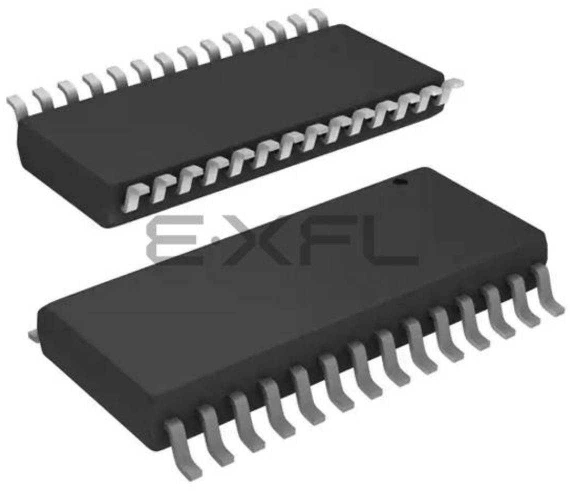
Microchip Technology - PIC18LF2550-I/SO Datasheet



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, HLVD, POR, PWM, WDT
Number of I/O	24
Program Memory Size	32KB (16K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	2V ~ 5.5V
Data Converters	A/D 10x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SOIC (0.295", 7.50mm Width)
Supplier Device Package	28-SOIC
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18lf2550-i-so

Table of Contents

TABLE 1-1: DEVICE FEATURES

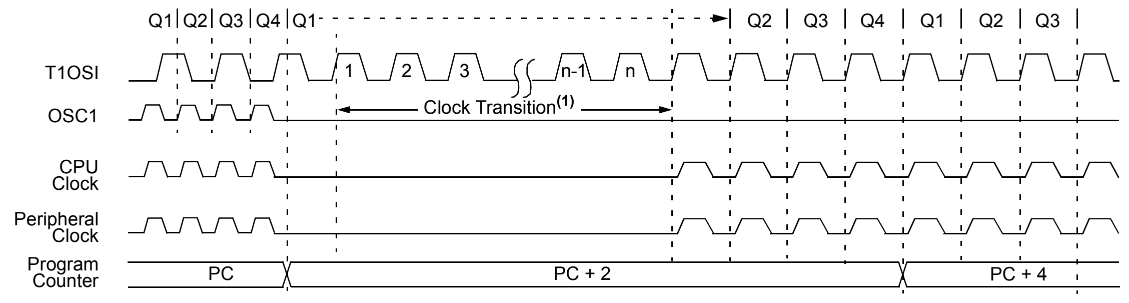
Features	PIC18F2455	PIC18F2550	PIC18F4455	PIC18F4550
Operating Frequency	DC – 48 MHz	DC – 48 MHz	DC – 48 MHz	DC – 48 MHz
Program Memory (Bytes)	24576	32768	24576	32768
Program Memory (Instructions)	12288	16384	12288	16384
Data Memory (Bytes)	2048	2048	2048	2048
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	19	19	20	20
I/O Ports	Ports A, B, C, (E)	Ports A, B, C, (E)	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	1	1
Enhanced Capture/ Compare/PWM Modules	0	0	1	1
Serial Communications	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART
Universal Serial Bus (USB) Module	1	1	1	1
Streaming Parallel Port (SPP)	No	No	Yes	Yes
10-Bit Analog-to-Digital Module	10 Input Channels	10 Input Channels	13 Input Channels	13 Input Channels
Comparators	2	2	2	2
Resets (and Delays)	POR, BOR, `RESET` Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{\text{MCLR}}$ (optional), WDT	POR, BOR, `RESET` Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{\text{MCLR}}$ (optional), WDT	POR, BOR, `RESET` Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{\text{MCLR}}$ (optional), WDT	POR, BOR, `RESET` Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{\text{MCLR}}$ (optional), WDT
Programmable Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled
Packages	28-Pin PDIP 28-Pin SOIC	28-Pin PDIP 28-Pin SOIC	40-Pin PDIP 44-Pin QFN 44-Pin TQFP	40-Pin PDIP 44-Pin QFN 44-Pin TQFP

SEC_RUN mode is entered by setting the SCS1:SCS0 bits to '01'. The device clock source is switched to the Timer1 oscillator (see Figure 3-1), the primary oscillator is shut down, the T1RUN bit (T1CON<6>) is set and the OSTS bit is cleared.

Note:	The Timer1 oscillator should already be running prior to entering SEC_RUN mode. If the T1OSCEN bit is not set when the SCS1:SCS0 bits are set to '01', entry to SEC_RUN mode will not occur. If the Timer1 oscillator is enabled but not yet running, device clocks will be delayed until the oscillator has started. In such situations, initial oscillator operation is far from stable and unpredictable operation may result.

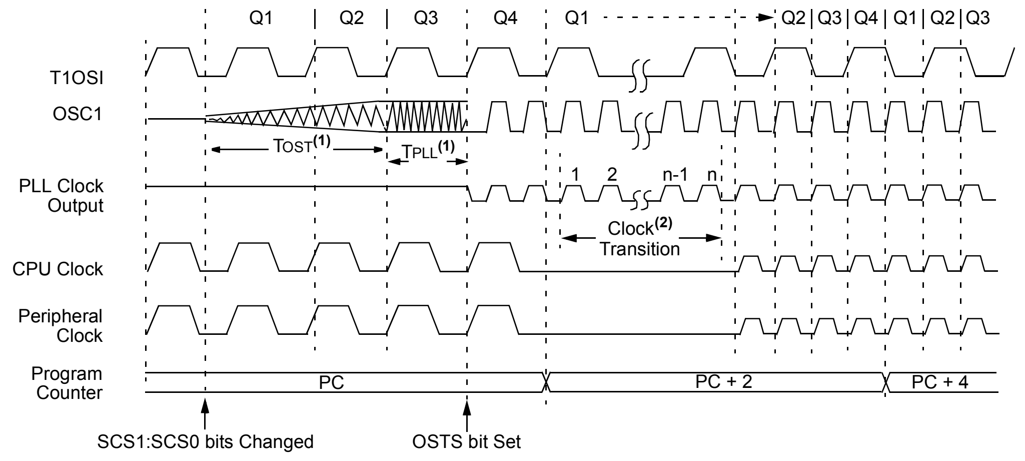
On transitions from SEC_RUN mode to PRI_RUN, the peripherals and CPU continue to be clocked from the Timer1 oscillator while the primary clock is started. When the primary clock becomes ready, a clock switch back to the primary clock occurs (see Figure 3-2). When the clock switch is complete, the T1RUN bit is cleared, the OSTS bit is set and the primary clock is providing the clock. The IDLEN and SCS bits are not affected by the wake-up; the Timer1 oscillator continues to run.

FIGURE 3-1: TRANSITION TIMING FOR ENTRY TO SEC_RUN MODE



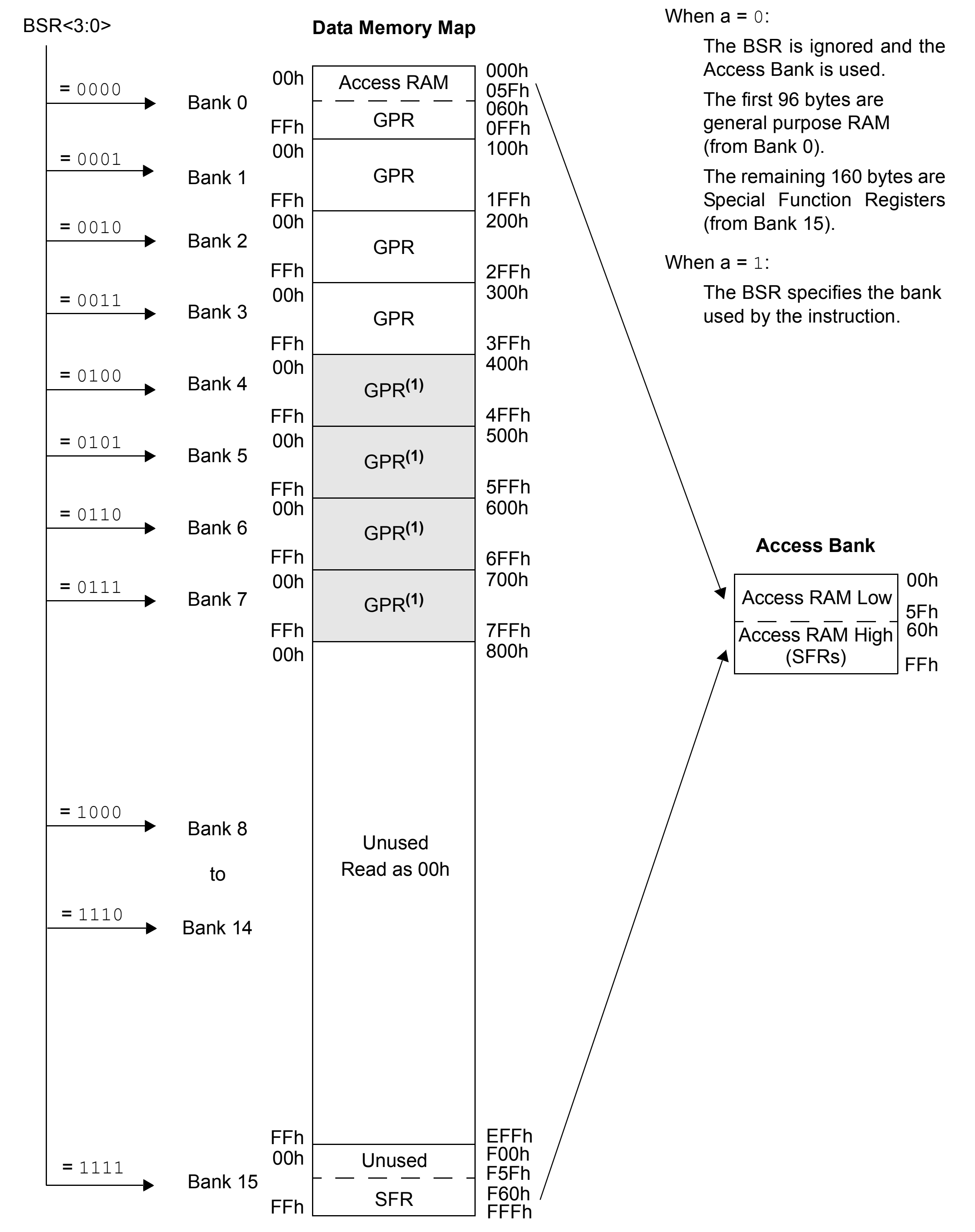
Note 1: Clock transition typically occurs within 2-4 TOSC.

FIGURE 3-2: TRANSITION TIMING FROM SEC_RUN MODE TO PRI_RUN MODE (HSPLL)



Note 1: TOST = 1024 TOSC; TPLL = 2 ms (approx). These intervals are not shown to scale.
2: Clock transition typically occurs within 2-4 TOSC.

FIGURE 5-5: DATA MEMORY MAP



Note 1: These banks also serve as RAM buffer for USB operation. See **Section 5.3.1 "USB RAM"** for more information.

NOTES:

9.5 IPR Registers

The IPR registers contain the individual priority bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Priority registers (IPR1 and IPR2). Using the priority bits requires that the Interrupt Priority Enable (IPEN) bit be set.

REGISTER 9-8: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
SPPIP[(1)]	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
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 **SPPIP:** Streaming Parallel Port Read/Write Interrupt Priority bit[(1)]
1 = High priority
0 = Low priority

bit 6 **ADIP:** A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RCIP:** EUSART Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXIP:** EUSART Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSPIP:** Master Synchronous Serial Port Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **CCP1IP:** CCP1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR2IP:** TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP:** TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

Note 1: This bit is reserved on 28-pin devices; always maintain this bit clear.

10.5 PORTE, TRISE and LATE Registers

Depending on the particular PIC18F2455/2550/4455/4550 device selected, PORTE is implemented in two different ways.

For 40/44-pin devices, PORTE is a 4-bit wide port. Three pins (RE0/AN5/CK1SPP, RE1/AN6/CK2SPP and RE2/AN7/OESPP) are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers. When selected as an analog input, these pins will read as '0's.

The corresponding Data Direction register is TRISE. Setting a TRISE bit (= 1) will make the corresponding PORTE pin an input (i.e., put the corresponding output driver in a high-impedance mode). Clearing a TRISE bit (= 0) will make the corresponding PORTE pin an output (i.e., put the contents of the output latch on the selected pin).

In addition to port data, the PORTE register (Register 10-1) also contains the RDPU control bit (PORTE<7>); this enables or disables the weak pull-ups on PORTD.

TRISE controls the direction of the RE pins, even when they are being used as analog inputs. The user must make sure to keep the pins configured as inputs when using them as analog inputs.

Note: On a Power-on Reset, RE2:RE0 are configured as analog inputs.

The Data Latch register (LATE) is also memory mapped. Read-modify-write operations on the LATE register read and write the latched output value for PORTE.

The fourth pin of PORTE ($\overline{\text{MCLR}}$/VPP/RE3) is an input only pin. Its operation is controlled by the MCLRE Configuration bit. When selected as a port pin (MCLRE = 0), it functions as a digital input only pin; as such, it does not have TRIS or LAT bits associated with its operation. Otherwise, it functions as the device's Master Clear input. In either configuration, RE3 also functions as the programming voltage input during programming.

Note: On a Power-on Reset, RE3 is enabled as a digital input only if Master Clear functionality is disabled.

EXAMPLE 10-5: INITIALIZING PORTE

```
CLRF    PORTE   ; Initialize PORTE by
                ; clearing output
                ; data latches
CLRF    LATE    ; Alternate method
                ; to clear output
                ; data latches
MOVLW   0Ah     ; Configure A/D
MOVWF   ADCON1  ; for digital inputs
MOVLW   03h     ; Value used to
                ; initialize data
                ; direction
MOVLW   07h     ; Turn off
MOVWF   CMCON   ; comparators
MOVWF   TRISC   ; Set RE<0> as inputs
                ; RE<1> as outputs
                ; RE<2> as inputs
```

10.5.1 PORTE IN 28-PIN DEVICES

For 28-pin devices, PORTE is only available when Master Clear functionality is disabled (MCLRE = 0). In these cases, PORTE is a single bit, input only port comprised of RE3 only. The pin operates as previously described.

REGISTER 10-1: PORTE REGISTER

R/W-0	U-0	U-0	U-0	R/W-x	R/W-0	R/W-0	R/W-0
RDPU[3]	—	—	—	RE3[1,2]	RE2[3]	RE1[3]	RE0[3]
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	**RDPU:** PORTD Pull-up Enable bit 1 = PORTD pull-ups are enabled by individual port latch values 0 = All PORTD pull-ups are disabled
bit 6-4	**Unimplemented:** Read as '0'
bit 3-0	**RE3:RE0:** PORTE Data Input bits[1,2,3]

Note 1: implemented only when Master Clear functionality is disabled (MCLRE Configuration bit = 0); otherwise, read as '0'.

2: RE3 is the only PORTE bit implemented on both 28-pin and 40/44-pin devices. All other bits are implemented only when PORTE is implemented (i.e., 40/44-pin devices).

3: Unimplemented in 28-pin devices; read as '0'.

12.7 Considerations in Asynchronous Counter Mode

Following a Timer1 interrupt and an update to the TMR1 registers, the Timer1 module uses a falling edge on its clock source to trigger the next register update on the rising edge. If the update is completed after the clock input has fallen, the next rising edge will not be counted.

If the application can reliably update TMR1 before the timer input goes low, no additional action is needed. Otherwise, an adjusted update can be performed following a later Timer1 increment. This can be done by monitoring TMR1L within the interrupt routine until it increments, and then updating the TMR1H:TMR1L register pair while the clock is low, or one-half of the period of the clock source. Assuming that Timer1 is being used as a Real-Time Clock, the clock source is a 32.768 kHz crystal oscillator; in this case, one-half period of the clock is 15.25 μs.

The Real-Time Clock application code in Example 12-1 shows a typical ISR for Timer1, as well as the optional code required if the update cannot be done reliably within the required interval.

EXAMPLE 12-1: IMPLEMENTING A REAL-TIME CLOCK USING A TIMER1 INTERRUPT SERVICE

```
RTCinit
           MOVLW   80h              ; Preload TMR1 register pair
           MOVWF   TMR1H            ; for 1 second overflow
           CLRF    TMR1L
           MOVLW   b'00001111'      ; Configure for external clock,
           MOVWF   T1CON            ; Asynchronous operation, external oscillator
           CLRF    secs             ; Initialize timekeeping registers
           CLRF    mins             ;
           MOVLW   .12
           MOVWF   hours
           BSF     PIE1, TMR1IE     ; Enable Timer1 interrupt
           RETURN
RTCisr
                                    ; Insert the next 4 lines of code when TMR1
                                    ; can not be reliably updated before clock pulse goes low
           BTFSC   TMR1L,0          ; wait for TMR1L to become clear
           BRA     $-2              ; (may already be clear)
           BTFSS   TMR1L,0          ; wait for TMR1L to become set
           BRA     $-2              ; TMR1 has just incremented
                                    ; If TMR1 update can be completed before clock pulse goes low
                                    ; Start ISR here
           BSF     TMR1H, 7         ; Preload for 1 sec overflow
           BCF     PIR1, TMR1IF     ; Clear interrupt flag
           INCF    secs, F          ; Increment seconds
           MOVLW   .59              ; 60 seconds elapsed?
           CPFSGT  secs
           RETURN                   ; No, done
           CLRF    secs             ; Clear seconds
           INCF    mins, F          ; Increment minutes
           MOVLW   .59              ; 60 minutes elapsed?
           CPFSGT  mins
           RETURN                   ; No, done
           CLRF    mins             ; clear minutes
           INCF    hours, F         ; Increment hours
           MOVLW   .23              ; 24 hours elapsed?
           CPFSGT  hours
           RETURN                   ; No, done
           CLRF    hours            ; Reset hours
           RETURN                   ; Done
```

The BDnSTAT byte of the BDT should always be the last byte updated when preparing to arm an endpoint. The SIE will clear the UOWN bit when a transaction has completed. The only exception to this is when KEN is enabled and/or BSTALL is enabled.

No hardware mechanism exists to block access when the UOWN bit is set. Thus, unexpected behavior can occur if the microcontroller attempts to modify memory when the SIE owns it. Similarly, reading such memory may produce inaccurate data until the USB peripheral returns ownership to the microcontroller.

17.4.1.2 BDnSTAT Register (CPU Mode)

When UOWN = `0`, the microcontroller core owns the BD. At this point, the other seven bits of the register take on control functions.

The Keep Enable bit, KEN (BDnSTAT<5>), determines if a BD stays enabled. If the bit is set, once the UOWN bit is set, it will remain owned by the SIE independent of the endpoint activity. This prevents the USTAT FIFO from being updated, as well as the transaction complete interrupt from being set for the endpoint. This feature should only be enabled when the Streaming Parallel Port is selected as the data I/O channel instead of USB RAM.

The Address Increment Disable bit, INCDIS (BDnSTAT<4>), controls the SIE's automatic address increment function. Setting INCDIS disables the auto-increment of the buffer address by the SIE for each byte transmitted or received. This feature should only be enabled when using the Streaming Parallel Port, where each data byte is processed to or from the same memory location.

The Data Toggle Sync Enable bit, DTSEN (BDnSTAT<3>), controls data toggle parity checking. Setting DTSEN enables data toggle synchronization by the SIE. When enabled, it checks the data packet's parity against the value of DTS (BDnSTAT<6>). If a packet arrives with an incorrect synchronization, the data will essentially be ignored. It will not be written to the USB RAM and the USB transfer complete interrupt flag will not be set. The SIE will send an ACK token back to the host to Acknowledge receipt, however. The effects of the DTSEN bit on the SIE are summarized in Table 17-3.

The Buffer Stall bit, BSTALL (BDnSTAT<2>), provides support for control transfers, usually one-time stalls on Endpoint 0. It also provides support for the SET_FEATURE/CLEAR_FEATURE commands specified in Chapter 9 of the USB specification; typically, continuous STALLs to any endpoint other than the default control endpoint.

The BSTALL bit enables buffer stalls. Setting BSTALL causes the SIE to return a STALL token to the host if a received token would use the BD in that location. The EPSTALL bit in the corresponding UEPn control register is set and a STALL interrupt is generated when a STALL is issued to the host. The UOWN bit remains set and the BDs are not changed unless a SETUP token is received. In this case, the STALL condition is cleared and the ownership of the BD is returned to the microcontroller core.

The BD9:BD8 bits (BDnSTAT<1:0>) store the two most significant digits of the SIE byte count; the lower 8 digits are stored in the corresponding BDnCNT register. See **Section 17.4.2 "BD Byte Count"** for more information.

TABLE 17-3: EFFECT OF DTSEN BIT ON ODD/EVEN (DATA0/DATA1) PACKET RECEPTION

OUT Packet from Host	BDnSTAT Settings		Device Response after Receiving Packet			
	DTSEN	DTS	Handshake	UOWN	TRNIF	BDnSTAT and USTAT Status
DATA0	1	0	ACK	0	1	Updated
DATA1	1	0	ACK	1	0	Not Updated
DATA1	1	1	ACK	0	1	Updated
DATA0	1	1	ACK	1	0	Not Updated
Either	0	x	ACK	0	1	Updated
Either, with error	x	x	NAK	1	0	Not Updated

Legend: x = don't care

18.2 Setup for USB Control

When the SPP is configured for USB operation, data can be clocked directly to and from the USB peripheral without intervention of the microcontroller; thus, no process time is required. Data is clocked into or out from the SPP with endpoint (address) information first, followed by one or more bytes of data, as shown in Figure 18-5. This is ideal for applications that require isochronous, large volume data movement.

The following steps are required to set up the SPP for USB control:

1. Configure the SPP as desired, including wait states and clocks.
2. Set the SPPOWN bit for USB ownership.
3. Set the buffer descriptor starting address (BDnADRL:BDnADRH) to FFFFh.
4. Set the KEN bit (BDnSTAT<5>) so the buffer descriptor is kept indefinitely by the SIE.
5. Set the INCDIS bit (BDnSTAT<4>) to disable automatic buffer address increment.
6. Set the SPPEN bit to enable the module.

Note:	If a USB endpoint is configured to use the SPP, the data transfer type of that endpoint must be isochronous only.

18.3 Setup for Microcontroller Control

The SPP can also act as a parallel port for the microcontroller. In this mode, the SPPEPS register (Register 18-3) provides status and address write control. Data is written to and read from the SPPDATA register. When the SPP is owned by the microcontroller, the SPP clock is driven by the instruction clock (FOSC/4).

The following steps are required to set up the SPP for microcontroller operation:

1. Configure the SPP as desired, including wait states and clocks.
2. Clear the SPPOWN bit.
3. Set SPPEN to enable the module.

18.3.1 SPP INTERRUPTS

When owned by the microcontroller core, control can generate an interrupt to notify the application when each read and write operation is completed. The interrupt flag bit is SPPIF (PIR1<7>) and is enabled by the SPPIE bit (PIE1<7>). Like all other microcontroller level interrupts, it can be set to a low or high priority. This is done with the SPPIP bit (IPR1<7>).

18.3.2 WRITING TO THE SPP

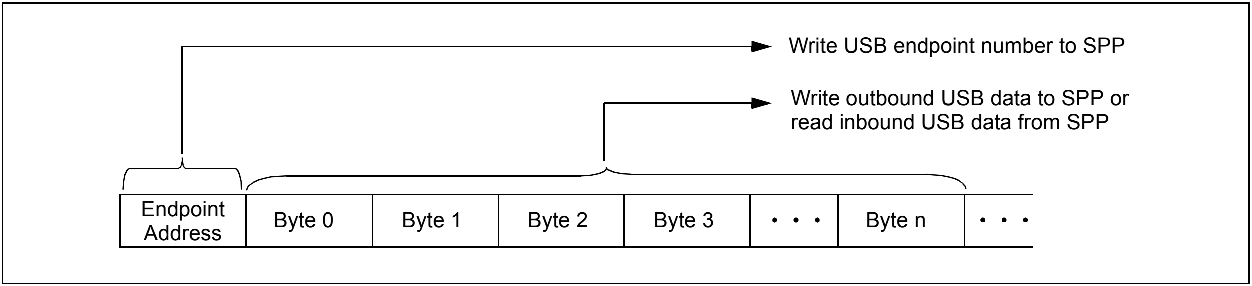
Once configured, writing to the SPP is performed by writing to the SPPEPS and SPPDATA registers. If the SPP is configured to clock out endpoint address information with the data, writing to the SPPEPS register initiates the address write cycle. Otherwise, the write is started by writing the data to the SPPDATA register. The SPPBUSY bit indicates the status of the address and the data write cycles.

The following is an example write sequence:

1. Write the 4-bit address to the SPPEPS register. The SPP automatically starts writing the address. If address write is not used, then skip to step 3.
2. Monitor the SPPBUSY bit to determine when the address has been sent. The duration depends on the wait states.
3. Write the data to the SPPDATA register. The SPP automatically starts writing the data.
4. Monitor the SPPBUSY bit to determine when the data has been sent. The duration depends on the wait states.
5. Go back to steps 1 or 3 to write a new address or data.

Note:	The SPPBUSY bit should be polled to make certain that successive writes to the SPPEPS or SPPDATA registers do not overrun the wait time due to the wait state setting.

FIGURE 18-5: TRANSFER OF DATA BETWEEN USB SIE AND SPP


Write USB endpoint number to SPP
Write outbound USB data to SPP or read inbound USB data from SPP
Endpoint Address
Byte 0
Byte 1
Byte 2
Byte 3
• • •
Byte n
• • •

REGISTER 19-6: SSPCON2: MSSP CONTROL REGISTER 2 (I^{2}C™ SLAVE MODE)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ADMSK5	ADMSK4	ADMSK3	ADMSK2	ADMSK1	SEN(1)
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 **GCEN:** General Call Enable bit (Slave mode only)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled

bit 6 **ACKSTAT:** Acknowledge Status bit
Unused in Slave mode.

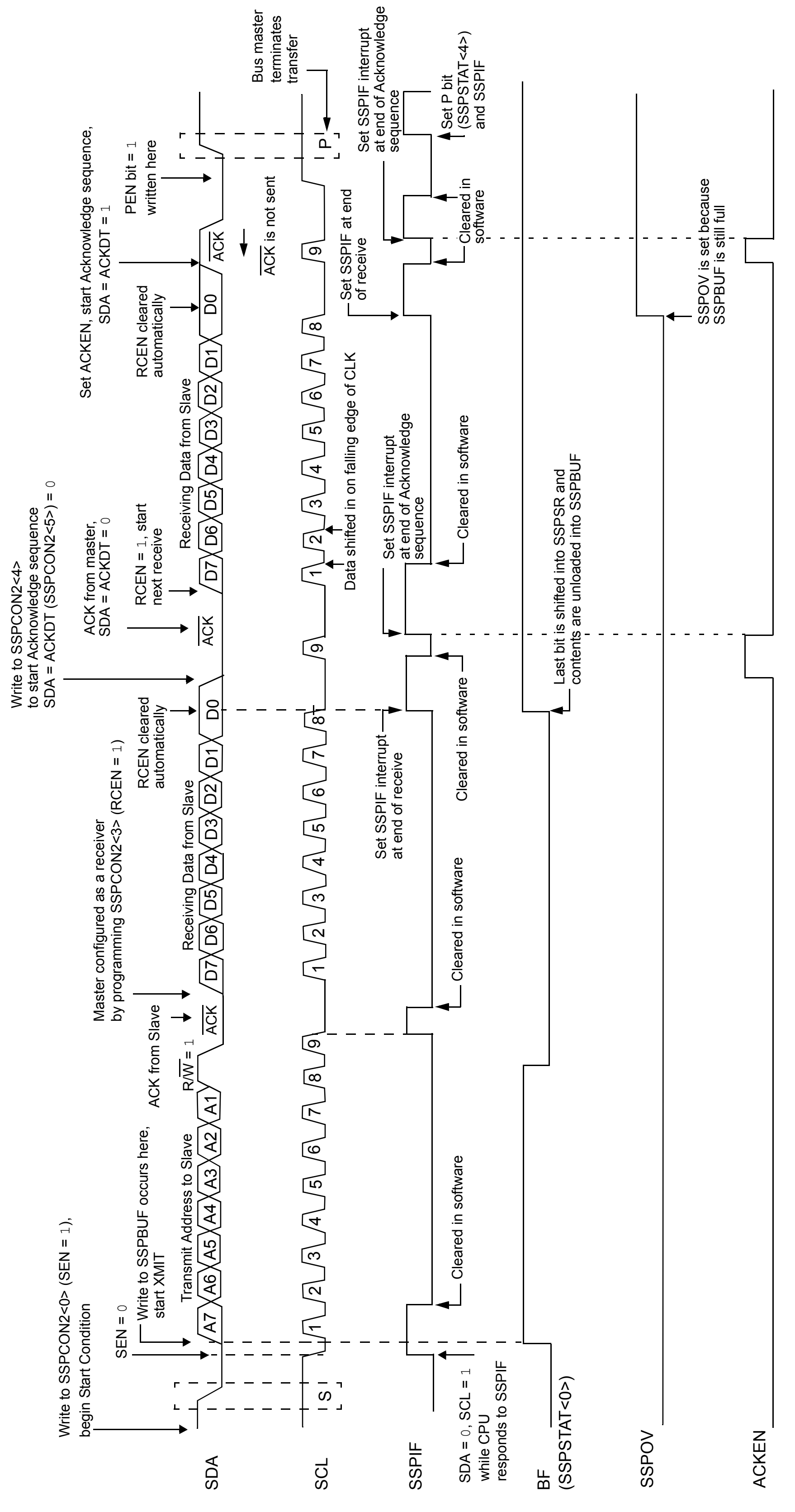
bit 5-2 **ADMSK5:ADMSK2:** Slave Address Mask Select bits
1 = Masking of corresponding bits of SSPADD enabled
0 = Masking of corresponding bits of SSPADD disabled

bit 1 **ADMSK1:** Slave Address Mask Select bit
In 7-Bit Addressing mode:
1 = Masking of SPADD<1> only enabled
0 = Masking of SPADD<1> only disabled
In 10-Bit Addressing mode:
1 = Masking of SSPADD<1:0> enabled
0 = Masking of SSPADD<1:0> disabled

bit 0 **SEN:** Stretch Enable bit(1)
1 = Clock stretching is enabled for both slave transmit and slave receive (stretch enabled)
0 = Clock stretching is disabled

Note 1: If the I^{2}C module is active, this bit may not be set (no spooling) and the SSPBUF may not be written (or writes to the SSPBUF are disabled).

FIGURE 19-24: I²C™ MASTER MODE WAVEFORM (RECEPTION, 7-BIT ADDRESS)


SDA
SCL
SSPIF
SDA = 0, SCL = 1 while CPU responds to SSPIF
BF (SSPSTAT<0>)
SSPOV
ACKEN
Write to SSPCON2<0> (SEN = 1), begin Start Condition
SEN = 0
Write to SSPBUF occurs here, start XMIT
Transmit Address to Slave
ACK from Slave
R/W = 1
Master configured as a receiver by programming SSPCON2<3> (RCEN = 1)
Receiving Data from Slave
RCEN cleared automatically
Write to SSPCON2<4> to start Acknowledge sequence SDA = ACKDT (SSPCON2<5>) = 0
ACK from master, SDA = ACKDT = 0
RCEN = 1, start next receive
Receiving Data from Slave
RCEN cleared automatically
Set ACKEN, start Acknowledge sequence, SDA = ACKDT = 1
PEN bit = 1 written here
ACK is not sent
Bus master terminates transfer
S
P
A7 A6 A5 A4 A3 A2 A1 ACK
D7 D6 D5 D4 D3 D2 D1 D0 ACK
D7 D6 D5 D4 D3 D2 D1 D0 ACK
1 2 3 4 5 6 7 8 9
Cleared in software
Set SSPIF interrupt at end of receive
Cleared in software
Set SSPIF interrupt at end of Acknowledge sequence
Data shifted in on falling edge of CLK
Cleared in software
Set SSPIF at end of receive
Set SSPIF interrupt at end of Acknowledge sequence
Cleared in software
Set P bit (SSPSTAT<4>) and SSPIF
Last bit is shifted into SSPSR and contents are unloaded into SSPBUF
SSPOV is set because SSPBUF is still full

19.4.17.2 Bus Collision During a Repeated Start Condition

During a Repeated Start condition, a bus collision occurs if:

a) A low level is sampled on SDA when SCL goes from low level to high level.
b) SCL goes low before SDA is asserted low, indicating that another master is attempting to transmit a data '1'.

When the user deasserts SDA and the pin is allowed to float high, the BRG is loaded with SSPADD<6:0> and counts down to '0'. The SCL pin is then deasserted and when sampled high, the SDA pin is sampled.

If SDA is low, a bus collision has occurred (i.e., another master is attempting to transmit a data '0', see Figure 19-31). If SDA is sampled high, the BRG is reloaded and begins counting. If SDA goes from high-to-low before the BRG times out, no bus collision occurs because no two masters can assert SDA at exactly the same time.

If SCL goes from high-to-low before the BRG times out and SDA has not already been asserted, a bus collision occurs. In this case, another master is attempting to transmit a data '1' during the Repeated Start condition (see Figure 19-32).

If, at the end of the BRG time-out, both SCL and SDA are still high, the SDA pin is driven low and the BRG is reloaded and begins counting. At the end of the count, regardless of the status of the SCL pin, the SCL pin is driven low and the Repeated Start condition is complete.

FIGURE 19-31: BUS COLLISION DURING A REPEATED START CONDITION (CASE 1)

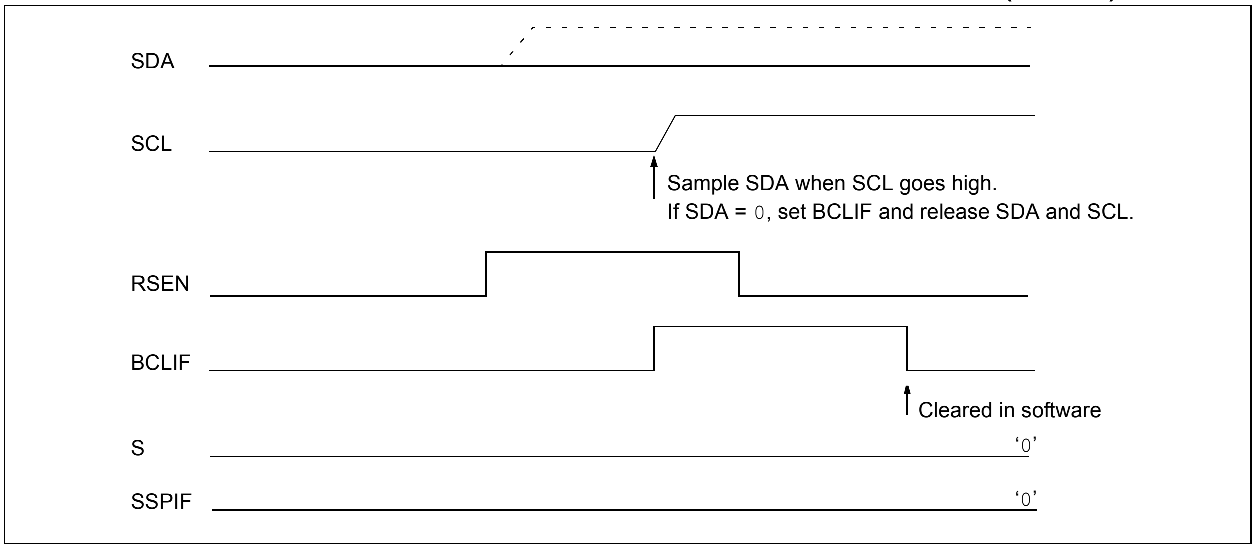
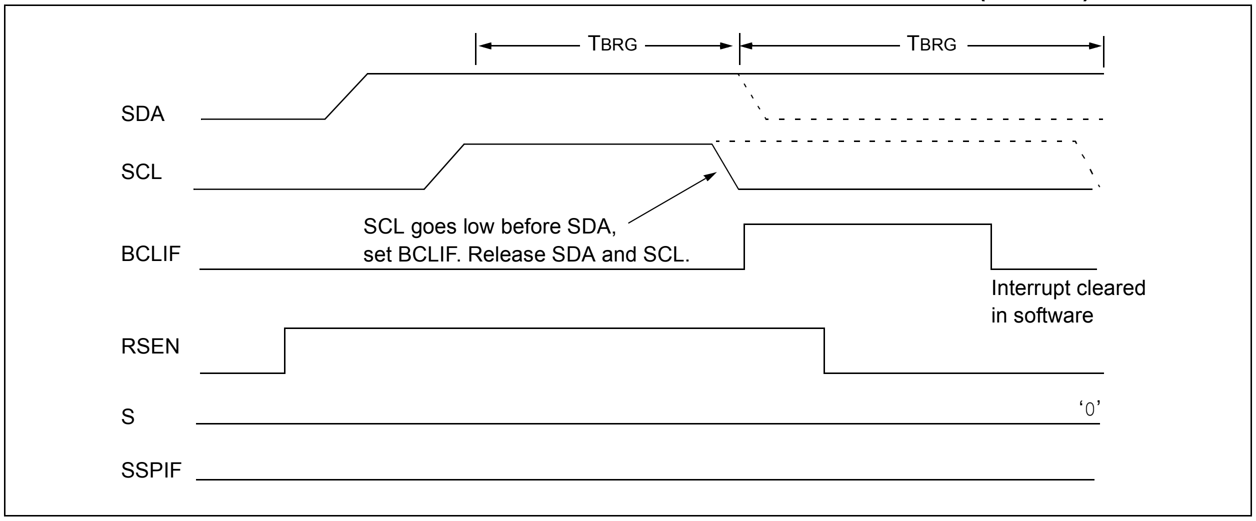

SDA
SCL
Sample SDA when SCL goes high.
If SDA = 0, set BCLIF and release SDA and SCL.
RSEN
BCLIF
Cleared in software
S
'0'
SSPIF
'0'

FIGURE 19-32: BUS COLLISION DURING REPEATED START CONDITION (CASE 2)


TBRG
TBRG
SDA
SCL
SCL goes low before SDA, set BCLIF. Release SDA and SCL.
BCLIF
Interrupt cleared in software
RSEN
S
'0'
SSPIF

LFSR — Load FSR

Syntax:	LFSR f, k
Operands:	$0 \le f \le 2$ $0 \le k \le 4095$
Operation:	$k \rightarrow$ FSRf
Status Affected:	None

Encoding:

1110	1110	00ff	k_{11}kkk
1111	0000	$k_7$kkk	kkkk

Description:	The 12-bit literal ‘k’ is loaded into the File Select Register pointed to by ‘f’.
Words:	2
Cycles:	2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal ‘k’ MSB	Process Data	Write literal ‘k’ MSB to FSRfH
Decode	Read literal ‘k’ LSB	Process Data	Write literal ‘k’ to FSRfL

Example: `LFSR 2, 3ABh`

After Instruction

FSR2H	=	03h
FSR2L	=	ABh

MOVF — Move f

Syntax:	MOVF f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$f \rightarrow$ dest
Status Affected:	N, Z

Encoding:

0101	00da	ffff	ffff

Description: The contents of register ‘f’ are moved to a destination dependent upon the status of ‘d’. If ‘d’ is ‘0’, the result is placed in W. If ‘d’ is ‘1’, the result is placed back in register ‘f’ (default). Location ‘f’ can be anywhere in the 256-byte bank.
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 26.2.3 “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 W

Example: `MOVF REG, 0, 0`

Before Instruction

REG	=	22h
W	=	FFh

After Instruction

REG	=	22h
W	=	22h

POP — Pop Top of Return Stack

Syntax:	POP
Operands:	None
Operation:	(TOS) → bit bucket
Status Affected:	None
Encoding:	`0000` `0000` `0000` `0110`
Description:	The TOS value is pulled off the return stack and is discarded. The TOS value then becomes the previous value that was pushed onto the return stack. This instruction is provided to enable the user to properly manage the return stack to incorporate a software stack.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Pop TOS value	No operation

Example:
```
POP
GOTO      NEW
```

Before Instruction

TOS	=	0031A2h
Stack (1 level down)	=	014332h

After Instruction

TOS	=	014332h
PC	=	NEW

PUSH — Push Top of Return Stack

Syntax:	PUSH
Operands:	None
Operation:	(PC + 2) → TOS
Status Affected:	None
Encoding:	`0000` `0000` `0000` `0101`
Description:	The PC + 2 is pushed onto the top of the return stack. The previous TOS value is pushed down on the stack. This instruction allows implementing a software stack by modifying TOS and then pushing it onto the return stack.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Push PC + 2 onto return stack	No operation	No operation

Example:
```
PUSH
```

Before Instruction

TOS	=	345Ah
PC	=	0124h

After Instruction

PC	=	0126h
TOS	=	0126h
Stack (1 level down)	=	345Ah

TBLWT Table Write

Syntax:	TBLWT (*; *+; *-; +*)
Operands:	None

Operation:	if TBLWT*,
	(TABLAT) → Holding Register,
	TBLPTR – No Change;
	if TBLWT*+,
	(TABLAT) → Holding Register,
	(TBLPTR) + 1 → TBLPTR;
	if TBLWT*-,
	(TABLAT) → Holding Register,
	(TBLPTR) – 1 → TBLPTR;
	if TBLWT+*,
	(TBLPTR) + 1 → TBLPTR;
	(TABLAT) → Holding Register

Status Affected:	None

Encoding:

0000	0000	0000	11nn nn=0 * =1 *+ =2 *- =3 +*

Description: This instruction uses the 3 LSBs of TBLPTR to determine which of the 8 holding registers the TABLAT is written to. The holding registers are used to program the contents of Program Memory (P.M.). (Refer to **Section 6.0 “Flash Program Memory”** for additional details on programming Flash memory.) The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2-Mbyte address range. The LSb of the TBLPTR selects which byte of the program memory location to access.

TBLPTR[0] = `0`:	Least Significant Byte of Program Memory Word
TBLPTR[0] = `1`:	Most Significant Byte of Program Memory Word

The `TBLWT` instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation
No operation	No operation (Read TABLAT)	No operation	No operation (Write to Holding Register)

TBLWT Table Write (Continued)

Example 1: `TBLWT *+;`

Before Instruction

TABLAT	=	55h
TBLPTR	=	00A356h
HOLDING REGISTER (00A356h)	=	FFh

After Instructions (table write completion)

TABLAT	=	55h
TBLPTR	=	00A357h
HOLDING REGISTER (00A356h)	=	55h

Example 2: `TBLWT +*;`

Before Instruction

TABLAT	=	34h
TBLPTR	=	01389Ah
HOLDING REGISTER (01389Ah)	=	FFh
HOLDING REGISTER (01389Bh)	=	FFh

After Instruction (table write completion)

TABLAT	=	34h
TBLPTR	=	01389Bh
HOLDING REGISTER (01389Ah)	=	FFh
HOLDING REGISTER (01389Bh)	=	34h

27.2 MPASM Assembler

The MPASM Assembler is a full-featured, universal macro assembler for all PIC MCUs.

The MPASM Assembler generates relocatable object files for the MPLINK Object Linker, Intel® standard HEX files, MAP files to detail memory usage and symbol reference, absolute LST files that contain source lines and generated machine code and COFF files for debugging.

The MPASM Assembler features include:

- Integration into MPLAB IDE projects
- User-defined macros to streamline assembly code
- Conditional assembly for multi-purpose source files
- Directives that allow complete control over the assembly process

27.3 MPLAB C18 and MPLAB C30 C Compilers

The MPLAB C18 and MPLAB C30 Code Development Systems are complete ANSI C compilers for Microchip's PIC18 and PIC24 families of microcontrollers and the dsPIC30 and dsPIC33 family of digital signal controllers. These compilers provide powerful integration capabilities, superior code optimization and ease of use not found with other compilers.

For easy source level debugging, the compilers provide symbol information that is optimized to the MPLAB IDE debugger.

27.4 MPLINK Object Linker/ MPLIB Object Librarian

The MPLINK Object Linker combines relocatable objects created by the MPASM Assembler and the MPLAB C18 C Compiler. It can link relocatable objects from precompiled libraries, using directives from a linker script.

The MPLIB Object Librarian manages the creation and modification of library files of precompiled code. When a routine from a library is called from a source file, only the modules that contain that routine will be linked in with the application. This allows large libraries to be used efficiently in many different applications.

The object linker/library features include:

- Efficient linking of single libraries instead of many smaller files
- Enhanced code maintainability by grouping related modules together
- Flexible creation of libraries with easy module listing, replacement, deletion and extraction

27.5 MPLAB ASM30 Assembler, Linker and Librarian

MPLAB ASM30 Assembler produces relocatable machine code from symbolic assembly language for dsPIC30F devices. MPLAB C30 C Compiler uses the assembler to produce its object file. The assembler generates relocatable object files that can then be archived or linked with other relocatable object files and archives to create an executable file. Notable features of the assembler include:

- Support for the entire dsPIC30F instruction set
- Support for fixed-point and floating-point data
- Command line interface
- Rich directive set
- Flexible macro language
- MPLAB IDE compatibility

27.6 MPLAB SIM Software Simulator

The MPLAB 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 SIM Software Simulator fully supports symbolic debugging using the MPLAB C18 and MPLAB C30 C Compilers, and the MPASM and MPLAB ASM30 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.

APPENDIX C: CONVERSION CONSIDERATIONS

This appendix discusses the considerations for converting from previous versions of a device to the ones listed in this data sheet. Typically, these changes are due to the differences in the process technology used. An example of this type of conversion is from a PIC16C74A to a PIC16C74B.

Not Applicable

APPENDIX D: MIGRATION FROM BASELINE TO ENHANCED DEVICES

This section discusses how to migrate from a Baseline device (i.e., PIC16C5X) to an Enhanced MCU device (i.e., PIC18FXXX).

The following are the list of modifications over the PIC16C5X microcontroller family:

Not Currently Available