set to a value such that a data underrun error condition occurs, the `URERR` bit (`NVMCON<8>`) will be set to indicate the condition.

Devices which implement RAM buffer row programming also implement one or two write latches. These are loaded using the `TBLWT` instructions and are used to perform word programming operations.

**Figure 4-4: Uncompressed and Compressed Storage Formats for Program Data**



**Example 4-6: Writing Program Memory from a Data RAM Buffer (in C)**

```
int data[64]; // Data to be programmed in RAM
int targetWriteAddressL; // bits<15:0>
int targetWriteAddressH; // bits<22:16>

NVMCON = 0x4002; // Row programming
NVMCONbits.RPDF = 0; // Select compressed format
NVMSRCADRL = (int)&data[0]; // Start address of data in RAM
NVMADRL = targetWriteAddressL;
NVMADRH = targetWriteAddressH;

__builtin_disi(5); // Disable interrupts for NVM unlock sequence
__builtin_write_NVM();
```

## 4.4 General Flash Programming Algorithms

Flash programming operations are controlled using the following Nonvolatile Memory (NVM) control registers:

- NVMCON
- NVMKEY
- NVMADRL/H
- NVMSRCADRL/H (some devices)

A complete programming sequence is necessary for programming or erasing the internal Flash in RTSP mode. Setting the WR bit (NVMCON<15>) starts the operation and the WR bit is automatically cleared when the operation is finished.

When performing Flash programming operations on the Active Partition (particularly in Standard Partition mode), the CPU will stall until the operation is complete. When programming the Inactive Partition, the CPU can continue to operate without stalling. The following sections outline programming algorithms that exhibit CPU stall and no stall.

### 4.4.1 ERASING PROGRAM MEMORY (ACTIVE PARTITION)

1. Set the NVMOPx bits (NVMCOM<3:0>) to '0011' to configure for page erase and set the WREN bit (NVMCOM<14>).
2. Write the starting address of the block to be erased into the NVMADRL/H registers.
3. Disable interrupts.
4. Write 55h to NVMKEY.
5. Write AAh to NVMKEY.
6. Set the WR bit (NVMCOM<15>) to start the erase cycle.
7. Execute two NOP instructions.

When the erase is done, the WR bit is cleared automatically.

### 4.4.2 ROW PROGRAMMING (ACTIVE PARTITION, STANDARD PARTITION MODE)

The user can program one row of program Flash memory at a time. To do this, it is necessary to erase the page containing the desired row.

The general process for row programming to the Active Partition is:

1. Read eight rows of program memory (512 instructions) and store in data RAM.
2. Update the program data in RAM with the desired new data.
3. Erase the block:
  - a) Set the NVMOPx bits (NVMCOM<3:0>) to '0011' to configure for page erase and set the WREN bit (NVMCOM<14>).
  - b) Write the starting address of the block to be erased into the NVMADRL/H registers.
  - c) Write 55h to NVMKEY.
  - d) Write AAh to NVMKEY.
  - e) Set the WR bit (NVMCOM<15>). The erase cycle begins and the CPU stalls for the duration of the erase cycle. When the erase is done, the WR bit is cleared automatically.
4. Write the first 64 instructions from data RAM into the program memory buffers (see [Section 4.2.7 "Table Write Holding Latches"](#)) or write the NVMSRCADR register with the starting address of the data stored in RAM.
5. Write the program block to Flash memory:
  - a) Set the NVMOPx bits to '0010' to configure for row programming and set the WREN bit.
  - b) Write 55h to NVMKEY.
  - c) Write AAh to NVMKEY.
  - d) Set the WR bit. The programming cycle begins and the CPU stalls for the duration of the write cycle. When the write to Flash memory is done, the WR bit is cleared automatically.
6. Repeat Steps 4 and 5, using the next available 64 instructions from the block in data RAM by incrementing the addresses in NVMADRL/H, until all 512 instructions are written back to Flash memory.

**Note:** Not all devices will exhibit CPU stall during a write or erase cycle. To avoid stalls, it is recommended to avoid reads or writes by the application to the row being erased or written.

For protection against accidental operations, the write initiate sequence for NVMKEY is required prior to any erase or program operation. After the programming command has been executed, the user must wait for the programming time until programming is complete. The two instructions following the start of the programming sequence should be `NOFS`.

**Note 1:** The number of rows, blocks and holding latches may vary from device to device; please refer to the specific device data sheet for actual numbers, as well as the complete reference code of Flash memory programming.

**2:** For devices with a single holding latch, the Flash program memory must be written to by word programming.

## 4.4.3 PROGRAMMING A PAGE IN THE INACTIVE PARTITION (DUAL PARTITION MODES)

Programming in Dual Partition modes requires special considerations. Because the CPU is able to continue executing instructions while the Inactive Partition is being programmed, CPU stalls will not occur.

The algorithm for erasing and reprogramming a page of data in one of the Dual Partition modes is as follows:

1. Erase the block:
  - a) Set the NVMOPx bits (NVMCOM<3:0>) to '0011' to configure for page erase.
  - b) Set the WREN bit (NVMCOM<14>).
  - c) Write the starting address of the block to be erased into the NVMADR registers with the page address.
  - d) Write 55h to NVMKEY.
  - e) Write AAh to NVMKEY.
  - f) Set the WR bit (NVMCOM<15>). The erase cycle begins and the CPU will remain running.
  - g) When the erase is done, the WR bit is cleared automatically and the NVM Write Complete Interrupt Flag (NVMIF) will occur.
2. Prepare the data to be programmed by filling the RAM buffer; alternately, load the write latches with `TBLWT` instructions with the data for the first row of memory (64 instructions).
3. Program the block:
  - a) Set the NVMOPx bits (NVMCON<3:0>) to '0010' to configure for row programming.
  - b) Set the WREN (NVMCON<14>) bit.
  - c) Write the starting address of the block to be written into the NVMADR registers with the row starting address.
  - d) Write 55h to NVMKEY.
  - e) Write AAh to NVMKEY.
  - f) Set the WR bit (NVMCOM<15>). The write cycle begins and the CPU will remain running.
  - g) When the erase is done, the WR bit is cleared automatically and the NVM Write Complete Interrupt Flag (NVMIF) will occur.
4. Repeat Steps 2 and 3 to program each of the remaining rows of data in the erased page.

## 4.4.4 PROGRAMMING THE ENTIRE INACTIVE PARTITION (DUAL PARTITION MODES)

To entirely update the code in the Inactive Partition:

1. Erase the Inactive Partition:
  - a) Set the NVMOPx bits (NVMCOM<3:0>) to '0100' to configure for Inactive Partition erase.
  - b) Set the WREN bit (NVMCOM<14>).
  - c) Write 55h to NVMKEY.
  - d) Write AAh to NVMKEY.
  - e) Set the WR bit (NVMCOM<15>). The erase cycle begins and the CPU will remain running during the cycle. When the erase is done, the WR bit is cleared automatically, and the NVM Write Complete Interrupt Flag (NVMIF) occurs.
2. Write each page of the Inactive Partition using page writes, as described in [Section 4.4.3 "Programming a Page in the Inactive Partition \(Dual Partition Modes\)"](#).
3. Verify the written data. One suggested method is to perform a CRC on the data to be written and verify the CRC value on the full partition to ensure the data was written correctly.

## 4.4.5 UPDATING THE ACTIVE PARTITION USING A BOOTLOADER

1. Erase and program the entire Inactive Partition as described in [Section 4.4.4 "Programming the Entire Inactive Partition \(Dual Partition Modes\)"](#).
2. Read the FBTSEQ Configuration register of the Active Partition.
3. Decrement the value by one and write to FBTSEQ of the Inactive Partition.
4. Force a partition swap:
  - a) If CPU stalls are not a concern, perform a device Reset. Since the Inactive Partition has a lower Boot Sequence Number, it will become the Active Partition after the Reset.
  - b) If a CPU stall is not acceptable, execute the `BOOTSWP` instruction.



## 5.0 PROGRAM SPACE VISIBILITY AND EXTENDED DATA SPACE (PSV AND EDS)

For all dsPIC33 and PIC24 devices, table instructions (see [Section 4.2 “Table Operation Instructions”](#)) can be used to access data within the program memory space. This is useful when data only needs to be read or written, one byte or word at a time. It is also possible to map 16K word pages of the program memory space into the upper 32 Kbytes of the data address space. This allows an effective expansion of the data space beyond its normal 64-Kbyte addressing limits, as well as transparent access without the use of table instructions.

All dsPIC33 and PIC24 devices are able to map any page in the implemented program memory space into the data space. This feature is known as Program Space Visibility (PSV).

Some devices expand PSV by memory-mapping certain peripherals to a specific range of virtual program memory pages. This feature is particularly useful for peripherals, such as the Advanced Graphics Controller, which has high data throughput requirements. This expansion of PSV is known as Extended Data Space (EDS).

PSV and EDS are implemented as features of the data memory. They are implemented differently for dsPIC33 and PIC24 devices. For a detailed description, refer to the “dsPIC33/PIC24 Family Reference Manual”, “**Data Memory**”. (dsPIC33, DS70595) and/or “**Data Memory**” (PIC24, DS39717).

### 5.1 PSV and Instruction Stalls

For more information about instruction stalls using PSV, refer to the “dsPIC33/PIC24 Family Reference Manual”, “**dsPIC33E Enhanced CPU**” (DS70005158).

## 6.0 REGISTER MAP

A summary of the SFRs associated with the Dual Partition Flash Program Memory is provided in [Table 6-1](#).

**Table 6-1: Special Function Registers Associated with Flash Program Memory<sup>(1)</sup>**

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets <sup>(2)</sup>
TBLPAG	—	—	—	—	—	—	—	—	Table Page Address Pointer								0000
NVMCON	WR	WREN	WRERR	NVMPIDL	SFTSWP	P2ACTIV	RPDF	URERR	—	—	—	—	NVMOP3	NVMOP2	NVMOP1	NVMOP0	0000
NVMKEY	—	—	—	—	—	—	—	—	NVMKEY<7:0>								0000
NVMSRCADRL	Data RAM Programming Buffer Start Address																0000
NVMSRCADRH	Data RAM Programming Buffer Address (EDS Operations Only)																0000
NVMADRL	Flash Program Memory Destination Address, Lower Byte (ADDR<15:0>)																0000
NVMADRH	—	—	—	—	—	—	—	—	Flash Program Memory Destination Address, Upper Byte (ADDR<23:16>)								0000

**Legend:** — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

**Note 1:** Please refer to the device data sheet for specific memory map details.

**2:** Reset value shown is for POR only. Value on other Reset states is dependent on the state of the memory write or erase operations at the time of Reset.

## 7.0 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 PIC24 or dsPIC33 product families, but the concepts are pertinent and could be used with modification and possible limitations.

The current application notes related to the Dual Partition Flash Program Memory are:

Title	Application Note #
-------	--------------------

No related application notes at this time.

<b>Note:</b> Please visit the Microchip web site ( <a href="http://www.microchip.com">www.microchip.com</a> ) for additional Application Notes and code examples for the PIC24 and dsPIC33 families of devices.
---

## 8.0 REVISION HISTORY

### **Revision A (March 2014)**

Original version of this document.

### **Revision B (February 2015)**

Changed the title and all instances of the phrase, “Dual Boot Flash Program Memory” to “Dual Partition Flash Program Memory” or “Dual Partition Flash”.

---

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

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

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

---

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

#### **Trademarks**

The Microchip name and logo, the Microchip logo, dsPIC, FlashFlex, flexPWR, JukeBlox, KEELoQ, KEELoQ logo, Klear, LANCheck, MediaLB, MOST, MOST logo, MPLAB, OptoLyzer, PIC, PICSTART, PIC<sup>32</sup> logo, RightTouch, SpyNIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

The Embedded Control Solutions Company and mTouch are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, ECAN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, KlearNet, KlearNet logo, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, RightTouch logo, REAL ICE, SQI, Serial Quad I/O, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademarks of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

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

© 2014-2015, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-63277-049-3

**QUALITY MANAGEMENT SYSTEM**  
**CERTIFIED BY DNV**  
**== ISO/TS 16949 ==**

*Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELoQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

## Worldwide Sales and Service

### AMERICAS

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

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

**Austin, TX**  
Tel: 512-257-3370

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

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

**Cleveland**  
Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643

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

**Detroit**  
Novi, MI  
Tel: 248-848-4000

**Houston, TX**  
Tel: 281-894-5983

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

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

**New York, NY**  
Tel: 631-435-6000

**San Jose, CA**  
Tel: 408-735-9110

**Canada - Toronto**  
Tel: 905-673-0699  
Fax: 905-673-6509

### ASIA/PACIFIC

**Asia Pacific Office**  
Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2943-5100  
Fax: 852-2401-3431

**Australia - Sydney**  
Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

**China - Beijing**  
Tel: 86-10-8569-7000  
Fax: 86-10-8528-2104

**China - Chengdu**  
Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889

**China - Chongqing**  
Tel: 86-23-8980-9588  
Fax: 86-23-8980-9500

**China - Dongguan**  
Tel: 86-769-8702-9880

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

**China - Hong Kong SAR**  
Tel: 852-2943-5100  
Fax: 852-2401-3431

**China - Nanjing**  
Tel: 86-25-8473-2460  
Fax: 86-25-8473-2470

**China - Qingdao**  
Tel: 86-532-8502-7355  
Fax: 86-532-8502-7205

**China - Shanghai**  
Tel: 86-21-5407-5533  
Fax: 86-21-5407-5066

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

**China - Shenzhen**  
Tel: 86-755-8864-2200  
Fax: 86-755-8203-1760

**China - Wuhan**  
Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

**China - Xian**  
Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256

### ASIA/PACIFIC

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

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

**India - Bangalore**  
Tel: 91-80-3090-4444  
Fax: 91-80-3090-4123

**India - New Delhi**  
Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632

**India - Pune**  
Tel: 91-20-3019-1500

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

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

**Korea - Daegu**  
Tel: 82-53-744-4301  
Fax: 82-53-744-4302

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

**Malaysia - Kuala Lumpur**  
Tel: 60-3-6201-9857  
Fax: 60-3-6201-9859

**Malaysia - Penang**  
Tel: 60-4-227-8870  
Fax: 60-4-227-4068

**Philippines - Manila**  
Tel: 63-2-634-9065  
Fax: 63-2-634-9069

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

**Taiwan - Hsin Chu**  
Tel: 886-3-5778-366  
Fax: 886-3-5770-955

**Taiwan - Kaohsiung**  
Tel: 886-7-213-7828

**Taiwan - Taipei**  
Tel: 886-2-2508-8600  
Fax: 886-2-2508-0102

**Thailand - Bangkok**  
Tel: 66-2-694-1351  
Fax: 66-2-694-1350

### EUROPE

**Austria - Wels**  
Tel: 43-7242-2244-39  
Fax: 43-7242-2244-393

**Denmark - Copenhagen**  
Tel: 45-4450-2828  
Fax: 45-4485-2829

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

**Germany - Dusseldorf**  
Tel: 49-2129-3766400

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

**Germany - Pforzheim**  
Tel: 49-7231-424750

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

**Italy - Venice**  
Tel: 39-049-7625286

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

**Poland - Warsaw**  
Tel: 48-22-3325737

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

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

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