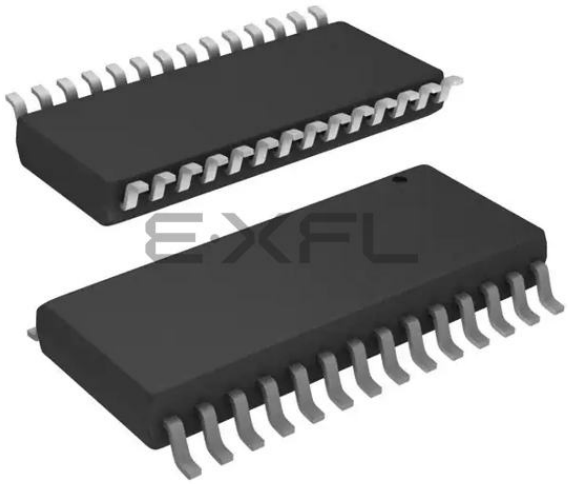


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

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

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	64MHz
Connectivity	I²C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, HLVD, POR, PWM, WDT
Number of I/O	25
Program Memory Size	32KB (16K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	2.3V ~ 5.5V
Data Converters	A/D 24x12b; D/A 1x5b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SOIC (0.295", 7.50mm Width)
Supplier Device Package	28-SOIC
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18f25k42-e-so

3.1 System Arbitration

The System Arbiter resolves memory access between the System Level Selections (i.e., Main, Interrupt Service Routine) and Peripheral Selection (i.e., DMA and Scanner) based on user-assigned priorities. Each of the system level and peripheral selections has its own priority selection registers. Memory access priority is resolved using the number written to the corresponding Priority registers, 0 being the highest priority and 4 being the lowest priority. The Default priorities are listed in Table 3-1.

In case the user wants to change priorities then ensure that each Priority register is written with a unique value from 0 to 4.

TABLE 3-1: DEFAULT PRIORITIES

Selection		Priority register Reset value
System Level	ISR	0
	MAIN	1

TABLE 3-1: DEFAULT PRIORITIES

Selection		Priority register Reset value
Peripheral	DMA1	2
	DMA2	3
	SCANNER	4

FIGURE 3-2: PIC18(L)F24/25K42 SYSTEM ARBITER BLOCK DIAGRAM

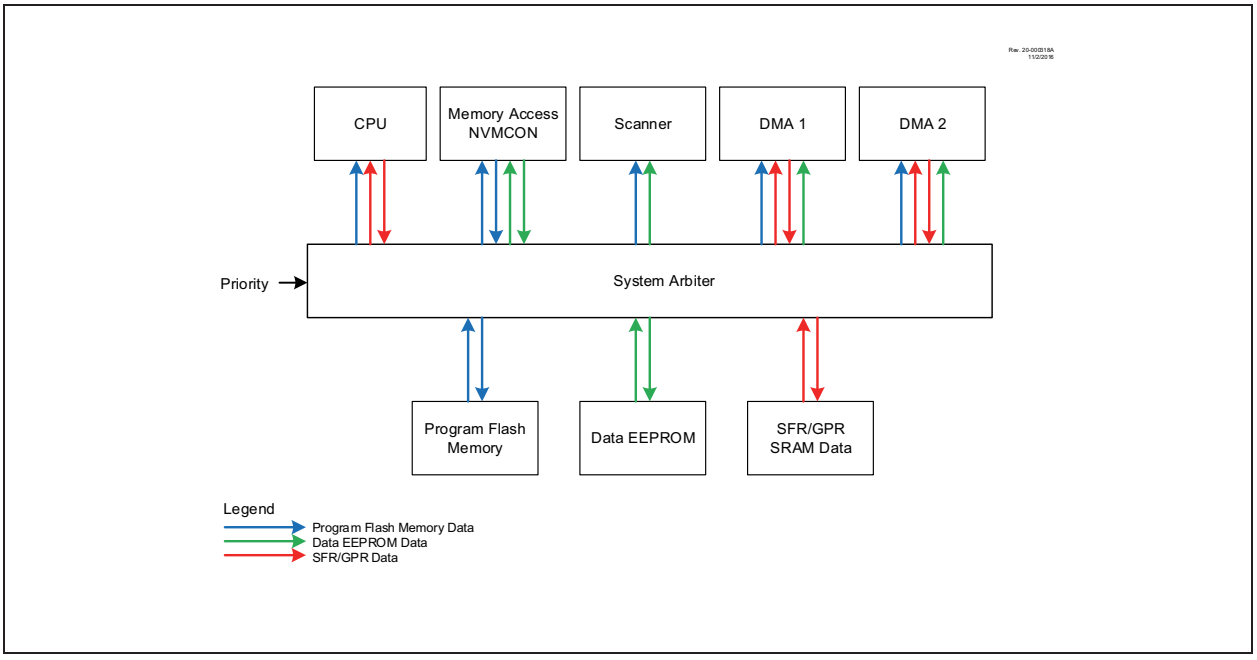


TABLE 3-2: SUMMARY OF REGISTERS ASSOCIATED WITH CPU

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on page
ISRPR	—	—	—	—	—	ISRPR2	ISRPR1	ISRPR0	21
MAINPR	—	—	—	—	—	MAINPR2	MAINPR1	MAINPR0	21
DMA1PR	—	—	—	—	—	DMA1PR2	DMA1PR1	DMA1PR0	21
DMA2PR	—	—	—	—	—	DMA2PR2	DMA2PR1	DMA2PR0	22
SCANPR	—	—	—	—	—	SCANPR2	SCANPR1	SCANPR0	22
PRLOCK	—	—	—	—	—	—	—	PRLOCKED	22

Legend: — = Unimplemented location, read as '0'.

4.5.2 ACCESS BANK

While the use of the BSR with an embedded 8-bit address allows users to address the entire range of data memory, it also means that the user must always ensure that the correct bank is selected. Otherwise, data may be read from or written to the wrong location. This can be disastrous if a GPR is the intended target of an operation, but an SFR is written to instead. Verifying and/or changing the BSR for each read or write to data memory can become very inefficient.

To streamline access for the most commonly used data memory locations, the data memory is configured with an Access Bank, which allows users to access a mapped block of memory without specifying a BSR. The Access Bank consists of the first 96 bytes of memory (00h-5Fh) in Bank 0 and the last 160 bytes of memory (60h-FFh) in Bank 63. The lower half is known as the “Access RAM” and is composed of GPRs. This upper half is also where some of the SFRs of the device are mapped. These two areas are mapped contiguously in the Access Bank and can be addressed in a linear fashion by an 8-bit address (Figure 4-4).

The Access Bank is used by core PIC18 instructions that include the Access RAM bit (the ‘a’ parameter in the instruction). When ‘a’ is equal to ‘1’, the instruction uses the BSR and the 8-bit address included in the opcode for the data memory address. When ‘a’ is ‘0’, however, the instruction is forced to use the Access Bank address map; the current value of the BSR is ignored entirely.

Using this “forced” addressing allows the instruction to operate on a data address in a single cycle, without updating the BSR first. For 8-bit addresses of 60h and above, this means that users can evaluate and operate on SFRs more efficiently. The Access RAM below 60h is a good place for data values that the user might need to access rapidly, such as immediate computational results or common program variables. Access RAM also allows for faster and more code efficient and switching of variables.

The mapping of the Access Bank is slightly different when the extended instruction set is enabled (XINST Configuration bit = 1). This is discussed in more detail in **Section 4.8.3 “Mapping the Access Bank in Indexed Literal Offset Mode”**.

4.5.3 GENERAL PURPOSE REGISTER FILE

PIC18 devices may have banked memory in the GPR area. This is data RAM, which is available for use by all instructions. GPRs start at the bottom of Bank 0 (address 0000h) and grow upwards towards the bottom of the SFR area. GPRs are not initialized by a Power-on Reset and are unchanged on all other Resets.

4.5.4 SPECIAL FUNCTION REGISTERS

The Special Function Registers (SFRs) are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. SFRs start at the top of data memory (3FFFh) and extend downward to occupy Bank 56 through 63 (3800h to 3FFFh). A list of these registers is given in Table 4-3 to Table 4-10.

The SFRs can be classified into two sets: those associated with the “core” device functionality (ALU, Resets and interrupts) and those related to the peripheral functions. The Reset and Interrupt registers are described in their respective chapters, while the ALU’s STATUS register is described later in this section. Registers related to the operation of a peripheral feature are described in the chapter for that peripheral.

The SFRs are typically distributed among the peripherals whose functions they control. Unused SFR locations are unimplemented and read as ‘0’s.

REGISTER 11-2: INTCON1: INTERRUPT CONTROL REGISTER 1

R-0/0	R-0/0	U-0	U-0	U-0	U-0	U-0	U-0
STAT<1:0>		—	—	—	—	—	—
bit 7							bit 0

Legend:		
HC = Bit is cleared by hardware		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	q = Value depends on condition

bit 7-6 **STAT<1:0>:** Interrupt State Status bits
11 =High priority ISR executing, high priority interrupt was received while a low priority ISR was executing
10 =High priority ISR executing, high priority interrupt was received in main routine
01 =Low priority ISR executing, low priority interrupt was received in main routine
00 =Main routine executing

bit 5-0 **Unimplemented**: Read as ‘0’

REGISTER 11-11: PIR8: PERIPHERAL INTERRUPT REGISTER 8

R/W/HS-0/0	R/W/HS-0/0	U-0	U-0	U-0	U-0	U-0	U-0
TMR5GIF	TMR5IF	—	—	—	—	—	—
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	HS = Bit is set in hardware

bit 7 **TMR5GIF:** TMR5 Gate Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
0 = Interrupt event has not occurred

bit 6 **TMR5IF:** TMR5 Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
0 = Interrupt event has not occurred

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

Note: Interrupt flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding enable bit, or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

REGISTER 11-12: PIR9: PERIPHERAL INTERRUPT REGISTER 9

U-0	U-0	U-0	U-0	R/W/HS-0/0	R/W/HS-0/0	R/W/HS-0/0	R/W/HS-0/0
—	—	—	—	CLC3IF	CWG3IF	CCP3IF	TMR6IF
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

bit 7-4 **Unimplemented:** Read as '0'

bit 3 **CLC3IF:** CLC3 Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
0 = Interrupt event has not occurred

bit 2 **CWG3IF:** CWG3 Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
0 = Interrupt event has not occurred

bit 1 **CCP3IF:** CCP3 Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
0 = Interrupt event has not occurred

bit 0 **TMR6IF:** TMR6 Interrupt Flag bit
1 = Interrupt has occurred (must be cleared by software)
0 = Interrupt event has not occurred

Note: Interrupt flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding enable bit, or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

REGISTER 13-3: WDTPSL: WWDT PRESCALE SELECT LOW BYTE REGISTER (READ-ONLY)

R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0
PSCNT<7:0>							
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	

bit 7-0 **PSCNT<7:0>:** Prescale Select Low Byte bits(1)

Note 1: The 18-bit WDT prescale value, PSCNT<17:0> includes the WDTPSL, WDTPSH and the lower bits of the WDTTMR registers. PSCNT<17:0> is intended for debug operations and should not be read during normal operation.

REGISTER 13-4: WDTPSH: WWDT PRESCALE SELECT HIGH BYTE REGISTER (READ-ONLY)

R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0
PSCNT<15:8>							
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	

bit 7-0 **PSCNT<15:8>:** Prescale Select High Byte bits(1)

Note 1: The 18-bit WDT prescale value, PSCNT<17:0> includes the WDTPSL, WDTPSH and the lower bits of the WDTTMR registers. PSCNT<17:0> is intended for debug operations and should not be read during normal operation.

REGISTER 13-5: WDTTMR: WDT TIMER REGISTER (READ-ONLY)

R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0	R-0/0
WDTTMR<4:0>					STATE	PSCNT<17:16>	
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

bit 7-3 **WDTTMR<4:0>:** Watchdog Window Value bits

WINDOW	WDT Window State		Open Percent
	Closed	Open	
111	N/A	00000-11111	100
110	00000-00011	00100-11111	87.5
101	00000-00111	01000-11111	75
100	00000-01011	01100-11111	62.5
011	00000-01111	10000-11111	50
010	00000-10011	10100-11111	37.5
001	00000-10111	11000-11111	25
000	00000-11011	11100-11111	12.5

bit 2 **STATE:** WDT Armed Status bit
1 = WDT is armed
0 = WDT is not armed

bit 1-0 **PSCNT<17:16>:** Prescale Select Upper Byte bits(1)

Note 1: The 18-bit WDT prescale value, PSCNT<17:0> includes the WDTPSL, WDTPSH and the lower bits of the WDTTMR registers. PSCNT<17:0> is intended for debug operations and should not be read during normal operation.

Example 14-3 shows the sequence to do a 16 x 16 unsigned multiplication. Equation 14-1 shows the algorithm that is used. The 32-bit result is stored in four registers (RES<3:0>).

EQUATION 14-1: 16 x 16 UNSIGNED MULTIPLICATION ALGORITHM

$$
\begin{aligned}
RES3:RES0 &= ARG1H:ARG1L \bullet ARG2H:ARG2L \\
&= (ARG1H \bullet ARG2H \bullet 2^{16}) + \\
&\quad (ARG1H \bullet ARG2L \bullet 2^{8}) + \\
&\quad (ARG1L \bullet ARG2H \bullet 2^{8}) + \\
&\quad (ARG1L \bullet ARG2L)
\end{aligned}
$$

EXAMPLE 14-3: 16 x 16 UNSIGNED MULTIPLY ROUTINE

```
    MOVF    ARG1L, W
    MULWF   ARG2L           ; ARG1L * ARG2L->
                            ; PRODH:PRODL
    MOVFF   PRODH, RES1     ;
    MOVFF   PRODL, RES0     ;
;
    MOVF    ARG1H, W
    MULWF   ARG2H           ; ARG1H * ARG2H->
                            ; PRODH:PRODL
    MOVFF   PRODH, RES3     ;
    MOVFF   PRODL, RES2     ;
;
    MOVF    ARG1L, W
    MULWF   ARG2H           ; ARG1L * ARG2H->
                            ; PRODH:PRODL
    MOVF    PRODL, W        ;
    ADDWF   RES1, F         ; Add cross
    MOVF    PRODH, W        ; products
    ADDWFC  RES2, F         ;
    CLRF    WREG            ;
    ADDWFC  RES3, F         ;
;
    MOVF    ARG1H, W        ;
    MULWF   ARG2L           ; ARG1H * ARG2L->
                            ; PRODH:PRODL
    MOVF    PRODL, W        ;
    ADDWF   RES1, F         ; Add cross
    MOVF    PRODH, W        ; products
    ADDWFC  RES2, F         ;
    CLRF    WREG            ;
    ADDWFC  RES3, F         ;
```

Example 14-4 shows the sequence to do a 16 x 16 signed multiply. Equation 14-2 shows the algorithm used. The 32-bit result is stored in four registers (RES<3:0>). To account for the sign bits of the arguments, the MSb for each argument pair is tested and the appropriate subtractions are done.

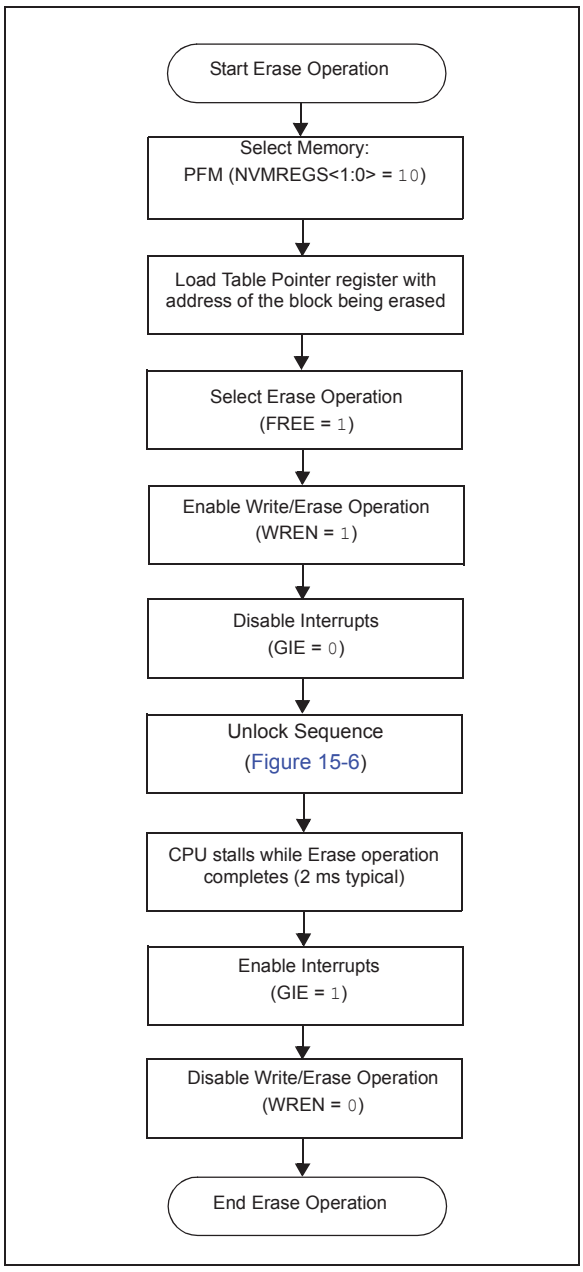
EQUATION 14-2: 16 x 16 SIGNED MULTIPLICATION ALGORITHM

$$
\begin{aligned}
RES3:RES0 &= ARG1H:ARG1L \bullet ARG2H:ARG2L \\
&= (ARG1H \bullet ARG2H \bullet 2^{16}) + \\
&\quad (ARG1H \bullet ARG2L \bullet 2^{8}) + \\
&\quad (ARG1L \bullet ARG2H \bullet 2^{8}) + \\
&\quad (ARG1L \bullet ARG2L) + \\
&\quad (-1 \bullet ARG2H\langle 7\rangle \bullet ARG1H:ARG1L \bullet 2^{16}) + \\
&\quad (-1 \bullet ARG1H\langle 7\rangle \bullet ARG2H:ARG2L \bullet 2^{16})
\end{aligned}
$$

EXAMPLE 14-4: 16 x 16 SIGNED MULTIPLY ROUTINE

```
    MOVF    ARG1L, W
    MULWF   ARG2L           ; ARG1L * ARG2L ->
                            ; PRODH:PRODL
    MOVFF   PRODH, RES1     ;
    MOVFF   PRODL, RES0     ;
;
    MOVF    ARG1H, W
    MULWF   ARG2H           ; ARG1H * ARG2H ->
                            ; PRODH:PRODL
    MOVFF   PRODH, RES3     ;
    MOVFF   PRODL, RES2     ;
;
    MOVF    ARG1L, W
    MULWF   ARG2H           ; ARG1L * ARG2H ->
                            ; PRODH:PRODL
    MOVF    PRODL, W        ;
    ADDWF   RES1, F         ; Add cross
    MOVF    PRODH, W        ; products
    ADDWFC  RES2, F         ;
    CLRF    WREG            ;
    ADDWFC  RES3, F         ;
;
    MOVF    ARG1H, W        ;
    MULWF   ARG2L           ; ARG1H * ARG2L ->
                            ; PRODH:PRODL
    MOVF    PRODL, W        ;
    ADDWF   RES1, F         ; Add cross
    MOVF    PRODH, W        ; products
    ADDWFC  RES2, F         ;
    CLRF    WREG            ;
    ADDWFC  RES3, F         ;
;
    BTFSS   ARG2H, 7        ; ARG2H:ARG2L neg?
    BRA     SIGN_ARG1       ; no, check ARG1
    MOVF    ARG1L, W        ;
    SUBWF   RES2            ;
    MOVF    ARG1H, W        ;
    SUBWFB  RES3
;
SIGN_ARG1
    BTFSS   ARG1H, 7        ; ARG1H:ARG1L neg?
    BRA     CONT_CODE       ; no, done
    MOVF    ARG2L, W        ;
    SUBWF   RES2            ;
    MOVF    ARG2H, W        ;
    SUBWFB  RES3
;
CONT_CODE
    :
```

FIGURE 15-7: PFM ROW ERASE FLOWCHART



15.1.6 WRITING TO PROGRAM FLASH MEMORY

The programming write block size is described in Table 15-2. Word or byte programming is not supported. Table writes are used internally to load the holding registers needed to program the memory. There are only as many holding registers as there are bytes in a write block. Refer to Table 15-2 for write latch size.

Since the table latch (TABLAT) is only a single byte, the TBLWT instruction needs to be executed multiple times for each programming operation. The write protection state is ignored for this operation. All of the table write operations will essentially be short writes because only the holding registers are written. NVMIF is not affected while writing to the holding registers.

After all the holding registers have been written, the programming operation of that block of memory is started by configuring the NVMCON1 register for a program memory write and performing the long write sequence.

If the PFM address in the TBLPTR is write-protected or if TBLPTR points to an invalid location, the WR bit is cleared without any effect and the WREER is signaled.

The long write is necessary for programming the program memory. CPU operation is suspended during a long write cycle and resumes when the operation is complete. The long write operation completes in one instruction cycle. When complete, WR is cleared in hardware and NVMIF is set and an interrupt will occur if NVMIE is also set. The latched data is reset to all '1s'. WREN is not changed.

The internal programming timer controls the write time. The write/erase voltages are generated by an on-chip charge pump, rated to operate over the voltage range of the device.

Note:	The default value of the holding registers on device Resets and after write operations is FFh. A write of FFh to a holding register does not modify that byte. This means that individual bytes of program memory may be modified, provided that the change does not attempt to change any bit from a '0' to a '1'. When modifying individual bytes, it is not necessary to load all holding registers before executing a long write operation.

15.3.5 WRITE VERIFY

Depending on the application, good programming practice may dictate that the value written to the memory should be verified against the original value. This should be used in applications where excessive writes can stress bits near the specification limit.

EXAMPLE 15-5: DATA EEPROM READ

```
; Data Memory Address to read
            CLRF      NVMCON1          ; Setup Data EEPROM Access
            MOVF      EE_ADDRL, W      ;
            MOVWF     NVMADRL          ; Setup Address
            BSF       NVMCON1, RD      ; Issue EE Read
            MOVF      NVMDAT, W        ; W = EE_DATA
```

EXAMPLE 15-6: DATA EEPROM WRITE

```
; Data Memory Address to write
            CLRF      NVMCON1          ; Setup Data EEPROM Access
            MOVF      EE_ADDRL, W      ;
            MOVWF     NVMADRL          ; Setup Address
; Data Memory Value to write
            MOVF      EE_DATA, W       ;
            MOVWF     NVMDAT           ;
; Enable writes
            BSF       NVMCON1, WREN    ;
; Disable interrupts
            BCF       INTCON0, GIE     ;
; Required unlock sequence
            MOVLW     55h              ;
            MOVWF     NVMCON2          ;
            MOVLW     AAh              ;
            MOVWF     NVMCON2          ;
; Set WR bit to begin write
            BSF       NVMCON1, WR      ;
; Enable INT
            BSF       INTCON0, GIE     ;
; Wait for interrupt, write done
            SLEEP                      ;
; Disable writes
            BCF       NVMCON1, WREN    ;
```

15.3.6 OPERATION DURING CODE-PROTECT

Data EEPROM Memory has its own code-protect bits in Configuration Words. External read and write operations are disabled if code protection is enabled.

If the Data EEPROM is write-protected or if NVMADR points an invalid address location, the WR bit is cleared without any effect. WRERR is signaled in this scenario.

15.3.7 PROTECTION AGAINST SPURIOUS WRITE

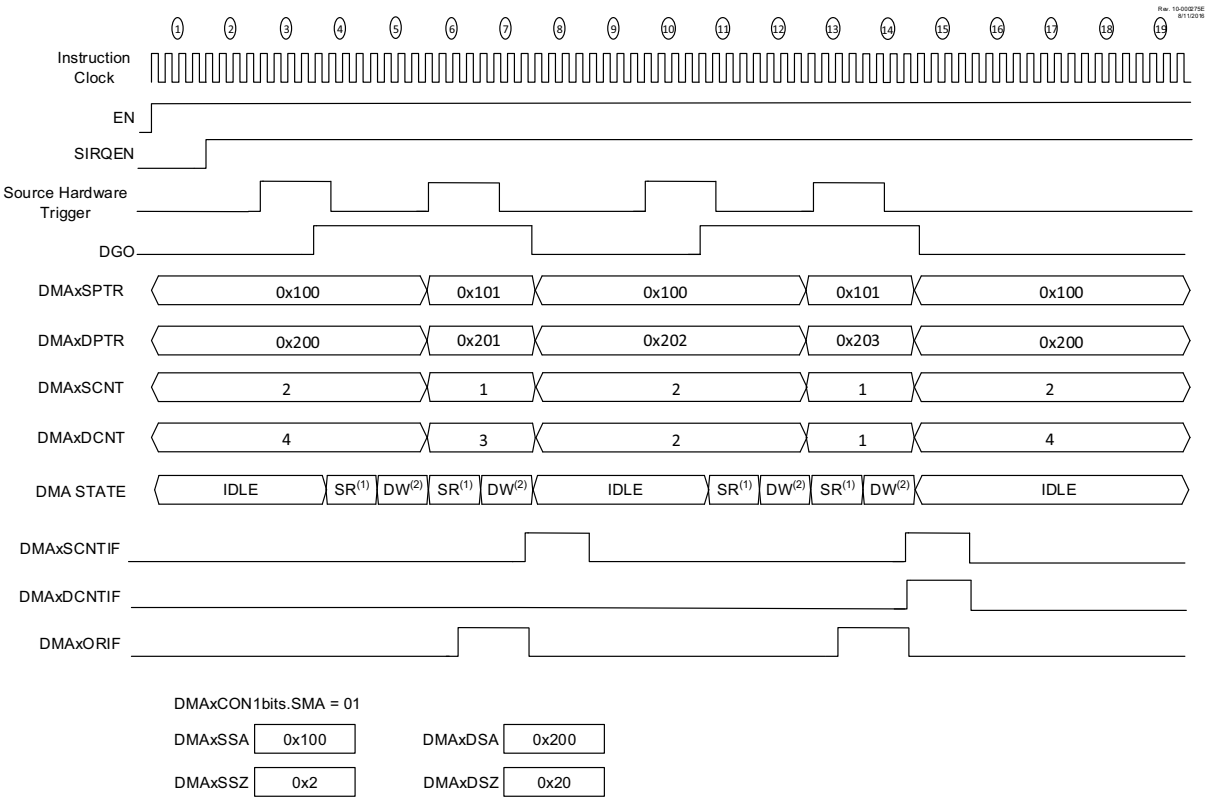
There are conditions when the user may not want to write to the Data EEPROM Memory. To protect against spurious EEPROM writes, various mechanisms have been implemented. On power-up, the WREN bit is cleared. In addition, writes to the EEPROM are blocked during the Power-up Timer period (TPWRT).

The unlock sequence and the WREN bit together help prevent an accidental write during brown-out, power glitch or software malfunction.

17.9.5 OVERRUN INTERRUPT

The Overrun Interrupt flag is set if the DMA receives a trigger to start a new message before the current message is completed.

FIGURE 17-9: OVERRUN INTERRUPT



Note 1: SR - Source Read

2: DW - Destination Write

REGISTER 17-12: DMAxSCNTL – DMAx SOURCE COUNT LOW REGISTER

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
SCNT<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n/n = Value at POR and BOR/Value at all other Resets	1 = bit is set	0 = bit is cleared	x = bit is unknown u = bit is unchanged

bit 7-0 **SCNT<7:0>:** Current Source Byte Count

REGISTER 17-13: DMAxSCNTH – DMAx SOURCE COUNT HIGH REGISTER

U-0	U-0	U-0	U-0	R-0	R-0	R-0	R-0
—	—	—	—	SCNT<11:8>			
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n/n = Value at POR and BOR/Value at all other Resets	1 = bit is set	0 = bit is cleared	x = bit is unknown u = bit is unchanged

bit 7-4 **Unimplemented**: Read as '0'

bit 3-0 **SCNT<11:8>:** Current Source Byte Count

REGISTER 17-14: DMAxDSAL – DMAx DESTINATION START ADDRESS LOW REGISTER

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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n/n = Value at POR and BOR/Value at all other Resets	1 = bit is set	0 = bit is cleared	x = bit is unknown u = bit is unchanged

bit 7-0 **DSA<7:0>:** Destination Start Address bits

28.0 COMPLEMENTARY WAVEFORM GENERATOR (CWG) MODULE

The Complementary Waveform Generator (CWG) produces half-bridge, full-bridge, and steering of PWM waveforms. It is backwards compatible with previous CCP functions. The PIC18(L)F2x/4xK42 family has three instances of the CWG module.

Each of the CWG modules has the following features:

- Six operating modes:
  - Synchronous Steering mode
  - Asynchronous Steering mode
  - Full-Bridge mode, Forward
  - Full-Bridge mode, Reverse
  - Half-Bridge mode
  - Push-Pull mode
- Output polarity control
- Output steering
- Independent 6-bit rising and falling event dead-band timers
  - Clocked dead band
  - Independent rising and falling dead-band enables
- Auto-shutdown control with:
  - Selectable shutdown sources
  - Auto-restart option
  - Auto-shutdown pin override control

28.1 Fundamental Operation

The CWG generates two output waveforms from the selected input source.

The off-to-on transition of each output can be delayed from the on-to-off transition of the other output, thereby, creating a time delay immediately where neither output is driven. This is referred to as dead time and is covered in **Section 28.6 "Dead-Band Control"**.

It may be necessary to guard against the possibility of circuit faults or a feedback event arriving too late or not at all. In this case, the active drive must be terminated before the Fault condition causes damage. This is referred to as auto-shutdown and is covered in **Section 28.10 "Auto-Shutdown"**.

28.2 Operating Modes

The CWG module can operate in six different modes, as specified by the MODE<2:0> bits of the CWGxCON0 register:

- Half-Bridge mode
- Push-Pull mode
- Asynchronous Steering mode
- Synchronous Steering mode
- Full-Bridge mode, Forward
- Full-Bridge mode, Reverse

All modes accept a single pulse data input, and provide up to four outputs as described in the following sections.

All modes include auto-shutdown control as described in **Section 28.10 "Auto-Shutdown"**

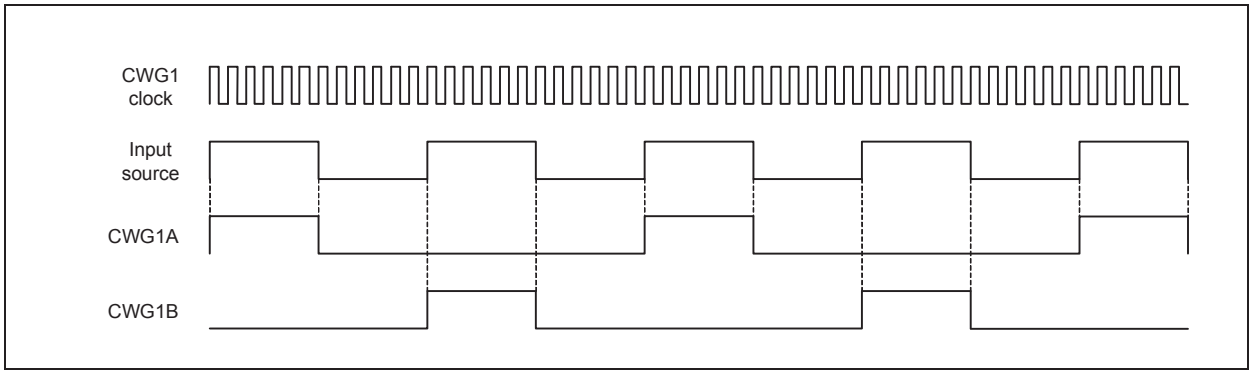
Note:	Except as noted for Full-bridge mode (**Section 28.2.3 "Full-Bridge Modes"**), mode changes should only be performed while EN = 0 (Register 28-1).

28.2.1 HALF-BRIDGE MODE

In Half-Bridge mode, two output signals are generated as true and inverted versions of the input as illustrated in Figure 28-2. A non-overlap (dead-band) time is inserted between the two outputs as described in **Section 28.6 "Dead-Band Control"**. The output steering feature cannot be used in this mode. A basic block diagram of this mode is shown in Figure 28-1.

The unused outputs CWGxC and CWGxD drive similar signals as CWGxA and CWGxB, with polarity independently controlled by the POLC and POLD bits of the CWGxCON1 register, respectively.

FIGURE 28-4: CWGx PUSH-PULL MODE OPERATION



28.2.3 FULL-BRIDGE MODES

In Forward and Reverse Full-Bridge modes, three outputs drive static values while the fourth is modulated by the input data signal. The mode selection may be toggled between forward and reverse by toggling the MODE<0> bit of the CWGxCON0 while keeping MODE<2:1> static, without disabling the CWG module. When connected as shown in Figure 28-5, the outputs are appropriate for a full-bridge motor driver. Each CWG output signal has independent polarity control, so the circuit can be adapted to high-active and low-active drivers. A simplified block diagram for the Full-Bridge modes is shown in Figure 28-6.

FIGURE 28-5: EXAMPLE OF FULL-BRIDGE APPLICATION

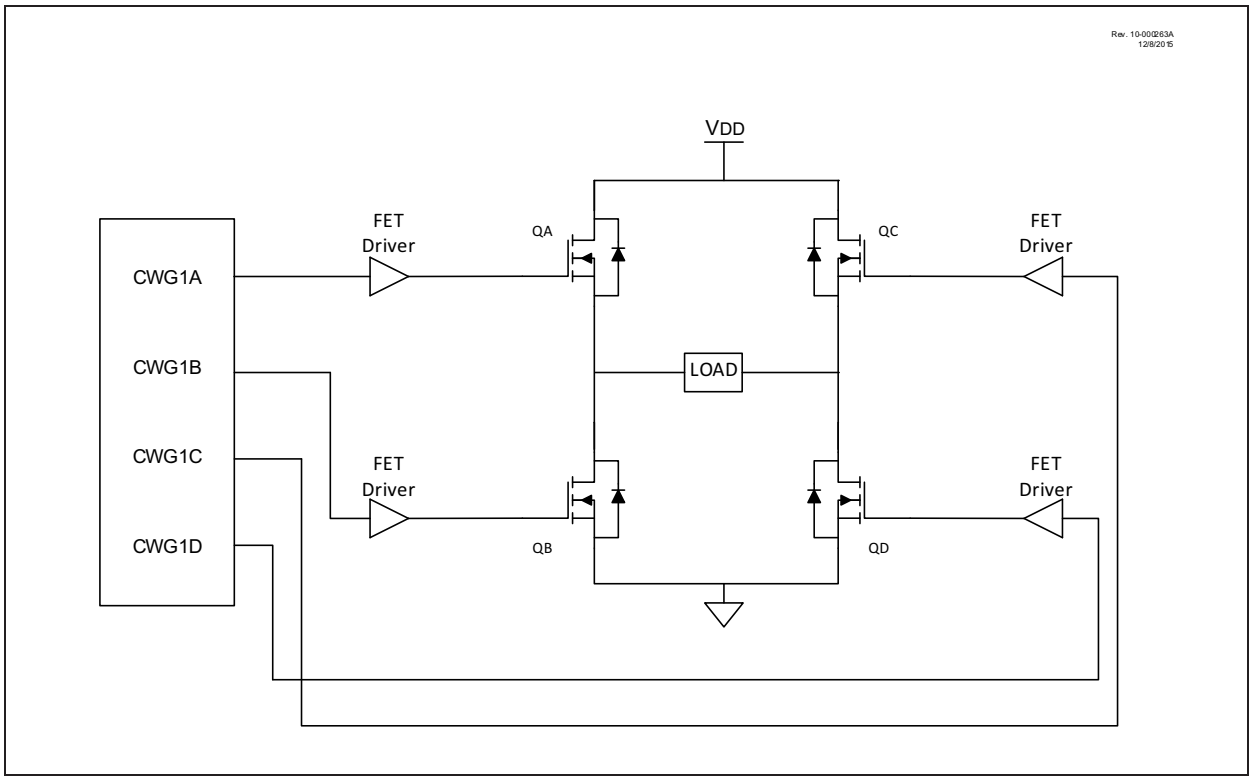


TABLE 29-1: CLCx DATA INPUT SELECTION

DyS<5:0> Value	CLCx Input Source
111111 [63]	Reserved
.	
.	
.	
110100 [52]	Reserved
110011 [51]	CWG3B_out
110010 [50]	CWG3A_out
110001 [49]	CWG2B_out
110000 [48]	CWG2A_out
101111 [47]	CWG1B_out
101110 [46]	CWG1A_out
101101 [45]	SPI1_ss_out
101100 [44]	SPI1_sck_out
101011 [43]	SPI1_sdo_out
101010 [42]	Reserved
101001 [41]	UART2_tx_out
101000 [40]	UART1_tx_out
100111 [39]	CLC4_out
100110 [38]	CLC3_out
100101 [37]	CLC2_out
100100 [36]	CLC1_out
100011 [35]	DSM1_out
100010 [34]	IOC_flag
100001 [33]	ZCD_out
100000 [32]	CMP2_out
011111 [31]	CMP1_out
011110 [30]	NCO1_out
011101 [29]	Reserved
011100 [28]	Reserved
011011 [27]	PWM8_out
011010 [26]	PWM7_out
011001 [25]	PWM6_out
011000 [24]	PWM5_out
010111 [23]	CCP4_out
010110 [22]	CCP3_out
010101 [21]	CCP2_out
010100 [20]	CCP1 _out
010011 [19]	SMT1_out
010010 [18]	TMR6_out
010001 [17]	TMR5 _overflow
010000 [16]	TMR4 _out
001111 [15]	TMR3 _overflow

TABLE 29-1: CLCx DATA INPUT SELECTION (CONTINUED)

DyS<5:0> Value	CLCx Input Source
001110 [14]	TMR2 _out
001101 [13]	TMR1 _overflow
001100 [12]	TMR0 _overflow
001011 [11]	CLKR _out
001010 [10]	ADCRC.clc_adc_clk
001001 [9]	SOSC
001000 [8]	MFINTOSC (32 kHz)
000111 [7]	MFINTOSC (500 kHz)
000110 [6]	LFINTOSC
000101 [5]	HFINTOSC
000100 [4]	FOSC
000011 [3]	CLCIN3PPS
000010 [2]	CLCIN2PPS
000001 [1]	CLCIN1PPS
000000 [0]	CLCIN0PPS

34.6 Slave Mode

34.6.1 SLAVE MODE TRANSMIT OPTIONS

The SDO output of the SPI module in Slave mode is controlled by the TXR bit of SPIxCON2, the TRIS bit associated with the SDO pin, the Slave Select input, and the current state of the TXFIFO. This control is summarized in Table 34-2. In this table, TRISxn refers to the bit in the TRIS register corresponding to the pin that SDO has been assigned with PPS, TXR is the Transmit Data Required Control bit of SPIxCON2, SS is the state of the Slave Select input, and TXBE is the TXFIFO Buffer Empty bit of SPIxSTATUS.

34.6.1.1 SDO Drive/Tri-state

The TRIS bit associated with the SDO pin controls whether the SDO pin will tri-state. When this TRIS bit is cleared, the pin will always be driving to a level, even when the SPI module is inactive. When the SPI module is inactive (either due to the master not clocking the SCK line or the SS being false), the SDO pin will be driven to the value of the LAT bit associated with the SDO pin. When the SPI module is active, its output is determined by both TXR and whether there is data in the TXFIFO.

When the TRIS bit associated with the SDO pin is set, the pin will only have an output level driven to it when TXR = 1 and the slave select input is true. In all other cases, the pin will be tri-stated.

34.6.1.2 SDO Output Data

The TXR bit controls the nature of the data that is transmitted in Slave mode. When TXR is set, transmitted data is taken from the TXFIFO. If the FIFO is empty, the most recently received data will be transmitted and the TXUIF flag will be set to indicate that a transmit FIFO underflow has occurred.

When TXR is cleared, the data will be taken from the TXFIFO, and the TXFIFO occupancy will not decrease. If the TXFIFO is empty, the most recently received data will be transmitted, and the TXUIF bit will not be set. However, if the TRIS bit associated with the SDO pin is set, clearing the TXR bit will cause the SPI module to not output any data to the SDO pin.

TABLE 34-2: SLAVE MODE TRANSMIT

TRISxn(1)	TXR	SS	TXBE	SDO State
0	0	FALSE	0	Drives state determined by LATxn(2)
0	0	FALSE	1	Drives state determined by LATxn(2)
0	0	TRUE	0	Outputs the oldest byte in the TXFIFO Does not remove data from the TXFIFO
0	0	TRUE	1	Outputs the most recently received byte
0	1	FALSE	0	Drives state determined by LATxn(2)
0	1	FALSE	1	Drives state determined by LATxn(2)
0	1	TRUE	0	Outputs the oldest byte in the TXFIFO Removes transmitted byte from the TXFIFO Decrements occupancy of TXFIFO
0	1	TRUE	1	Outputs the most recently received byte Sets the TXUIF bit of SPIxINTF
1	0	FALSE	0	Tri-stated
1	0	FALSE	1	Tri-stated
1	0	TRUE	0	Tri-stated
1	0	TRUE	1	Tri-stated
1	1	FALSE	0	Tri-stated
1	1	FALSE	1	Tri-stated
1	1	TRUE	0	Outputs the oldest byte in the TXFIFO Removes transmitted byte from the TXFIFO Decrements occupancy of TXFIFO
1	1	TRUE	1	Outputs the most recently received byte Sets the TXUIF bit of SPIxINTF

Note 1: TRISxn is the bit in the TRISx register corresponding to the pin that SDO has been assigned with PPS.
2: LATxn is the bit in the LATx register corresponding to the pin that SDO has been assigned with PPS.

REGISTER 34-12: SPIxTxB: SPI TRANSMIT BUFFER REGISTER

W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'

bit 7-0 **TXB<7:0>**: Transmit Buffer bits (write only)

If TXFIFO is not full:

Writing to this register adds the data to the top of the TXFIFO and increases the occupancy of the TXFIFO write pointer

If TXFIFO is full:

Writing to this register does not affect the data in the TXFIFO or the write pointer, and the TXWE bit of SPIxSTATUS will be set

REGISTER 34-13: SPIxCLK: SPI CLOCK SELECTION REGISTER

U-0	U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	—	CLKSEL3	CLKSEL2	CLKSEL1	CLKSEL0
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'

bit 7-4 **Unimplemented**: Read as '0'

bit 3-0 **CLKSEL<3:0>**: SPI Clock Source Selection bits

1111-1001 = Reserved
1000 = SMT_match
0111 = TMR6_Postscaled
0110 = TMR4_Postscaled
0101 = TMR2_Postscaled
0100 = TMR0_overflow
0011 = CLKREF
0010 = MFINTOSC
0001 = HFINTOSC
0000 = FOSC

REGISTER 38-2: ADCON1: ADC CONTROL REGISTER 1

R/W-0/0	R/W-0/0	R/W-0/0	U-0	U-0	U-0	U-0	R/W-0/0
PPOL	IPEN	GPOL	—	—	—	—	DSEN
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

bit 7 **PPOL:** Precharge Polarity bit
If PRE>0x00:

PPOL	Action During 1st Precharge Stage	
	External (selected analog I/O pin)	Internal (AD sampling capacitor)
1	Connected to V_{DD}	C_{HOLD} connected to V_{SS}
0	Connected to V_{SS}	C_{HOLD} connected to V_{DD}

Otherwise:
The bit is ignored

bit 6 **IPEN:** A/D Inverted Precharge Enable bit
If DSEN = 1
1 = The precharge and guard signals in the second conversion cycle are the opposite polarity of the first cycle
0 = Both Conversion cycles use the precharge and guards specified by ADPPOL and ADGPOL
Otherwise:
The bit is ignored

bit 5 **GPOL:** Guard Ring Polarity Selection bit
1 = ADC guard Ring outputs start as digital high during Precharge stage
0 = ADC guard Ring outputs start as digital low during Precharge stage

bit 4-1 **Unimplemented:** Read as '0'

bit 0 **DSEN:** Double-sample enable bit
1 = Two conversions are performed on each trigger. Data from the first conversion appears in PREV
0 = One conversion is performed for each trigger

REGISTER 39-2: DAC1CON1: DAC DATA REGISTER

U-0	U-0	U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
—	—	—	DATA<4:0>				
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

bit 7-5 **Unimplemented:** Read as '0'

bit 4-0 **DATA<4:0>:** Data Input Register for DAC bits

TABLE 39-2: SUMMARY OF REGISTERS ASSOCIATED WITH THE DAC MODULE

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on page
DAC1CON0	EN	—	OE1	OE2	PSS<1:0>		—	NSS	644
DAC1CON1	—	—	—	DATA<4:0>					645

Legend: — = Unimplemented location, read as '0'. Shaded cells are not used with the DAC module.

43.0 INSTRUCTION SET SUMMARY

PIC18(L)F2x/4xK42 devices incorporate the standard set of PIC18 core instructions, as well as an extended set of instructions, for the optimization of code that is recursive or that utilizes a software stack. The extended set is discussed later in this section.

43.1 Standard Instruction Set

The standard PIC18 instruction set adds many enhancements to the previous PIC® MCU instruction sets, while maintaining an easy migration from these PIC® MCU instruction sets. Most instructions are a single program memory word (16 bits), but there are four instructions that require two-program memory locations and two that require three-program memory locations.

Each single-word instruction is a 16-bit word divided into an opcode, which specifies the instruction type and one or more operands, which further specify the operation of the instruction.

The instruction set is highly orthogonal and is grouped into four basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal** operations
- **Control** operations

The PIC18 instruction set summary in Table 43-2 lists **byte-oriented**, **bit-oriented**, **literal** and **control** operations. Table 43-1 shows the opcode field descriptions.

Most **byte-oriented** instructions have three operands:

1. The file register (specified by ‘f’)
2. The destination of the result (specified by ‘d’)
3. The accessed memory (specified by ‘a’)

The file register designator ‘f’ specifies which file register is to be used by the instruction. The destination designator ‘d’ specifies where the result of the operation is to be placed. If ‘d’ is zero, the result is placed in the WREG register. If ‘d’ is one, the result is placed in the file register specified in the instruction.

All **bit-oriented** instructions have three operands:

1. The file register (specified by ‘f’)
2. The bit in the file register (specified by ‘b’)
3. The accessed memory (specified by ‘a’)

The bit field designator ‘b’ selects the number of the bit affected by the operation, while the file register designator ‘f’ represents the number of the file in which the bit is located.

The literal instructions may use some of the following operands:

- A literal value to be loaded into a file register (specified by ‘k’)
- The desired FSR register to load the literal value into (specified by ‘f’)
- No operand required (specified by ‘—’)

The control instructions may use some of the following operands:

- A program memory address (specified by ‘n’)
- The mode of the CALL or RETURN instructions (specified by ‘s’)
- The mode of the table read and table write instructions (specified by ‘m’)
- No operand required (specified by ‘—’)

All instructions are a single word, except for four double-word instructions. These instructions were made double-word to contain the required information in 32 bits. In the second word, the four MSbs are ‘1’s. If this second word is executed as an instruction (by itself), it will execute as a NOP.

All single-word instructions are executed in a single instruction cycle, unless a conditional test is true or the program counter is changed as a result of the instruction. In these cases, the execution takes two instruction cycles, with the additional instruction cycle(s) executed as a NOP.

The double-word instructions execute in two instruction cycles.

One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 μs. If a conditional test is true, or the program counter is changed as a result of an instruction, the instruction execution time is 2 μs. Two-word branch instructions (if true) would take 3 μs.

Figure 43-1 shows the general formats that the instructions can have. All examples use the convention ‘nnh’ to represent a hexadecimal number.

The Instruction Set Summary, shown in Table 43-2, lists the standard instructions recognized by the Microchip Assembler (MPASM™).

Section 43.1.1 “Standard Instruction Set” provides a description of each instruction.