dsPIC33/PIC24 Program Memory

HIGHLIGHTS

This section of the manual contains the following topics:

Note: This family reference manual section is meant to serve as a complement to device data sheets. Depending on the device variant, this manual section may not apply to all dsPIC33/PIC24 devices.

Please consult the note at the beginning of the **"Memory Organization"** and **"Flash Program Memory"** chapters in the current device data sheet to check whether this document supports the device you are using.

Device data sheets and family reference manual sections are available for download from the Microchip Worldwide Website at: http://www.microchip.com.

1.0 PROGRAM MEMORY ADDRESS MAP

dsPIC33/PIC24 devices have a 4M x 24-bit program memory address space. Figure 1-1 shows a typical program memory map for dsPIC33/PIC24 family devices. Figure 1-2 provides an example of the program memory map for devices that also implement auxiliary memory.

The program memory space can be accessed through the following methods:

- 23-bit Program Counter (PC)
- Table Read (`TBLRD`) instruction
- Program Space Visibility (PSV) mapping any 32-Kbyte segment of program memory into the data memory address space

The program memory address space in dsPIC33/PIC24 devices is divided into two equal halves, referred to as the User Memory Space and the Configuration Memory Space.

The User Memory Space is comprised of the following areas:

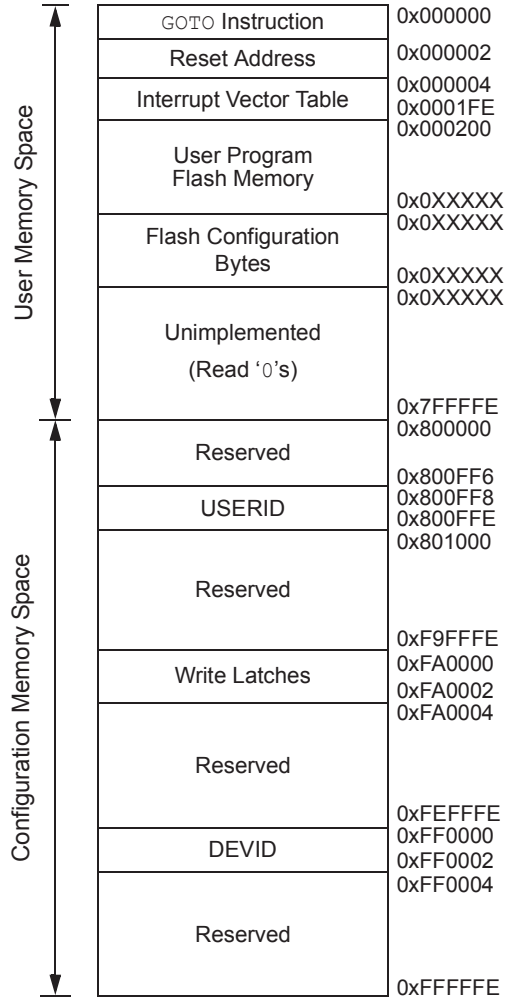
- User Program Flash Memory
- Flash Configuration Bytes (if applicable; refer to the **"Special Features"** chapter of the specific device data sheet for availability)
- Auxiliary Program Flash Memory (if applicable; refer to the **"Memory Organization"** chapter of the specific device data sheet for availability)

For devices that support auxiliary program Flash memory, instructions in the auxiliary program Flash memory can be executed by the CPU, without stalling it, while the user program memory is being erased and/or programmed. Similarly, instructions in the user program memory can be executed by the CPU while the auxiliary program memory is being erased and/or programmed, without Stalls.

The Configuration Memory Space consists of the following areas:

- Device Configuration registers (if applicable; refer to the **"Special Features"** chapter of the specific device data sheet for availability)
- Either USERID or One-Time-Programmable (OTP) locations to store serialization and other application-specific data (if applicable; refer to the **"Special Features"** chapter of the device data sheet for specific implementation details)
- Write latches, which are used for programming user and auxiliary Flash memory (the number of latches is device-dependent; refer to the **"Memory Organization"** chapter of the specific device data sheet for the number of available write latches)
- DEVID locations, which contain the device ID and revision ID. Refer to the *"Programming Specification"* for your device, which is available for download from the Microchip Website (www.microchip.com) for more information

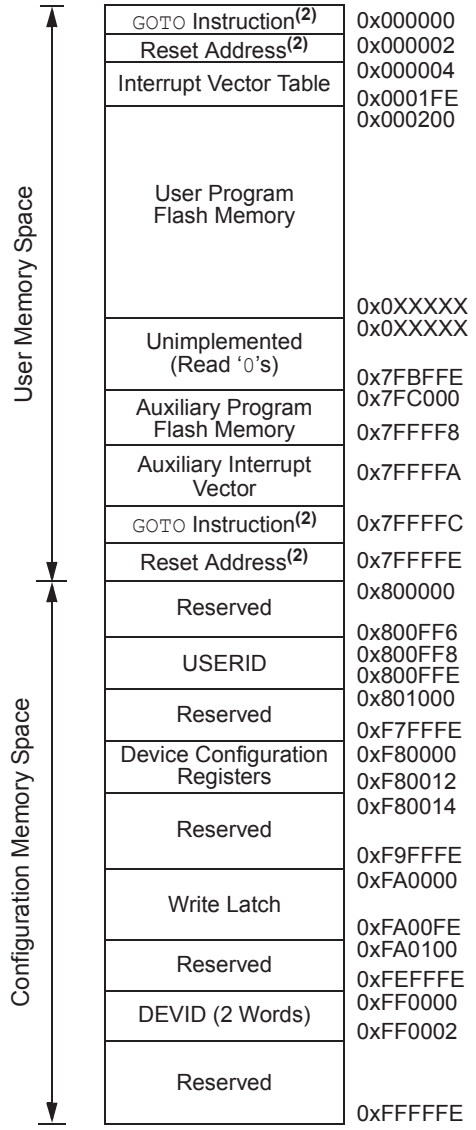
Figure 1-1: dsPIC33/PIC24 Program Memory Map for Devices without Auxiliary Memory



Note 1: Memory areas are not shown to scale.

2: This memory map is for reference only. Refer to the **"Memory Organization"** chapter of the specific device data sheet for exact memory addresses.

Figure 1-2: dsPIC33/PIC24 Program Memory Map for Devices with Auxiliary Memory



Note 1: Memory areas are not shown to scale.

2: Reset location is controlled by the Reset Target Vector Select bit (RSTPRI). Refer to the **"Special Features"** chapter of the specific device data sheet for more information.

3: This memory map is for reference only. Refer to the **"Memory Organization"** chapter in the specific device data sheet for exact memory addresses.

2.0 CONTROL REGISTERS

There are two registers that can be used to manage the program Flash:

- TBLPAG: Table Page Register
- DSRPAG: Data Space Read Page Register

Register 2-1: TBLPAG: Table Page Register

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

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TBLPAG<7:0>							
bit 7							bit 0

Legend:			
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 **TBLPAG<7:0>:** Table Page Address bits

The 8-bit Table Address Page bits are concatenated with the W register to form a 23-bit effective program memory address plus a Byte Select bit.

Register 2-2: DSRPAG: Data Space Read Page Register(1,2,3)

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0
—	—	—	—	—	—	DSRPAG<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1
DSRPAG<7:0>							
bit 7							bit 0

Legend:			
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 15-10 **Unimplemented:** Read as '0'

bit 9-0 **DSRPAG<9:0>:** Data Space Read Page Pointer bits

Note 1: When DSRPAG = 0x000, attempts to read from the paged Data Space (DS) window will cause an address error trap.

2: DSRPAG is reset to 0x001.

3: The Program Space (PS) can be read using DSRPAG values of 0x200 or greater.

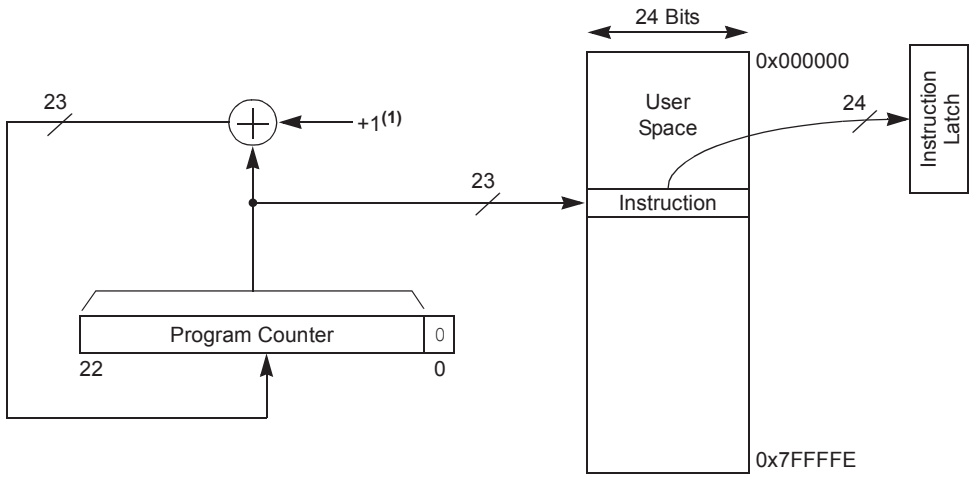
3.0 PROGRAM COUNTER

The PC increments by two with the Least Significant bit (LSb) set to '0' to provide compatibility with Data Space Addressing. Sequential instruction words are addressed in the 4M program memory space by PC<22:1>. Each instruction word is 24 bits wide.

The LSb of the program memory address (PC<0>) is reserved as a Byte Select bit for program memory accesses, from Data Space, that use Program Space Visibility (PSV) or table instructions. For instruction fetches via the PC, the Byte Select bit is not required, so PC<0> is always set to '0'. For more information on the PSV mode of operation, see **Section 5.0 "Program Space Visibility from Data Space"**.

Figure 3-1 illustrates an instruction fetch example. Note that incrementing PC<22:1> by one is equivalent to adding two to PC<22:0>.

Figure 3-1: Instruction Fetch Example



Note 1: An increment of one to PC<22:1> is equivalent to PC<22:0> + 2.

4.0 READING PROGRAM MEMORY USING TABLE INSTRUCTIONS

The Table Read instruction offers a direct method of reading the least significant word (lsw) and the Most Significant Byte (MSB) of any instruction word, within Program Space, without going through Data Space, which is preferable for some applications. For information on programming Flash memory, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Flash Programming"** (DS70000609), which is available from the Microchip website (www.microchip.com).

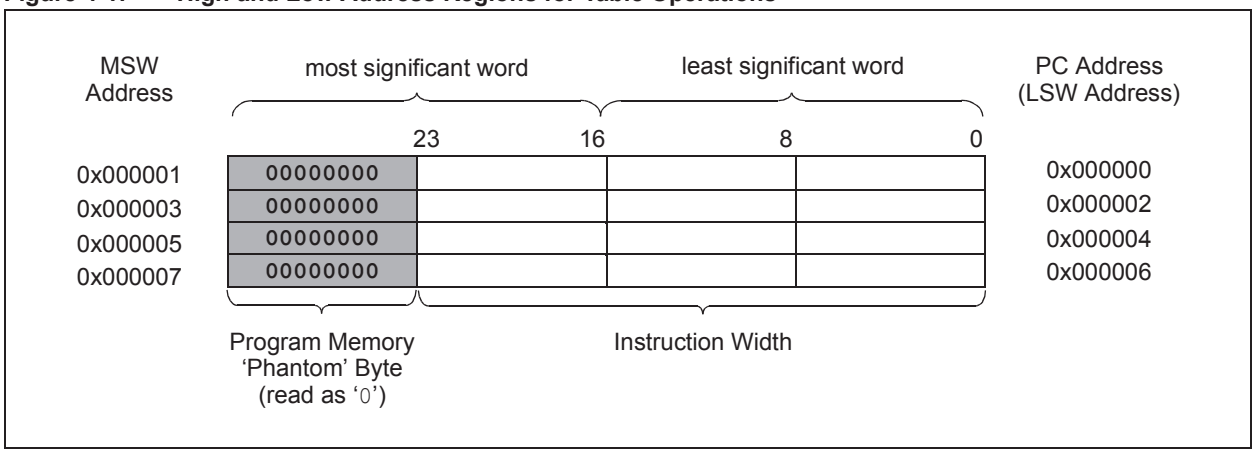
4.1 Table Instruction Summary

A set of table instructions is provided to move byte-sized or word-sized data between Program Space and Data Space. The Table Read instructions, in conjunction with the TBLPAG register, are used to read from the program memory space into data memory space. There are two Table Read instructions: `TBLRDL` (Table Read Low) and `TBLRDH` (Table Read High).

For table instructions, program memory can be regarded as two 16-bit, word-wide address spaces, residing side by side, each with the same address range (as illustrated in Figure 4-1). This allows Program Space to be accessed as byte or aligned word-addressable, 16-bit wide, 64-Kbyte pages (i.e., same as Data Space).

The `TBLRDL` instruction accesses the least significant data word of the program memory and `TBLRDH` accesses the upper word. Because program memory is only 24 bits wide, the upper byte from this latter space does not exist, although it is addressable; it is therefore termed, the "phantom" byte.

Figure 4-1: High and Low Address Regions for Table Operations



4.2 Table Address Generation

Figure 4-2 illustrates how for all table instructions, a W register address value is concatenated with the 8-bit Table Page (TBLPAG) register to form a 24-bit effective Program Space address, including a Byte Select bit (bit 0). Because there are 16 bits of Program Space address provided from the W register, the data table page size in program memory is 32K words. Figure 4-3 shows the organization of the table pages in the Program Space.

Note: In the event of an overflow or underflow, the Effective Address (EA) will wrap to the beginning of the current page.

Figure 4-2: Address Generation for Table Operations

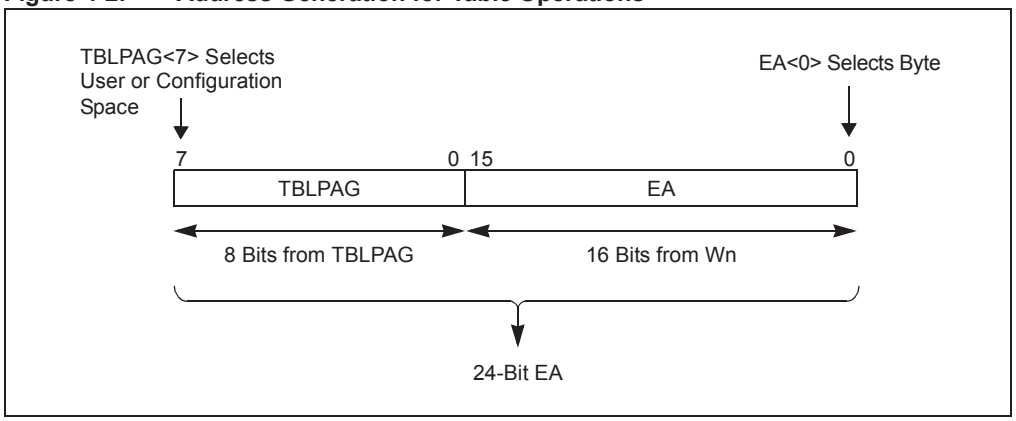
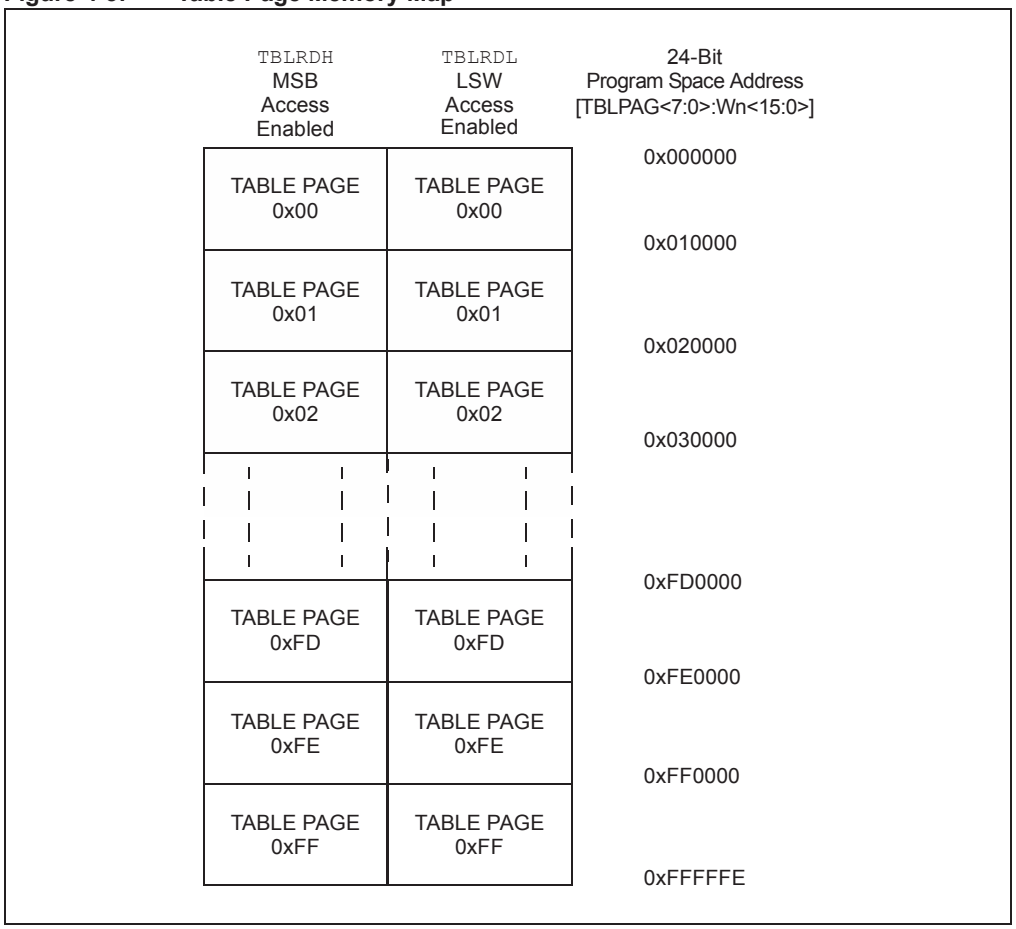


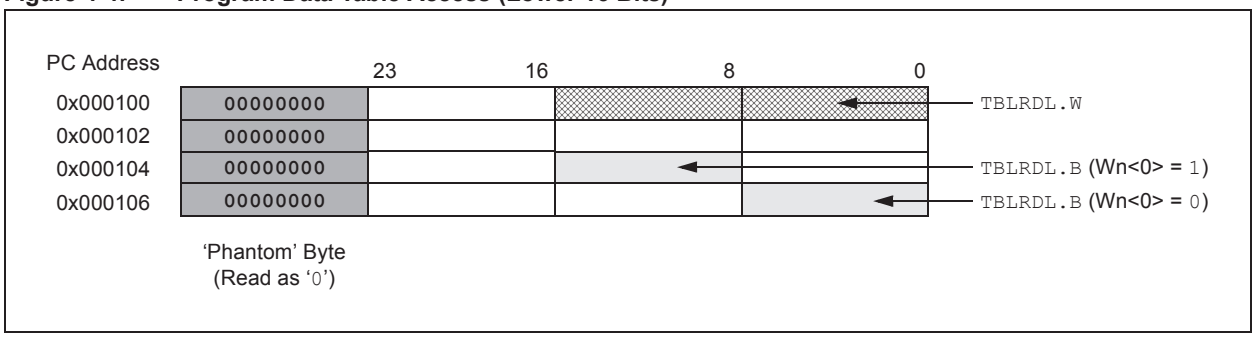
Figure 4-3: Table Page Memory Map



4.3 Program Memory Low Word Access

The TBLRDL instruction is used to access the lower 16 bits of program memory data. The LSb of the W register, which is used as a pointer, is ignored for word-wide table accesses. For byte-wide accesses, the LSb of the W register address determines which byte is read. Figure 4-4 demonstrates the program memory data regions accessed by the TBLRDL instruction.

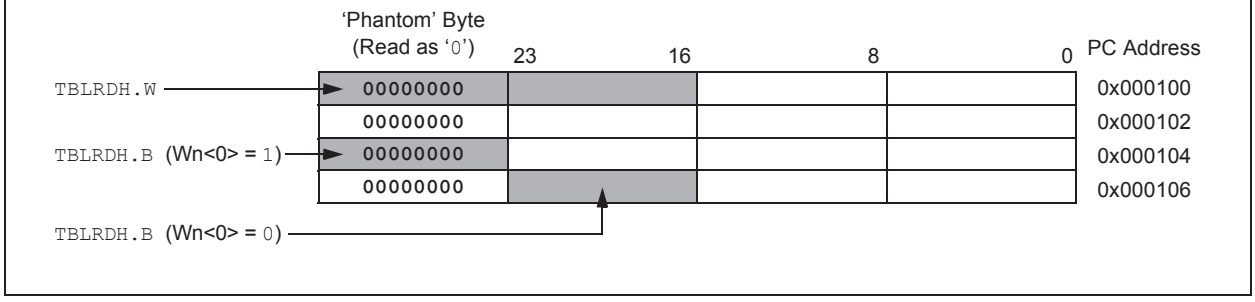
Figure 4-4: Program Data Table Access (Lower 16 Bits)



4.4 Program Memory High Word Access

The TBLRDH instruction is used to access the upper eight bits of the program memory data. Figure 4-5 illustrates how these instructions also support Word or Byte Access modes for orthogonality, but the high byte of the program memory data will always return '0'.

Figure 4-5: Program Data Table Access (Upper 8 Bits)



4.5 Accessing Program Memory Using Table Instructions

In Example 4-1, table instructions are used to access the program memory using an assembly language subroutine. In Example 4-2, program memory is accessed using the built-in functions, `__builtin_tblpage` and `__builtin_tbloffset`, that are provided by the MPLAB® XC16 C compiler.

Example 4-2 uses the `space(prog)` attribute to allocate the buffer in program memory. The MPLAB XC16 Compiler also has built-in functions, such as `__builtin_tblpage` and `__builtin_tbloffset`, that can be used to access the buffer. For more information, refer to the *"MPLAB XC16 C Compiler User's Guide"* (DS50002071).

Example 4-1: Using Table Instructions to Access Program Memory

```
extern   long  MemRead (unsigned int TablePage, unsigned int TableOffset);
unsigned long  Data1, Data2, Data3;

int      main(void)
{
    /* Read Configuration Register addresses 0xF80000 and 0xF80002 */
    Data1 = MemRead (0xF8, 0x0006);
    Data2 = MemRead (0xF8, 0x0008);
    Data3 = MemRead (0xF8, 0x000A);

    while(1);
}

.section .text
.global _MemRead

;************************
; Function _MemRead:
;
; W0 = TBLPAG value
; W1 = Table Offset
; Return: Data in W1:W0
;************************
_MemRead:
    MOV      W0, TBLPAG
    NOP
    TBLRDL   [W1], W0
    TBLRDH   [W1], W1
    RETURN
```

Example 4-2: Using MPLAB® XC16 C Compiler to Access Program Memory

```
int  prog_data[10] __attribute__((space(prog))) = {0x0000, 0x1111, 0x2222,
0x3333, 0x4444, 0x5555, 0x6666, 0x7777, 0x8888, 0x9999};

unsigned   int lowWord[10], highWord[10];
unsigned   int tableOffset, loopCount;

int  main(void)
{
     TBLPAG = __builtin_tblpage (prog_data);
     tableOffset = __builtin_tbloffset (prog_data);

     /* Read all 10 constants into the lowWord and highWord arrays */
     for (loopCount = 0; loopCount < 10; loopCount ++)
     {
        lowWord[loopCount] = __builtin_tblrdl (tableOffset);
        highWord[loopCount] = __builtin_tblrdh (tableOffset);
        tableOffset +=2;
     }

     while(1);
}
```

5.0 PROGRAM SPACE VISIBILITY FROM DATA SPACE

The upper 32 Kbytes of the dsPIC33/PIC24 data memory address space can optionally be mapped into any 16K word Program Space page. The PSV mode of operation provides transparent access of stored constant data from X Data Space without the need to use special instructions (i.e., `TBLRD`, `TBLWT` instructions).

5.1 PSV Configuration

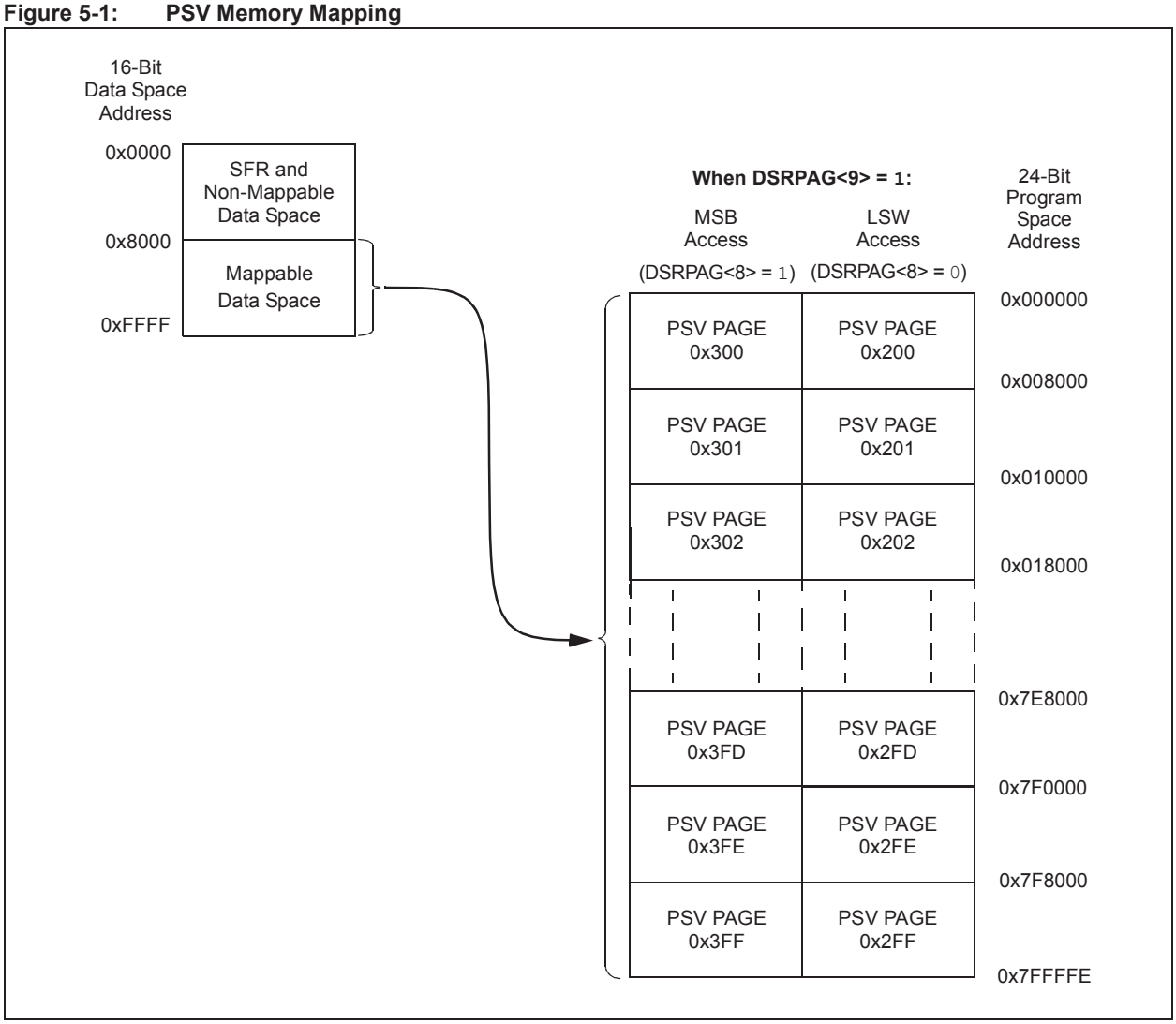
The dsPIC33/PIC24 core extends the available Data Space through a paging scheme to make it appear linear for pre-modified and post-modified Effective Addresses.

The upper half of the base Data Space address (0x8000 to 0xFFFF) is used with the 10-bit Data Space Read Page (DSRPAG) register to form a PSV address and can address eight Mbytes of PSV address space. The paged memory scheme provides access to multiple 32-Kbyte windows in the PSV memory. The PSV in the paged data memory space is illustrated in Figure 5-1.

Program Space (PS) can be read with a DSRPAG register of 0x200 or greater. Reads from PS are supported using the DSRPAG register. Writes to PS are not supported; therefore, the Data Space Write Page (DSWPAG) register is dedicated exclusively to Data Space (DS), including Extended Data Space (EDS).

For more information on the paged memory scheme, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Data Memory"** (DS70595).

Figure 5-1: PSV Memory Mapping

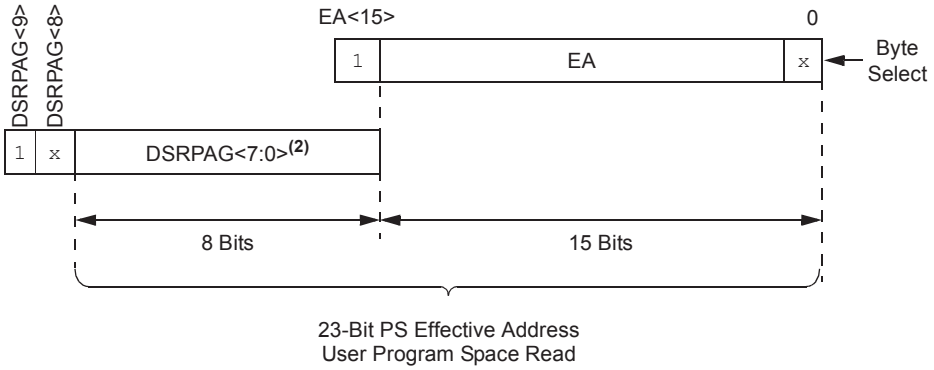


5.1.1 PSV ADDRESS GENERATION

Allocating different Page registers for read and write access allows the architecture to support data movement from different PSV pages to EDS pages, by configuring DSRPAG and DSWPAG to address PSV and EDS space, respectively. The data can be moved from PSV to EDS space by a single instruction.

Figure 5-2 illustrates the generation of the PSV address. The 15 Least Significant bits (LSbs) of the PSV address are provided by the W register that contains the Effective Address. The Most Significant bit (MSb) of the W register is not used to form the address. Instead, the MSb specifies whether to perform a PSV access from program memory space or a normal access from the data memory space. If the Effective Address of the W register is 0x8000 or greater, the data access will occur from program memory space, depending on the page selected by the DSRPAG register. All data access occurs from the data memory when the Effective Address of the W register is less than 0x8000.

Figure 5-2: PSV Address Generation



EA<15>	DSRPAG		Operation and Target
	<9>	<8>	
1	1	1	PSV, Upper Word (MSB)(1)
1	1	0	PSV, Lower Word (lsw) (byte selected by EA<0>)(1)
1	0	x	EDS Read (uses DSPAG<8:0> for a 24-bit EA)
0	x	x	Data Memory Space

Note 1: PSV access is only valid for DSPAG<7:0> values for 0x200 to 0x3FF, inclusive.

The remaining address bits are provided by the eight LSbs of the Data Space Read Page register (DSRPAG<7:0>). The DSRPAG<7:0> bits are concatenated with the 15 LSbs of the W register holding the Effective Address, and the MSb is forced to '0', thereby forming a 24-bit program memory address.

> **Note:** PSV can only be used to access values in the program memory space. Table instructions must be used to access values in the user configuration space.

The LSb of the W register value is used as a Byte Select bit, which allows instructions using PSV to operate in Byte or Word mode.

The PSV address is split into lsw and MSB. When DSRPAG<9:8> = 10, the lsw 16 bits of the 24-bit PS word can be accessed using PSV. When DSRPAG<9:8> = 11, the MSB of the 24-bit PS word can be accessed using PSV. The range of valid DSRPAG values for a lsw read starts at DSRPAG = 0x200 and the range of valid DSRPAG values for a MSB read starts at DSRPAG = 0x300.

5.2 PSV Timing

All instructions that use PSV require five instruction cycles to complete execution.

5.2.1 USING PSV IN A REPEAT LOOP

Instructions that use PSV with Indirect Addressing mode, using a post-modification offset of +2 or -2 within a REPEAT loop, eliminate some of the cycle count overhead required for the instruction access from program memory. These instructions have an effective execution throughput of one instruction cycle per iteration. However, the following iterations of the REPEAT loop will execute in five instruction cycles:

- First iteration
- Instruction execution prior to exiting the loop due to an interrupt
- Instruction execution upon re-entering the loop after an interrupt is serviced

The last iteration of the REPEAT loop will execute in six instruction cycles.

If the PSV Addressing mode uses an offset range other than +2 or -2 within a REPEAT loop, five instruction cycles are needed to execute each iteration of the loop.

Note: Unlike PSV accesses, a TBLRDL/H instruction requires five instruction cycles for each iteration.

5.2.2 PSV AND INSTRUCTION STALLS

For more information about instruction Stalls using PSV, refer to the appropriate *"dsPIC33/PIC24 Family Reference Manual"* chapter, **"CPU"** or **"dsPIC33 Enhanced CPU"** (DS70359 or DS70005158, respectively) specified in the device data sheet.

5.3 PSV Code Examples

Example 5-1 illustrates how to create a buffer and access the buffer in the compiler-managed PSV section. The auto_psv space is the compiler-managed PSV section. Sections greater than 32K are allowed and automatically managed. By default, the compiler places all const qualified variables into the auto_psv space.

When auto_psv is used, the compiler will save/restore the DSRPAG register dynamically, as needed. The tool chain will arrange for the DSRPAG to be correctly initialized in the compiler run-time start-up code.

Example 5-1: Compiler-Managed PSV Access

```
const int m[5] __attribute__((space(auto_psv))) = {1, 2, 3, 4, 5};
int x[5] = {10, 20, 30, 40, 50};
int   sum;

int   vectordot (int *, int *);

int   main(void)
{
      // Compiler-managed PSV
      sum = vectordot ((int *) m, x);

      while(1);
}

int   vectordot (int *m, int *x)
{
      int i, sum = 0;
      for (i = 0; i < 5; i ++)
           sum += (*m++) * (*x++);
      return (sum);
}
```

Note: The auto_psv option must be used if the user application is using both PSV and EDS accesses on a device with more than 28 Kbytes of RAM.

Example 5-2 illustrates buffer placement and access in the user-managed PSV section. The `psv space` is the user-managed PSV section. Example 5-3 illustrates the placement of constant data in program memory and accesses this data through the PSV data window using an assembly program.

Example 5-2: User-Managed PSV Access

```
const int m[5] = {1, 2, 3, 4, 5};
const int m1[5] __attribute__ ((space(psv))) = {2, 4, 6, 8, 10};
const int m2[5] __attribute__ ((space(psv))) = {3, 6, 9, 12, 15};
int   x[5] = {10, 20, 30, 40, 50};
int   sum, sum1, sum2;

int   vectordot (int *, int *);

int   main(void)
{
      int temp;
      temp = DSRPAG;                        // Save original PSV page value
      DSRPAG = __builtin_psvpage (m1);
      sum1 = vectordot ((int *) m1, x);
      DSRPAG = __builtin_psvpage (m2);
      sum2 = vectordot ((int *) m2, x);
      DSRPAG = temp;                        // Restore original PSV page value
      sum = vectordot ((int *) m, x);
      while(1);
}

int   vectordot (int *m, int *x)
{
      int i, sum = 0;
           for (i = 0; i < 5; i ++)
           sum += (*m++) * (*x++);
      return (sum);
}
```

Example 5-3: PSV Code Example in Assembly

```
    .section .const, psv
fib_data:
    .word 0, 1, 2, 3, 5, 8, 13

    ; Start of code section
    .text
    .global __main
__main:

    ; Set DSRPAG to the page that contains the “fib_data” array
    MOVPAG #psvpage(fib_data), DSRPAG
    ; Set up W0 as a pointer to “fib_data” through the PSV data window
    MOV    #psvoffset(fib_data), W0
    ; Load the data values into registers W1 - W7
    MOV    [W0++], W1
    MOV    [W0++], W2
    MOV    [W0++], W3
    MOV    [W0++], W4
    MOV    [W0++], W5
    MOV    [W0++], W6
    MOV    [W0++], W7

done:
    BRA done

    RETURN
```

6.0 PROGRAM MEMORY WRITES

There are two methods by which the user application can program Flash memory:

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

For more information on RTSP, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Flash Programming"** (DS70000609). For more information on ICSP, refer to the specific *"Flash Programming Specification"* for your device, which can be obtained from the Microchip website (www.microchip.com).

7.0 ERROR CORRECTING CODE (ECC)

In order to improve program memory performance and durability, select dsPIC33 and PIC24 devices include Error Correcting Code (ECC) functionality as an integral part of the Flash memory controller. ECC can determine the presence of single-bit errors in program data, including which bit is in error, and correct the data without user intervention. When implemented, ECC is automatic and cannot be disabled.

When data is written to program memory, ECC generates a 7-bit Hamming code parity value for every two (24-bit) instruction words. The data is stored in blocks of 48 data bits and seven parity bits; parity data is not memory-mapped and is inaccessible. When the data is read back, the ECC calculates parity on it and compares it to the previously stored parity value. If a parity mismatch occurs, there are two possible outcomes:

- Single-bit errors are automatically identified and corrected on read back. An optional device-level interrupt (ECCSBEIF) is also generated.
- Double-bit errors will generate a generic hard trap. If special exception handling for the trap is not implemented, a device Reset will also occur.

To use the single-bit error interrupt, set the ECC Single-Bit Error Interrupt Enable (ECCSBEIE) bit and configure the ECCSBEIPx bits to set the appropriate interrupt priority.

On some devices, Fault injection and additional error information are made available. Refer to the device-specific data sheet for availability. In addition to the ECCSBEIF flag, the ECCSTATL register contains the parity information for single-bit errors. The SECOUT<7:0> bits field contains the expected calculated SEC parity and the SECIN<7:0> bits contain the actual value from a Flash read operation. The SECSYNDx bits (ECCSTATH<7:0>) indicate the bit position of the single-bit error within the 48-bit pair of instruction words. When no error is present, SECINx equals SECOUTx and SECSYNDx is zero.

Double-bit errors result in a generic hard trap. The ECCDBE bit (INTCON4<1>) will be set to identify the source of the hard trap. If no Interrupt Service Routine is implemented for the hard trap, a device Reset will also occur. The ECCSTATH register contains double-bit error status information. The DEDOUT bit is the expected calculated DED parity and DEDIN is the actual value from a Flash read operation. When no error is present, DEDIN equals DEDOUT.

7.1 ECC Fault Injection

To test Fault handling, an EEC error can be generated. Both single and double-bit errors can be generated in both the read and write data paths. Read path Fault injection first reads the Flash data and then modifies it prior to entering the ECC logic. Write path Fault injection modifies the actual data prior to it being written into the target Flash and will cause an EEC error on a subsequent Flash read. The following procedure is used to inject a Fault:

1. Load the Flash target address into the ECCADDR register.
2. Select 1st Fault bit determined by FLT1PTRx (ECCCONH<7:0>). The target bit is inverted to create the Fault.
3. If a double Fault is desired, select the 2nd Fault bit determined by FLT2PTRx (ECCCONH<15:8>); otherwise, set to all '1's.
4. Write the NVMKEY unlock sequence (see specific device data sheet for sequence).
5. Enable the ECC Fault injection logic by setting the FLTINJ bit (ECCCONL<0>).
6. Perform a read or write to the Flash target address.

8.0 PROGRAM MEMORY LOW-POWER MODE

The voltage regulator for the program Flash memory can be placed in Standby mode when the device is in Sleep mode, resulting in a reduction in device Power-Down Current (IPD).

When the VREGSF bit (RCON<11>) is equal to '0', the Flash memory voltage regulator goes into Standby mode during Sleep. When the VREGSF bit is equal to '1', the Flash memory voltage regulator is active during Sleep mode; however, this mode increases the device wake-up delay.

9.0 REGISTER MAP

A summary of the registers associated with the dsPIC33/PIC24 Program Memory is provided in Table 9-1.

Table 9-1: CPU Core Register Map

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
PCL	Program Counter Low Register															—	0000
PCH	—	—	—	—	—	—	—	—	—	Program Counter High Register							0000
DSRPAG	—	—	—	—	—	—	DSRPAG<9:0>										0001
DSWPAG	—	—	—	—	—	—	—	DSWPAG<8:0>									0001
TBLPAG	—	—	—	—	—	—	—	—	TBLPAG<7:0>								0000
ECCONL	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	FLTINJ	0000
ECCONH	FLT2PTR<7:0>								FLT1PTR<7:0>								0000
ECCADDRL	ECCADDR<15:8>								ECCADDR<7:0>								0000
ECCADDRH	—	—	—	—	—	—	—	—	ECCADDR<23:16>								0000
ECCSTATL	SECOUT<7:0>								SECIN<7:0>								0000
ECCSTATH							DEDOUT	DEDIN	SECSYND<7:0>								0000

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

10.0 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 dsPIC33/PIC24 Product Families, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the dsPIC33/PIC24 Program Memory module are:

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

Note: For additional Application Notes and code examples for the dsPIC33/PIC24 families of devices, visit the Microchip website (www.microchip.com).

11.0 REVISION HISTORY

Revision A (September 2009)

This is the initial released version of this document.

Revision B (July 2010)

This revision includes the following updates:

- All code examples have been updated (see Example 4-1 through Example 5-3)
- Updated the Program Memory Map (see Figure 1-2)
- Updated the first paragraph and the shaded note in **Section 4.1 "Table Instruction Summary"**
- Added a shaded note after Figure 4-1 with information on writing to the TBLPAG register
- Updated **Section 4.2 "Table Address Generation"**
- Updated the second sentence in **Section 4.3 "Program Memory Low Word Access"**
- Added the new figure Table Page Memory Map (see Figure 4-3) in **Section 4.4 "Program Memory High Word Access"**
- Added a shaded note and updated the last paragraph in **Section 5.1 "PSV Configuration"**
- Updated the Paged Data Memory Space (see Figure 5-1)
- Updated the PSV Address Generation (see Figure 5-2)
- Changed the number of required instruction cycles from two to five throughout **Section 5.2 "PSV Timing"**
- Added a shaded note after Example 5-1 with information on using the `auto_psv` option
- Added a reference to the *"dsPIC33/PIC24 Flash Programming Specification"* (DS70619) to **Section 6.0 "Program Memory Writes"**

Revision C (December 2011)

This revision includes the following updates:

- Updated **Section 1.0 "Program Memory Address Map"**
- Updated the existing Program Memory Map for devices with auxiliary memory (see Figure 1-2)
- Added a new Program Memory Map for devices without auxiliary memory (see Figure 1-1)
- Updated Using Table Instructions to Access Program Memory (see Example 4-1)
- Updated Using MPLAB® C Compiler to Access Program Memory (see Example 4-2)
- Removed **4.4.5 "Data Storage in Program Memory"**
- Removed **4.5.2 "PSV Mapping with X and Y Data Space"**
- Updated Compiler-Managed PSV Access (see Example 5-1)
- Updated User-Managed PSV Access (see Example 5-2)
- Updated **Section 6.0 "Program Memory Writes"**
- Updated **Section 8.0 "Program Memory Low-Power Mode"**
- Updated the Register Map table (see Table 9-1)
- Minor updates to text and formatting were incorporated throughout the document

Revision D (November 2014)

Updates the document format and removes the previously assigned master section number as part of the realignment of dsPIC33 technical documentation. The document reference number format is also updated.

Updates the document title to “dsPIC33/PIC24 Program Memory” for clarity.

Adds **Section 7.0 “Error Correcting Code (ECC)”**. Subsequent sections are renumbered accordingly.

Updates **Section 1.0 “Program Memory Address Map”** to mention OTP locations, in addition to USERID locations

Reorganizes **Section 4.0 “Reading Program Memory Using Table Instructions”** to include the “Table Memory Map” (formerly Figure 4-5) with the text of **Section 4.2 “Table Address Generation”**, as Figure 4-3.

Updates **Section 5.0 “Program Space Visibility from Data Space”** by revising Figures 5-1 and 5-2 for clarity, and removing Figures 5-3 and 5-4 as redundant. Adds the subhead, **Section 5.1.1 “PSV Address Generation”** to delineate topics without changing the previously existing text.

Other minor changes to text and typographic changes throughout the document.

Revision E (October 2018)

Updates **Section 7.0 “Error Correcting Code (ECC)”**.

Updates Table 9-1.

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.

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.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Trademarks

ISBN: 978-1-5224-3699-7

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

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

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
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

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

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

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

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

Finland - Espoo
Tel: 358-9-4520-820

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

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

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

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

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

Italy - Padova
Tel: 39-049-7625286

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

Norway - Trondheim
Tel: 47-7288-4388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

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

Sweden - Gothenberg
Tel: 46-31-704-60-40

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

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