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, POR, PWM, WDT
Number of I/O	17
Program Memory Size	8KB (4K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	256 x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	A/D 12x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	20-SSOP (0.209", 5.30mm Width)
Supplier Device Package	20-SSOP
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18lf13k22-i-ss

TABLE 3-2: REGISTER FILE SUMMARY (PIC18(L)F1XK22) (CONTINUED)

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Details on page:
SPBRGH	EUSART Baud Rate Generator Register, High Byte								0000 0000	247, 182
SPBRG	EUSART Baud Rate Generator Register, Low Byte								0000 0000	247, 182
RCREG	EUSART Receive Register								0000 0000	247, 175
TXREG	EUSART Transmit Register								0000 0000	247, 172
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	0000 0010	247, 179
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	247, 180
EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0	0000 0000	247, 45, 54
EEDATA	EEPROM Data Register								0000 0000	247, 45, 54
EECON2	EEPROM Control Register 2 (not a physical register)								0000 0000	247, 45, 54
EECON1	EEPGD	CFGS	—	FREE	WRERR	WREN	WR	RD	xx-0 x000	247, 45, 54
IPR2	OSCFIP	C1IP	C2IP	EEIP	BCLIP	—	TMR3IP	—	1111 1-1-	248, 70
PIR2	OSCFIF	C1IF	C2IF	EEIF	BCLIF	—	TMR3IF	—	0000 0-0-	248, 66
PIE2	OSCFIE	C1IE	C2IE	EEIE	BCLIE	—	TMR3IE	—	0000 0-0-	248, 68
IPR1	—	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	-111 1111	248, 69
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	-000 0000	248, 65
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	-000 0000	248, 67
OSCTUNE	INTSRC	PLLEN	TUN5	TUN4	TUN3	TUN2	TUN1	TUN0	0000 0000	248, 19
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	1111 1111	248, 84
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	—	—	—	—	1111 ----	248, 80
TRISA	—	—	TRISA5	TRISA4	—(3)	TRISA2	TRISA1	TRISA0	--11 1111	248, 75
LATC	LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0	xxxx xxxx	248, 85
LATB	LATB7	LATB6	LATB5	LATB4	—	—	—	—	xxxx ----	248, 80
LATA	—	—	LATA5	LATA4	—	LATA2	LATA1	LATA0	--xx -xxx	248, 76
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	248, 84
PORTB	RB7	RB6	RB5	RB4	—	—	—	—	xxxx ----	248, 80
PORTA	—	—	RA5	RA4	RA3(2)	RA2	RA1	RA0	--xx xxxx	248, 75
ANSELH	—	—	—	—	ANS11	ANS10	ANS9	ANS8	---- 1111	248, 89
ANSEL	ANS7	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0	1111 1111	248, 88
IOCB	IOCB7	IOCB6	IOCB5	IOCB4	—	—	—	—	0000 ----	248, 81
IOCA	—	—	IOCA5	IOCA4	IOCA3	IOCA2	IOCA1	IOCA0	--00 0000	248, 76
WPUB	WPUB7	WPUB6	WPUB5	WPUB4	—	—	—	—	1111 ----	248, 81
WPUA	—	—	WPUA5	WPUA4	WPUA3	WPUA2	WPUA1	WPUA0	--11 1111	245, 76
SLRCON	—	—	—	—	—	SLRC	SLRB	SLRA	---- -111	248, 90
SSPMSK	MSK7	MSK6	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	1111 1111	248, 146
CM1CON0	C1ON	C1OUT	C1OE	C1POL	C1SP	C1R	C1CH1	C1CH0	0000 0000	248, 216
CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	C1HYS	C2HYS	C1SYNC	C2SYNC	0000 0000	248, 220
CM2CON0	C2ON	C2OUT	C2OE	C2POL	C2SP	C2R	C2CH1	C2CH0	0000 0000	248, 217
SRCON1	SRSPE	SRSCKE	SRSC2E	SRSC1E	SRRPE	SRRCKE	SRRC2E	SRRC1E	0000 0000	248, 230
SRCON0	SRLEN	SRCLK2	SRCLK1	SRCLK0	SRQEN	SRNQEN	SRPS	SRPR	0000 0000	248, 229

Legend: x = unknown, u = unchanged, – = unimplemented, q = value depends on condition

Note 1: The SBOREN bit is only available when the BOREN<1:0> Configuration bits = 01; otherwise it is disabled and reads as '0'. See **Section 22.4 "Brown-out Reset (BOR)"**.

2: The RA3 bit is only available when Master Clear Reset is disabled (MCLRE Configuration bit = 0). Otherwise, RA3 reads as '0'. This bit is read-only.

3: Unimplemented, read as '1'.

3.5.3 MAPPING THE ACCESS BANK IN INDEXED LITERAL OFFSET MODE

The use of Indexed Literal Offset Addressing mode effectively changes how the first 96 locations of Access RAM (00h to 5Fh) are mapped. Rather than containing just the contents of the bottom section of Bank 0, this mode maps the contents from a user defined “window” that can be located anywhere in the data memory space. The value of FSR2 establishes the lower boundary of the addresses mapped into the window, while the upper boundary is defined by FSR2 plus 95 (5Fh). Addresses in the Access RAM above 5Fh are mapped as previously described (see **Section 3.3.2 “Access Bank”**). An example of Access Bank remapping in this addressing mode is shown in Figure 3-10.

Remapping of the Access Bank applies *only* to operations using the Indexed Literal Offset mode. Operations that use the BSR (Access RAM bit is ‘1’) will continue to use direct addressing as before.

3.6 PIC18 Instruction Execution and the Extended Instruction Set

Enabling the extended instruction set adds eight additional commands to the existing PIC18 instruction set. These instructions are executed as described in **Section 24.2 “Extended Instruction Set”**.

FIGURE 3-10: REMAPPING THE ACCESS BANK WITH INDEXED LITERAL OFFSET ADDRESSING

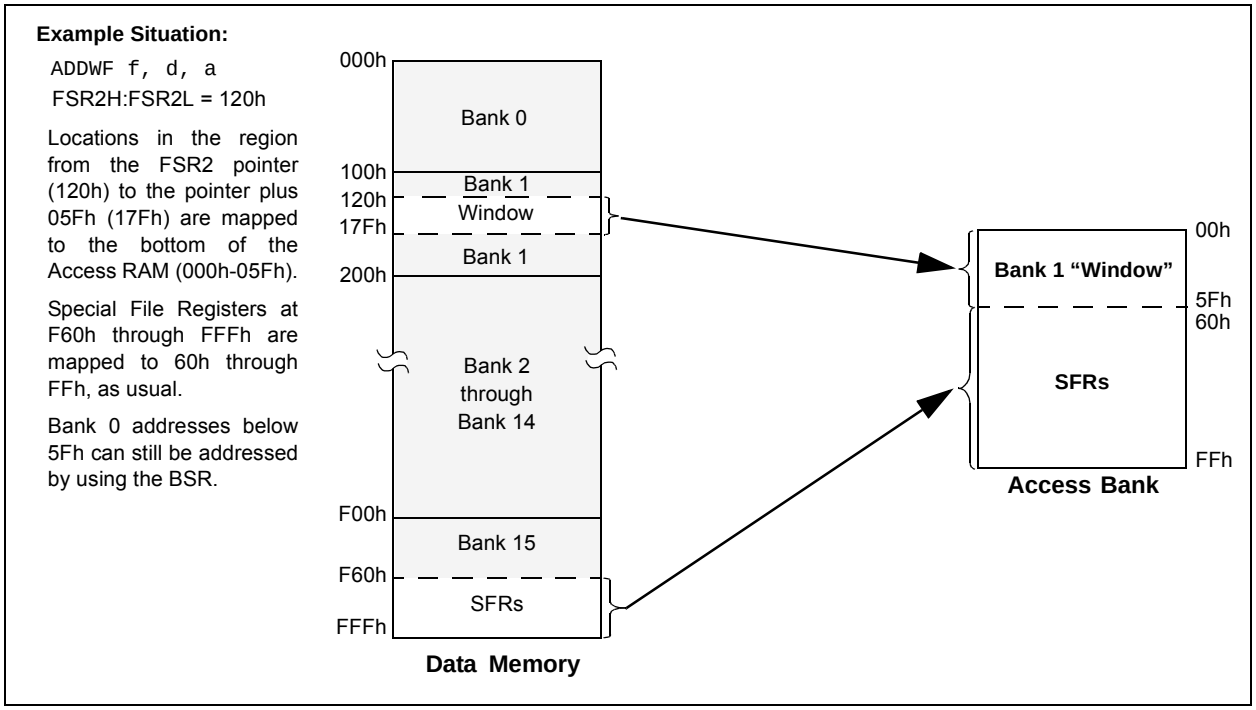
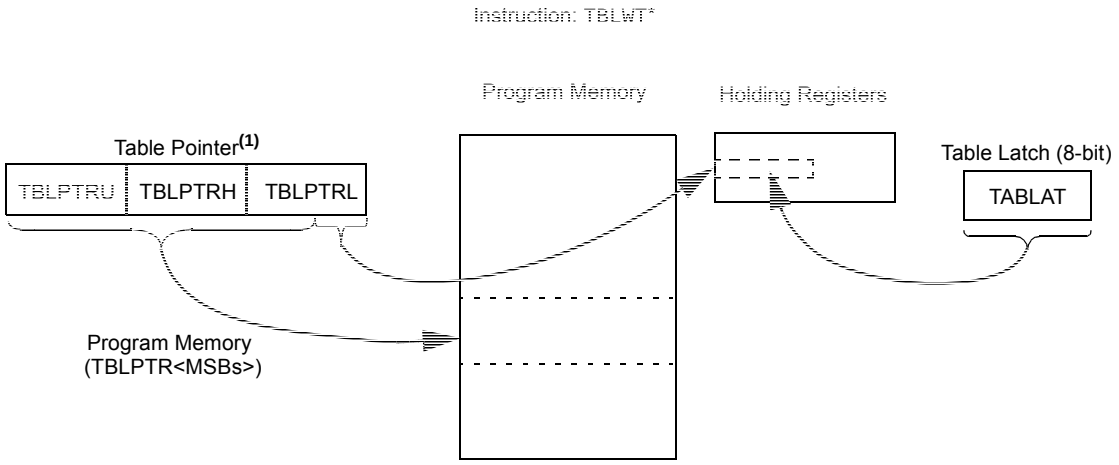


FIGURE 4-2: TABLE WRITE OPERATION



Note 1: During table writes the Table Pointer does not point directly to Program Memory. The LSBs of TBLPRTL actually point to an address within the write block holding registers. The MSBs of the Table Pointer determine where the write block will eventually be written. The process for writing the holding registers to the program memory array is discussed in **Section 4.5 "Writing to Flash Program Memory"**.

4.2 Control Registers

Several control registers are used in conjunction with the TBLRD and TBLWT instructions. These include the:

- EECON1 register
- EECON2 register
- TABLAT register
- TBLPTR registers

4.2.1 EECON1 AND EECON2 REGISTERS

The EECON1 register (Register 4-1) is the control register for memory accesses. The EECON2 register is not a physical register; it is used exclusively in the memory write and erase sequences. Reading EECON2 will read all '0's.

The EEPGD control bit determines if the access will be a program or data EEPROM memory access. When EEPGD is clear, any subsequent operations will operate on the data EEPROM memory. When EEPGD is set, any subsequent operations will operate on the program memory.

The CFGS control bit determines if the access will be to the Configuration/Calibration registers or to program memory/data EEPROM memory. When CFGS is set, subsequent operations will operate on Configuration registers regardless of EEPGD (see **Section 23.0 "Special Features of the CPU"**). When CFGS is clear, memory selection access is determined by EEPGD.

The FREE bit allows the program memory erase operation. When FREE is set, an erase operation is initiated on the next WR command. When FREE is clear, only writes are enabled.

The WREN bit, when set, will allow a write operation. The WREN bit is clear on power-up.

The WRERR bit is set by hardware when the WR bit is set and cleared when the internal programming timer expires and the write operation is complete.

Note:	During normal operation, the WRERR is read as '1'. This can indicate that a write operation was prematurely terminated by a Reset, or a write operation was attempted improperly.

The WR control bit initiates write operations. The WR bit cannot be cleared, only set, by firmware. Then WR bit is cleared by hardware at the completion of the write operation.

Note:	The EEIF interrupt flag bit of the PIR2 register is set when the write is complete. The EEIF flag stays set until cleared by firmware.

4.5 Writing to Flash Program Memory

The programming block size is 8 or 16 bytes, depending on the device (See Table 4-1). Word or byte programming is not supported.

Table writes are used internally to load the holding registers needed to program the Flash memory. There are only as many holding registers as there are bytes in a write block (See Table 4-1).

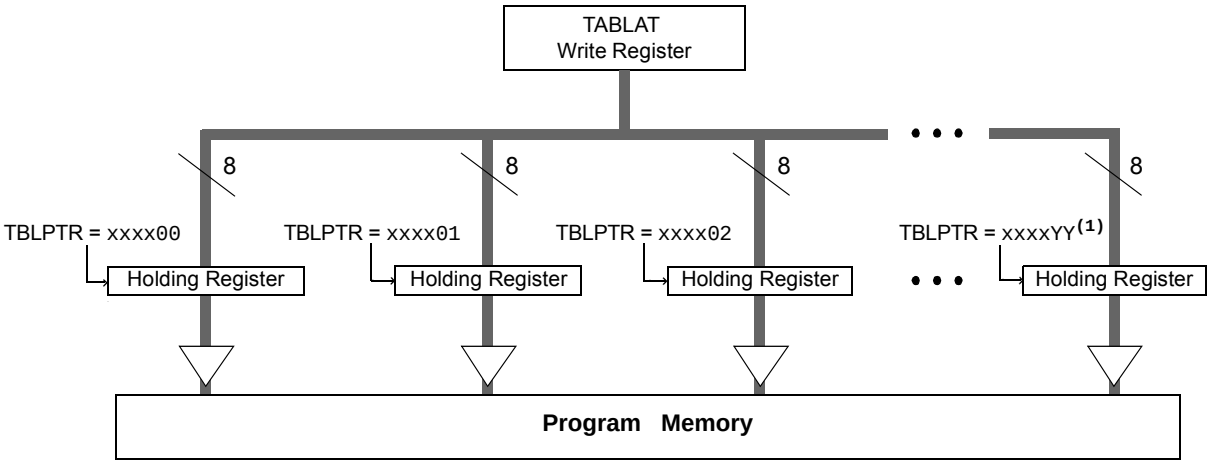
Since the Table Latch (TABLAT) is only a single byte, the TBLWT instruction may need to be executed 8, or 16 times, depending on the device, for each programming operation. All of the table write operations will essentially be short writes because only the holding registers are written. After all the holding registers have been written, the programming operation of that block of memory is started by configuring the EECON1 register for a program memory write and performing the long write sequence.

The long write is necessary for programming the internal Flash. Instruction execution is halted during a long write cycle. The long write will be terminated by the internal programming timer.

The EEPROM on-chip 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.

FIGURE 4-5: TABLE WRITES TO FLASH PROGRAM MEMORY



Note 1: YY = x7 or xF for 8 or 16 byte write blocks, respectively.

4.5.1 FLASH PROGRAM MEMORY WRITE SEQUENCE

The sequence of events for programming an internal program memory location should be:

1. Read 64 bytes into RAM.
2. Update data values in RAM as necessary.
3. Load Table Pointer register with address being erased.
4. Execute the block erase procedure.
5. Load Table Pointer register with address of first byte being written.
6. Write the 8 or 16 byte block into the holding registers with auto-increment.
7. Set the EECON1 register for the write operation:
   - set EEPGD bit to point to program memory;
   - clear the CFGS bit to access program memory;
   - set WREN to enable byte writes.
8. Disable interrupts.
9. Write 55h to EECON2.
10. Write 0AAh to EECON2.
11. Set the WR bit. This will begin the write cycle.
12. The CPU will stall for duration of the write (about 2 ms using internal timer).
13. Re-enable interrupts.
14. Repeat steps 6 to 13 for each block until all 64 bytes are written.
15. Verify the memory (table read).

This procedure will require about 6 ms to update each write block of memory. An example of the required code is given in Example 4-3.

Note:	Before setting the WR bit, the Table Pointer address needs to be within the intended address range of the bytes in the holding registers.

5.3 Reading the Data EEPROM Memory

To read a data memory location, the user must write the address to the EEADR register, clear the EEPGD control bit of the EECON1 register and then set control bit, RD. The data is available on the very next instruction cycle; therefore, the EEDATA register can be read by the next instruction. EEDATA will hold this value until another read operation, or until it is written to by the user (during a write operation).

The basic process is shown in Example 5-1.

5.4 Writing to the Data EEPROM Memory

To write an EEPROM data location, the address must first be written to the EEADR register and the data written to the EEDATA register. The sequence in Example 5-2 must be followed to initiate the write cycle.

The write will not begin if this sequence is not exactly followed (write 55h to EECON2, write 0AAh to EECON2, then set WR bit) for each byte. It is strongly recommended that interrupts be disabled during this code segment.

Additionally, the WREN bit in EECON1 must be set to enable writes. This mechanism prevents accidental writes to data EEPROM due to unexpected code execution (i.e., runaway programs). The WREN bit should be kept clear at all times, except when updating the EEPROM. The WREN bit is not cleared by hardware.

After a write sequence has been initiated, EECON1, EEADR and EEDATA cannot be modified. The WR bit will be inhibited from being set unless the WREN bit is set. Both WR and WREN cannot be set with the same instruction.

At the completion of the write cycle, the WR bit is cleared by hardware and the EEPROM Interrupt Flag bit, EEIF, is set. The user may either enable this interrupt or poll this bit. EEIF must be cleared by software.

5.5 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 5-1: DATA EEPROM READ

```
        MOVLW   DATA_EE_ADDR    ;
        MOVWF   EEADR           ; Data Memory Address to read
        BCF     EECON1, EEPGD   ; Point to DATA memory
        BCF     EECON1, CFGS    ; Access EEPROM
        BSF     EECON1, RD      ; EEPROM Read
        MOVF    EEDATA, W       ; W = EEDATA
```

EXAMPLE 5-2: DATA EEPROM WRITE

```
                MOVLW   DATA_EE_ADDR_LOW    ;
                MOVWF   EEADR               ; Data Memory Address to write
                MOVLW   DATA_EE_DATA        ;
                MOVWF   EEDATA              ; Data Memory Value to write
                BCF     EECON1, EEPGD       ; Point to DATA memory
                BCF     EECON1, CFGS        ; Access EEPROM
                BSF     EECON1, WREN        ; Enable writes
                BCF     INTCON, GIE         ; Disable Interrupts
                MOVLW   55h                 ;
   Required     MOVWF   EECON2              ; Write 55h
   Sequence     MOVLW   0AAh                ;
                MOVWF   EECON2              ; Write 0AAh
                BSF     EECON1, WR          ; Set WR bit to begin write
                BSF     INTCON, GIE         ; Enable Interrupts

                                            ; User code execution
                BCF     EECON1, WREN        ; Disable writes on write complete (EEIF set)
```

REGISTER 8-6: PORTB: PORTB REGISTER

R/W-x	R/W-x	R/W-x	R/W-x	U-0	U-0	U-0	U-0
RB7	RB6	RB5	RB4	—	—	—	—
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 7-4 **RB<7:4>**: PORTB I/O Pin bit
1 = Port pin is >VIH
0 = Port pin is <VIL

bit 3-0 **Unimplemented**: Read as '0'

REGISTER 8-7: TRISB: PORTB TRI-STATE REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	U-0	U-0	U-0	U-0
TRISB7	TRISB6	TRISB5	TRISB4	—	—	—	—
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 7-4 **TRISB<7:4>:** PORTB Tri-State Control bit
1 = PORTB pin configured as an input (tri-stated)
0 = PORTB pin configured as an output

bit 3-0 **Unimplemented**: Read as '0'

REGISTER 8-8: LATB: PORTB DATA LATCH REGISTER

R/W-x	R/W-x	R/W-x	R/W-x	U-0	U-0	U-0	U-0
LATB7	LATB6	LATB5	LATB4	—	—	—	—
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 7-4 **LATB<7:4>**: RB<7:4> Port I/O Output Latch Register bits

bit 3-0 **Unimplemented**: Read as '0'

14.3.2 OPERATION

The MSSP module functions are enabled by setting SSPEN bit of the SSPCON1 register.

The SSPCON1 register allows control of the I^2C operation. Four mode selection bits of the SSPCON1 register allow one of the following I^2C modes to be selected:

- I^2C Master mode, clock = (FOSC/(4*(SSPADD + 1))
- I^2C Slave mode (7-bit address)
- I^2C Slave mode (10-bit address)
- I^2C Slave mode (7-bit address) with Start and Stop bit interrupts enabled
- I^2C Slave mode (10-bit address) with Start and Stop bit interrupts enabled
- I^2C Firmware Controlled Master mode, slave is Idle

Selection of any I^2C mode with the SSPEN bit set, forces the SCL and SDA pins to be open-drain, provided these pins are programmed to inputs by setting the appropriate TRIS bits

Note:	To ensure proper operation of the module, pull-up resistors must be provided externally to the SCL and SDA pins.

14.3.3 SLAVE MODE

In Slave mode, the SCL and SDA pins must be configured as inputs. The MSSP module will override the input state with the output data when required (slave-transmitter).

The I^2C Slave mode hardware will always generate an interrupt on an address match. Through the mode select bits, the user can also choose to interrupt on Start and Stop bits

When an address is matched, or the data transfer after an address match is received, the hardware automatically will generate the Acknowledge ($\overline{ACK}$) pulse and load the SSPBUF register with the received value currently in the SSPSR register.

Any combination of the following conditions will cause the MSSP module not to give this $\overline{ACK}$ pulse:

- The Buffer Full bit, BF bit of the SSPSTAT register, is set before the transfer is received.
- The overflow bit, SSPOV bit of the SSPCON1 register, is set before the transfer is received.

In this case, the SSPSR register value is not loaded into the SSPBUF, but bit SSPIF of the PIR1 register is set. The BF bit is cleared by reading the SSPBUF register, while bit SSPOV is cleared through software.

The SCL clock input must have a minimum high and low for proper operation. The high and low times of the I^2C specification, as well as the requirement of the MSSP module, are shown in **Section 26.0 "Electrical Specifications"**.

14.3.3.1 Addressing

Once the MSSP module has been enabled, it waits for a Start condition to occur. Following the Start condition, the eight bits are shifted into the SSPSR register. All incoming bits are sampled with the rising edge of the clock (SCL) line. The value of register SSPSR<7:1> is compared to the value of the SSPADD register. The address is compared on the falling edge of the eighth clock (SCL) pulse. If the addresses match and the BF and SSPOV bits are clear, the following events occur:

1. The SSPSR register value is loaded into the SSPBUF register.
2. The Buffer Full bit, BF, is set.
3. An $\overline{ACK}$ pulse is generated.
4. MSSP Interrupt Flag bit, SSPIF of the PIR1 register, is set (interrupt is generated, if enabled) on the falling edge of the ninth SCL pulse.

In 10-bit Address mode, two address bytes need to be received by the slave. The five Most Significant bits (MSbs) of the first address byte specify if this is a 10-bit address. Bit R/$\overline{W}$ of the SSPSTAT register must specify a write so the slave device will receive the second address byte. For a 10-bit address, the first byte would equal '11110 A9 A8 0', where 'A9' and 'A8' are the two MSbs of the address. The sequence of events for 10-bit address is as follows, with steps 7 through 9 for the slave-transmitter:

1. Receive first (high) byte of address (bits SSPIF, BF and UA of the SSPSTAT register are set).
2. Read the SSPBUF register (clears bit BF) and clear flag bit, SSPIF.
3. Update the SSPADD register with second (low) byte of address (clears bit UA and releases the SCL line).
4. Receive second (low) byte of address (bits SSPIF, BF and UA are set). If the address matches then the SCL is held until the next step. Otherwise the SCL line is not held.
5. Read the SSPBUF register (clears bit BF) and clear flag bit, SSPIF.
6. Update the SSPADD register with the first (high) byte of address. (This will clear bit UA and release a held SCL line.)
7. Receive Repeated Start condition.
8. Receive first (high) byte of address with R/$\overline{W}$ bit set (bits SSPIF, BF, R/$\overline{W}$ are set).
9. Read the SSPBUF register (clears bit BF) and clear flag bit, SSPIF.
10. Load SSPBUF with byte the slave is to transmit, sets the BF bit.
11. Set the CKP bit to release SCL.

15.3 EUSART Baud Rate Generator (BRG)

The Baud Rate Generator (BRG) is an 8-bit or 16-bit timer that is dedicated to the support of both the asynchronous and synchronous EUSART operation. By default, the BRG operates in 8-bit mode. Setting the BRG16 bit of the BAUDCON register selects 16-bit mode.

The SPBRGH:SPBRG register pair determines the period of the free running baud rate timer. In Asynchronous mode the multiplier of the baud rate period is determined by both the BRGH bit of the TXSTA register and the BRG16 bit of the BAUDCON register. In Synchronous mode, the BRGH bit is ignored.

Table 15-3 contains the formulas for determining the baud rate. Example 15-1 provides a sample calculation for determining the baud rate and baud rate error.

Typical baud rates and error values for various asynchronous modes have been computed for your convenience and are shown in Table 15-5. It may be advantageous to use the high baud rate (BRGH = 1), or the 16-bit BRG (BRG16 = 1) to reduce the baud rate error. The 16-bit BRG mode is used to achieve slow baud rates for fast oscillator frequencies.

Writing a new value to the SPBRGH, SPBRG register pair causes the BRG timer to be reset (or cleared). This ensures that the BRG does not wait for a timer overflow before outputting the new baud rate.

If the system clock is changed during an active receive operation, a receive error or data loss may result. To avoid this problem, check the status of the RCIDL bit to make sure that the receive operation is Idle before changing the system clock.

EXAMPLE 15-1: CALCULATING BAUD RATE ERROR

For a device with FOSC of 16 MHz, desired baud rate of 9600, Asynchronous mode, 8-bit BRG:

$$Desired\ Baud\ Rate = \frac{FOSC}{64([SPBRGH:SPBRG] + 1)}$$

Solving for SPBRGH:SPBRG:

$$X = \left(\frac{FOSC}{64^* (\text{Desired Baud Rate})}\right)^{-1}$$

$$= \left(\frac{16{,}000{,}000}{64^* 9600}\right)^{-1}$$

$$= [25.042] = 25$$

$$Calculated\ Baud\ Rate = \frac{16000000}{64(25 + 1)}$$

$$= 9615$$

$$Error = \frac{Calc.\ Baud\ Rate - Desired\ Baud\ Rate}{Desired\ Baud\ Rate}$$

$$= \frac{(9615 - 9600)}{9600} = 0.16\%$$

TABLE 15-3: BAUD RATE FORMULAS

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	FOSC/[64 (n+1)]
0	0	1	8-bit/Asynchronous	FOSC/[16 (n+1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	FOSC/[4 (n+1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGH, SPBRG register pair

TABLE 15-4: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

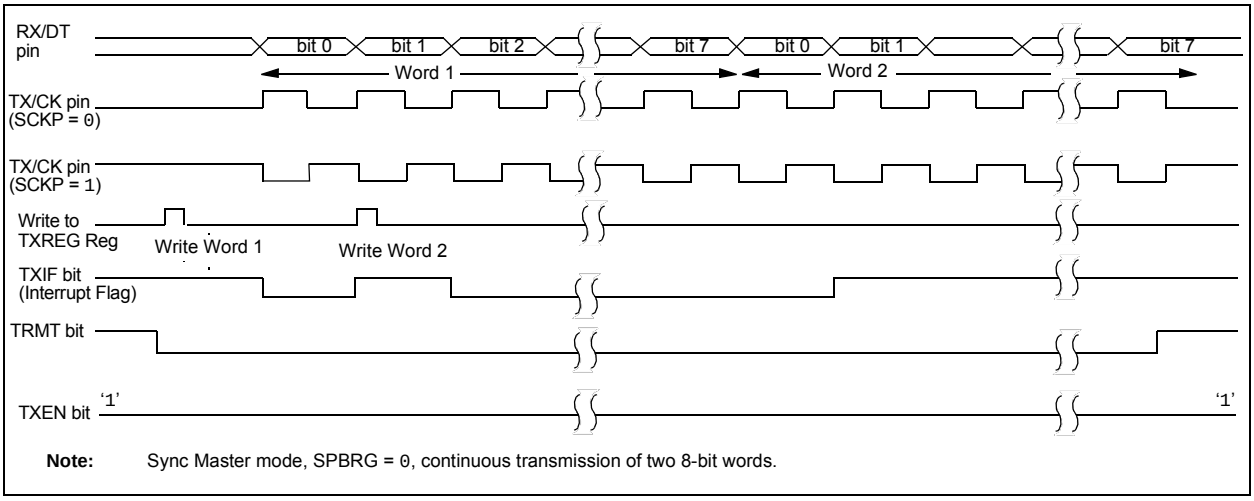
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
BAUDCON	ABDOVF	RCIDL	DTRXP	CKTXP	BRG16	—	WUE	ABDEN	247
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	247
SPBRG	EUSART Baud Rate Generator Register, Low Byte								247
SPBRGH	EUSART Baud Rate Generator Register, High Byte								247
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	247

Legend: — = unimplemented, read as '0'. Shaded cells are not used by the BRG.

15.4.1.5 Synchronous Master Transmission Set-up

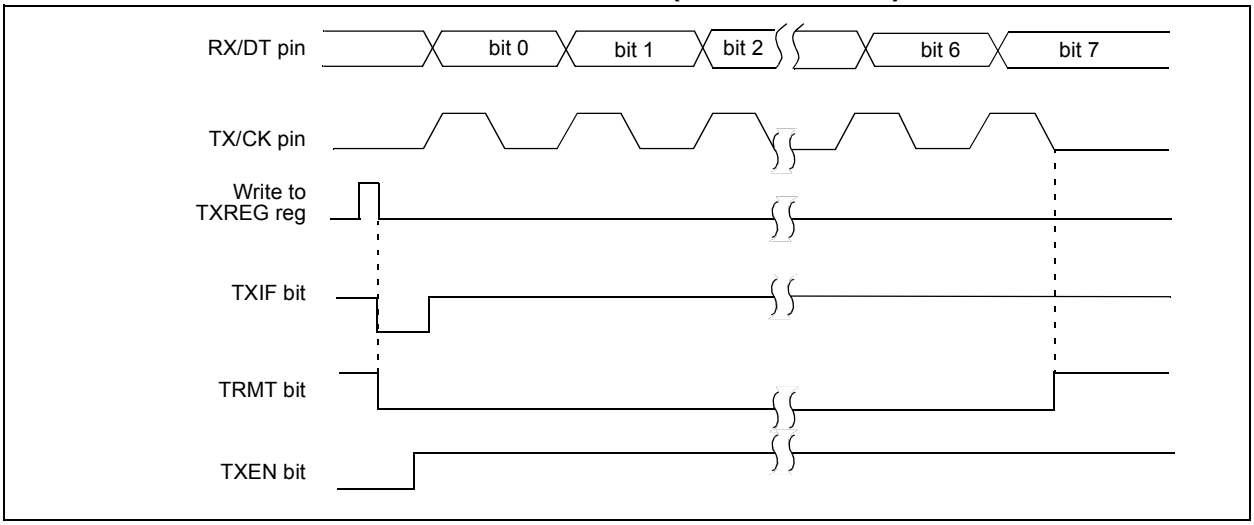
1. Initialize the SPBRGH, SPBRG register pair and the BRGH and BRG16 bits to achieve the desired baud rate (see **Section 15.3 "EUSART Baud Rate Generator (BRG)"**).
2. Enable the synchronous master serial port by setting bits SYNC, SPEN and CSRC. Set the TRIS bits corresponding to the RX/DT and TX/CK I/O pins.
3. Disable Receive mode by clearing bits SREN and CREN.
4. Enable Transmit mode by setting the TXEN bit.
5. If 9-bit transmission is desired, set the TX9 bit.
6. If interrupts are desired, set the TXIE, GIE and PEIE interrupt enable bits.
7. If 9-bit transmission is selected, the ninth bit should be loaded in the TX9D bit.
8. Start transmission by loading data to the TXREG register.

FIGURE 15-10: SYNCHRONOUS TRANSMISSION



Note: Sync Master mode, SPBRG = 0, continuous transmission of two 8-bit words.

FIGURE 15-11: SYNCHRONOUS TRANSMISSION (THROUGH TXEN)



19.0 SR LATCH

The module consists of a single SR latch with multiple Set and Reset inputs as well as selectable latch output. The SR latch module includes the following features:

- Programmable input selection
- SR latch output is available internally/externally
- Selectable Q and $\overline{Q}$ output
- Firmware Set and Reset
- SR Latch

19.1 Latch Operation

The latch is a Set-Reset latch that does not depend on a clock source. Each of the Set and Reset inputs are active-high. The latch can be Set or Reset by CxOUT, INT1 pin, or variable clock. Additionally the SRPS and the SRPR bits of the SRCON0 register may be used to Set or Reset the SR latch, respectively. The latch is reset-dominant, therefore, if both Set and Reset inputs are high the latch will go to the Reset state. Both the SRPS and SRPR bits are self resetting which means that a single write to either of the bits is all that is necessary to complete a latch Set or Reset operation.

19.2 Latch Output

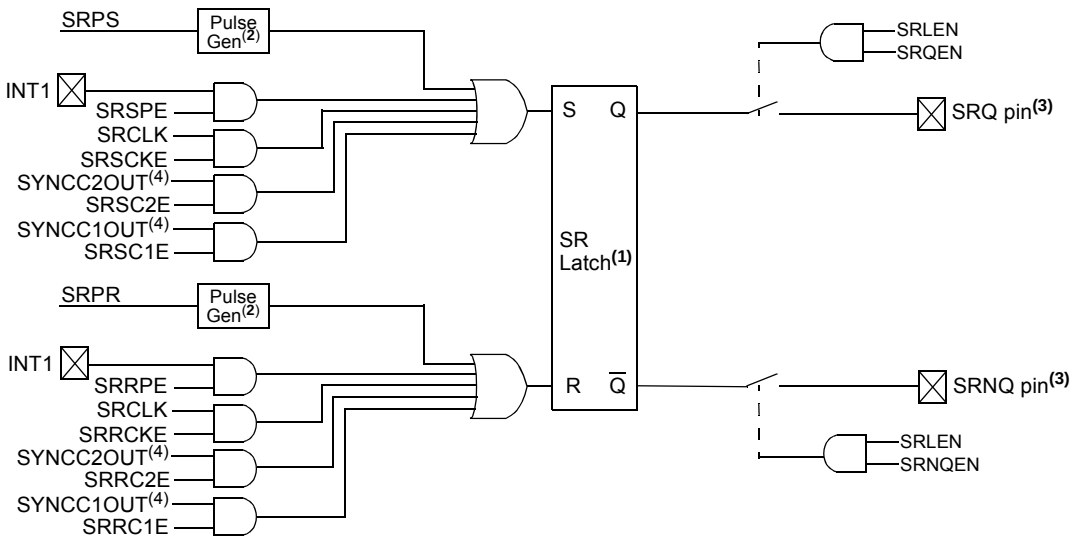
The SRQEN and SRNQEN bits of the SRCON0 register control the latch output selection. Both of the SR latch's outputs may be directly output to an independent I/O pin. Control is determined by the state of bits SRQEN and SRNQEN in registers SRCON0.

The applicable TRIS bit of the corresponding port must be cleared to enable the port pin output driver.

19.3 Effects of a Reset

Upon any device Reset, the SR latch is not initialized. The user's firmware is responsible to initialize the latch output before enabling it to the output pins.

FIGURE 19-1: SR LATCH SIMPLIFIED BLOCK DIAGRAM



Note 1: If R = 1 and S = 1 simultaneously, Q = 0, $\overline{Q}$ = 1
2: Pulse generator causes a 2 Q-state pulse width.
3: Output shown for reference only. See I/O port pin block diagram for more detail.
4: Name denotes the source of connection at the comparator output.

21.0 DIGITAL-TO-ANALOG CONVERTER (DAC) MODULE

The Digital-to-Analog Converter supplies a variable voltage reference, ratiometric with the input source, with 32 selectable output levels.

The input of the DAC can be connected to:

- External V_{REF} pins
- V_{DD} supply voltage
- FVR (Fixed Voltage Reference)

The output of the DAC can be configured to supply a reference voltage to the following:

- Comparator positive input
- ADC input channel
- DAC1OUT pin

The Digital-to-Analog Converter (DAC) can be enabled by setting the D1EN bit of the VREFCON1 register.

21.1 Output Voltage Selection

The DAC has 32 voltage level ranges. The 32 levels are set with the DAC1R<4:0> bits of the VREFCON2 register.

The DAC output voltage is determined by the following equations:

EQUATION 21-1: DAC OUTPUT VOLTAGE

$$V_{OUT} = \left((V_{SRC+} - V_{SRC-}) \times \frac{DACR<4:0>}{2^5} \right) + V_{SRC-}$$

$$V_{SRC+} = V_{DD}, V_{REF+} \text{ or } FVR1$$

$$V_{SRC-} = V_{SS} \text{ or } V_{REF-}$$

21.2 Ratiometric Output Level

The DAC output value is derived using a resistor ladder with each end of the ladder tied to a positive and negative voltage reference input source. If the voltage of either input source fluctuates, a similar fluctuation will result in the DAC output value.

The value of the individual resistors within the ladder can be found in **Section 26.0 "Electrical Specifications"**.

21.3 Low-Power Voltage State

In order for the DAC module to consume the least amount of power, one of the two voltage reference input sources to the resistor ladder must be disconnected. Either the positive voltage source, (V_{SRC+}), or the negative voltage source, (V_{SRC-}) can be disabled.

The negative voltage source is disabled by setting the D1LPS bit in the VREFCON1 register. Clearing the D1LPS bit in the VREFCON1 register disables the positive voltage source.

21.4 Output Clamped to Positive Voltage Source

The DAC output voltage can be set to V_{SRC+} with the least amount of power consumption by performing the following:

- Clearing the D1EN bit in the VREFCON1 register.
- Setting the D1LPS bit in the VREFCON1 register.
- Configuring the D1PSS bits to the proper positive source.
- Configuring the DAC1Rx bits to '11111' in the VREFCON2 register.

This is also the method used to output the voltage level from the FVR to an output pin. See **Section 21.6 "DAC Voltage Reference Output"** for more information.

21.5 Output Clamped to Negative Voltage Source

The DAC output voltage can be set to V_{SRC-} with the least amount of power consumption by performing the following:

- Clearing the D1EN bit in the VREFCON1 register.
- Clearing the DAC1R bit in the VREFCON1 register.
- Configuring the D1PSS bits to the proper negative source.
- Configuring the DAC1Rx bits to '00000' in the VREFCON2 register.

This allows the comparator to detect a zero-crossing while not consuming additional current through the DAC module.

21.6 DAC Voltage Reference Output

The DAC can be output to the DAC1OUT (CV_{REF}) pin by setting the DAC1OE bit of the VREFCON1 register to '1'. Selecting the DAC reference voltage for output on the DAC1OUT pin automatically overrides the digital output buffer and digital input threshold detector functions of that pin. Reading the DAC1OUT pin when it has been configured for DAC reference voltage output will always return a '0'.

Due to the limited current drive capability, a buffer must be used on the DAC voltage reference output for external connections to DAC1OUT. Figure 21-2 shows an example buffering technique.

TABLE 22-4: INITIALIZATION CONDITIONS FOR ALL REGISTERS

Register	Address	Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets, WDT Reset, RESET Instruction, Stack Resets	Wake-up via WDT or Interrupt
TOSU	FFFh	---0 0000	---0 0000	---0 uuuu(3)
TOSH	FFEh	0000 0000	0000 0000	uuuu uuuu(3)
TOSL	FFDh	0000 0000	0000 0000	uuuu uuuu(3)
STKPTR	FFCh	00-0 0000	uu-0 0000	uu-u uuuu(3)
PCLATU	FFBh	---0 0000	---0 0000	---u uuuu
PCLATH	FFAh	0000 0000	0000 0000	uuuu uuuu
PCL	FF9h	0000 0000	0000 0000	PC + 2(2)
TBLPTRU	FF8h	---0 0000	---0 0000	---u uuuu
TBLPTRH	FF7h	0000 0000	0000 0000	uuuu uuuu
TBLPTRL	FF6h	0000 0000	0000 0000	uuuu uuuu
TABLAT	FF5h	0000 0000	0000 0000	uuuu uuuu
PRODH	FF4h	xxxx xxxx	uuuu uuuu	uuuu uuuu
PRODL	FF3h	xxxx xxxx	uuuu uuuu	uuuu uuuu
INTCON	FF2h	0000 000x	0000 000u	uuuu uuuu(1)
INTCON2	FF1h	1111 -1-1	1111 -1-1	uuuu -u-u(1)
INTCON3	FF0h	11-0 0-00	11-0 0-00	uu-u u-uu(1)
INDF0	FEFh	N/A	N/A	N/A
POSTINC0	FEEh	N/A	N/A	N/A
POSTDEC0	FEDh	N/A	N/A	N/A
PREINC0	FECh	N/A	N/A	N/A
PLUSW0	FEBh	N/A	N/A	N/A
FSR0H	FEAh	---- 0000	---- 0000	---- uuuu
FSR0L	FE9h	xxxx xxxx	uuuu uuuu	uuuu uuuu
WREG	FE8h	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF1	FE7h	N/A	N/A	N/A
POSTINC1	FE6h	N/A	N/A	N/A
POSTDEC1	FE5h	N/A	N/A	N/A
PREINC1	FE4h	N/A	N/A	N/A
PLUSW1	FE3h	N/A	N/A	N/A

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition. Shaded cells indicate conditions do not apply for the designated device.

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: See Table 22-3 for Reset value for specific condition.

BCF **Bit Clear f**

Syntax: BCF f, b {,a}

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: $0 \rightarrow$ f<b>

Status Affected: None

Encoding:

1001	bbba	ffff	ffff

Description: Bit 'b' in register 'f' is cleared.
If 'a' is '0', the Access Bank is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `BCF FLAG_REG, 7, 0`

Before Instruction
FLAG_REG = C7h
After Instruction
FLAG_REG = 47h

BN **Branch if Negative**

Syntax: BN n

Operands: $-128 \le n \le 127$

Operation: if NEGATIVE bit is '1'
$(PC) + 2 + 2n \rightarrow PC$

Status Affected: None

Encoding:

1110	0110	nnnn	nnnn

Description: If the NEGATIVE bit is '1', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC + 2 + 2n. This instruction is then a 2-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example:
```
HERE    BN   Jump
```

Before Instruction
PC = address (HERE)
After Instruction
If NEGATIVE = 1;
PC = address (Jump)
If NEGATIVE = 0;
PC = address (HERE + 2)

DAW **Decimal Adjust W Register**

Syntax: DAW

Operands: None

Operation: If [W<3:0> > 9] or [DC = 1] then
(W<3:0>) + 6 → W<3:0>;
else
(W<3:0>) → W<3:0>;

If [W<7:4> + DC > 9] or [C = 1] then
(W<7:4>) + 6 + DC → W<7:4>;
else
(W<7:4>) + DC → W<7:4>

Status Affected: C

Encoding:

0000	0000	0000	0111

Description: DAW adjusts the 8-bit value in W, resulting from the earlier addition of two variables (each in packed BCD format) and produces a correct packed BCD result.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register W	Process Data	Write W

Example1:

```
DAW
```

Before Instruction
W = A5h
C = 0
DC = 0

After Instruction
W = 05h
C = 1
DC = 0

Example 2:

Before Instruction
W = CEh
C = 0
DC = 0

After Instruction
W = 34h
C = 1
DC = 0

DECF **Decrement f**

Syntax: DECF f {,d {,a}}

Operands: 0 ≤ f ≤ 255
d ∈ [0,1]
a ∈ [0,1]

Operation: (f) – 1 → dest

Status Affected: C, DC, N, OV, Z

Encoding:

0000	01da	ffff	ffff

Description: Decrement register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default).
If 'a' is '0', the Access Bank is selected.
If 'a' is '1', the BSR is used to select the GPR bank (default).
If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: DECF CNT, 1, 0

Before Instruction
CNT = 01h
Z = 0

After Instruction
CNT = 00h
Z = 1

RCALL **Relative Call**

Syntax: RCALL n

Operands: $-1024 \leq n \leq 1023$

Operation: $(PC) + 2 \rightarrow TOS$,
$(PC) + 2 + 2n \rightarrow PC$

Status Affected: None

Encoding:

1101	1nnn	nnnn	nnnn

Description: Subroutine call with a jump up to 1K from the current location. First, return address (PC + 2) is pushed onto the stack. Then, add the 2's complement number '2n' to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC + 2 + 2n. This instruction is a 2-cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'n' PUSH PC to stack	Process Data	Write to PC
No operation	No operation	No operation	No operation

Example: HERE RCALL Jump

Before Instruction
PC = Address (HERE)
After Instruction
PC = Address (Jump)
TOS = Address (HERE + 2)

RESET **Reset**

Syntax: RESET

Operands: None

Operation: Reset all registers and flags that are affected by a $\overline{MCLR}$ Reset.

Status Affected: All

Encoding:

0000	0000	1111	1111

Description: This instruction provides a way to execute a $\overline{MCLR}$ Reset by software.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Start Reset	No operation	No operation

Example: RESET

After Instruction
Registers = Reset Value
Flags* = Reset Value

24.2.2 EXTENDED INSTRUCTION SET

ADDFSR **Add Literal to FSR**

Syntax: ADDFSR f, k

Operands: $0 \le k \le 63$
$f \in [0, 1, 2]$

Operation: FSR(f) + k → FSR(f)

Status Affected: None

Encoding:

1110	1000	ffkk	kkkk

Description: The 6-bit literal 'k' is added to the contents of the FSR specified by 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to FSR

Example: `ADDFSR 2, 23h`

Before Instruction
FSR2 = 03FFh
After Instruction
FSR2 = 0422h

ADDULNK **Add Literal to FSR2 and Return**

Syntax: ADDULNK k

Operands: $0 \le k \le 63$

Operation: FSR2 + k → FSR2,
(TOS) → PC

Status Affected: None

Encoding:

1110	1000	11kk	kkkk

Description: The 6-bit literal 'k' is added to the contents of FSR2. A RETURN is then executed by loading the PC with the TOS.
The instruction takes two cycles to execute; a NOP is performed during the second cycle.
This may be thought of as a special case of the ADDFSR instruction, where f = 3 (binary '11'); it operates only on FSR2.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to FSR
No Operation	No Operation	No Operation	No Operation

Example: `ADDULNK 23h`

Before Instruction
FSR2 = 03FFh
PC = 0100h
After Instruction
FSR2 = 0422h
PC = (TOS)

Note: All PIC18 instructions may take an optional label argument preceding the instruction mnemonic for use in symbolic addressing. If a label is used, the instruction syntax then becomes: {label} instruction argument(s).

24.2.3 BYTE-ORIENTED AND BIT-ORIENTED INSTRUCTIONS IN INDEXED LITERAL OFFSET MODE

Note:	Enabling the PIC18 instruction set extension may cause legacy applications to behave erratically or fail entirely.

In addition to eight new commands in the extended set, enabling the extended instruction set also enables Indexed Literal Offset Addressing mode (**Section 3.5.1 "Indexed Addressing with Literal Offset"**). This has a significant impact on the way that many commands of the standard PIC18 instruction set are interpreted.

When the extended set is disabled, addresses embedded in opcodes are treated as literal memory locations: either as a location in the Access Bank ('a' = 0), or in a GPR bank designated by the BSR ('a' = 1). When the extended instruction set is enabled and 'a' = 0, however, a file register argument of 5Fh or less is interpreted as an offset from the pointer value in FSR2 and not as a literal address. For practical purposes, this means that all instructions that use the Access RAM bit as an argument – that is, all byte-oriented and bit-oriented instructions, or almost half of the core PIC18 instructions – may behave differently when the extended instruction set is enabled.

When the content of FSR2 is 00h, the boundaries of the Access RAM are essentially remapped to their original values. This may be useful in creating backward compatible code. If this technique is used, it may be necessary to save the value of FSR2 and restore it when moving back and forth between C and assembly routines in order to preserve the Stack Pointer. Users must also keep in mind the syntax requirements of the extended instruction set (see **Section 24.2.3.1 "Extended Instruction Syntax with Standard PIC18 Commands"**).

Although the Indexed Literal Offset Addressing mode can be very useful for dynamic stack and pointer manipulation, it can also be very annoying if a simple arithmetic operation is carried out on the wrong register. Users who are accustomed to the PIC18 programming must keep in mind that, when the extended instruction set is enabled, register addresses of 5Fh or less are used for Indexed Literal Offset Addressing.

Representative examples of typical byte-oriented and bit-oriented instructions in the Indexed Literal Offset Addressing mode are provided on the following page to show how execution is affected. The operand conditions shown in the examples are applicable to all instructions of these types.

24.2.3.1 Extended Instruction Syntax with Standard PIC18 Commands

When the extended instruction set is enabled, the file register argument, 'f', in the standard byte-oriented and bit-oriented commands is replaced with the literal offset value, 'k'. As already noted, this occurs only when 'f' is less than or equal to 5Fh. When an offset value is used, it must be indicated by square brackets ("[ ]"). As with the extended instructions, the use of brackets indicates to the compiler that the value is to be interpreted as an index or an offset. Omitting the brackets, or using a value greater than 5Fh within brackets, will generate an error in the MPASM™ assembler.

If the index argument is properly bracketed for Indexed Literal Offset Addressing, the Access RAM argument is never specified; it will automatically be assumed to be '0'. This is in contrast to standard operation (extended instruction set disabled) when 'a' is set on the basis of the target address. Declaring the Access RAM bit in this mode will also generate an error in the MPASM assembler.

The destination argument, 'd', functions as before.

In the latest versions of the MPASM assembler, language support for the extended instruction set must be explicitly invoked. This is done with either the command line option, /y, or the PE directive in the source listing.

24.2.4 CONSIDERATIONS WHEN ENABLING THE EXTENDED INSTRUCTION SET

It is important to note that the extensions to the instruction set may not be beneficial to all users. In particular, users who are not writing code that uses a software stack may not benefit from using the extensions to the instruction set.

Additionally, the Indexed Literal Offset Addressing mode may create issues with legacy applications written to the PIC18 assembler. This is because instructions in the legacy code may attempt to address registers in the Access Bank below 5Fh. Since these addresses are interpreted as literal offsets to FSR2 when the instruction set extension is enabled, the application may read or write to the wrong data addresses.

When porting an application to the PIC18(L)F1XK22, it is very important to consider the type of code. A large, re-entrant application that is written in 'C' and would benefit from efficient compilation will do well when using the instruction set extensions. Legacy applications that heavily use the Access Bank will most likely not benefit from using the extended instruction set.

25.6 MPLAB X SIM Software Simulator

The MPLAB X SIM Software Simulator allows code development in a PC-hosted environment by simulating the PIC MCUs and dsPIC DSCs on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a comprehensive stimulus controller. Registers can be logged to files for further run-time analysis. The trace buffer and logic analyzer display extend the power of the simulator to record and track program execution, actions on I/O, most peripherals and internal registers.

The MPLAB X SIM Software Simulator fully supports symbolic debugging using the MPLAB XC Compilers, and the MPASM and MPLAB Assemblers. The software simulator offers the flexibility to develop and debug code outside of the hardware laboratory environment, making it an excellent, economical software development tool.

25.7 MPLAB REAL ICE In-Circuit Emulator System

The MPLAB REAL ICE In-Circuit Emulator System is Microchip's next generation high-speed emulator for Microchip Flash DSC and MCU devices. It debugs and programs all 8, 16 and 32-bit MCU, and DSC devices with the easy-to-use, powerful graphical user interface of the MPLAB X IDE.

The emulator is connected to the design engineer's PC using a high-speed USB 2.0 interface and is connected to the target with either a connector compatible with in-circuit debugger systems (RJ-11) or with the new high-speed, noise tolerant, Low-Voltage Differential Signal (LVDS) interconnection (CAT5).

The emulator is field upgradable through future firmware downloads in MPLAB X IDE. MPLAB REAL ICE offers significant advantages over competitive emulators including full-speed emulation, run-time variable watches, trace analysis, complex breakpoints, logic probes, a ruggedized probe interface and long (up to three meters) interconnection cables.

25.8 MPLAB ICD 3 In-Circuit Debugger System

The MPLAB ICD 3 In-Circuit Debugger System is Microchip's most cost-effective, high-speed hardware debugger/programmer for Microchip Flash DSC and MCU devices. It debugs and programs PIC Flash microcontrollers and dsPIC DSCs with the powerful, yet easy-to-use graphical user interface of the MPLAB IDE.

The MPLAB ICD 3 In-Circuit Debugger probe is connected to the design engineer's PC using a high-speed USB 2.0 interface and is connected to the target with a connector compatible with the MPLAB ICD 2 or MPLAB REAL ICE systems (RJ-11). MPLAB ICD 3 supports all MPLAB ICD 2 headers.

25.9 PICkit 3 In-Circuit Debugger/ Programmer

The MPLAB PICkit 3 allows debugging and programming of PIC and dsPIC Flash microcontrollers at a most affordable price point using the powerful graphical user interface of the MPLAB IDE. The MPLAB PICkit 3 is connected to the design engineer's PC using a full-speed USB interface and can be connected to the target via a Microchip debug (RJ-11) connector (compatible with MPLAB ICD 3 and MPLAB REAL ICE). The connector uses two device I/O pins and the Reset line to implement in-circuit debugging and In-Circuit Serial Programming™ (ICSP™).

25.10 MPLAB PM3 Device Programmer

The MPLAB PM3 Device Programmer is a universal, CE compliant device programmer with programmable voltage verification at V_{DDMIN} and V_{DDMAX} for maximum reliability. It features a large LCD display (128 x 64) for menus and error messages, and a modular, detachable socket assembly to support various package types. The ICSP cable assembly is included as a standard item. In Stand-Alone mode, the MPLAB PM3 Device Programmer can read, verify and program PIC devices without a PC connection. It can also set code protection in this mode. The MPLAB PM3 connects to the host PC via an RS-232 or USB cable. The MPLAB PM3 has high-speed communications and optimized algorithms for quick programming of large memory devices, and incorporates an MMC card for file storage and data applications.

FIGURE 27-17: MEMLOW TYPICAL RC_RUN 31 kHz IDD

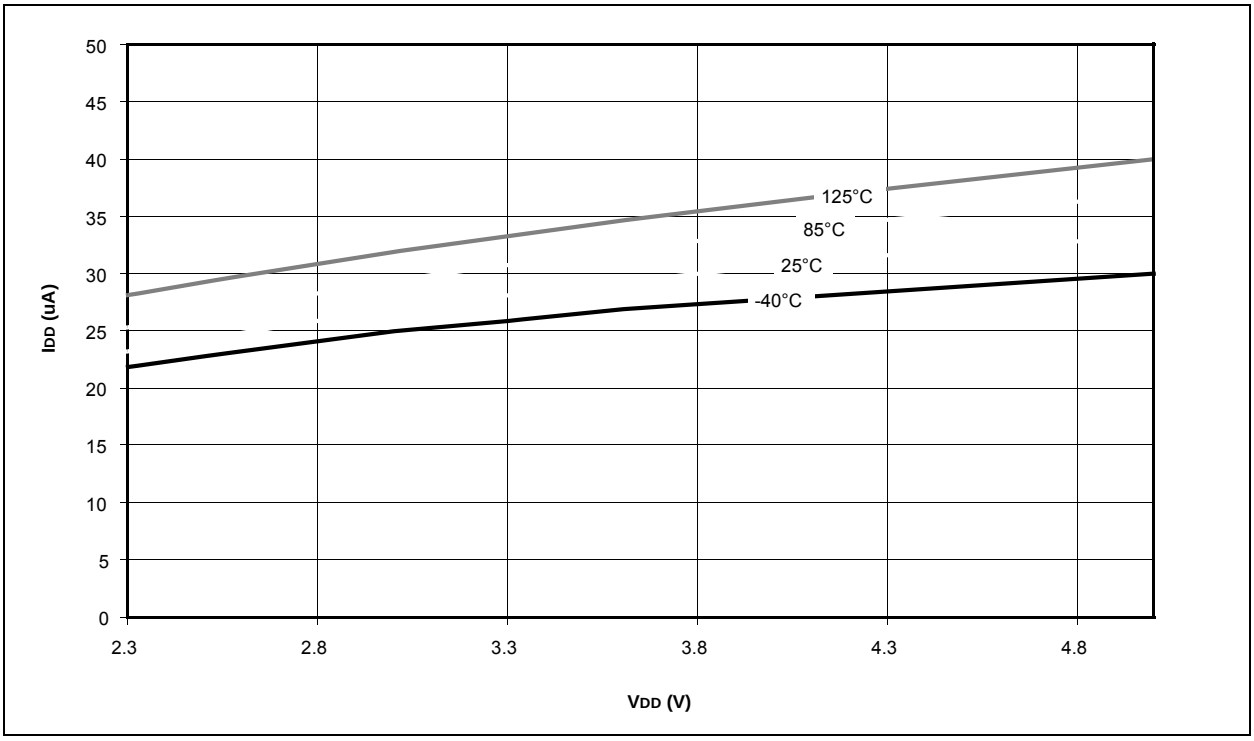
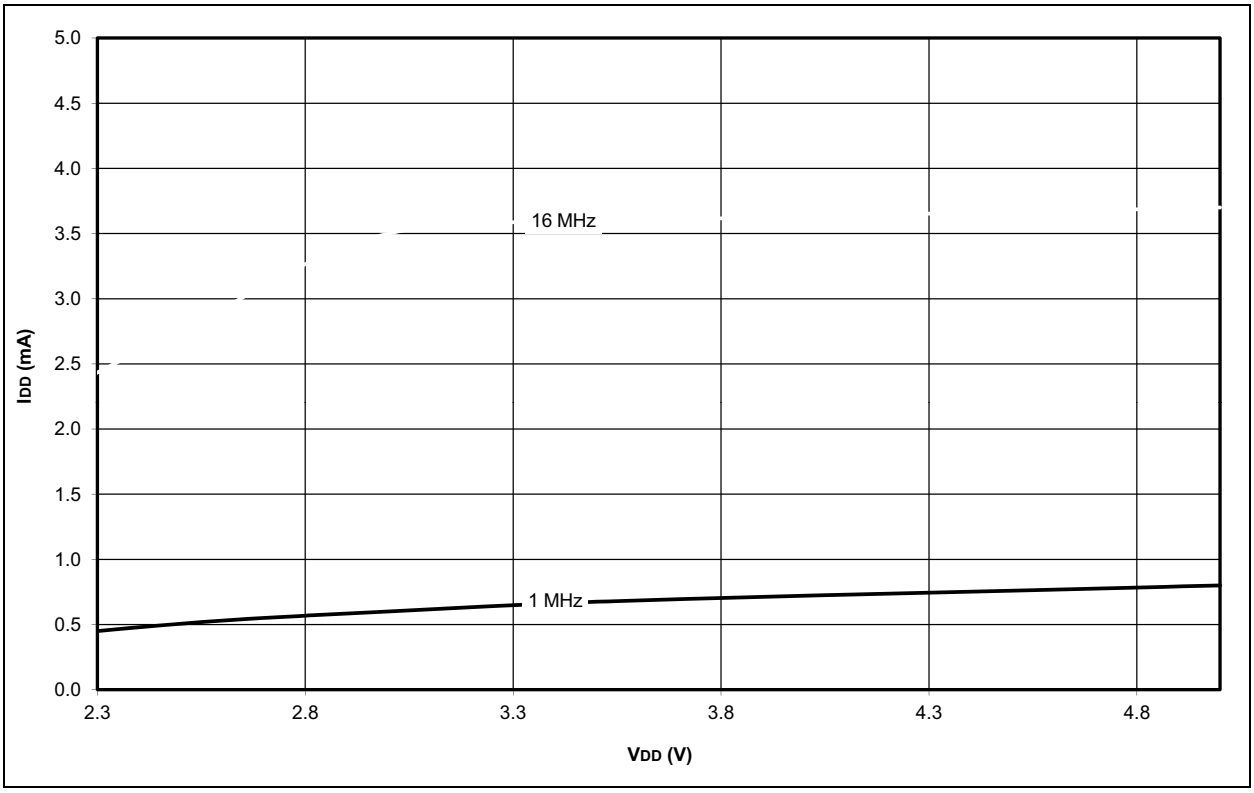


FIGURE 27-18: MEMLOW TYPICAL RC_RUN IDD

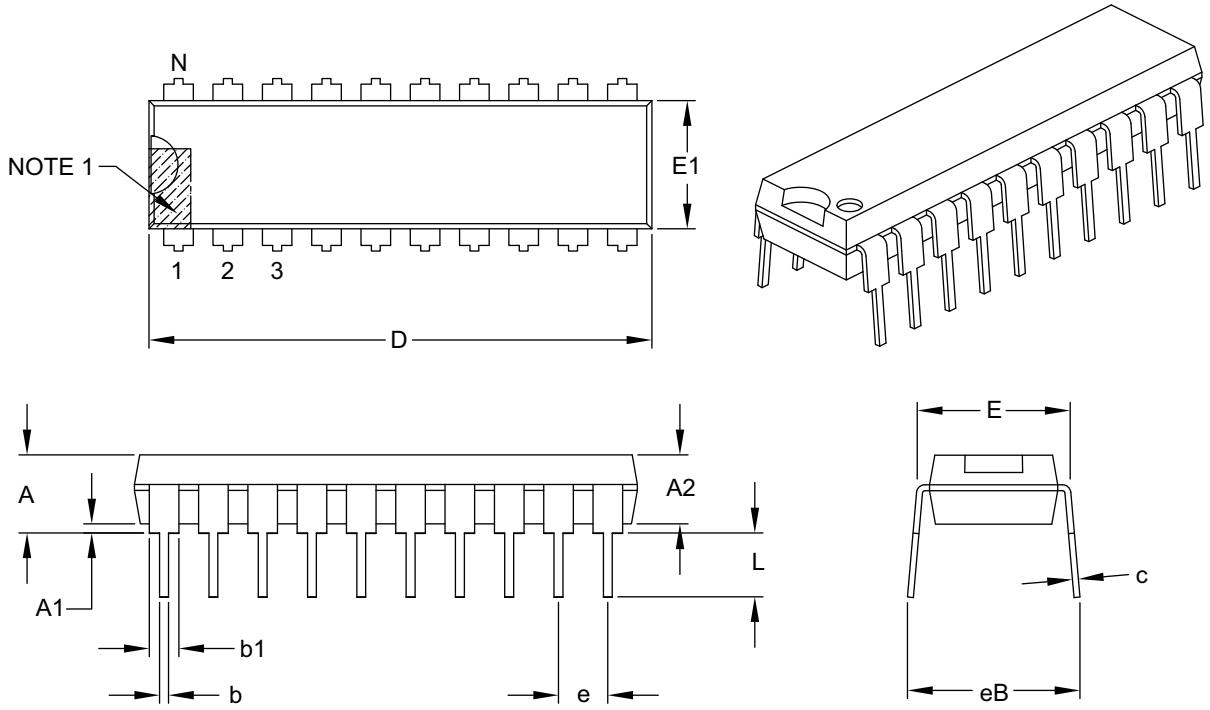


28.2 Package Details

The following sections give the technical details of the packages.

20-Lead Plastic Dual In-Line (P) – 300 mil Body [PDIP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



Units		INCHES		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		20	
Pitch	e		.100 BSC	
Top to Seating Plane	A	–	–	.210
Molded Package Thickness	A2	.115	.130	.195
Base to Seating Plane	A1	.015	–	–
Shoulder to Shoulder Width	E	.300	.310	.325
Molded Package Width	E1	.240	.250	.280
Overall Length	D	.980	1.030	1.060
Tip to Seating Plane	L	.115	.130	.150
Lead Thickness	c	.008	.010	.015
Upper Lead Width	b1	.045	.060	.070
Lower Lead Width	b	.014	.018	.022
Overall Row Spacing §	eB	–	–	.430

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-019B