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	48MHz
Connectivity	I²C, SPI, UART/USART, USB
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	14
Program Memory Size	8KB (4K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	512 x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	A/D 11x10b
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/pic18lf13k50-i-ss

TABLE 2-2: EXAMPLES OF DELAYS DUE TO CLOCK SWITCHING

Switch From	Switch To	Oscillator Delay
Sleep/POR	LFINTOSC HFINTOSC	Oscillator Warm-up Delay (TWARM)
Sleep/POR	LP, XT, HS	1024 clock cycles
Sleep/POR	EC, RC	8 clock cycles

2.9 4x Phase Lock Loop Frequency Multiplier

A Phase-Locked Loop (PLL) circuit is provided as an option for users who wish to use a lower frequency external oscillator or to operate at 32 MHz with the HFINTOSC. The PLL is designed for an input frequency from 4 MHz to 12 MHz. The PLL multiplies its input frequency by a factor of four when the PLL is enabled. This may be useful for customers who are concerned with EMI, due to high-frequency crystals.

Two bits control the PLL: the PLLEN bit of the CONFIG1H Configuration register and the SPLLEN bit of the OSCTUNE register. The PLL is enabled when the PLLEN bit is set and it is under software control when the PLLEN bit is cleared.

TABLE 2-3: PLL CONFIGURATION

PLLEN	SPLLEN	PLL Status
1	x	PLL enabled
0	1	PLL enabled
0	0	PLL disabled

2.9.1 32 MHZ INTERNAL OSCILLATOR FREQUENCY SELECTION

The Internal Oscillator Block can be used with the 4X PLL associated with the External Oscillator Block to produce a 32 MHz internal system clock source. The following settings are required to use the 32 MHz internal clock source:

- The FOSC bits in CONFIG1H must be set to use the INTOSC source as the device system clock (FOSC<3:0> = 1000 or 1001).
- The SCS bits in the OSCCON register must be cleared to use the clock determined by FOSC<3:0> in CONFIG1H (SCS<1:0> = 00).
- The IRCF bits in the OSCCON register must be set to the 8 MHz HFINTOSC set to use (IRCF<2:0> = 110).
- The SPLLEN bit in the OSCTUNE register must be set to enable the 4xPLL, or the PLLEN bit of CONFIG1H must be programmed to a ‘1’.

Note:	When using the PLLEN bit of CONFIG1H, the 4xPLL cannot be disabled by software and the 8 MHz HFINTOSC option will no longer be available.

The 4xPLL is not available for use with the internal oscillator when the SCS bits of the OSCCON register are set to ‘1x’. The SCS bits must be set to ‘00’ to use the 4xPLL with the internal oscillator.

2.10 CPU Clock Divider

The CPU Clock Divider allows the system clock to run at a slower speed than the Low/Full Speed USB module clock while sharing the same clock source. Only the oscillator defined by the settings of the FOSC bits of the CONFIG1H Configuration register may be used with the CPU Clock Divider. The CPU Clock Divider is controlled by the CPUDIV bits of the CONFIG1L Configuration register. Setting the CPUDIV bits will set the system clock to:

- Equal the clock speed of the USB module
- Half the clock speed of the USB module
- One third the clock speed of the USB module
- One fourth the clock speed of the USB module

For more information on the CPU Clock Divider, see Figure 2-1 and Register 24-1 CONFIG1L.

3.1.2.2 Return Stack Pointer (STKPTR)

The STKPTR register (Register 3-1) contains the Stack Pointer value, the STKFUL (stack full) bit and the STKUNF (Stack Underflow) bits. The value of the Stack Pointer can be 0 through 31. The Stack Pointer increments before values are pushed onto the stack and decrements after values are popped off the stack. On Reset, the Stack Pointer value will be zero. The user may read and write the Stack Pointer value. This feature can be used by a Real-Time Operating System (RTOS) for return stack maintenance.

After the PC is pushed onto the stack 31 times (without popping any values off the stack), the STKFUL bit is set. The STKFUL bit is cleared by software or by a POR.

The action that takes place when the stack becomes full depends on the state of the STVREN (Stack Overflow Reset Enable) Configuration bit. (Refer to **Section 24.1 "Configuration Bits"** for a description of the device Configuration bits.) If STVREN is set (default), the 31st push will push the (PC + 2) value onto the stack, set the STKFUL bit and reset the device. The STKFUL bit will remain set and the Stack Pointer will be set to zero.

If STVREN is cleared, the STKFUL bit will be set on the 31st push and the Stack Pointer will increment to 31. Any additional pushes will not overwrite the 31st push and STKPTR will remain at 31.

When the stack has been popped enough times to unload the stack, the next pop will return a value of zero to the PC and sets the STKUNF bit, while the Stack Pointer remains at zero. The STKUNF bit will remain set until cleared by software or until a POR occurs.

Note:	Returning a value of zero to the PC on an underflow has the effect of vectoring the program to the Reset vector, where the stack conditions can be verified and appropriate actions can be taken. This is not the same as a Reset, as the contents of the SFRs are not affected.

3.1.2.3 PUSH and POP Instructions

Since the Top-of-Stack is readable and writable, the ability to push values onto the stack and pull values off the stack without disturbing normal program execution is a desirable feature. The PIC18 instruction set includes two instructions, PUSH and POP, that permit the TOS to be manipulated under software control. TOSU, TOSH and TOSL can be modified to place data or a return address on the stack.

The PUSH instruction places the current PC value onto the stack. This increments the Stack Pointer and loads the current PC value onto the stack.

The POP instruction discards the current TOS by decrementing the Stack Pointer. The previous value pushed onto the stack then becomes the TOS value.

REGISTER 3-1: STKPTR: STACK POINTER REGISTER

R/C-0	R/C-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STKFUL(1)	STKUNF(1)	—	SP4	SP3	SP2	SP1	SP0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented	C = Clearable only bit
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **STKFUL:** Stack Full Flag bit(1)
1 = Stack became full or overflowed
0 = Stack has not become full or overflowed

bit 6 **STKUNF:** Stack Underflow Flag bit(1)
1 = Stack underflow occurred
0 = Stack underflow did not occur

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

bit 4-0 **SP<4:0>:** Stack Pointer Location bits

Note 1: Bit 7 and bit 6 are cleared by user software or by a POR.

3.5.3 MAPPING THE ACCESS BANK IN INDEXED LITERAL OFFSET MODE

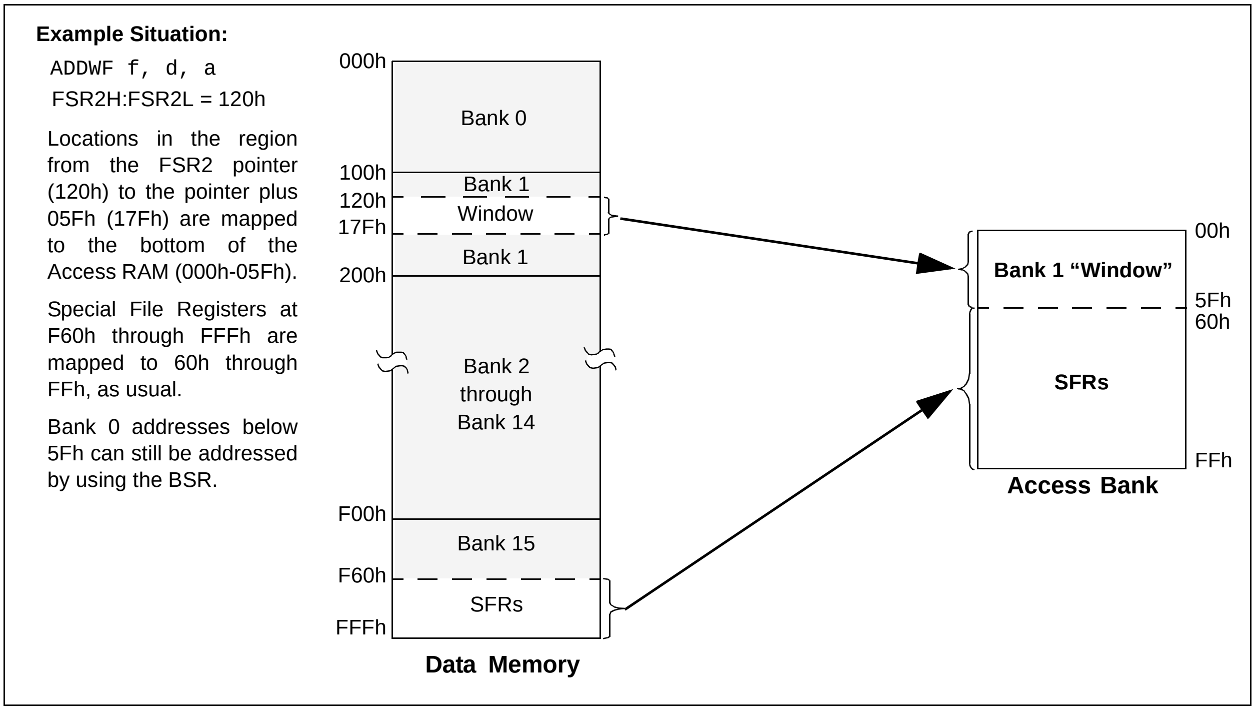
The use of Indexed Literal Offset Addressing mode effectively changes how the first 96 locations of Access RAM (00h to 5Fh) are mapped. Rather than containing just the contents of the bottom section of Bank 0, this mode maps the contents from a user defined "window" that can be located anywhere in the 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 3.3.3 "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 25.2 "Extended Instruction Set"**.

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



EXAMPLE 4-3: WRITING TO FLASH PROGRAM MEMORY

```
                    MOVLW    D'64'                   ; number of bytes in erase block
                    MOVWF    COUNTER
                    MOVLW    BUFFER_ADDR_HIGH        ; point to buffer
                    MOVWF    FSR0H
                    MOVLW    BUFFER_ADDR_LOW
                    MOVWF    FSR0L
                    MOVLW    CODE_ADDR_UPPER         ; Load TBLPTR with the base
                    MOVWF    TBLPTRU                 ; address of the memory block
                    MOVLW    CODE_ADDR_HIGH
                    MOVWF    TBLPTRH
                    MOVLW    CODE_ADDR_LOW
                    MOVWF    TBLPTRL
READ_BLOCK
                    TBLRD*+                          ; read into TABLAT, and inc
                    MOVF     TABLAT, W               ; get data
                    MOVWF    POSTINC0                ; store data
                    DECFSZ   COUNTER                 ; done?
                    BRA      READ_BLOCK              ; repeat
MODIFY_WORD
                    MOVLW    BUFFER_ADDR_HIGH        ; point to buffer
                    MOVWF    FSR0H
                    MOVLW    BUFFER_ADDR_LOW
                    MOVWF    FSR0L
                    MOVLW    NEW_DATA_LOW            ; update buffer word
                    MOVWF    POSTINC0
                    MOVLW    NEW_DATA_HIGH
                    MOVWF    INDF0
ERASE_BLOCK
                    MOVLW    CODE_ADDR_UPPER         ; load TBLPTR with the base
                    MOVWF    TBLPTRU                 ; address of the memory block
                    MOVLW    CODE_ADDR_HIGH
                    MOVWF    TBLPTRH
                    MOVLW    CODE_ADDR_LOW
                    MOVWF    TBLPTRL
                    BSF      EECON1, EEPGD           ; point to Flash program memory
                    BCF      EECON1, CFGS            ; access Flash program memory
                    BSF      EECON1, WREN            ; enable write to memory
                    BSF      EECON1, FREE            ; enable Erase operation
                    BCF      INTCON, GIE             ; disable interrupts
                    MOVLW    55h
Required            MOVWF    EECON2                  ; write 55h
Sequence            MOVLW    0AAh
                    MOVWF    EECON2                  ; write 0AAh
                    BSF      EECON1, WR              ; start erase (CPU stall)
                    BSF      INTCON, GIE             ; re-enable interrupts
                    TBLRD*-                          ; dummy read decrement
                    MOVLW    BUFFER_ADDR_HIGH        ; point to buffer
                    MOVWF    FSR0H
                    MOVLW    BUFFER_ADDR_LOW
                    MOVWF    FSR0L
WRITE_BUFFER_BACK
                    MOVLW    BlockSize               ; number of bytes in holding register
                    MOVWF    COUNTER
                    MOVLW    D'64'/BlockSize         ; number of write blocks in 64 bytes
                    MOVWF    COUNTER2
WRITE_BYTE_TO_HREGS
                    MOVF     POSTINC0, W             ; get low byte of buffer data
                    MOVWF    TABLAT                  ; present data to table latch
                    TBLWT+*                          ; write data, perform a short write
                                                     ; to internal TBLWT holding register.
```

EXAMPLE 4-3: WRITING TO FLASH PROGRAM MEMORY (CONTINUED)

```
                 DECFSZ  COUNTER              ; loop until holding registers are full
                 BRA     WRITE_WORD_TO_HREGS
PROGRAM_MEMORY
                 BSF     EECON1, EEPGD        ; point to Flash program memory
                 BCF     EECON1, CFGS         ; access Flash program memory
                 BSF     EECON1, WREN         ; enable write to memory
                 BCF     INTCON, GIE          ; disable interrupts
                 MOVLW   55h
    Required     MOVWF   EECON2               ; write 55h
    Sequence     MOVLW   0AAh
                 MOVWF   EECON2               ; write 0AAh
                 BSF     EECON1, WR           ; start program (CPU stall)
                 DCFSZ   COUNTER2             ; repeat for remaining write blocks
                 BRA     WRITE_BYTE_TO_HREGS  ;
                 BSF     INTCON, GIE          ; re-enable interrupts
                 BCF     EECON1, WREN         ; disable write to memory
```

4.5.2 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.

4.5.3 UNEXPECTED TERMINATION OF WRITE OPERATION

If a write is terminated by an unplanned event, such as loss of power or an unexpected Reset, the memory location just programmed should be verified and reprogrammed if needed. If the write operation is interrupted by a $\overline{\text{MCLR}}$ Reset or a WDT Time-out Reset during normal operation, the WRERR bit will be set which the user can check to decide whether a rewrite of the location(s) is needed.

4.5.4 PROTECTION AGAINST SPURIOUS WRITES

To protect against spurious writes to Flash program memory, the write initiate sequence must also be followed. See **Section 24.0 “Special Features of the CPU”** for more detail.

4.6 Flash Program Operation During Code Protection

See **Section 24.3 “Program Verification and Code Protection”** for details on code protection of Flash program memory.

TABLE 4-3: REGISTERS ASSOCIATED WITH PROGRAM FLASH MEMORY

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TBLPTRU	—	—	bit 21	Program Memory Table Pointer Upper Byte (TBLPTR<20:16>)					275
TBPLTRH	Program Memory Table Pointer High Byte (TBLPTR<15:8>)								275
TBLPTRL	Program Memory Table Pointer Low Byte (TBLPTR<7:0>)								275
TABLAT	Program Memory Table Latch								275
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RABIE	TMR0IF	INT0IF	RABIF	275
EECON2	EEPROM Control Register 2 (not a physical register)								277
EECON1	EEPGD	CFGS	—	FREE	WRERR	WREN	WR	RD	277
IPR2	OSCFIP	C1IP	C2IP	EEIP	BCLIP	USBIP	TMR3IP	—	278
PIR2	OSCFIF	C1IF	C2IF	EEIF	BCLIF	USBIF	TMR3IF	—	278
PIE2	OSCFIE	C1IE	C2IE	EEIE	BCLIE	USBIE	TMR3IE	—	278

Legend: — = unimplemented, read as ‘0’. Shaded cells are not used during Flash/EEPROM access.

REGISTER 5-1: EECON1: DATA EEPROM CONTROL 1 REGISTER

R/W-x	R/W-x	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	CFGS	—	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit		
S = Bit can be set by software, but not cleared		U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 7 **EEPGD:** Flash Program or Data EEPROM Memory Select bit
1 = Access Flash program memory
0 = Access data EEPROM memory

bit 6 **CFGS:** Flash Program/Data EEPROM or Configuration Select bit
1 = Access Configuration registers
0 = Access Flash program or data EEPROM memory

bit 5 **Unimplemented:** Read as ‘0’

bit 4 **FREE:** Flash Row (Block) Erase Enable bit
1 = Erase the program memory block addressed by TBLPTR on the next WR command (cleared by completion of erase operation)
0 = Perform write-only

bit 3 **WRERR:** Flash Program/Data EEPROM Error Flag bit(1)
1 = A write operation is prematurely terminated (any Reset during self-timed programming in normal operation, or an improper write attempt)
0 = The write operation completed

bit 2 **WREN:** Flash Program/Data EEPROM Write Enable bit
1 = Allows write cycles to Flash program/data EEPROM
0 = Inhibits write cycles to Flash program/data EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a data EEPROM erase/write cycle or a program memory erase cycle or write cycle. (The operation is self-timed and the bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) by software.)
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read (Read takes one cycle. RD is cleared by hardware. The RD bit can only be set (not cleared) by software. RD bit cannot be set when EEPGD = 1 or CFGS = 1.)
0 = Does not initiate an EEPROM read

Note 1: When a WRERR occurs, the EEPGD and CFGS bits are not cleared. This allows tracing of the error condition.

7.0 INTERRUPTS

The PIC18(L)F1XK50 devices have multiple interrupt sources and an interrupt priority feature that allows most interrupt sources to be assigned a high priority level or a low priority level. The high priority interrupt vector is at 0008h and the low priority interrupt vector is at 0018h. A high priority interrupt event will interrupt a low priority interrupt that may be in progress.

There are ten registers which are used to control interrupt operation. These registers are:

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

It is recommended that the Microchip header files supplied with MPLAB® IDE be used for the symbolic bit names in these registers. This allows the assembler/compiler to automatically take care of the placement of these bits within the specified register.

In general, interrupt sources have three bits to control their operation. They are:

- **Flag bit** to indicate that an interrupt event occurred
- **Enable bit** that allows program execution to branch to the interrupt vector address when the flag bit is set
- **Priority bit** to select high priority or low priority

7.1 Mid-Range Compatibility

When the IPEN bit is cleared (default state), the interrupt priority feature is disabled and interrupts are compatible with PIC® microcontroller mid-range devices. In Compatibility mode, the interrupt priority bits of the IPRx registers have no effect. The PEIE bit of the INTCON register is the global interrupt enable for the peripherals. The PEIE bit disables only the peripheral interrupt sources and enables the peripheral interrupt sources when the GIE bit is also set. The GIE bit of the INTCON register is the global interrupt enable which enables all non-peripheral interrupt sources and disables all interrupt sources, including the peripherals. All interrupts branch to address 0008h in Compatibility mode.

7.2 Interrupt Priority

The interrupt priority feature is enabled by setting the IPEN bit of the RCON register. When interrupt priority is enabled the GIE and PEIE global interrupt enable bits of Compatibility mode are replaced by the GIEH high priority, and GIEL low priority, global interrupt enables. When set, the GIEH bit of the INTCON register enables all interrupts that have their associated IPRx register or INTCONx register priority bit set (high priority). When clear, the GIEH bit disables all interrupt sources including those selected as low priority. When clear, the GIEL bit of the INTCON register disables only the interrupts that have their associated priority bit cleared (low priority). When set, the GIEL bit enables the low priority sources when the GIEH bit is also set.

When the interrupt flag, enable bit and appropriate global interrupt enable bit are all set, the interrupt will vector immediately to address 0008h for high priority, or 0018h for low priority, depending on level of the interrupting source's priority bit. Individual interrupts can be disabled through their corresponding interrupt enable bits.

7.3 Interrupt Response

When an interrupt is responded to, the global interrupt enable bit is cleared to disable further interrupts. The GIE bit is the global interrupt enable when the IPEN bit is cleared. When the IPEN bit is set, enabling interrupt priority levels, the GIEH bit is the high priority global interrupt enable and the GIEL bit is the low priority global interrupt enable. High priority interrupt sources can interrupt a low priority interrupt. Low priority interrupts are not processed while high priority interrupts are in progress.

The return address is pushed onto the stack and the PC is loaded with the interrupt vector address (0008h or 0018h). Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits in the INTCONx and PIRx registers. The interrupt flag bits must be cleared by software before re-enabling interrupts to avoid repeating the same interrupt.

The "return from interrupt" instruction, `RETFIE`, exits the interrupt routine and sets the GIE bit (GIEH or GIEL if priority levels are used), which re-enables interrupts.

For external interrupt events, such as the INT pins or the PORTB interrupt-on-change, the interrupt latency will be three to four instruction cycles. The exact latency is the same for one-cycle or 2-cycle instructions. Individual interrupt flag bits are set, regardless of the status of their corresponding enable bits or the global interrupt enable bit.

TABLE 9-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
PORTA	—	—	RA5(1)	RA4(1)	RA3(2)	—	RA1(3)	RA0(3)	278
LATA	—	—	LATA5(1)	LATA4(1)	—	—	—	—	278
TRISA	—	—	TRISA5(1)	TRISA4(1)	—	—	—	—	278
ANSEL	ANS7	ANS6	ANS5	ANS4	ANS3	—	—	—	278
SLRCON	—	—	—	—	—	SLRC	SLRB	SLRA	278
IOCA	—	—	IOCA5	IOCA4	IOCA3(2)	—	IOCA1(3)	IOCA0(3)	278
WPUA	—	—	WPUA5	WPUA4	WPUA3(2)	—	—	—	275
UCON	—	PPBRST	SE0	PKTDIS	USBEN	RESUME	SUSPND	—	278
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RABIE	TMR0IF	INT0IF	RABIF	275
INTCON2	$\overline{\text{RABPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RABIP	275

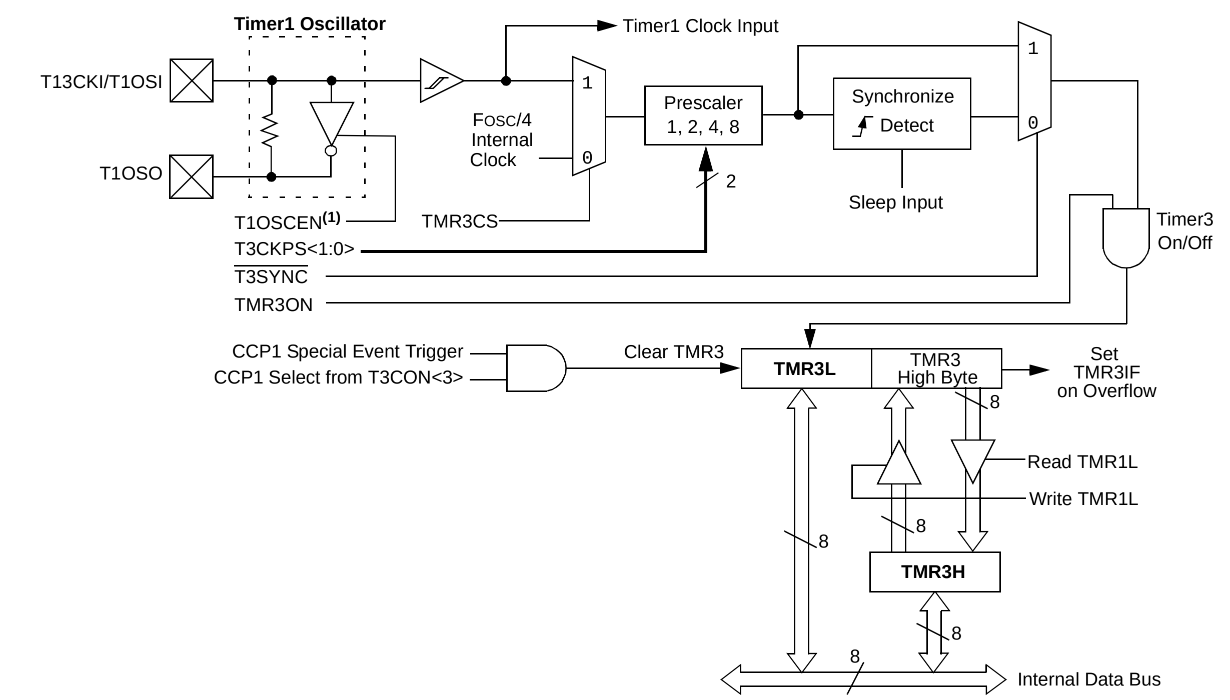
Legend: — = unimplemented, read as '0'. Shaded cells are not used by PORTA.

Note 1: RA<5:4> and their associated latch and data direction bits are enabled as I/O pins based on oscillator configuration; otherwise, they are read as '0'.

2: Implemented only when Master Clear functionality is disabled (MCLRE Configuration bit = 0).

3: RA1 and RA0 are only available as port pins when the USB module is disabled (UCON<3> = 0).

FIGURE 13-2: TIMER3 BLOCK DIAGRAM (16-BIT READ/WRITE MODE)



Note 1: When enable bit, T1OSCEN, is cleared, the inverter and feedback resistor are turned off to eliminate power drain.

13.2 Timer3 16-Bit Read/Write Mode

Timer3 can be configured for 16-bit reads and writes (see Figure 13-2). When the RD16 control bit of the T3CON register is set, the address for TMR3H is mapped to a buffer register for the high byte of Timer3. A read from TMR3L will load the contents of the high byte of Timer3 into the Timer3 High Byte Buffer register. This provides the user with the ability to accurately read all 16 bits of Timer1 without having to determine whether a read of the high byte, followed by a read of the low byte, has become invalid due to a rollover between reads.

A write to the high byte of Timer3 must also take place through the TMR3H Buffer register. The Timer3 high byte is updated with the contents of TMR3H when a write occurs to TMR3L. This allows a user to write all 16 bits to both the high and low bytes of Timer3 at once.

The high byte of Timer3 is not directly readable or writable in this mode. All reads and writes must take place through the Timer3 High Byte Buffer register.

Writes to TMR3H do not clear the Timer3 prescaler. The prescaler is only cleared on writes to TMR3L.

13.3 Using the Timer1 Oscillator as the Timer3 Clock Source

The Timer1 internal oscillator may be used as the clock source for Timer3. The Timer1 oscillator is enabled by setting the T1OSCEN bit of the T1CON register. To use it as the Timer3 clock source, the TMR3CS bit must also be set. As previously noted, this also configures Timer3 to increment on every rising edge of the oscillator source.

The Timer1 oscillator is described in **Section 11.0 “Timer1 Module”**.

13.4 Timer3 Interrupt

The TMR3 register pair (TMR3H:TMR3L) increments from 0000h to FFFFh and overflows to 0000h. The Timer3 interrupt, if enabled, is generated on overflow and is latched in interrupt flag bit, TMR3IF of the PIR2 register. This interrupt can be enabled or disabled by setting or clearing the Timer3 Interrupt Enable bit, TMR3IE of the PIE2 register.

14.4.2.1 Direction Change in Full-Bridge Mode

In the Full-Bridge mode, the P1M1 bit in the CCP1CON register allows users to control the forward/reverse direction. When the application firmware changes this direction control bit, the module will change to the new direction on the next PWM cycle.

A direction change is initiated in software by changing the P1M1 bit of the CCP1CON register. The following sequence occurs prior to the end of the current PWM period:

- The modulated outputs (P1B and P1D) are placed in their inactive state.
- The associated unmodulated outputs (P1A and P1C) are switched to drive in the opposite direction.
- PWM modulation resumes at the beginning of the next period.

See Figure 14-10 for an illustration of this sequence.

The Full-Bridge mode does not provide dead-band delay. As one output is modulated at a time, dead-band delay is generally not required. There is a situation where dead-band delay is required. This situation occurs when both of the following conditions are true:

1. The direction of the PWM output changes when the duty cycle of the output is at or near 100%.
2. The turn off time of the power switch, including the power device and driver circuit, is greater than the turn on time.

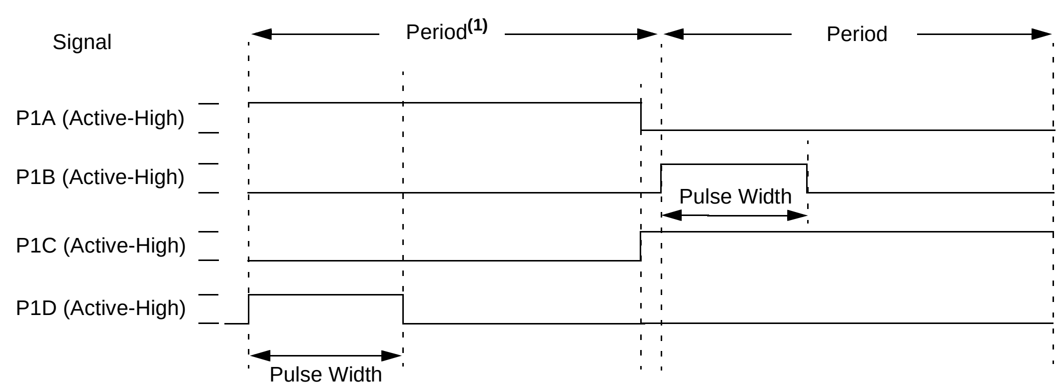
Figure 14-11 shows an example of the PWM direction changing from forward to reverse, at a near 100% duty cycle. In this example, at time t1, the output P1A and P1D become inactive, while output P1C becomes active. Since the turn off time of the power devices is longer than the turn on time, a shoot-through current will flow through power devices QC and QD (see Figure 14-8) for the duration of 't'. The same phenomenon will occur to power devices QA and QB for PWM direction change from reverse to forward.

If changing PWM direction at high duty cycle is required for an application, two possible solutions for eliminating the shoot-through current are:

1. Reduce PWM duty cycle for one PWM period before changing directions.
2. Use switch drivers that can drive the switches off faster than they can drive them on.

Other options to prevent shoot-through current may exist.

FIGURE 14-10: EXAMPLE OF PWM DIRECTION CHANGE



Note 1: The direction bit P1M1 of the CCP1CON register is written any time during the PWM cycle.

14.4.8 OPERATION IN POWER-MANAGED MODES

In Sleep mode, all clock sources are disabled. Timer2 will not increment and the state of the module will not change. If the ECCP pin is driving a value, it will continue to drive that value. When the device wakes up, it will continue from this state. If Two-Speed Start-ups are enabled, the initial start-up frequency from HFINTOSC and the postscaler may not be stable immediately.

In PRI_IDLE mode, the primary clock will continue to clock the ECCP module without change. In all other power-managed modes, the selected power-managed mode clock will clock Timer2. Other power-managed mode clocks will most likely be different than the primary clock frequency.

14.4.8.1 Operation with Fail-Safe Clock Monitor

If the Fail-Safe Clock Monitor is enabled, a clock failure will force the device into the RC_RUN Power-Managed mode and the OSCFIF bit of the PIR2 register will be set. The ECCP will then be clocked from the internal oscillator clock source, which may have a different clock frequency than the primary clock.

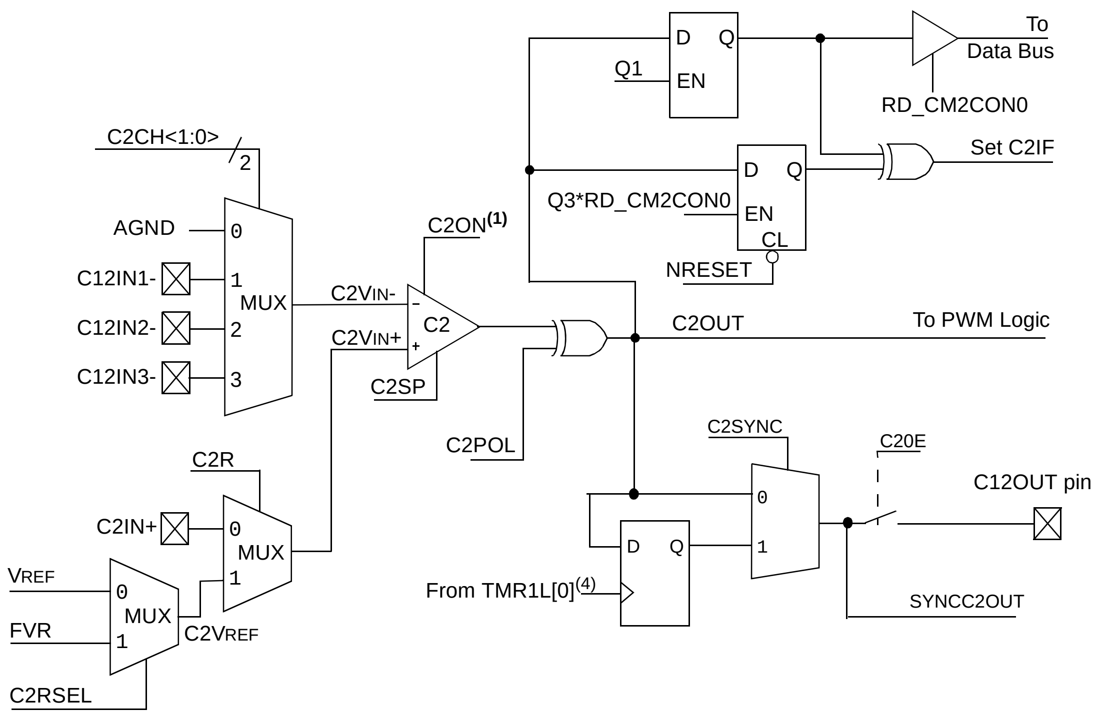
See the previous section for additional details.

14.4.9 EFFECTS OF A RESET

Both Power-on Reset and subsequent Resets will force all ports to Input mode and the CCP registers to their Reset states.

This forces the enhanced CCP module to reset to a state compatible with the standard CCP module.

FIGURE 18-3: COMPARATOR C2 SIMPLIFIED BLOCK DIAGRAM



Note 1: When C2ON = 0, the C2 comparator will produce a ‘0’ output to the XOR Gate.
2: Q1 and Q3 are phases of the four-phase system clock (FOSC).
3: Q1 is held high during Sleep mode.
4: Positive going pulse generated on both falling and rising edges of the bit.

REGISTER 24-13: DEVID1: DEVICE ID REGISTER 1 FOR PIC18(L)F1XK50

R	R	R	R	R	R	R	R
DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0
bit 7							bit 0

Legend:	
R = Readable bit	U = Unimplemented bit, read as ‘0’
-n = Value when device is unprogrammed	C = Clearable only bit

bit 7-5 **DEV<2:0>:** Device ID bits
`010` = PIC18F13K50
`011` = PIC18F14K50

bit 4-0 **REV<4:0>:** Revision ID bits
These bits are used to indicate the device revision.

REGISTER 24-14: DEVID2: DEVICE ID REGISTER 2 FOR PIC18(L)F1XK50

R	R	R	R	R	R	R	R
DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3
bit 7							bit 0

Legend:	
R = Readable bit	U = Unimplemented bit, read as ‘0’
-n = Value when device is unprogrammed	C = Clearable only bit

bit 7-0 **DEV<10:3>:** Device ID bits
These bits are used with the DEV<2:0> bits in the Device ID Register 1 to identify the part number.
`0010 0000` = PIC18(L)F1XK50 devices

Note 1: These values for DEV<10:3> may be shared with other devices. The specific device is always identified by using the entire DEV<10:0> bit sequence.

BCF	Bit Clear f
Syntax:	BCF f, b {,a}
Operands:	0 ≤ f ≤ 255 0 ≤ b ≤ 7 a ∈ [0,1]
Operation:	0 → 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 ≤ 95 (5Fh). See **Section 25.2.3 "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 ≤ n ≤ 127
Operation:	if NEGATIVE bit is '1' (PC) + 2 + 2n → 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`)

COMF — Complement f

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

Operands: $0 \leq f \leq 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: $(\bar{f}) \rightarrow$ dest

Status Affected: N, Z

Encoding:

0001	11da	ffff	ffff

Description: The contents of register ‘f’ are complemented. 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 $f \leq 95$ (5Fh). See **Section 25.2.3 “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: `COMF REG, 0, 0`

Before Instruction
REG = 13h
After Instruction
REG = 13h
W = ECh

CPFSEQ — Compare f with W, skip if f = W

Syntax: CPFSEQ f {,a}

Operands: $0 \leq f \leq 255$
$a \in [0,1]$

Operation: (f) – (W),
skip if (f) = (W)
(unsigned comparison)

Status Affected: None

Encoding:

0110	001a	ffff	ffff

Description: Compares the contents of data memory location ‘f’ to the contents of W by performing an unsigned subtraction.
If ‘f’ = W, then the fetched instruction is discarded and a NOP is executed instead, making this a 2-cycle instruction.
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 $f \leq 95$ (5Fh). See **Section 25.2.3 “Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode”** for details.

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register ‘f’	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:

```
HERE      CPFSEQ REG, 0
NEQUAL    :
EQUAL     :
```

Before Instruction
PC Address = HERE
W = ?
REG = ?
After Instruction
If REG = W;
PC = Address (EQUAL)
If REG ≠ W;
PC = Address (NEQUAL)

XORWF Exclusive OR W with f

Syntax:	XORWF f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) .XOR. (f) $\rightarrow$ dest
Status Affected:	N, Z
Encoding:	`0001` | `10da` | `ffff` | `ffff`
Description:	Exclusive OR the contents of W with register 'f'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in the 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 25.2.3 "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: `XORWF REG, 1, 0`

Before Instruction
- REG = AFh
- W = B5h

After Instruction
- REG = 1Ah
- W = B5h

CALLW — Subroutine Call Using WREG

Syntax:	CALLW
Operands:	None
Operation:	(PC + 2) $\rightarrow$ TOS, (W) $\rightarrow$ PCL, (PCLATH) $\rightarrow$ PCH, (PCLATU) $\rightarrow$ PCU
Status Affected:	None
Encoding:	`0000 0000 0001 0100`
Description	First, the return address (PC + 2) is pushed onto the return stack. Next, the contents of W are written to PCL; the existing value is discarded. Then, the contents of PCLATH and PCLATU are latched into PCH and PCU, respectively. The second cycle is executed as a `NOP` instruction while the new next instruction is fetched. Unlike `CALL`, there is no option to update W, Status or BSR.
Words:	1
Cycles:	2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read WREG	PUSH PC to stack	No operation
No operation	No operation	No operation	No operation

Example: `HERE CALLW`

Before Instruction

PC	=	address (HERE)
PCLATH	=	10h
PCLATU	=	00h
W	=	06h

After Instruction

PC	=	001006h
TOS	=	address (HERE + 2)
PCLATH	=	10h
PCLATU	=	00h
W	=	06h

MOVSF — Move Indexed to f

Syntax:	MOVSF [z_s], f_d
Operands:	$0 \le z_s \le 127$ $0 \le f_d \le 4095$
Operation:	$((FSR2) + z_s) \rightarrow f_d$
Status Affected:	None
Encoding: 1st word (source) 2nd word (destin.)	 `1110 1011 0zzz zzzz`$_s$ `1111 ffff ffff ffff`$_d$
Description:	The contents of the source register are moved to destination register 'f_d'. The actual address of the source register is determined by adding the 7-bit literal offset 'z_s' in the first word to the value of FSR2. The address of the destination register is specified by the 12-bit literal 'f_d' in the second word. Both addresses can be anywhere in the 4096-byte data space (000h to FFFh). The `MOVSF` instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register. If the resultant source address points to an indirect addressing register, the value returned will be 00h.
Words:	2
Cycles:	2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Determine source addr	Determine source addr	Read source reg
Decode	No operation No dummy read	No operation	Write register 'f' (dest)

Example: `MOVSF [05h], REG2`

Before Instruction

FSR2	=	80h
Contents of 85h	=	33h
REG2	=	11h

After Instruction

FSR2	=	80h
Contents of 85h	=	33h
REG2	=	33h

ADDWF **ADD W to Indexed (Indexed Literal Offset mode)**

Syntax:	ADDWF [k] {,d}
Operands:	0 ≤ k ≤ 95 d ∈ [0,1]
Operation:	(W) + ((FSR2) + k) → dest
Status Affected:	N, OV, C, DC, Z
Encoding:	`0010` `01d0` `kkkk` `kkkk`
Description:	The contents of W are added to the contents of the register indicated by FSR2, offset by the value 'k'. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f' (default).
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `ADDWF [OFST] , 0`

Before Instruction
- W = 17h
- OFST = 2Ch
- FSR2 = 0A00h
- Contents of 0A2Ch = 20h

After Instruction
- W = 37h
- Contents of 0A2Ch = 20h

BSF **Bit Set Indexed (Indexed Literal Offset mode)**

Syntax:	BSF [k], b
Operands:	0 ≤ f ≤ 95 0 ≤ b ≤ 7
Operation:	1 → ((FSR2) + k)<b>
Status Affected:	None
Encoding:	`1000` `bbb0` `kkkk` `kkkk`
Description:	Bit 'b' of the register indicated by FSR2, offset by the value 'k', is set.
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `BSF [FLAG_OFST], 7`

Before Instruction
- FLAG_OFST = 0Ah
- FSR2 = 0A00h
- Contents of 0A0Ah = 55h

After Instruction
- Contents of 0A0Ah = D5h

SETF **Set Indexed (Indexed Literal Offset mode)**

Syntax:	SETF [k]
Operands:	0 ≤ k ≤ 95
Operation:	FFh → ((FSR2) + k)
Status Affected:	None
Encoding:	`0110` `1000` `kkkk` `kkkk`
Description:	The contents of the register indicated by FSR2, offset by 'k', are set to FFh.
Words:	1
Cycles:	1

Q Cycle Activity:

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

Example: `SETF [OFST]`

Before Instruction
- OFST = 2Ch
- FSR2 = 0A00h
- Contents of 0A2Ch = 00h

After Instruction
- Contents of 0A2Ch = FFh

TABLE 27-5: MEMORY PROGRAMMING REQUIREMENTS

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ T_A ≤ +125°C				
Param. No.	**Sym.**	**Characteristic**	**Min.**	**Typ.†**	**Max.**	**Units**	**Conditions**
		Internal Program Memory Programming Specifications(1)					
D110	V_{PP}	Voltage on $\overline{MCLR}$/V_{PP}/RA3 pin	8	—	9	V	**(Note 3, Note 4)**
D113	I_{DDP}	Supply Current during Programming	—	—	10	mA	
		Data EEPROM Memory(2)					
D120	E_D	Byte Endurance	100K	—	—	E/W	-40°C to +85°C
D121	V_{DRW}	V_{DD} for Read/Write	V_{DDMIN}	—	V_{DDMAX}	V	Using EECON to read/write
D122	T_{DEW}	Erase/Write Cycle Time	—	3	4	ms	
D123	T_{RETD}	Characteristic Retention	—	40	—	Year	Provided no other specifications are violated
D124	T_{REF}	Number of Total Erase/Write Cycles before Refresh(2)	1M	10M	—	E/W	-40°C to +85°C
D130		**Program Flash Memory**					
	E_P	Cell Endurance	10k	—	—	E/W	Temperature during programming: 10°C ≤ T_A ≤ 40°C
D131	V_{PR}	V_{DD} for Read	V_{DDMIN}	—	V_{DDMAX}	V	
D131A		Voltage on $\overline{MCLR}$/V_{PP} during Erase/Program	8.0	—	9.0	V	Temperature during programming: 10°C ≤ T_A ≤ 40°C
D131B	V_{BE}	V_{DD} for Bulk Erase	2.7	—	V_{DDMAX}	V	Temperature during programming: 10°C ≤ T_A ≤ 40°C
D132	V_{PEW}	V_{DD} for Write or Row Erase	2.2 V_{DDMIN}	— —	V_{DDMAX} V_{DDMAX}	V	PIC18LF1XK50 PIC18F1XK50
D132A	I_{PPPGM}	Current on $\overline{MCLR}$/V_{PP} during Erase/Write	—	1.0	—	mA	Temperature during programming: 10°C ≤ T_A ≤ 40°C
D132B	I_{DDPGM}	Current on V_{DD} during Erase/Write	—	5.0	—	mA	Temperature during programming: 10°C ≤ T_A ≤ 40°C
D133	T_{PEW}	Erase/Write cycle time	—	2.0	2.8	ms	Temperature during programming: 10°C ≤ T_A ≤ 40°C
D134	T_{RETD}	Characteristic Retention	—	40	—	Year	Provided no other specifications are violated

† Data in “Typ.” column is at 3.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: These specifications are for programming the on-chip program memory through the use of table write instructions.
2: Refer to **Section 5.8 “Using the Data EEPROM”** for a more detailed discussion on data EEPROM endurance.
3: Required only if single-supply programming is disabled.
4: The MPLAB ICD 2 does not support variable V_{PP} output. Circuitry to limit the ICD 2 V_{PP} voltage must be placed between the ICD 2 and target system when programming or debugging with the ICD 2.

PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	[X][(1)]	X	/XX	XXX
Device	Tape and Reel Option	Temperature Range	Package	Pattern

Device:	PIC18F13K50[(1)], PIC18F14K50[(1)], PIC18LF13K50[(1)], PIC18LF14K50	
Tape and Reel Option:	Blank = Standard packaging (tube or tray) T = Tape and Reel[(1)]	
Temperature Range:	I = -40°C to +85°C (Industrial) E = -40°C to +125°C (Extended)	
Package:	P = PDIP SO = SOIC SS = SSOP MQ = QFN	
Pattern:	QTP, SQTP, Code or Special Requirements (blank otherwise)	

Examples:

a) PIC18F14K50-E/P 301 = Extended temp., PDIP package, Extended VDD limits, QTP pattern #301.

b) PIC18LF14K50-E/SO = Extended temp., SOIC package.

c) PIC18LF14K50-E/P = Extended temp., PDIP package.

d) PIC18LF14K50-E/MQ = Extended temp., QFN package.

e) PIC18F14K50-I/P = Industrial temp., PDIP package.

Note 1: Tape and Reel identifier only appears in the catalog part number description. This identifier is used for ordering purposes and is not printed on the device package. Check with your Microchip Sales Office for package availability with the Tape and Reel option.