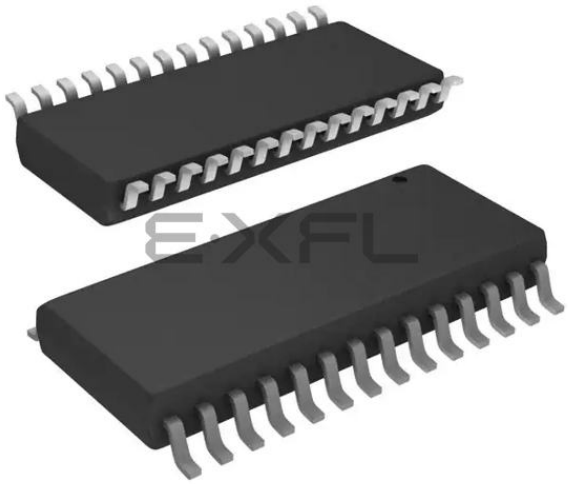


Welcome to E-XFL.COM

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	Obsolete
Core Processor	PIC
Core Size	8-Bit
Speed	40MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	23
Program Memory Size	32KB (16K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	1.5K x 8
Voltage - Supply (Vcc/Vdd)	2V ~ 5.5V
Data Converters	A/D 5x10b
Oscillator Type	External
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/pic18lf252t-i-sog

TABLE 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset RESET Instruction Stack Resets	Wake-up via WDT or Interrupt
IPR2	242	442	252	452	---1 1111	---1 1111	---u uuuu
PIR2	242	442	252	452	---0 0000	---0 0000	---u uuuu(1)
PIE2	242	442	252	452	---0 0000	---0 0000	---u uuuu
IPR1	242	442	252	452	1111 1111	1111 1111	uuuu uuuu
	242	442	252	452	-111 1111	-111 1111	-uuu uuuu
PIR1	242	442	252	452	0000 0000	0000 0000	uuuu uuuu(1)
	242	442	252	452	-000 0000	-000 0000	-uuu uuuu(1)
PIE1	242	442	252	452	0000 0000	0000 0000	uuuu uuuu
	242	442	252	452	-000 0000	-000 0000	-uuu uuuu
TRISE	242	442	252	452	0000 -111	0000 -111	uuuu -uuu
TRISD	242	442	252	452	1111 1111	1111 1111	uuuu uuuu
TRISC	242	442	252	452	1111 1111	1111 1111	uuuu uuuu
TRISB	242	442	252	452	1111 1111	1111 1111	uuuu uuuu
TRISA(5,6)	242	442	252	452	-111 1111(5)	-111 1111(5)	-uuu uuuu(5)
LATE	242	442	252	452	---- -xxx	---- -uuu	---- -uuu
LATD	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATC	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATB	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATA(5,6)	242	442	252	452	-xxx xxxx(5)	-uuu uuuu(5)	-uuu uuuu(5)
PORTE	242	442	252	452	---- -000	---- -000	---- -uuu
PORTD	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTC	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTB	242	442	252	452	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA(5,6)	242	442	252	452	-x0x 0000(5)	-u0u 0000(5)	-uuu uuuu(5)

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 3-2 for RESET value for specific condition.

5: Bit 6 of PORTA, LATA, and TRISA are enabled in ECIO and RCIO Oscillator modes only. In all other Oscillator modes, they are disabled and read ’0’.

6: Bit 6 of PORTA, LATA and TRISA are not available on all devices. When unimplemented, they are read ’0’.

4.9 Data Memory Organization

The data memory is implemented as static RAM. Each register in the data memory has a 12-bit address, allowing up to 4096 bytes of data memory. Figure 4-6 and Figure 4-7 show the data memory organization for the PIC18FXX2 devices.

The data memory map is divided into as many as 16 banks that contain 256 bytes each. The lower 4 bits of the Bank Select Register (BSR<3:0>) select which bank will be accessed. The upper 4 bits for the BSR are not implemented.

The data memory contains Special Function Registers (SFR) and General Purpose Registers (GPR). The SFRs are used for control and status of the controller and peripheral functions, while GPRs are used for data storage and scratch pad operations in the user's application. The SFRs start at the last location of Bank 15 (0xFFF) and extend downwards. Any remaining space beyond the SFRs in the Bank may be implemented as GPRs. GPRs start at the first location of Bank 0 and grow upwards. Any read of an unimplemented location will read as '0's.

The entire data memory may be accessed directly or indirectly. Direct addressing may require the use of the BSR register. Indirect addressing requires the use of a File Select Register (FSRn) and a corresponding Indirect File Operand (INDFn). Each FSR holds a 12-bit address value that can be used to access any location in the Data Memory map without banking.

The instruction set and architecture allow operations across all banks. This may be accomplished by indirect addressing or by the use of the `MOVFF` instruction. The `MOVFF` instruction is a two-word/two-cycle instruction that moves a value from one register to another.

To ensure that commonly used registers (SFRs and select GPRs) can be accessed in a single cycle, regardless of the current BSR values, an Access Bank is implemented. A segment of Bank 0 and a segment of Bank 15 comprise the Access RAM. Section 4.10 provides a detailed description of the Access RAM.

4.9.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly or indirectly. Indirect addressing operates using a File Select Register and corresponding Indirect File Operand. The operation of indirect addressing is shown in Section 4.12.

Enhanced MCU devices may have banked memory in the GPR area. GPRs are not initialized by a Power-on Reset and are unchanged on all other RESETS.

Data RAM is available for use as GPR registers by all instructions. The top half of Bank 15 (0xF80 to 0xFFF) contains SFRs. All other banks of data memory contain GPR registers, starting with Bank 0.

4.9.2 SPECIAL FUNCTION REGISTERS

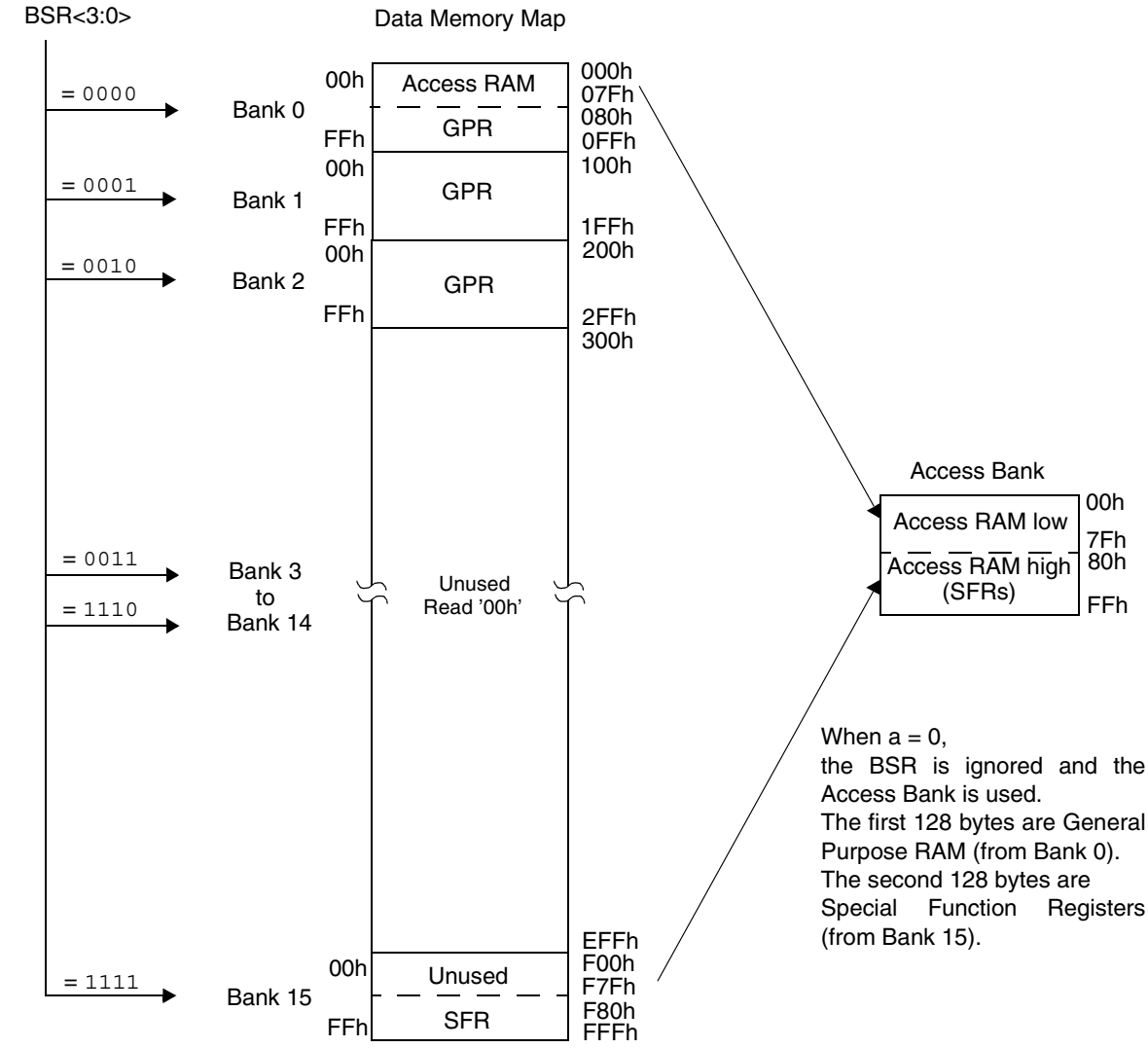
The Special Function Registers (SFRs) are registers used by the CPU and Peripheral Modules for controlling the desired operation of the device. These registers are implemented as static RAM. A list of these registers is given in Table 4-1 and Table 4-2.

The SFRs can be classified into two sets; those associated with the "core" function and those related to the peripheral functions. Those registers related to the "core" are described in this section, while those related to the operation of the peripheral features are described in the section of that peripheral feature.

The SFRs are typically distributed among the peripherals whose functions they control.

The unused SFR locations will be unimplemented and read as '0's. See Table 4-1 for addresses for the SFRs.

FIGURE 4-6: DATA MEMORY MAP FOR PIC18F242/442



4.12 Indirect Addressing, INDF and FSR Registers

Indirect addressing is a mode of addressing data memory, where the data memory address in the instruction is not fixed. An FSR register is used as a pointer to the data memory location that is to be read or written. Since this pointer is in RAM, the contents can be modified by the program. This can be useful for data tables in the data memory and for software stacks. Figure 4-9 shows the operation of indirect addressing. This shows the moving of the value to the data memory address specified by the value of the FSR register.

Indirect addressing is possible by using one of the INDF registers. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = 0), will read 00h. Writing to the INDF register indirectly, results in a no operation. The FSR register contains a 12-bit address, which is shown in Figure 4-10.

The INDFn register is not a physical register. Addressing INDFn actually addresses the register whose address is contained in the FSRn register (FSRn is a pointer). This is indirect addressing.

Example 4-4 shows a simple use of indirect addressing to clear the RAM in Bank1 (locations 100h-1FFh) in a minimum number of instructions.

EXAMPLE 4-4: HOW TO CLEAR RAM (BANK1) USING INDIRECT ADDRESSING

```
        LFSR  FSR0 ,0x100  ;
NEXT    CLRF  POSTINC0     ; Clear INDF
                           ; register and
                           ; inc pointer
        BTFSS FSR0H, 1     ; All done with
                           ; Bank1?
        GOTO  NEXT         ; NO, clear next
CONTINUE                   ; YES, continue
```

There are three indirect addressing registers. To address the entire data memory space (4096 bytes), these registers are 12-bit wide. To store the 12-bits of addressing information, two 8-bit registers are required. These indirect addressing registers are:

1. FSR0: composed of FSR0H:FSR0L
2. FSR1: composed of FSR1H:FSR1L
3. FSR2: composed of FSR2H:FSR2L

In addition, there are registers INDF0, INDF1 and INDF2, which are not physically implemented. Reading or writing to these registers activates indirect addressing, with the value in the corresponding FSR register being the address of the data. If an instruction writes a value to INDF0, the value will be written to the address pointed to by FSR0H:FSR0L. A read from INDF1 reads the data from the address pointed to by FSR1H:FSR1L. INDFn can be used in code anywhere an operand can be used.

If INDF0, INDF1 or INDF2 are read indirectly via an FSR, all '0's are read (zero bit is set). Similarly, if INDF0, INDF1 or INDF2 are written to indirectly, the operation will be equivalent to a NOP instruction and the STATUS bits are not affected.

4.12.1 INDIRECT ADDRESSING OPERATION

Each FSR register has an INDF register associated with it, plus four additional register addresses. Performing an operation on one of these five registers determines how the FSR will be modified during indirect addressing.

When data access is done to one of the five INDFn locations, the address selected will configure the FSRn register to:

- Do nothing to FSRn after an indirect access (no change) - INDFn
- Auto-decrement FSRn after an indirect access (post-decrement) - POSTDECn
- Auto-increment FSRn after an indirect access (post-increment) - POSTINCn
- Auto-increment FSRn before an indirect access (pre-increment) - PREINCn
- Use the value in the WREG register as an offset to FSRn. Do not modify the value of the WREG or the FSRn register after an indirect access (no change) - PLUSWn

When using the auto-increment or auto-decrement features, the effect on the FSR is not reflected in the STATUS register. For example, if the indirect address causes the FSR to equal '0', the Z bit will not be set.

Incrementing or decrementing an FSR affects all 12 bits. That is, when FSRnL overflows from an increment, FSRnH will be incremented automatically.

Adding these features allows the FSRn to be used as a stack pointer, in addition to its uses for table operations in data memory.

Each FSR has an address associated with it that performs an indexed indirect access. When a data access to this INDFn location (PLUSWn) occurs, the FSRn is configured to add the signed value in the WREG register and the value in FSR to form the address before an indirect access. The FSR value is not changed.

If an FSR register contains a value that points to one of the INDFn, an indirect read will read 00h (zero bit is set), while an indirect write will be equivalent to a NOP (STATUS bits are not affected).

If an indirect addressing operation is done where the target address is an FSRnH or FSRnL register, the write operation will dominate over the pre- or post-increment/decrement functions.

EXAMPLE 5-3: 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   DATA_ADDR_HIGH              ; point to buffer
            MOVWF   FSR0H
            MOVLW   DATA_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 Row Erase operation
            BCF     INTCON,GIE                  ; disable interrupts
            MOVLW   55h
            MOVWF   EECON2                      ; write 55h
            MOVLW   AAh
            MOVWF   EECON2                      ; write AAh
            BSF     EECON1,WR                   ; start erase (CPU stall)
            BSF     INTCON,GIE                  ; re-enable interrupts
            TBLRD*-                             ; dummy read decrement
WRITE_BUFFER_BACK
            MOVLW   8                           ; number of write buffer groups of 8 bytes
            MOVWF   COUNTER_HI
            MOVLW   BUFFER_ADDR_HIGH            ; point to buffer
            MOVWF   FSR0H
            MOVLW   BUFFER_ADDR_LOW
            MOVWF   FSR0L
PROGRAM_LOOP
            MOVLW   8                           ; number of bytes in holding register
            MOVWF   COUNTER
WRITE_WORD_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.
            DECFSZ  COUNTER                     ; loop until buffers are full
            BRA     WRITE_WORD_TO_HREGS
```

6.3 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 (EECON1<7>), clear the CFGS control bit (EECON1<6>), and then set control bit RD (EECON1<0>). The data is available for 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).

EXAMPLE 6-1: 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 program FLASH or Data EEPROM memory
BSF     EECON1, RD     ; EEPROM Read
MOVF    EEDATA, W      ; W = EEDATA
```

6.4 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. Then the sequence in Example 6-2 must be followed to initiate the write cycle.

The write will not initiate if the above sequence is not exactly followed (write 55h to EECON2, write AAh 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 EDATA cannot be modified. The WR bit will be inhibited from being set unless the WREN bit is set. The WREN bit must be set on a previous instruction. Both WR and WREN cannot be set with the same instruction.

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

EXAMPLE 6-2: DATA EEPROM WRITE

```
                MOVLW   DATA_EE_ADDR   ;
                MOVWF   EEADR          ; Data Memory Address to read
                MOVLW   DATA_EE_DATA   ;
                MOVWF   EEDATA         ; Data Memory Value to write
                BCF     EECON1, EEPGD  ; Point to DATA memory
                BCF     EECON1, CFGS   ; Access program FLASH or Data EEPROM memory
                BSF     EECON1, WREN   ; Enable writes


                BCF     INTCON, GIE    ; Disable interrupts
  Required      MOVLW   55h            ;
  Sequence      MOVWF   EECON2         ; Write 55h
                MOVLW   AAh            ;
                MOVWF   EECON2         ; Write AAh
                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)
```

8.6 INT0 Interrupt

External interrupts on the RB0/INT0, RB1/INT1 and RB2/INT2 pins are edge triggered: either rising, if the corresponding INTEDGx bit is set in the INTCON2 register, or falling, if the INTEDGx bit is clear. When a valid edge appears on the RBx/INTx pin, the corresponding flag bit INTxF is set. This interrupt can be disabled by clearing the corresponding enable bit INTxE. Flag bit INTxF must be cleared in software in the Interrupt Service Routine before re-enabling the interrupt. All external interrupts (INT0, INT1 and INT2) can wake-up the processor from SLEEP, if bit INTxE was set prior to going into SLEEP. If the global interrupt enable bit GIE is set, the processor will branch to the interrupt vector following wake-up.

Interrupt priority for INT1 and INT2 is determined by the value contained in the interrupt priority bits, INT1IP (INTCON3<6>) and INT2IP (INTCON3<7>). There is no priority bit associated with INT0. It is always a high priority interrupt source.

8.7 TMR0 Interrupt

In 8-bit mode (which is the default), an overflow (FFh $\rightarrow$ 00h) in the TMR0 register will set flag bit TMR0IF. In 16-bit mode, an overflow (FFFFh $\rightarrow$ 0000h) in the TMR0H:TMR0L registers will set flag bit TMR0IF. The interrupt can be enabled/disabled by setting/clearing enable bit T0IE (INTCON<5>). Interrupt priority for Timer0 is determined by the value contained in the interrupt priority bit TMR0IP (INTCON2<2>). See Section 10.0 for further details on the Timer0 module.

8.8 PORTB Interrupt-on-Change

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled/disabled by setting/clearing enable bit, RBIE (INTCON<3>). Interrupt priority for PORTB interrupt-on-change is determined by the value contained in the interrupt priority bit, RBIP (INTCON2<0>).

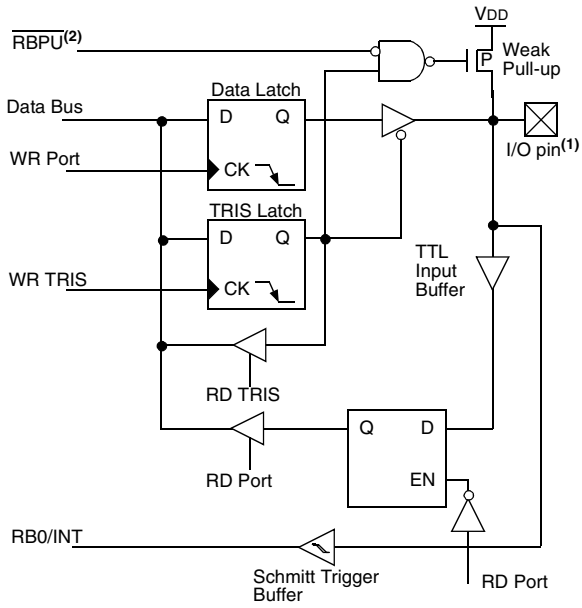
8.9 Context Saving During Interrupts

During an interrupt, the return PC value is saved on the stack. Additionally, the WREG, STATUS and BSR registers are saved on the fast return stack. If a fast return from interrupt is not used (See Section 4.3), the user may need to save the WREG, STATUS and BSR registers in software. Depending on the user's application, other registers may also need to be saved. Equation 8-1 saves and restores the WREG, STATUS and BSR registers during an Interrupt Service Routine.

EXAMPLE 8-1: SAVING STATUS, WREG AND BSR REGISTERS IN RAM

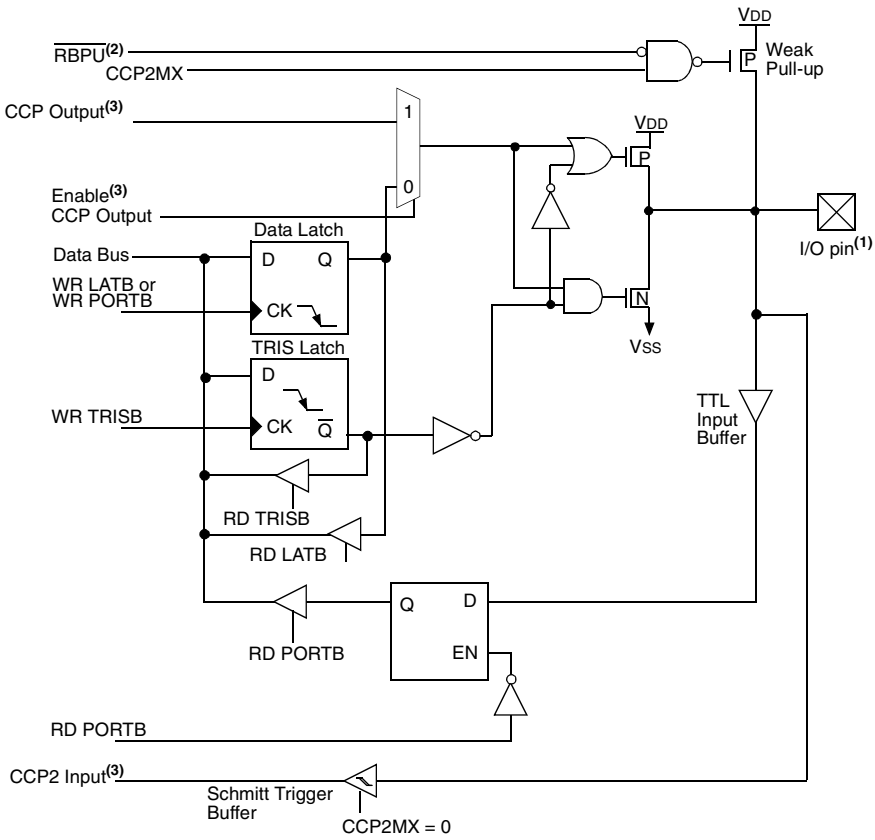
```
MOVWF   W_TEMP                          ; W_TEMP is in virtual bank
MOVFF   STATUS, STATUS_TEMP             ; STATUS_TEMP located anywhere
MOVFF   BSR,    BSR_TEMP                ; BSR located anywhere
;
; USER ISR CODE
;
MOVFF   BSR_TEMP,   BSR                 ; Restore BSR
MOVF    W_TEMP,     W                   ; Restore WREG
MOVFF   STATUS_TEMP,STATUS              ; Restore STATUS
```

FIGURE 9-5: BLOCK DIAGRAM OF RB2:RB0 PINS



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RBPU}}$ bit (OPTION_REG<7>).

FIGURE 9-6: BLOCK DIAGRAM OF RB3 PIN



Note 1: I/O pin has diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate DDR bit(s) and clear the $\overline{\text{RBPU}}$ bit (INTCON2<7>).
3: The CCP2 input/output is multiplexed with RB3 if the CCP2MX bit is enabled (='0') in the configuration register.

13.0 TIMER3 MODULE

The Timer3 module timer/counter has the following features:

- 16-bit timer/counter
  (two 8-bit registers; TMR3H and TMR3L)
- Readable and writable (both registers)
- Internal or external clock select
- Interrupt-on-overflow from FFFFh to 0000h
- RESET from CCP module trigger

Figure 13-1 is a simplified block diagram of the Timer3 module.

Register 13-1 shows the Timer3 control register. This register controls the Operating mode of the Timer3 module and sets the CCP clock source.

Register 11-1 shows the Timer1 control register. This register controls the Operating mode of the Timer1 module, as well as contains the Timer1 oscillator enable bit (T1OSCEN), which can be a clock source for Timer3.

REGISTER 13-1: T3CON: TIMER3 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit
1 = Enables register Read/Write of Timer3 in one 16-bit operation
0 = Enables register Read/Write of Timer3 in two 8-bit operations

bit 6-3 **T3CCP2:T3CCP1:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the clock source for compare/capture CCP modules
01 = Timer3 is the clock source for compare/capture of CCP2,
Timer1 is the clock source for compare/capture of CCP1
00 = Timer1 is the clock source for compare/capture CCP modules

bit 5-4 **T3CKPS1:T3CKPS0**: Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

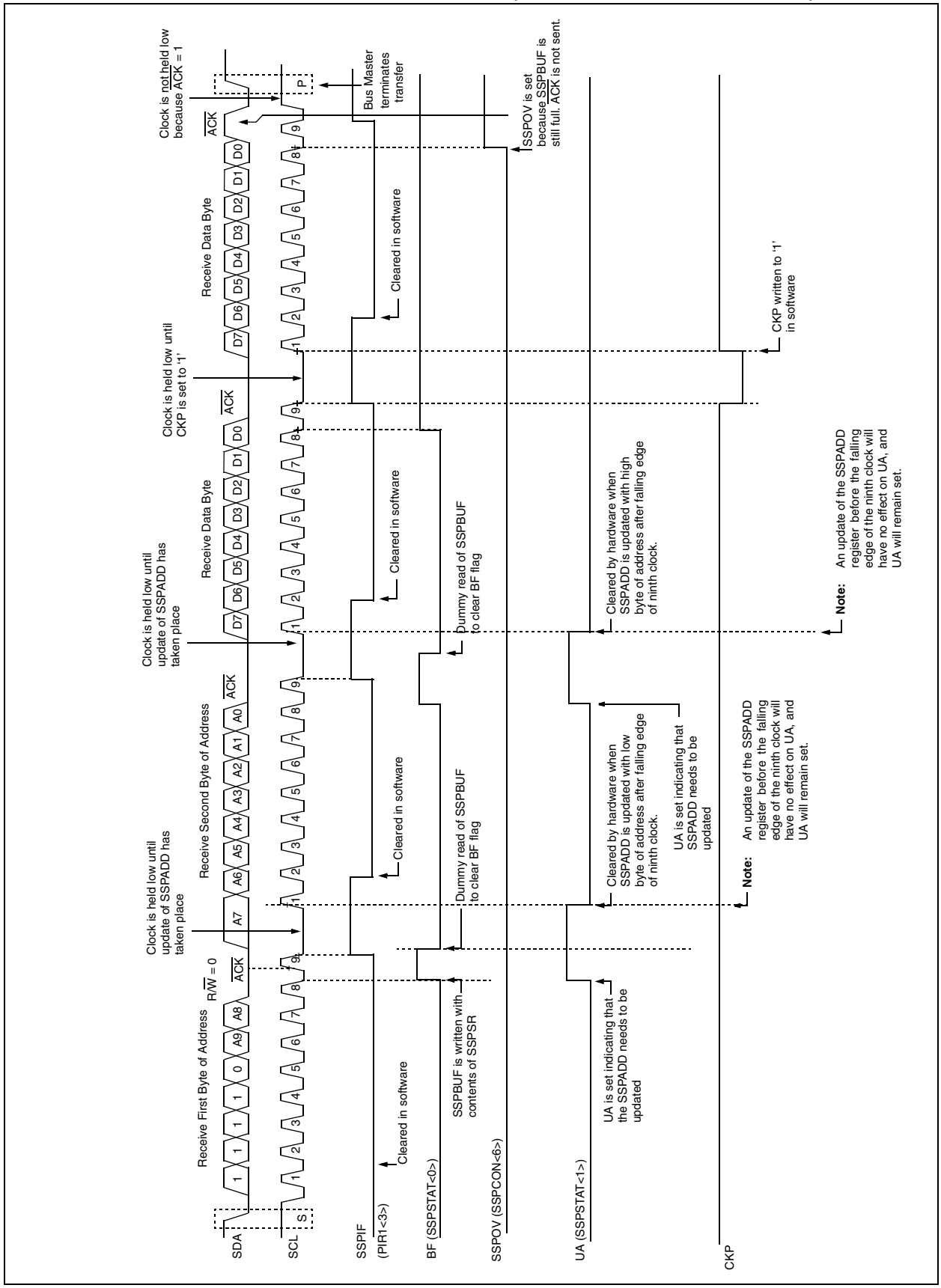
bit 2 **$\overline{\text{T3SYNC}}$:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the system clock comes from Timer1/Timer3)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T1CKI
(on the rising edge after the first falling edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

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

FIGURE 15-14: I²C SLAVE MODE TIMING SEN = 1 (RECEPTION, 10-BIT ADDRESS)



15.4.17.1 Bus Collision During a START Condition

During a START condition, a bus collision occurs if:

a) SDA or SCL are sampled low at the beginning of the START condition (Figure 15-26).

b) SCL is sampled low before SDA is asserted low (Figure 15-27).

During a START condition, both the SDA and the SCL pins are monitored.

If the SDA pin is already low, or the SCL pin is already low, then all of the following occur:

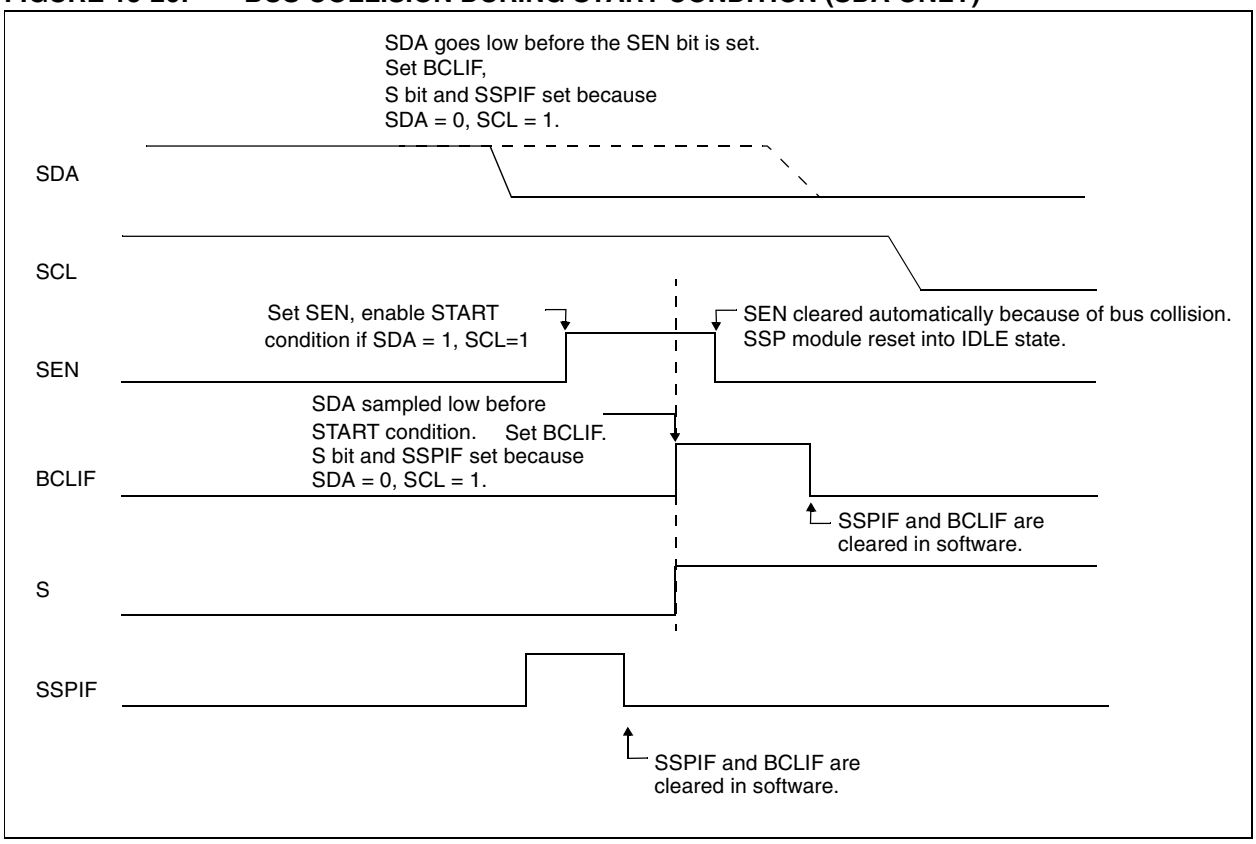
- the START condition is aborted,
- the BCLIF flag is set, and
- the MSSP module is reset to its IDLE state (Figure 15-26).

The START condition begins with the SDA and SCL pins de-asserted. When the SDA pin is sampled high, the baud rate generator is loaded from SSPADD<6:0> and counts down to 0. If the SCL pin is sampled low while SDA is high, a bus collision occurs, because it is assumed that another master is attempting to drive a data '1' during the START condition.

If the SDA pin is sampled low during this count, the BRG is reset and the SDA line is asserted early (Figure 15-28). If, however, a '1' is sampled on the SDA pin, the SDA pin is asserted low at the end of the BRG count. The baud rate generator is then reloaded and counts down to 0, and during this time, if the SCL pins are sampled as '0', a bus collision does not occur. At the end of the BRG count, the SCL pin is asserted low.

Note: The reason that bus collision is not a factor during a START condition is that no two bus masters can assert a START condition at the exact same time. Therefore, one master will always assert SDA before the other. This condition does not cause a bus collision, because the two masters must be allowed to arbitrate the first address following the START condition. If the address is the same, arbitration must be allowed to continue into the data portion, Repeated START or STOP conditions.

FIGURE 15-26: BUS COLLISION DURING START CONDITION (SDA ONLY)



NOTES:

16.1 USART Baud Rate Generator (BRG)

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. In Asynchronous mode, bit BRGH (TXSTA<2>) also controls the baud rate. In Synchronous mode, bit BRGH is ignored. Table 16-1 shows the formula for computation of the baud rate for different USART modes, which only apply in Master mode (internal clock).

Given the desired baud rate and Fosc, the nearest integer value for the SPBRG register can be calculated using the formula in Table 16-1. From this, the error in baud rate can be determined.

Example 16-1 shows the calculation of the baud rate error for the following conditions:

- FOSC = 16 MHz
- Desired Baud Rate = 9600
- BRGH = 0
- SYNC = 0

It may be advantageous to use the high baud rate (BRGH = 1) even for slower baud clocks. This is because the FOSC/(16(X + 1)) equation can reduce the baud rate error in some cases.

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

16.1.1 SAMPLING

The data on the RC7/RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX pin.

EXAMPLE 16-1: CALCULATING BAUD RATE ERROR

Desired Baud Rate = FOSC / (64 (X + 1))

Solving for X:

X = ((FOSC / Desired Baud Rate) / 64) – 1
X = ((16000000 / 9600) / 64) – 1
X = [25.042] = 25

Calculated Baud Rate = 16000000 / (64 (25 + 1))
= 9615

$$\text{Error} = \frac{\text{(Calculated Baud Rate – Desired Baud Rate)}}{\text{Desired Baud Rate}}$$

= (9615 – 9600) / 9600
= 0.16%

TABLE 16-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64(X+1))	Baud Rate = FOSC/(16(X+1))
1	(Synchronous) Baud Rate = FOSC/(4(X+1))	N/A

Legend: X = value in SPBRG (0 to 255)

TABLE 16-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

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

20.1 Instruction Set

ADDLW **ADD literal to W**

Syntax: [*label*] ADDLW k

Operands: $0 \le k \le 255$

Operation: $(W) + k \rightarrow W$

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

Encoding:

0000	1111	kkkk	kkkk

Description: The contents of W are added to the 8-bit literal 'k' and the result is placed in W.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ADDLW 0x15`

Before Instruction
W = 0x10

After Instruction
W = 0x25

ADDWF **ADD W to f**

Syntax: [*label*] ADDWF f [,d [,a]

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

Operation: $(W) + (f) \rightarrow dest$

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

Encoding:

0010	01da	ffff	ffff

Description: Add W to 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 will be selected. If 'a' is 1, the BSR is used.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `ADDWF REG, 0, 0`

Before Instruction
W = 0x17
REG = 0xC2

After Instruction
W = 0xD9
REG = 0xC2

NEGF **Negate f**

Syntax: [*label*] NEGF f [,a]

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

Operation: $(\overline{f}) + 1 \rightarrow f$

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

Encoding:

0110	110a	ffff	ffff

Description: Location ‘f’ is negated using two’s complement. The result is placed in the data memory location 'f'. If ’a’ is 0, the Access Bank will be selected, overriding the BSR value. If ’a’ = 1, then the bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `NEGF REG, 1`

Before Instruction
REG = `0011 1010` [0x3A]
After Instruction
REG = `1100 0110` [0xC6]

NOP **No Operation**

Syntax: [*label*] NOP

Operands: None

Operation: No operation

Status Affected: None

Encoding:

0000 1111	0000 xxxx	0000 xxxx	0000 xxxx

Description: No operation.

Words: 1

Cycles: 1

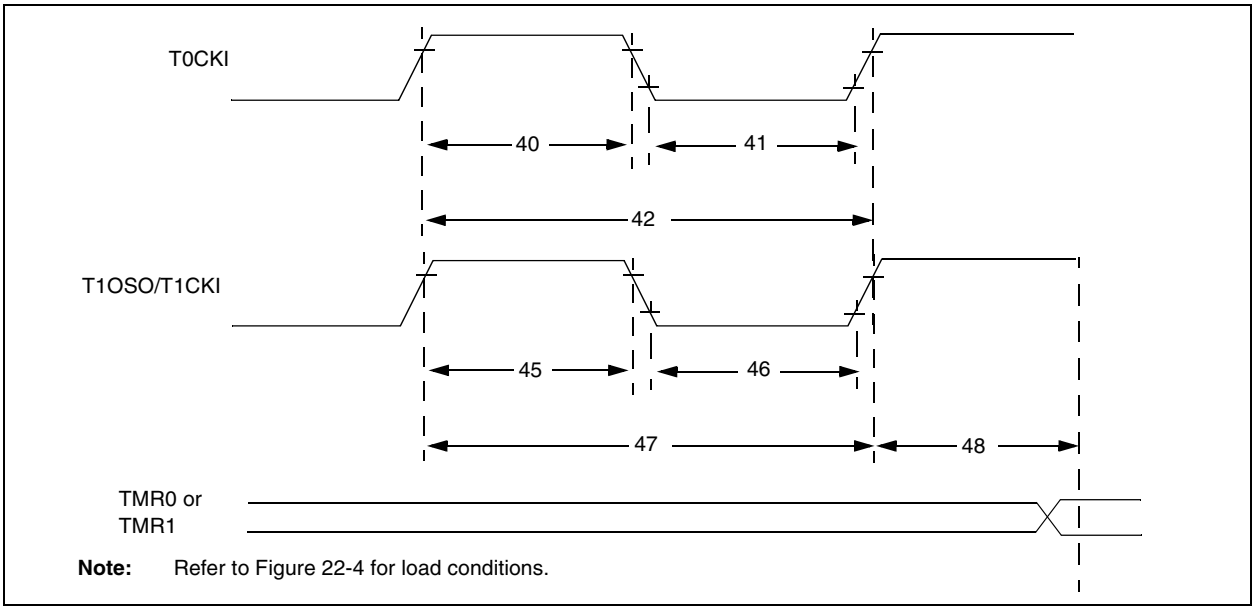
Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

Example:

None.

FIGURE 22-9: TIMER0 AND TIMER1 EXTERNAL CLOCK TIMINGS



Note: Refer to Figure 22-4 for load conditions.

TABLE 22-8: TIMER0 AND TIMER1 EXTERNAL CLOCK REQUIREMENTS

Param No.	Symbol	Characteristic			Min	Max	Units	Conditions
40	Tt0H	T0CKI High Pulse Width		No Prescaler	0.5TCY + 20	—	ns	
				With Prescaler	10	—	ns	
41	Tt0L	T0CKI Low Pulse Width		No Prescaler	0.5TCY + 20	—	ns	
				With Prescaler	10	—	ns	
42	Tt0P	T0CKI Period		No Prescaler	TCY + 10	—	ns	
				With Prescaler	Greater of: 20 ns or $\frac{T_{CY} + 40}{N}$	—	ns	N = prescale value (1, 2, 4,..., 256)
45	Tt1H	T1CKI High Time	Synchronous, no prescaler		0.5TCY + 20	—	ns	
			Synchronous, with prescaler	PIC18**F**XXX	10	—	ns	
				PIC18**LF**XXX	25	—	ns	
			Asynchronous	PIC18**F**XXX	30	—	ns	
				PIC18**LF**XXX	50	—	ns	
46	Tt1L	T1CKI Low Time	Synchronous, no prescaler		0.5TCY + 5	—	ns	
			Synchronous, with prescaler	PIC18**F**XXX	10	—	ns	
				PIC18**LF**XXX	25	—	ns	
			Asynchronous	PIC18**F**XXX	30	—	ns	
				PIC18**LF**XXX	50	—	ns	
47	Tt1P	T1CKI input period	Synchronous		Greater of: 20 ns or $\frac{T_{CY} + 40}{N}$	—	ns	N = prescale value (1, 2, 4, 8)
			Asynchronous		60	—	ns	
	Ft1	T1CKI oscillator input frequency range			DC	50	kHz	
48	Tcke2tmrl	Delay from external T1CKI clock edge to timer increment			2 TOSC	7 TOSC	—	

FIGURE 23-9: TYPICAL IDD vs. FOSC OVER VDD (EC MODE)

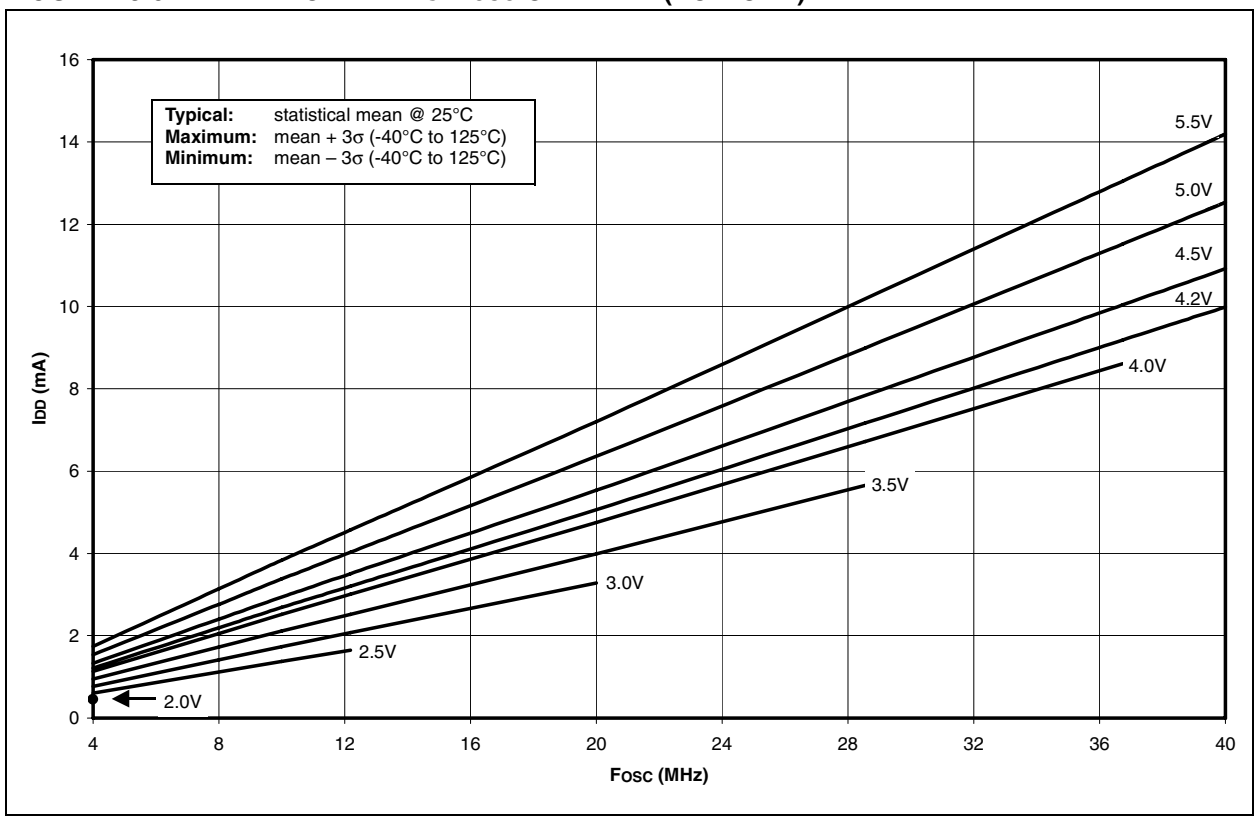


FIGURE 23-10: MAXIMUM IDD vs. FOSC OVER VDD (EC MODE)

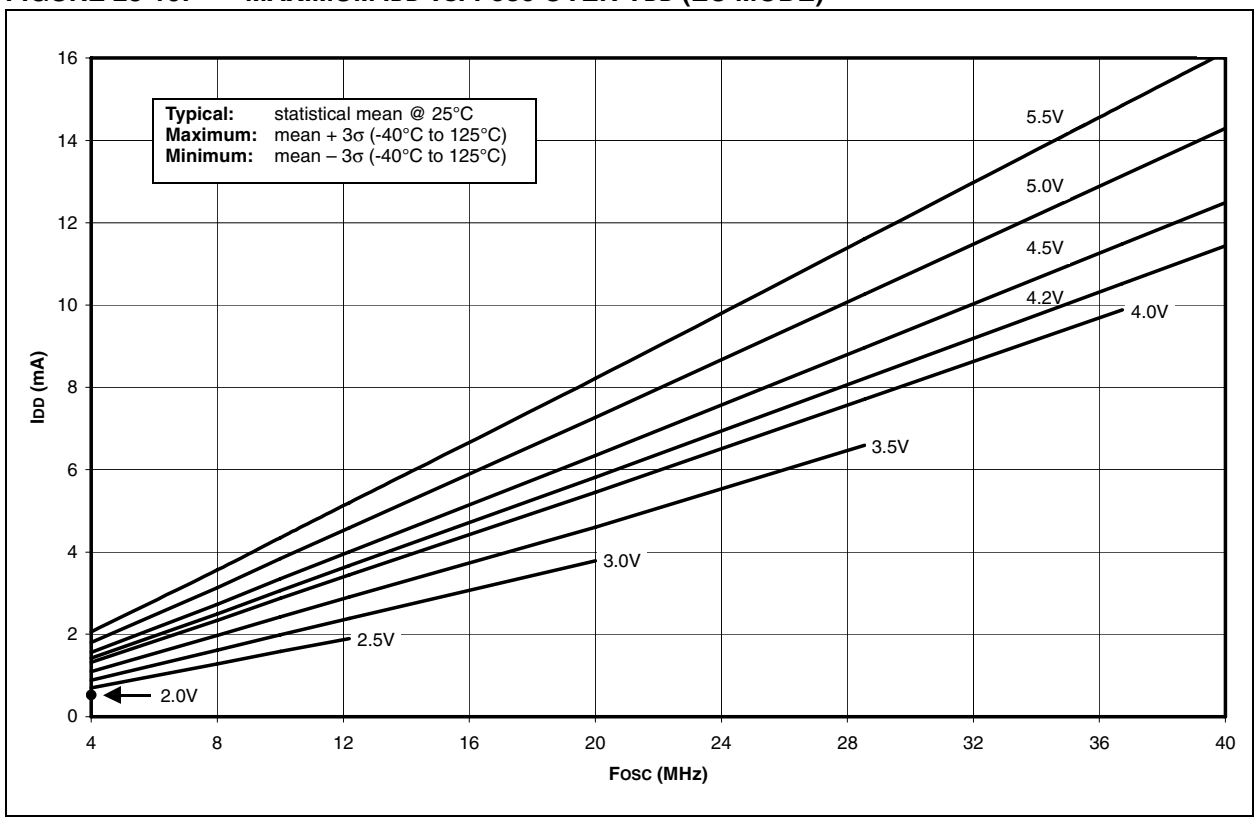


FIGURE 23-15: I_{PD} vs. V_{DD}, -40°C TO +125°C (SLEEP MODE, ALL PERIPHERALS DISABLED)

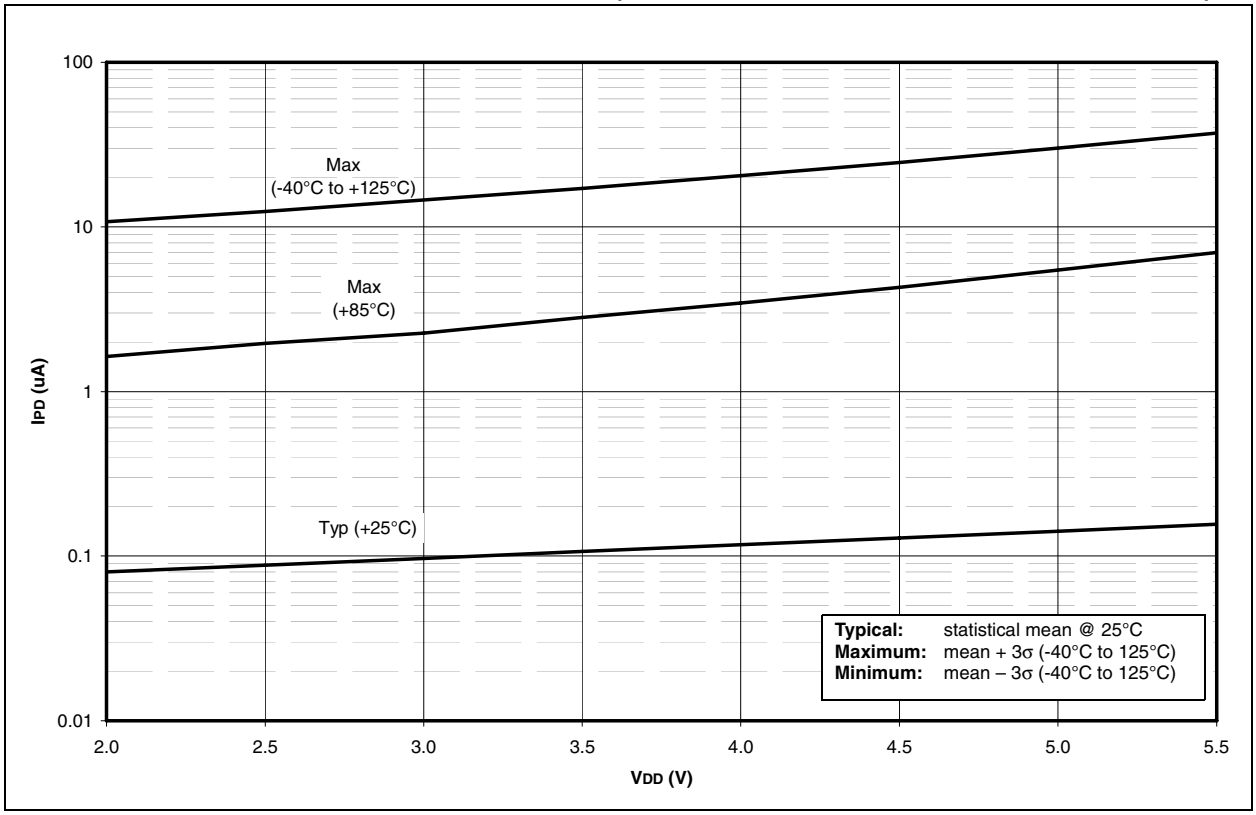


FIGURE 23-16: ΔI_{BOR} vs. V_{DD} OVER TEMPERATURE (BOR ENABLED, V_{BOR} = 2.00 - 2.16V)

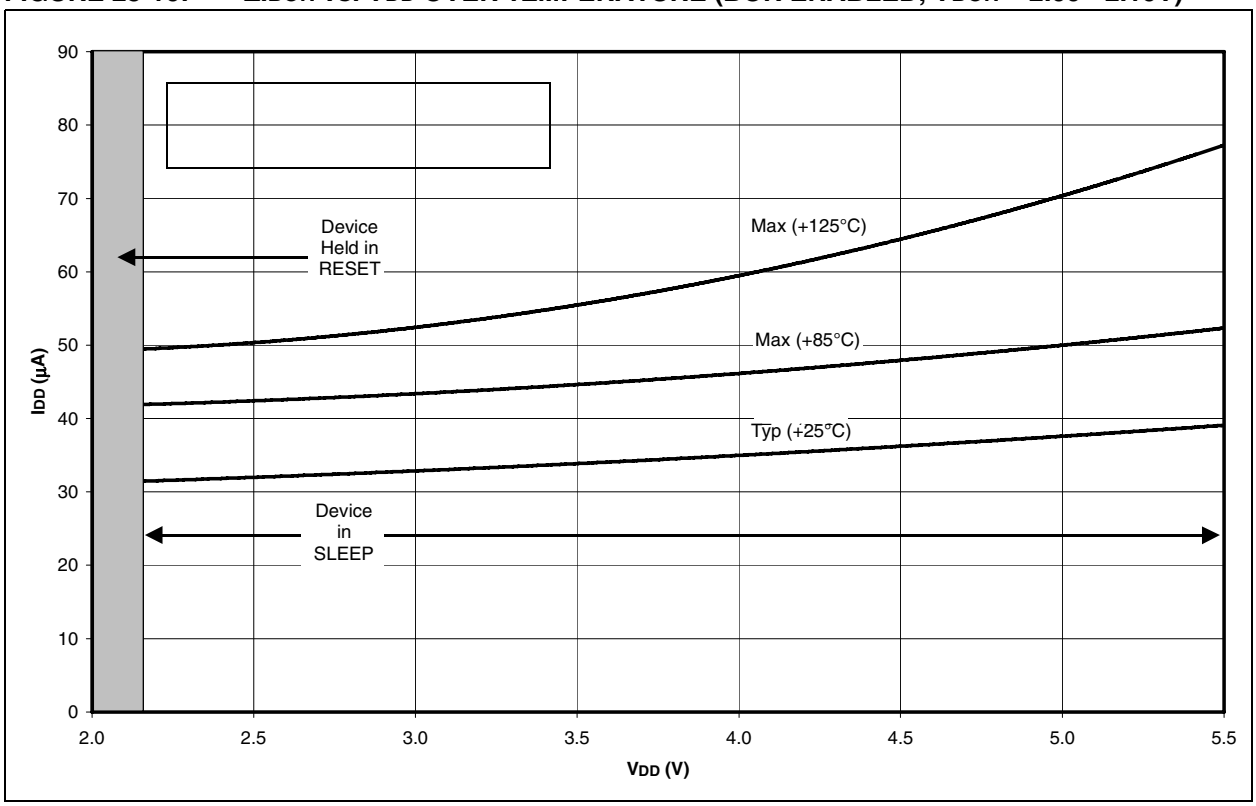


FIGURE 23-27: MINIMUM AND MAXIMUM VIN vs. VDD (I2C INPUT, -40°C TO +125°C)

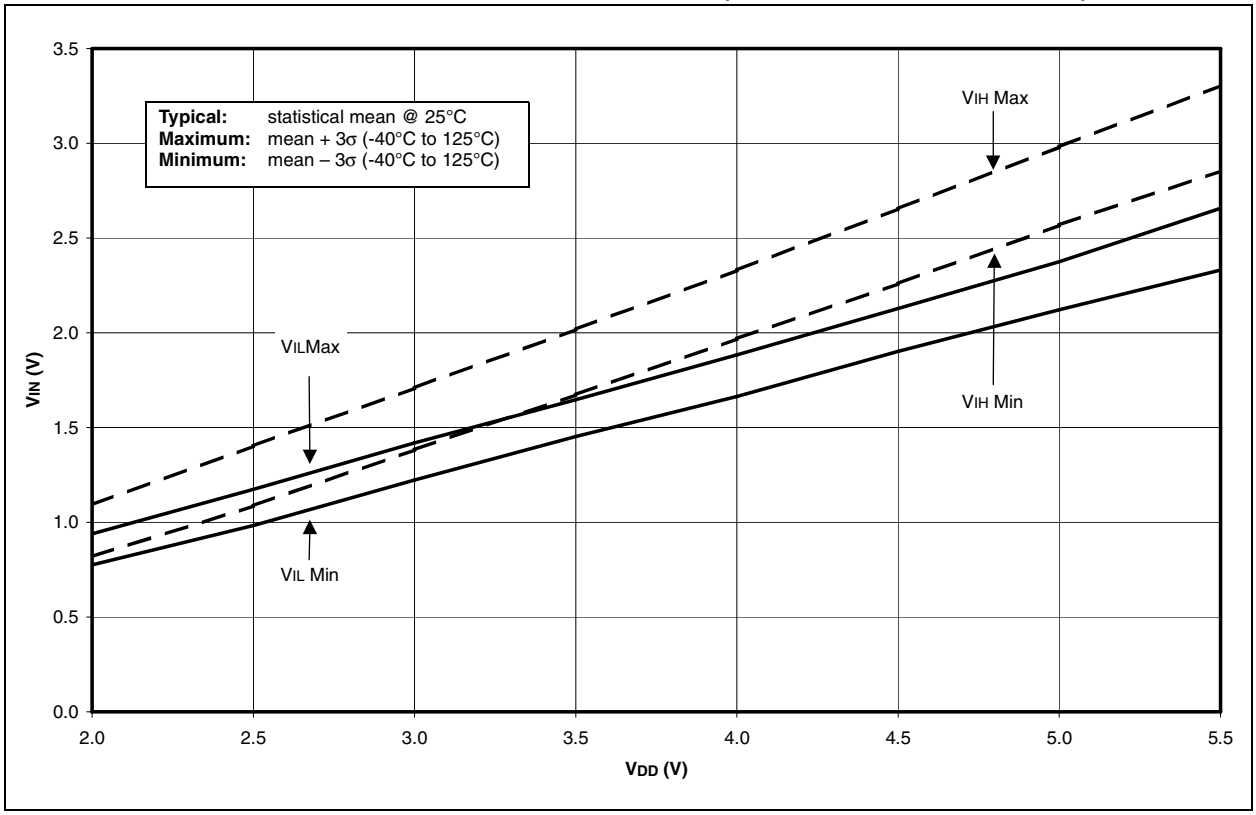
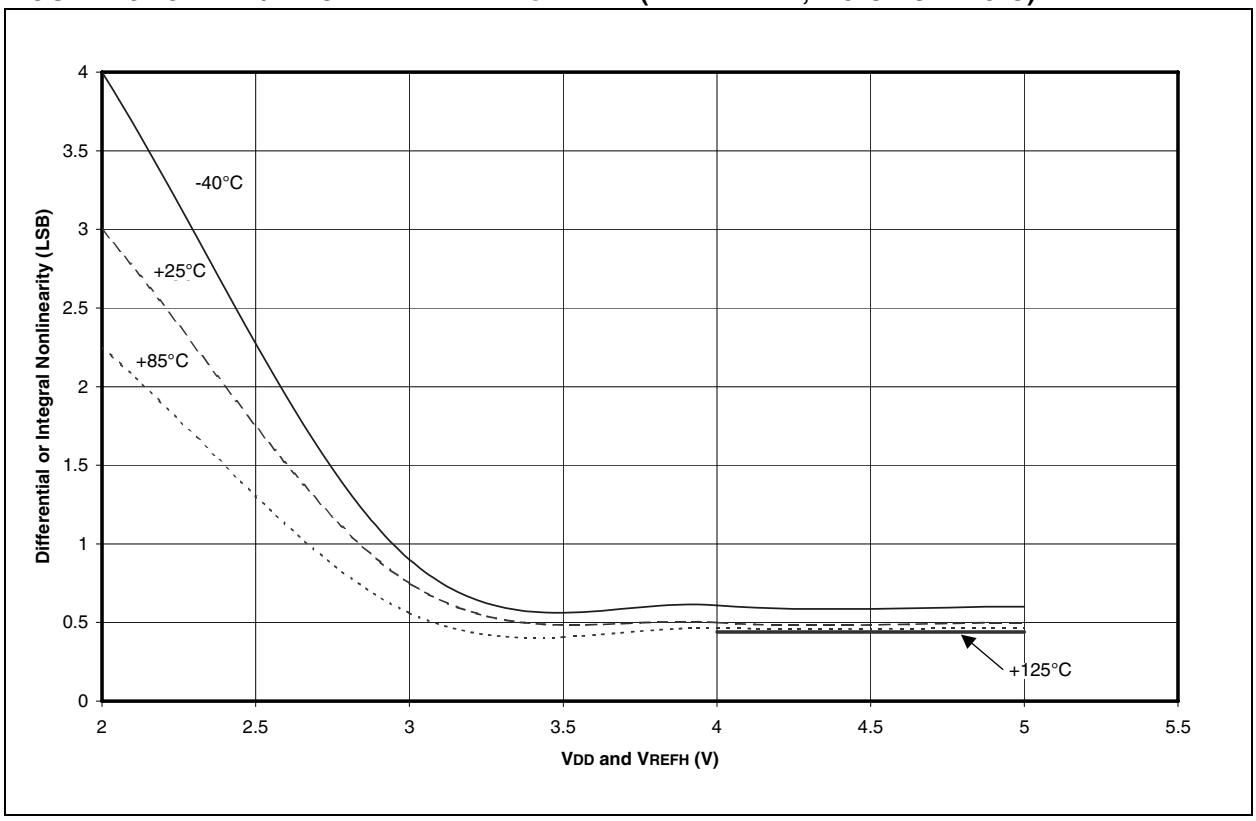


FIGURE 23-28: A/D NON-LINEARITY vs. VREFH (VDD = VREFH, -40°C TO +125°C)



PIC18FXX2 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	/XX	XXX
Device		Temperature Range	Package	Pattern

Device	PIC18FXX2(1), PIC18FXX2T(2); VDD range 4.2V to 5.5V PIC18LFXX2(1), PIC18LFXX2T(2); VDD range 2.5V to 5.5V
Temperature Range	I = -40°C to +85°C (Industrial) E = -40°C to +125°C (Extended)
Package	PT = TQFP (Thin Quad Flatpack) SO = SOIC SP = Skinny Plastic DIP P = PDIP L = PLCC
Pattern	QTP, SQTP, Code or Special Requirements (blank otherwise)

Examples:

a) PIC18LF452 - I/P 301 = Industrial temp., PDIP package, Extended VDD limits, QTP pattern #301.

b) PIC18LF242 - I/SO = Industrial temp., SOIC package, Extended VDD limits.

c) PIC18F442 - E/P = Extended temp., PDIP package, normal VDD limits.

Note 1: F = Standard Voltage range
LF = Wide Voltage Range

2: T = in tape and reel - SOIC, PLCC, and TQFP packages only.