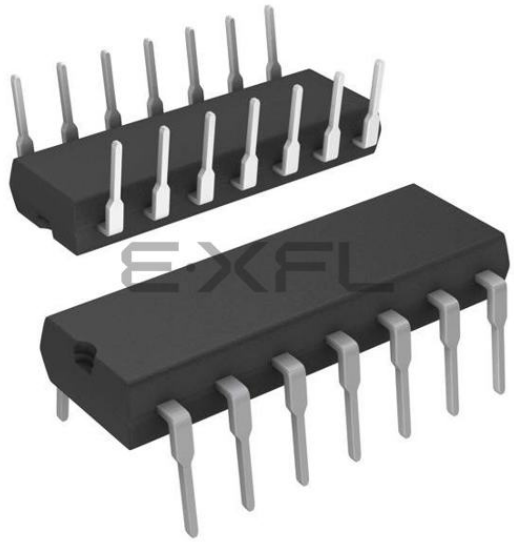


**Welcome to E-XFL.COM**

## 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 | 32MHz |
| Connectivity | I²C, SPI, UART/USART |
| Peripherals | Brown-out Detect/Reset, POR, PWM, WDT |
| Number of I/O | 11 |
| Program Memory Size | 7KB (4K x 14) |
| Program Memory Type | FLASH |
| EEPROM Size | - |
| RAM Size | 256 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 | Through Hole |
| Package / Case | 14-DIP (0.300", 7.62mm) |
| Supplier Device Package | 14-PDIP |
| Purchase URL | https://www.e-xfl.com/product-detail/microchip-technology/pic16lf1554-i-p |

## 3.0 MEMORY ORGANIZATION

These devices contain the following types of memory:

- Program Memory
  - Configuration Words
  - Device ID
  - User ID
  - Flash Program Memory
- Data Memory
  - Core Registers
  - Special Function Registers
  - General Purpose RAM
  - Common RAM

The following features are associated with access and control of program memory and data memory:

- PCL and PCLATH
- Stack
- Indirect Addressing

### 3.1 Program Memory Organization

The enhanced mid-range core has a 15-bit program counter capable of addressing a 32K x 14 program memory space. Table 3-1 shows the memory sizes implemented. Accessing a location above these boundaries will cause a wrap-around within the implemented memory space. The Reset vector is at 0000h and the interrupt vector is at 0004h (see Figure 3-1).

**TABLE 3-1: DEVICE SIZES AND ADDRESSES**

| Device | Program Memory Space (Words) | Last Program Memory Address | High-Endurance Flash Memory Address Range (1) |
|---|---|---|---|
| PIC16LF1554 | 4,096 | 0FFFh | 0F80h-0FFFh |
| PIC16LF1559 | 8,192 | 1FFFh | 1F80h-1FFFh |

**Note 1:** High-endurance Flash applies to low byte of each address in the range.

**TABLE 3-9: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)**

| Address | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bank 1** | | | | | | | | | | | |
| 080h | INDF0(1) | Addressing this location uses contents of FSR0H/FSR0L to address data memory (not a physical register) | | | | | | | | xxxx xxxx | uuuu uuuu |
| 081h | INDF1(1) | Addressing this location uses contents of FSR1H/FSR1L to address data memory (not a physical register) | | | | | | | | xxxx xxxx | uuuu uuuu |
| 082h | PCL(1) | Program Counter (PC) Least Significant Byte | | | | | | | | 0000 0000 | 0000 0000 |
| 083h | STATUS(1) | — | — | — | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | ---1 1000 | ---q quuu |
| 084h | FSR0L(1) | Indirect Data Memory Address 0 Low Pointer | | | | | | | | 0000 0000 | uuuu uuuu |
| 085h | FSR0H(1) | Indirect Data Memory Address 0 High Pointer | | | | | | | | 0000 0000 | 0000 0000 |
| 086h | FSR1L(1) | Indirect Data Memory Address 1 Low Pointer | | | | | | | | 0000 0000 | uuuu uuuu |
| 087h | FSR1H(1) | Indirect Data Memory Address 1 High Pointer | | | | | | | | 0000 0000 | 0000 0000 |
| 088h | BSR(1) | — | — | — | BSR<4:0> | | | | | ---0 0000 | ---0 0000 |
| 089h | WREG(1) | Working Register | | | | | | | | 0000 0000 | uuuu uuuu |
| 08Ah | PCLATH(1) | — | Write Buffer for the upper 7 bits of the Program Counter | | | | | | | -000 0000 | -000 0000 |
| 08Bh | INTCON(1) | GIE | PEIE | TMR0IE | INTE | IOCIE | TMR0IF | INTF | IOCIF | 0000 0000 | 0000 0000 |
| 08Ch | TRISA | — | — | TRISA5 | TRISA4 | — | TRISA2 | TRISA1 | TRISA0 | --11 1111 | --11 1111 |
| 08Dh | TRISB(2) | Unimplemented | | | | | | | | — | — |
| | TRISB(3) | TRISB7 | TRISB6 | TRISB5 | TRISB4 | — | — | — | — | 1111 ---- | 1111 ---- |
| 08Eh | TRISC(2) | — | — | TRISC5 | TRISC4 | TRISC3 | TRISC2 | TRISC1 | TRISC0 | --11 1111 | --11 1111 |
| | TRISC(3) | TRISC7 | TRISC6 | TRISC5 | TRISC4 | TRISC3 | TRISC2 | TRISC1 | TRISC0 | 1111 1111 | 1111 1111 |
| 091h | PIE1 | TMR1GIE | AD1IE | RCIE | TXIE | SSP1IE | — | TMR2IE | TMR1IE | 0000 0-00 | 0000 0-00 |
| 092h | PIE2 | — | AD2IE | — | — | BCLIE | — | — | — | -0-- 0--- | -0-- 0--- |
| 095h | OPTION_REG | $\overline{WPUEN}$ | INTEDG | TMR0CS | TMR0SE | PSA | PS<2:0> | | | 1111 1111 | 1111 1111 |
| 096h | PCON | STKOVF | STKUNF | — | $\overline{RWDT}$ | $\overline{RMCLR}$ | $\overline{RI}$ | $\overline{POR}$ | $\overline{BOR}$ | 00-1 11qq | 00-q qquu |
| 097h | WDTCON | — | — | WDTPS<4:0> | | | | | SWDTEN | --01 0110 | --01 0110 |
| 098h | — | Unimplemented | | | | | | | | — | — |
| 099h | OSCCON | SPLLEN | IRCF<3:0> | | | | — | SCS<1:0> | | 0011 1-00 | 0011 1-00 |
| 09Ah | OSCSTAT | — | PLLSR | — | HFIOFR | — | — | LFIOFR | HFIOFS | -0-0 --00 | -q-q -q0q |
| 09Bh | ADRESL/ AD1RES0L(4) | ADC1 Result Register 0 Low | | | | | | | | xxxx xxxx | uuuu uuuu |
| 09Ch | ADRESH/ AD1RES0H(4) | ADC1 Result Register 0 High | | | | | | | | xxxx xxxx | uuuu uuuu |
| 09Dh | ADCON0/ AD1CON0(4) | — | CHS4 | CHS3 | CHS2 | CHS1 | CHS0 | GO/$\overline{DONE}$1 | AD1ON | -000 0000 | -000 0000 |
| 09Eh | ADCON1/ ADCOMCON(4) | ADFM | ADCS<2:0> | | | — | GO/ $\overline{DONE}$_ALL | ADPREF<1:0> | | 0000 -000 | 0000 -000 |
| 09Fh | ADCON2/ AD1CON2(4) | — | TRIGSEL<2:0> | | | — | — | — | — | -000 ---- | -000 ---- |

**Legend:** x = unknown, u = unchanged, q = depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations unimplemented, read as '0'.

**Note 1:** These registers can be accessed from any bank.

**2:** PIC16LF1554.

**3:** PIC16LF1559.

**4:** These registers/bits are available at two address locations, in Bank 1 and Bank 14.

**TABLE 3-9: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)**

| Address | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bank 2** | | | | | | | | | | | |
| 100h | INDF0(1) | Addressing this location uses contents of FSR0H/FSR0L to address data memory (not a physical register) | | | | | | | | xxxx xxxx | uuuu uuuu |
| 101h | INDF1(1) | Addressing this location uses contents of FSR1H/FSR1L to address data memory (not a physical register) | | | | | | | | xxxx xxxx | uuuu uuuu |
| 102h | PCL(1) | Program Counter (PC) Least Significant Byte | | | | | | | | 0000 0000 | 0000 0000 |
| 103h | STATUS(1) | — | — | — | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C | ---1 1000 | ---q quuu |
| 104h | FSR0L(1) | Indirect Data Memory Address 0 Low Pointer | | | | | | | | 0000 0000 | uuuu uuuu |
| 105h | FSR0H(1) | Indirect Data Memory Address 0 High Pointer | | | | | | | | 0000 0000 | 0000 0000 |
| 106h | FSR1L(1) | Indirect Data Memory Address 1 Low Pointer | | | | | | | | 0000 0000 | uuuu uuuu |
| 107h | FSR1H(1) | Indirect Data Memory Address 1 High Pointer | | | | | | | | 0000 0000 | 0000 0000 |
| 108h | BSR(1) | — | — | — | BSR<4:0> | | | | | ---0 0000 | ---0 0000 |
| 109h | WREG(1) | Working Register | | | | | | | | 0000 0000 | uuuu uuuu |
| 10Ah | PCLATH(1) | — | Write Buffer for the upper 7 bits of the Program Counter | | | | | | | -000 0000 | -000 0000 |
| 10Bh | INTCON(1) | GIE | PEIE | TMR0IE | INTE | IOCIE | TMR0IF | INTF | IOCIF | 0000 0000 | 0000 0000 |
| 10Ch | LATA | — | — | LATA5 | LATA4 | — | LATA2 | LATA1 | LATA0 | --xx xxxx | --uu uuuu |
| 10Dh | LATB(2) | Unimplemented | | | | | | | | — | — |
| | LATB(3) | LATB7 | LATB6 | LATB5 | LATB4 | — | — | — | — | xxxx ---- | uuuu ---- |
| 10Eh | LATC(2) | — | — | LATC5 | LATC4 | LATC3 | LATC2 | LATC1 | LATC0 | --xx xxxx | --uu uuuu |
| | LATC(3) | LATC7 | LATC6 | LATC5 | LATC4 | LATC3 | LATC2 | LATC1 | LATC0 | xxxx xxxx | uuuu uuuu |
| 10Fh | — | Unimplemented | | | | | | | | — | — |
| 110h | — | Unimplemented | | | | | | | | — | — |
| 111h | — | Unimplemented | | | | | | | | — | — |
| 112h | — | Unimplemented | | | | | | | | — | — |
| 113h | — | Unimplemented | | | | | | | | — | — |
| 114h | — | Unimplemented | | | | | | | | — | — |
| 115h | — | Unimplemented | | | | | | | | — | — |
| 116h | BORCON | SBOREN | BORFS | — | — | — | — | — | BORRDY | 10-- ---q | uu-- ---u |
| 117h | FVRCON | FVREN | FVRRDY | TSEN | TSRNG | — | — | ADFVR<1:0> | | 0q00 --00 | 0q00 --00 |
| 118h | — | Unimplemented | | | | | | | | — | — |
| 119h | — | Unimplemented | | | | | | | | — | — |
| 11Ah | — | Unimplemented | | | | | | | | — | — |
| 11Bh | — | Unimplemented | | | | | | | | — | — |
| 11Ch | — | Unimplemented | | | | | | | | — | — |
| 11Dh | APFCON | RXDTSEL | SDOSEL | SSSEL | SDSEL | — | TXCKSEL | GRDBSEL | GRDASEL | 0000 -000 | 0000 -000 |
| 11Eh | — | Unimplemented | | | | | | | | — | — |
| 11Fh | — | Unimplemented | | | | | | | | — | — |

**Legend:** x = unknown, u = unchanged, q = depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations unimplemented, read as '0'.

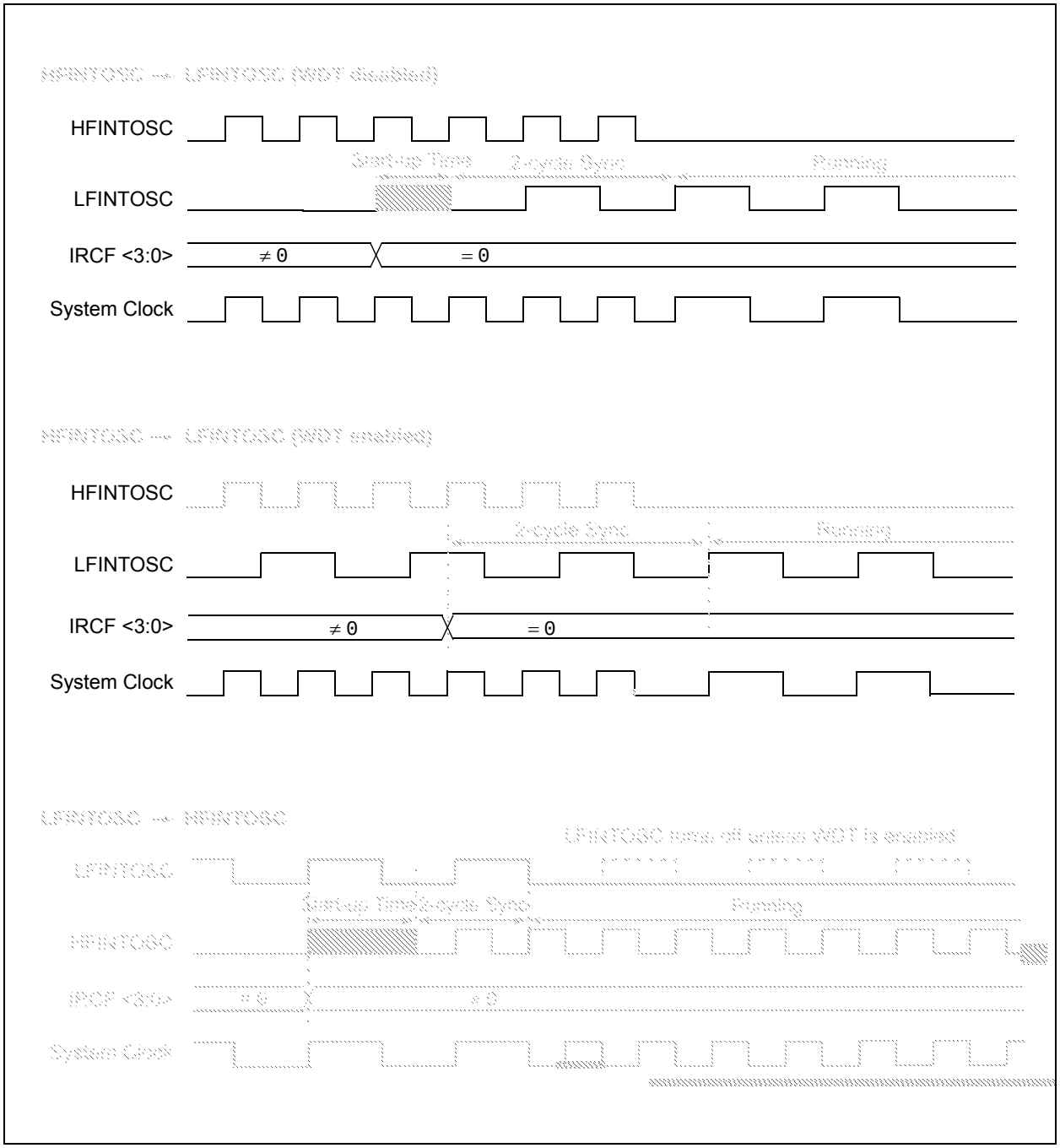
**Note 1:** These registers can be accessed from any bank.

**2:** PIC16LF1554.

**3:** PIC16LF1559.

**4:** These registers/bits are available at two address locations, in Bank 1 and Bank 14.

**FIGURE 5-3: INTERNAL OSCILLATOR SWITCH TIMING**



## 7.0 INTERRUPTS

The interrupt feature allows certain events to preempt normal program flow. Firmware is used to determine the source of the interrupt and act accordingly. Some interrupts can be configured to wake the MCU from Sleep mode.

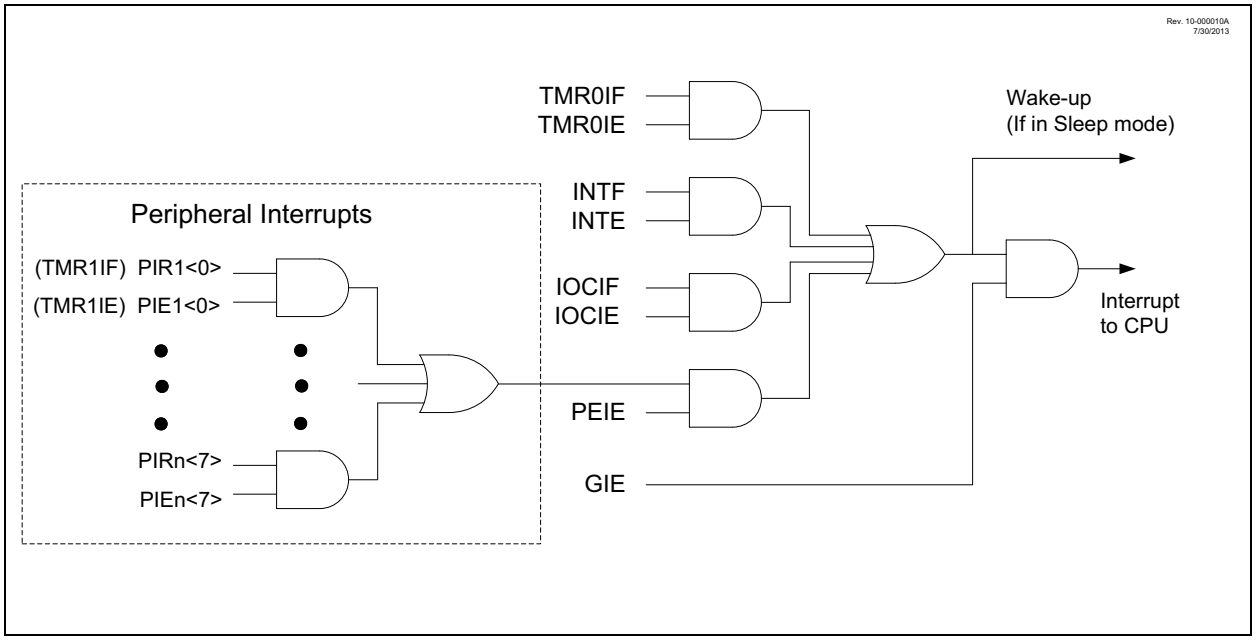
This chapter contains the following information for interrupts:

- Operation
- Interrupt latency
- Interrupts during Sleep
- INT pin
- Automatic context saving

Many peripherals produce interrupts. Refer to the corresponding chapters for details.

A block diagram of the interrupt logic is shown in Figure 7-1.

**FIGURE 7-1: INTERRUPT LOGIC**



**REGISTER 11-6: WPUA: WEAK PULL-UP PORTA REGISTER(1,2)**

| U-0 | U-0 | R/W-1/1 | R/W-1/1 | R/W-1/1 | R/W-1/1 | R/W-1/1 | R/W-1/1 |
|---|---|---|---|---|---|---|---|
| — | — | WPUA5 | WPUA4 | WPUA3 | WPUA2 | WPUA1 | WPUA0 |
| bit 7 | | | | | | | bit 0 |

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

bit 7-6 **Unimplemented:** Read as '0'

bit 5-0 **WPUA<5:0>**: Weak Pull-up Register bits(3)
1 = Pull-up enabled
0 = Pull-up disabled

**Note 1:** Global $\overline{\text{WPUEN}}$ bit of the OPTION_REG register must be cleared for individual pull-ups to be enabled.
**2:** The weak pull-up device is automatically disabled if the pin is configured as an output.
**3:** For the WPUA3 bit, when MCLRE = 1, weak pull-up is internally enabled, but not reported here.

**TABLE 11-3: SUMMARY OF REGISTERS ASSOCIATED WITH PORTA**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Register on Page |
|---|---|---|---|---|---|---|---|---|---|
| ANSELA | — | — | ANSA5 | ANSA4 | — | ANSA2 | ANSA1 | ANSA0 | 109 |
| APFCON | RXDTSEL | SDOSEL | SSSEL | SDSEL | — | TXCKSEL | GRDBSEL | GRDASEL | 106 |
| LATA | — | — | LATA5 | LATA4 | — | LATA2 | LATA1 | LATA0 | 109 |
| OPTION_REG | $\overline{\text{WPUEN}}$ | INTEDG | TMR0CS | TMR0SE | PSA | PS<2:0> | | | 166 |
| PORTA | — | — | RA5 | RA4 | RA3 | RA2 | RA1 | RA0 | 108 |
| TRISA | — | — | TRISA5 | TRISA4 | —(1) | TRISA2 | TRISA1 | TRISA0 | 108 |
| WPUA | — | — | WPUA5 | WPUA4 | WPUA3 | WPUA2 | WPUA1 | WPUA0 | 110 |

**Legend:** x = unknown, u = unchanged, – = unimplemented locations read as '0'. Shaded cells are not used by PORTA.
**Note 1:** Unimplemented, read as '1'.

**TABLE 11-4: SUMMARY OF CONFIGURATION WORD WITH PORTA**

| Name | Bits | Bit -/7 | Bit -/6 | Bit 13/5 | Bit 12/4 | Bit 11/3 | Bit 10/2 | Bit 9/1 | Bit 8/0 | Register on Page |
|---|---|---|---|---|---|---|---|---|---|---|
| CONFIG1 | 13:8 | — | — | — | — | $\overline{\text{CLKOUTEN}}$ | BOREN<1:0> | | — | 53 |
| | 7:0 | $\overline{\text{CP}}$ | MCLRE | $\overline{\text{PWRTE}}$ | WDTE<1:0> | | — | FOSC<2:0> | | |

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

**REGISTER 11-11: WPUB: WEAK PULL-UP PORTB REGISTER(1,2)**

| R/W-1/1 | R/W-1/1 | R/W-1/1 | R/W-1/1 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| WPUB7 | WPUB6 | WPUB5 | WPUB4 | — | — | — | — |
| bit 7 | | | | | | | bit 0 |

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

bit 7-4 **WPUB<7:4>**: Weak Pull-up Register bits
1 = Pull-up enabled
0 = Pull-up disabled

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

**Note 1:** Global $\overline{\text{WPUEN}}$ bit of the OPTION_REG register must be cleared for individual pull-ups to be enabled.
**2:** The weak pull-up device is automatically disabled if the pin is configured as an output.

**TABLE 11-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB(1)**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Register on Page |
|---|---|---|---|---|---|---|---|---|---|
| ANSELB | ANSB7 | ANSB6 | ANSB5 | ANSB4 | — | — | — | — | 113 |
| APFCON | RXDTSEL | SDOSEL | SSSEL | SDSEL | — | TXCKSEL | GRDBSEL | GRDASEL | 106 |
| LATB | LATB7 | LATB6 | LATB5 | LATB4 | — | — | — | — | 113 |
| OPTION_REG | $\overline{\text{WPUEN}}$ | INTEDG | TMR0CS | TMR0SE | PSA | PS<2:0> | | | 166 |
| PORTB | RB7 | RB6 | RB5 | RB4 | — | — | — | — | 112 |
| TRISB | TRISB7 | TRISB6 | TRISB5 | TRISB4 | — | — | — | — | 112 |
| WPUB | WPUB7 | WPUB6 | WPUB5 | WPUB4 | — | — | — | — | 114 |

**Legend:** x = unknown, u = unchanged, – = unimplemented locations read as '0'. Shaded cells are not used by PORTB.
**Note 1:** PIC16LF1559 only.

**TABLE 11-7: SUMMARY OF CONFIGURATION WORD WITH PORTB**

| Name | Bits | Bit -/7 | Bit -/6 | Bit 13/5 | Bit 12/4 | Bit 11/3 | Bit 10/2 | Bit 9/1 | Bit 8/0 | Register on Page |
|---|---|---|---|---|---|---|---|---|---|---|
| CONFIG1 | 13:8 | — | — | — | — | $\overline{\text{CLKOUTEN}}$ | BOREN<1:0> | | — | 53 |
| | 7:0 | $\overline{\text{CP}}$ | MCLRE | $\overline{\text{PWRTE}}$ | WDTE<1:0> | | — | FOSC<1:0> | | |

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

### 14.4 ADC Acquisition Time

To ensure accurate temperature measurements, the user must wait at least 200 μs after the ADC input multiplexer is connected to the temperature indicator output before the conversion is performed. In addition, the user must wait 200 μs between sequential conversions of the temperature indicator output.

**TABLE 14-2: SUMMARY OF REGISTERS ASSOCIATED WITH THE TEMPERATURE INDICATOR**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Register on page |
|---|---|---|---|---|---|---|---|---|---|
| FVRCON | FVREN | FVRRDY | TSEN | TSRNG | — | — | ADFVR<1:0> | | 118 |

**Legend:** Shaded cells are unused by the temperature indicator module.

### 15.2.6 INDIVIDUAL ADC CONVERSION PROCEDURE

This is an example procedure for using the ADCx to perform an Analog-to-Digital conversion:

1. Configure Port:
   - Disable pin output driver (Refer to the TRISx register)
   - Configure pin as analog (Refer to the ANSELx register)
   - Disable weak pull-ups either globally (refer to the OPTION_REG register) or individually (Refer to the appropriate WPUx register)
2. Configure the ADCx module:
   - Select ADCx conversion clock
   - Configure voltage reference
   - Select ADCx input channel
   - Turn on ADCx module
3. Configure ADCx interrupt (optional):
   - Clear ADCx interrupt flag
   - Enable ADCx interrupt
   - Enable peripheral interrupt
   - Enable global interrupt**(1)**
4. Wait the required acquisition time**(2)**.
5. Start conversion by setting the GO/$\overline{\text{DONE}}$x bit.
6. Wait for ADCx conversion to complete by one of the following:
   - Polling the GO/$\overline{\text{DONE}}$x bit
   - Waiting for the ADCx interrupt (interrupts enabled)
7. Read ADCx Result.
8. Clear the ADCx interrupt flag (required if interrupt is enabled).

**Note 1:** The global interrupt can be disabled if the user is attempting to wake-up from Sleep and resume in-line code execution.

**2:** Refer to **Section 15.4 "ADC Acquisition Requirements"**.

**EXAMPLE 15-1: ADC CONVERSION**

```
;This code block configures the ADC1
;for polling, Vdd and Vss references, FRC
;oscillator and AN0 input.
;
;Conversion start and polling for completion
;are included.
;
BANKSEL   ADCON1       ;
MOVLW     B'11110000'  ;Right justify, FRC
                       ;oscillator
MOVWF     ADCON1       ;VDD is VREFH
BANKSEL   TRISA        ;
BSF       TRISA,0      ;Set RA0 to input
BANKSEL   ANSELA       ;
BSF       ANSELA,0     ;Set RA0 to analog
BANKSEL   WPUA
BCF       WPUA,0       ;Disable RA0 weak
                       pull-up
BANKSEL   ADCON0       ;
MOVLW     B'00000001'  ;Select channel AN0
MOVWF     ADCON0       ;Turn ADC On
MOVLW     .5
MOVWF     AAD1ACQ      ;Acquisiton delay
BSF       ADCON0,ADGO  ;Start conversion
BTFSC     ADCON0,ADGO  ;Is conversion done?
GOTO      $-1          ;No, test again
BANKSEL   AD1RES0H     ;
MOVF      AD1RES0H,W   ;Read upper 2 bits
MOVWF     RESULTHI     ;store in GPR space
BANKSEL   AD1RES0L     ;
MOVF      AD1RES0L,W   ;Read lower 8 bits
MOVWF     RESULTLO     ;Store in GPR space
```

**REGISTER 16-4: AAD2CH: HARDWARE CVD 2 SECONDARY CHANNEL SELECT REGISTER(1,2,3,4)**

| U-0 | R/W-0/0 | R/W-0/0 | R/W-0/0 | R/W-0/0 | R/W-0/0 | R/W-0/0 | R/W-0/0 |
|---|---|---|---|---|---|---|---|
| — | CH26 | CH25 | CH24 | CH23 | CH22 | CH21 | CH20 |
| bit 7 | | | | | | | bit 0 |

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

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

bit 6 **CH26:** Channel 26 to ADC2 Connection bit(5)
- 1 = AN26 is connected to ADC2
- 0 = AN26 is not connected to ADC2

bit 5 **CH25:** Channel 25 to ADC2 Connection bit(5)
- 1 = AN25 is connected to ADC2
- 0 = AN25 is not connected to ADC2

bit 4 **CH24:** Channel 24 to ADC2 Connection bit(5)
- 1 = AN24 is connected to ADC2
- 0 = AN24 is not connected to ADC2

bit 3 **CH23:** Channel 23 to ADC2 Connection bit
- 1 = AN23 is connected to ADC2
- 0 = AN23 is not connected to ADC2

bit 2 **CH22:** Channel 22 to ADC2 Connection bit
- 1 = AN22 is connected to ADC2
- 0 = AN22 is not connected to ADC2

bit 1 **CH21:** Channel 21 to ADC2 Connection bit
- 1 = AN21 is connected to ADC2
- 0 = AN21 is not connected to ADC2

bit 0 **CH20:** Channel 20 to ADC2 Connection bit
- 1 = AN20 is connected to ADC2
- 0 = AN20 is not connected to ADC2

**Note 1:** This register selects secondary channels which are connected in parallel to the primary channel selected in AAD2CON0. Precharge bias is applied to both the primary and secondary channels.

**2:** If the same channel is selected as both primary and secondary then the selection as primary takes precedence.

**3:** Enabling these bits automatically overrides the corresponding TRIS bit to tri-state the selected pin.

**4:** In the same way that the CHS bits in AAD2CON0 only close the switch when the ADC is enabled, these connections and the TRISx overrides are only active if the ADC is enabled by setting ADxON.

**5:** PIC16LF1559 only. Unimplemented/Read as '0' on PIC16LF1554.

## 19.0 TIMER2 MODULE

The Timer2 module incorporates the following features:

- 8-bit Timer and Period registers (TMR2 and PR2, respectively)
- Readable and writable (both registers)
- Software programmable prescaler (1:1, 1:4, 1:16, and 1:64)
- Software programmable postscaler (1:1 to 1:16)
- Interrupt on TMR2 match with PR2

See Figure 19-1 for a block diagram of Timer2.

**FIGURE 19-1: TIMER2 BLOCK DIAGRAM**

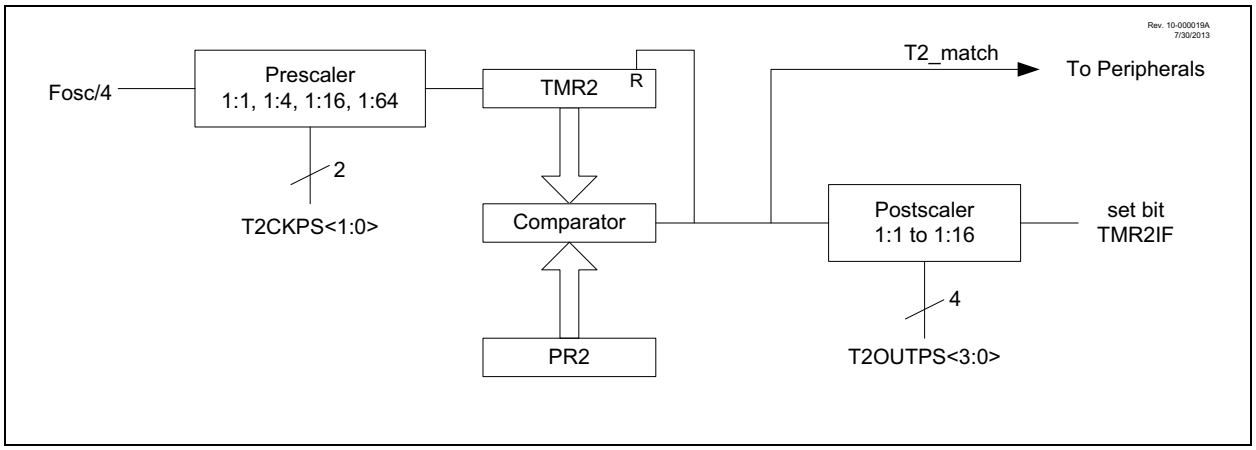
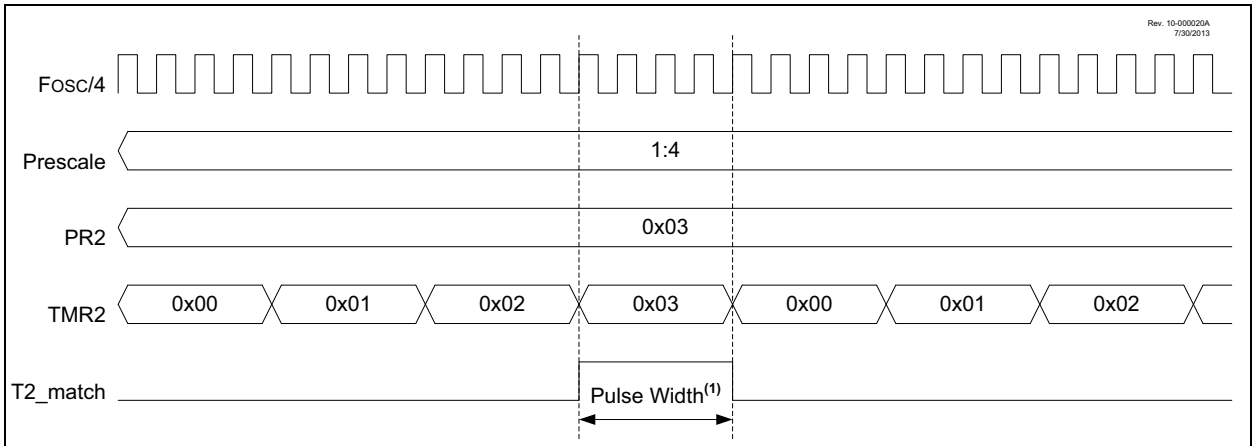



**FIGURE 19-2: TIMER2 TIMING DIAGRAM**




**Note 1:** The Pulse Width of T2_match is equal to the scaled input of TMR2.

## 20.3 I²C MODE OVERVIEW

The Inter-Integrated Circuit Bus (I²C) is a multi-master serial data communication bus. Devices communicate in a master/slave environment where the master devices initiate the communication. A Slave device is controlled through addressing.

The I²C bus specifies two signal connections:

- Serial Clock (SCL)
- Serial Data (SDA)

Figure 20-2 and Figure 20-3 show the block diagrams of the MSSP module when operating in I²C mode.

Both the SCL and SDA connections are bidirectional open-drain lines, each requiring pull-up resistors for the supply voltage. Pulling the line to ground is considered a logical zero and letting the line float is considered a logical one.

Figure 20-11 shows a typical connection between two processors configured as master and slave devices.

The I²C bus can operate with one or more master devices and one or more slave devices.

There are four potential modes of operation for a given device:

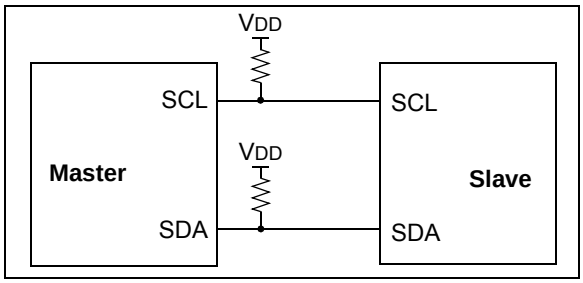
- Master Transmit mode
  (master is transmitting data to a slave)
- Master Receive mode
  (master is receiving data from a slave)
- Slave Transmit mode
  (slave is transmitting data to a master)
- Slave Receive mode
  (slave is receiving data from the master)

To begin communication, a master device starts out in Master Transmit mode. The master device sends out a Start bit followed by the address byte of the slave it intends to communicate with. This is followed by a single Read/Write bit, which determines whether the master intends to transmit to or receive data from the slave device.

If the requested slave exists on the bus, it will respond with an Acknowledge bit, otherwise known as an $\overline{ACK}$. The master then continues in either Transmit mode or Receive mode and the slave continues in the complement, either in Receive mode or Transmit mode, respectively.

A Start bit is indicated by a high-to-low transition of the SDA line while the SCL line is held high. Address and data bytes are sent out, Most Significant bit (MSb) first. The Read/Write bit is sent out as a logical one when the master intends to read data from the slave, and is sent out as a logical zero when it intends to write data to the slave.

**FIGURE 20-11: I²C MASTER/ SLAVE CONNECTION**




The Acknowledge bit ($\overline{ACK}$) is an active-low signal, which holds the SDA line low to indicate to the transmitter that the slave device has received the transmitted data and is ready to receive more.

The transition of a data bit is always performed while the SCL line is held low. Transitions that occur while the SCL line is held high are used to indicate Start and Stop bits.

If the master intends to write to the slave, then it repeatedly sends out a byte of data, with the slave responding after each byte with an $\overline{ACK}$ bit. In this example, the master device is in Master Transmit mode and the slave is in Slave Receive mode.

If the master intends to read from the slave, then it repeatedly receives a byte of data from the slave, and responds after each byte with an $\overline{ACK}$ bit. In this example, the master device is in Master Receive mode and the slave is Slave Transmit mode.

On the last byte of data communicated, the master device may end the transmission by sending a Stop bit. If the master device is in Receive mode, it first sends a not $\overline{ACK}$ bit in place of an $\overline{ACK}$ and then terminates the transfer with a Stop bit.

In some cases, the master may want to maintain control of the bus and re-initiate another transmission. If so, the master device may send another Start bit in place of the Stop bit or last $\overline{ACK}$ bit when it is in receive mode.

The I²C bus specifies three message protocols;

- Single message where a master writes data to a slave.
- Single message where a master reads data from a slave.
- Combined message where a master initiates a minimum of two writes, or two reads, or a combination of writes and reads, to one or more slaves.

When one device is transmitting a logical one, or letting the line float, and a second device is transmitting a logical zero, or holding the line low, the first device can detect that the line is not a logical one. This detection, when used on the SCL line, is called clock stretching. Clock stretching gives slave devices a mechanism to control the flow of data. When this detection is used on

the SDA line, it is called arbitration. Arbitration ensures that there is only one master device communicating at any single time.

### 20.3.1 CLOCK STRETCHING

When a slave device has not completed processing data, it can delay the transfer of more data through the process of clock stretching. An addressed slave device may hold the SCL clock line low after receiving or sending a bit, indicating that it is not yet ready to continue. The master that is communicating with the slave will attempt to raise the SCL line in order to transfer the next bit, but will detect that the clock line has not yet been released. Because the SCL connection is open-drain, the slave has the ability to hold that line low until it is ready to continue communicating.

Clock stretching allows receivers that cannot keep up with a transmitter to control the flow of incoming data.

### 20.3.2 ARBITRATION

Each master device must monitor the bus for Start and Stop bits. If the device detects that the bus is busy, it cannot begin a new message until the bus returns to an Idle state.

However, there are three conditions under which a collision can occur:

a) Two master devices may try to initiate a transmission on or about the same time
b) A device acting as multiple devices on the bus (one of which is acting as a master) may collide with another master which is trying to access a slave address on the first device
c) Two slaves may respond to a general call read at the same time

#### 20.3.2.1 Multi-Master Collision

In the first condition each master transmitter checks the level of the SDA data line and compares it to the level that it expects to find. The first transmitter to observe that the two levels do not match loses arbitration and must stop transmitting on the SDA line. For example, if one transmitter holds the SDA line to a logical one (lets it float) and a second transmitter holds it to a logical zero (pulls it low), the result is that the SDA line will be low.

The first transmitter then observes that the level of the line is different than expected and concludes that another transmitter is communicating. The first transmitter to notice this difference is the one that loses arbitration and must stop driving the SDA line. If this transmitter is also a master device, it must also stop driving the SCL line. Then it can monitor the lines for a Stop condition before trying to reissue its transmission. In the meantime, the other device that has not noticed any difference between the expected and actual levels on the SDA line continues with its original transmission. It can do so without any complications because, so far, the transmission appears exactly as expected with no other transmitter disturbing the message.

#### 20.3.2.2 Multi-Master with Slave Recovery

In the second condition there are essentially three entities on the bus: a master and a slave that reside within the same device, and a second master attempting to access the slave. If the master coupled to the slave loses the arbitration, then the slave must be able to recover and correctly respond to the second master. To accomplish this, the master which shares a device with the slave must use the $I^2C$ Firmware Controller Master mode of operation. This mode utilizes software to perform the master function, while the slave monitors the bus as a separate entity allowing it to respond to the second master.

To detect a collision, each device transmitting on the bus must check the level of the SDA data line and compare it to the level that it expects to find. The first transmitter to observe that the two levels do not match loses arbitration, and it must drop off the bus as well as stop transmitting on the SDA line if it is acting as a master. For example, if one transmitter holds the SDA line to a logical one (lets it float) and a second transmitter holds it to a logical zero (pulls it low), the result is that the SDA line will be low.

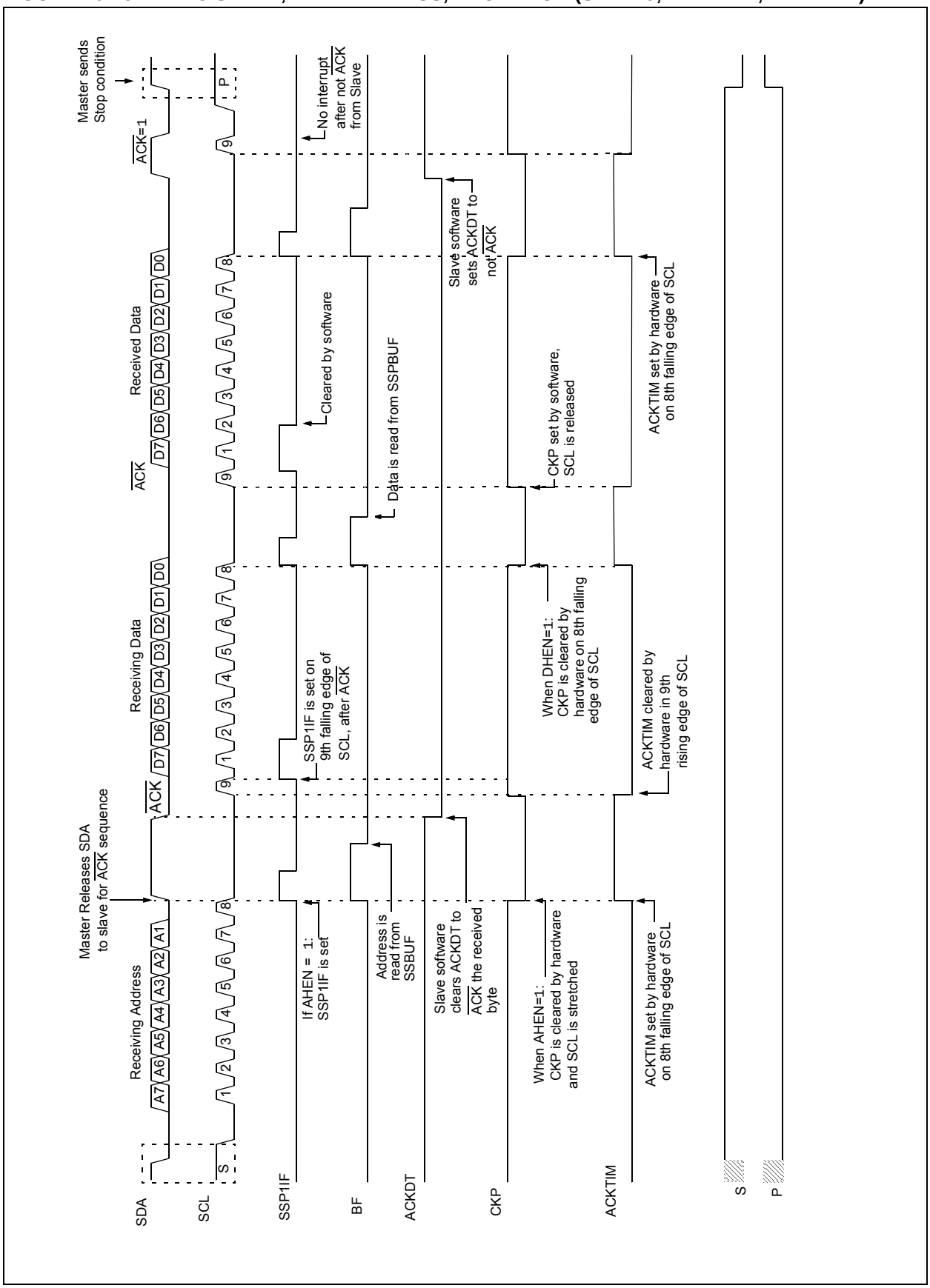
If two master devices send a message to two different slave devices at the address stage, the master sending the lower slave address always wins arbitration. When two master devices send messages to the same slave address, and addresses can sometimes refer to multiple slaves, the arbitration process must continue into the data stage.

#### 20.3.2.3 Multi-Slave Collision

When a system attempts a slave read from the general call address, there is the possibility that more than one slave devices may attempt to return data to the master. When this happens, a read collision occurs and the slaves which fail the arbitration must fall off the bus and allow the winning device to continue its data transmission. Once the winning device has completed its data transfer, the master can then initiate an additional read from the general call address to retrieve the second slave's data. In large systems this may require multiple reads by the master. Several protocols use this feature and the MSSP hardware incorporates a detection mechanism in the slave read to detect the condition and respond appropriately.

In most designs, arbitration usually occurs very rarely, but it is a necessary process for proper multi-master support or even multi-slave systems that use general call address reads.

**FIGURE 20-16: I²C SLAVE, 7-BIT ADDRESS, RECEPTION (SEN = 0, AHEN = 1, DHEN = 1)**



### 20.6.5 I²C MASTER MODE REPEATED START CONDITION TIMING

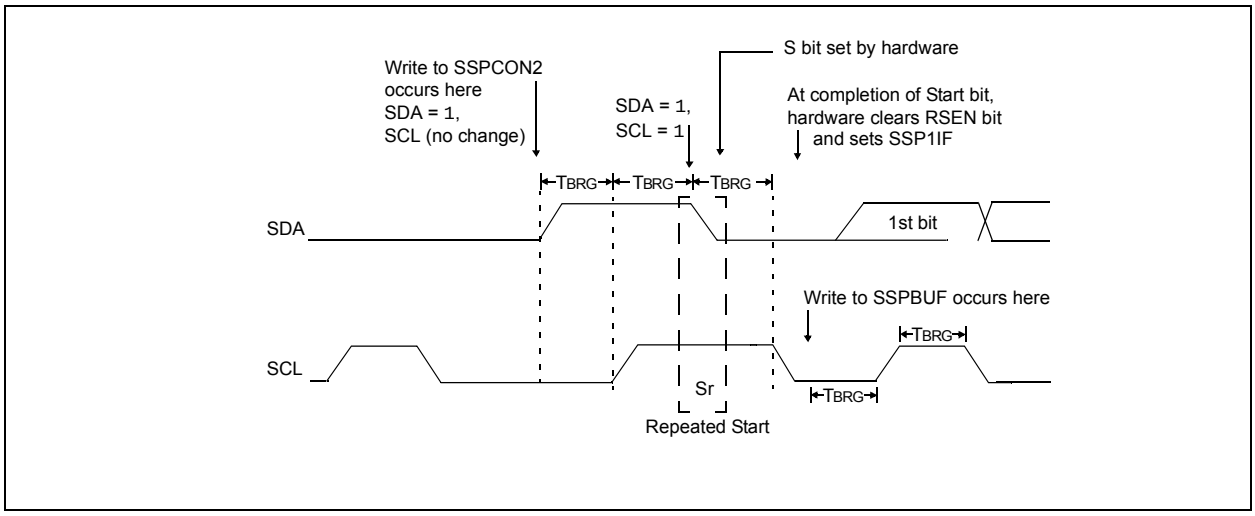
A Repeated Start condition (Figure 20-27) occurs when the RSEN bit of the SSPCON2 register is programmed high and the master state machine is no longer active. When the RSEN bit is set, the SCL pin is asserted low. When the SCL pin is sampled low, the Baud Rate Generator is loaded and begins counting. The SDA pin is released (brought high) for one Baud Rate Generator count (TBRG). When the Baud Rate Generator times out, if SDA is sampled high, the SCL pin will be deasserted (brought high). When SCL is sampled high, the Baud Rate Generator is reloaded and begins counting. SDA and SCL must be sampled high for one TBRG. This action is then followed by assertion of the SDA pin (SDA = 0) for one TBRG while SCL is high. SCL is asserted low. Following this, the RSEN bit of the SSPCON2 register will be automatically cleared and the Baud Rate Generator will not be reloaded, leaving the SDA pin held low. As soon as a Start condition is detected on the SDA and SCL pins, the S bit of the SSPSTAT register will be set. The SSP1IF bit will not be set until the Baud Rate Generator has timed out.

**Note 1:** If RSEN is programmed while any other event is in progress, it will not take effect.

**2:** A bus collision during the Repeated Start condition occurs if:

- SDA is sampled low when SCL goes from low-to-high.
- SCL goes low before SDA is asserted low. This may indicate that another master is attempting to transmit a data '1'.

**FIGURE 20-27: REPEAT START CONDITION WAVEFORM**




### 20.6.6 I²C MASTER MODE TRANSMISSION

Transmission of a data byte, a 7-bit address or the other half of a 10-bit address is accomplished by simply writing a value to the SSPBUF register. This action will set the Buffer Full flag bit, BF and allow the Baud Rate Generator to begin counting and start the next transmission. Each bit of address/data will be shifted out onto the SDA pin after the falling edge of SCL is asserted. SCL is held low for one Baud Rate Generator rollover count (TBRG). Data should be valid before SCL is released high. When the SCL pin is released high, it is held that way for TBRG. The data on the SDA pin must remain stable for that duration and some hold time after the next falling edge of SCL. After the eighth bit is shifted out (the falling edge of the eighth clock), the BF flag is cleared and the master releases SDA. This allows the slave device being addressed to respond with an $\overline{\text{ACK}}$ bit during the ninth bit time if an address match occurred, or if data was received properly. The status of $\overline{\text{ACK}}$ is written into the ACKSTAT bit on the rising edge of the ninth clock. If the master receives an Acknowledge, the Acknowledge Status bit, ACKSTAT, is cleared. If not, the bit is set. After the ninth clock, the SSP1IF bit is set and the master clock (Baud Rate Generator) is suspended until the next data byte is loaded into the SSPBUF, leaving SCL low and SDA unchanged (Figure 20-28).

After the write to the SSPBUF, each bit of the address will be shifted out on the falling edge of SCL until all seven address bits and the R/$\overline{\text{W}}$ bit are completed. On the falling edge of the eighth clock, the master will release the SDA pin, allowing the slave to respond with an Acknowledge. On the falling edge of the ninth clock, the master will sample the SDA pin to see if the address was recognized by a slave. The status of the $\overline{\text{ACK}}$ bit is loaded into the ACKSTAT Status bit of the SSPCON2 register. Following the falling edge of the ninth clock transmission of the address, the SSP1IF is set, the BF flag is cleared and the Baud Rate Generator is turned off until another write to the SSPBUF takes place, holding SCL low and allowing SDA to float.

### 21.4.4 BREAK CHARACTER SEQUENCE

The EUSART module has the capability of sending the special Break character sequences that are required by the LIN bus standard. A Break character consists of a Start bit, followed by 12 ‘0’ bits and a Stop bit.

To send a Break character, set the SENDB and TXEN bits of the TXSTA register. The Break character transmission is then initiated by a write to the TXREG. The value of data written to TXREG will be ignored and all ‘0’s will be transmitted.

The SENDB bit is automatically reset by hardware after the corresponding Stop bit is sent. This allows the user to preload the transmit FIFO with the next transmit byte following the Break character (typically, the Sync character in the LIN specification).

The TRMT bit of the TXSTA register indicates when the transmit operation is active or idle, just as it does during normal transmission. See Figure 21-9 for the timing of the Break character sequence.

#### 21.4.4.1 Break and Sync Transmit Sequence

The following sequence will start a message frame header made up of a Break, followed by an auto-baud Sync byte. This sequence is typical of a LIN bus master.

1. Configure the EUSART for the desired mode.
2. Set the TXEN and SENDB bits to enable the Break sequence.
3. Load the TXREG with a dummy character to initiate transmission (the value is ignored).
4. Write ‘55h’ to TXREG to load the Sync character into the transmit FIFO buffer.
5. After the Break has been sent, the SENDB bit is reset by hardware and the Sync character is then transmitted.

When the TXREG becomes empty, as indicated by the TXIF, the next data byte can be written to TXREG.

### 21.4.5 RECEIVING A BREAK CHARACTER

The Enhanced EUSART module can receive a Break character in two ways.

The first method to detect a Break character uses the FERR bit of the RCSTA register and the received data as indicated by RCREG. The Baud Rate Generator is assumed to have been initialized to the expected baud rate.

A Break character has been received when;

- RCIF bit is set
- FERR bit is set
- RCREG = 00h

The second method uses the Auto-Wake-up feature described in **Section 21.4.3 “Auto-Wake-up on Break”**. By enabling this feature, the EUSART will sample the next two transitions on RX/DT, cause an RCIF interrupt, and receive the next data byte followed by another interrupt.

Note that following a Break character, the user will typically want to enable the Auto-Baud Detect feature. For both methods, the user can set the ABDEN bit of the BAUDCON register before placing the EUSART in Sleep mode.

**FIGURE 21-9: SEND BREAK CHARACTER SEQUENCE**

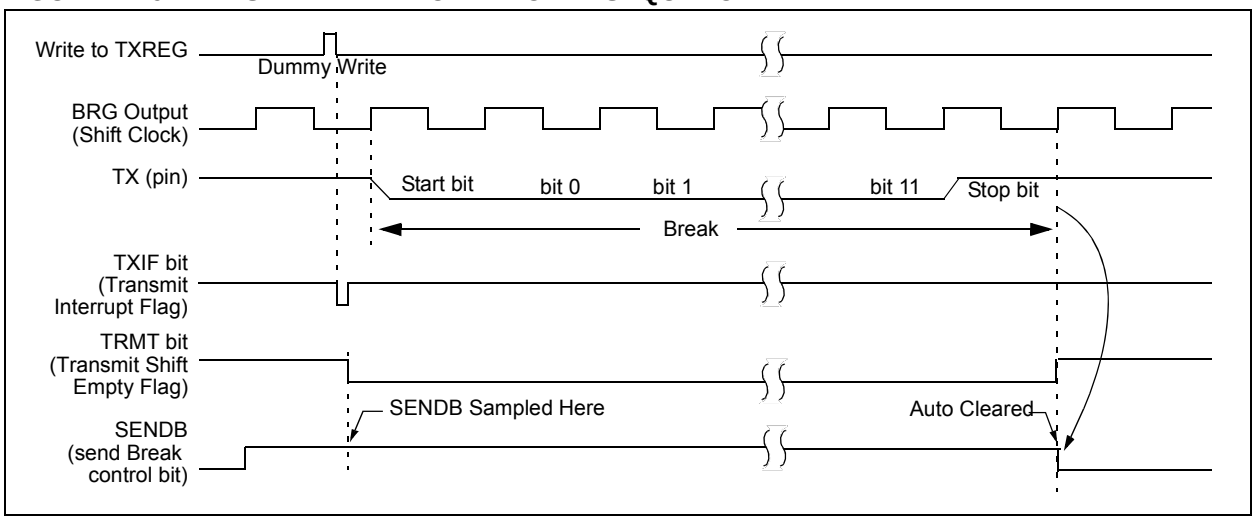


**TABLE 24-3: ENHANCED MID-RANGE INSTRUCTION SET (CONTINUED)**

| Mnemonic, Operands | | Description | Cycles | 14-Bit Opcode MSb | | | LSb | Status Affected | Notes |
|---|---|---|---|---|---|---|---|---|---|
| CONTROL OPERATIONS | | | | | | | | | |
| BRA | k | Relative Branch | 2 | 11 | 001k | kkkk | kkkk | | |
| BRW | – | Relative Branch with W | 2 | 00 | 0000 | 0000 | 1011 | | |
| CALL | k | Call Subroutine | 2 | 10 | 0kkk | kkkk | kkkk | | |
| CALLW | – | Call Subroutine with W | 2 | 00 | 0000 | 0000 | 1010 | | |
| GOTO | k | Go to address | 2 | 10 | 1kkk | kkkk | kkkk | | |
| RETFIE | – | Return from interrupt | 2 | 00 | 0000 | 0000 | 1001 | | |
| RETLW | k | Return with literal in W | 2 | 11 | 0100 | kkkk | kkkk | | |
| RETURN | – | Return from Subroutine | 2 | 00 | 0000 | 0000 | 1000 | | |
| INHERENT OPERATIONS | | | | | | | | | |
| CLRWDT | – | Clear Watchdog Timer | 1 | 00 | 0000 | 0110 | 0100 | $\overline{TO}$, $\overline{PD}$ | |
| NOP | – | No Operation | 1 | 00 | 0000 | 0000 | 0000 | | |
| OPTION | – | Load OPTION_REG register with W | 1 | 00 | 0000 | 0110 | 0010 | | |
| RESET | – | Software device Reset | 1 | 00 | 0000 | 0000 | 0001 | | |
| SLEEP | – | Go into Standby mode | 1 | 00 | 0000 | 0110 | 0011 | $\overline{TO}$, $\overline{PD}$ | |
| TRIS | f | Load TRIS register with W | 1 | 00 | 0000 | 0110 | 0fff | | |
| C-COMPILER OPTIMIZED | | | | | | | | | |
| ADDFSR | n, k | Add Literal k to FSRn | 1 | 11 | 0001 | 0nkk | kkkk | | |
| MOVIW | n mm | Move Indirect FSRn to W with pre/post inc/dec modifier, mm | 1 | 00 | 0000 | 0001 | 0nmm kkkk | Z | 2, 3 |
| | k[n] | Move INDFn to W, Indexed Indirect. | 1 | 11 | 1111 | 0nkk | 1nmm | Z | 2 |
| MOVWI | n mm | Move W to Indirect FSRn with pre/post inc/dec modifier, mm | 1 | 00 | 0000 | 0001 | kkkk | | 2, 3 |
| | k[n] | Move W to INDFn, Indexed Indirect. | 1 | 11 | 1111 | 1nkk | | | 2 |

**Note 1:** If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

**2:** If this instruction addresses an INDF register and the MSb of the corresponding FSR is set, this instruction will require one additional instruction cycle.

**3:** See Table in the MOVIW and MOVWI instruction descriptions.

**TABLE 25-7: CLOCK OSCILLATOR TIMING REQUIREMENTS**

| **Standard Operating Conditions (unless otherwise stated)**<br>Operating temperature -40°C ≤ TA ≤ +125°C | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Param. No.** | **Sym.** | **Characteristic** | **Min.** | **Typ.†** | **Max.** | **Units** | **Conditions** |
| OS01 | FOSC | External CLKIN Frequency(1) | DC | — | 0.5 | MHz | EC Oscillator mode (low) |
| | | | DC | — | 4 | MHz | EC Oscillator mode (medium) |
| | | | DC | — | 20 | MHz | EC Oscillator mode (high) |
| OS02 | TOSC | External CLKIN Period(1) | 50 | — | ∞ | ns | EC mode |
| OS03 | TCY | Instruction Cycle Time(1) | 200 | — | DC | ns | TCY = FOSC/4 |

* These parameters are characterized but not tested.

† 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:** Instruction cycle period (TCY) equals four times the input oscillator time base period. All specified values are based on characterization data for that particular oscillator type under standard operating conditions with the device executing code. Exceeding these specified limits may result in an unstable oscillator operation and/or higher than expected current consumption. All devices are tested to operate at "min" values with an external clock applied to CLKIN pin. When an external clock input is used, the "max" cycle time limit is "DC" (no clock) for all devices.

**TABLE 25-8: OSCILLATOR PARAMETERS**

| **Standard Operating Conditions (unless otherwise stated)**<br>Operating Temperature -40°C ≤ TA ≤ +125°C | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Param. No.** | **Sym.** | **Characteristic** | **Min.** | **Typ.†** | **Max.** | **Units** | **Conditions** |
| OS08 | HFOSC | Internal Calibrated HFINTOSC Frequency(1) | — | 16.0 | — | MHz | 0°C ≤ TA ≤ +85°C |
| OS08A | HFTOL | Frequency Tolerance | — | ±3 | — | % | 25°C, 16 MHz |
| | | | — | ±6 | — | % | 0°C ≤ TA ≤ +85°C, 16 MHz |
| OS09 | LFOSC | Internal LFINTOSC Frequency | — | 31 | — | kHz | -40°C ≤ TA ≤ +125°C |
| OS10* | TWARM | HFINTOSC<br>Wake-up from Sleep Start-up Time<br>LFINTOSC<br>Wake-up from Sleep Start-up Time | — | 5<br><br>0.5 | 8 | µs<br><br>ms | — |

* These parameters are characterized but not tested.

† 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:** To ensure these oscillator frequency tolerances, VDD and VSS must be capacitively decoupled as close to the device as possible. 0.1 µF and 0.01 µF values in parallel are recommended.

**Note:** Unless otherwise noted $C_{IN}$ = 0.1 µF and $T_A$ = 25°C.

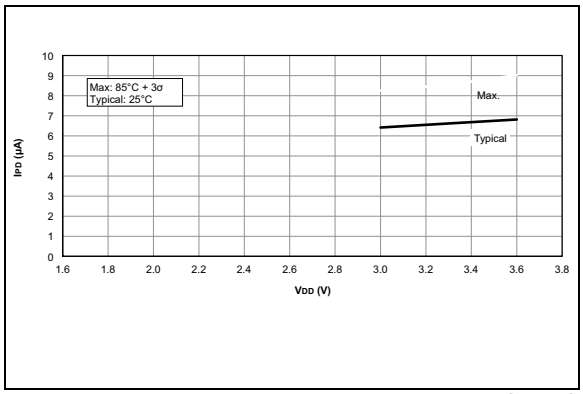



***FIGURE 26-13:*** *IPD, Brown-out Reset (BOR)*

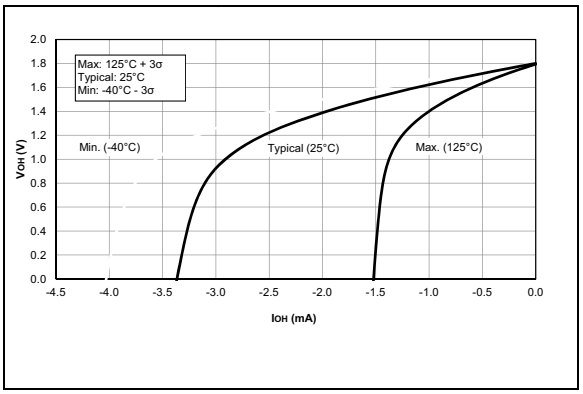



***FIGURE 26-16:*** *VOH vs. IOH, over Temperature, VDD=1.8V*

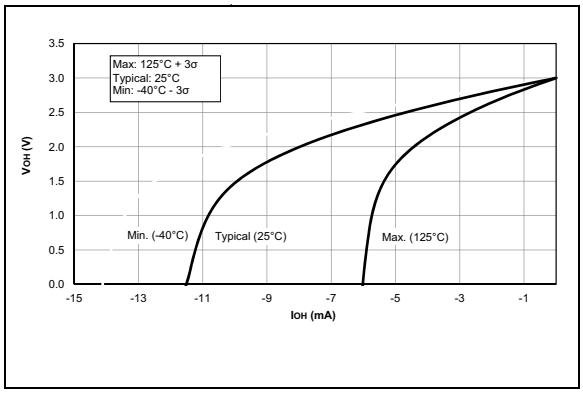



***FIGURE 26-14:*** *VOH vs. IOH, over Temperature, VDD=3.0V*

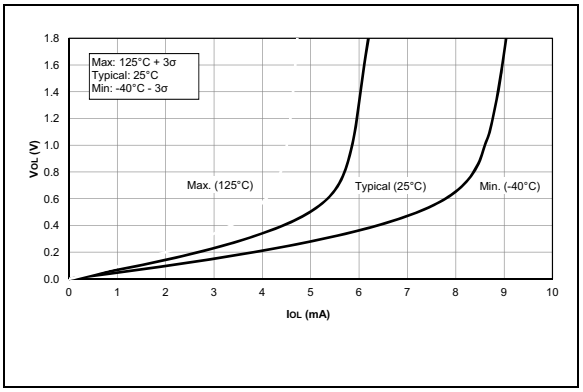



***FIGURE 26-17:*** *VOL vs. IOL, over Temperature, VDD=1.8V*

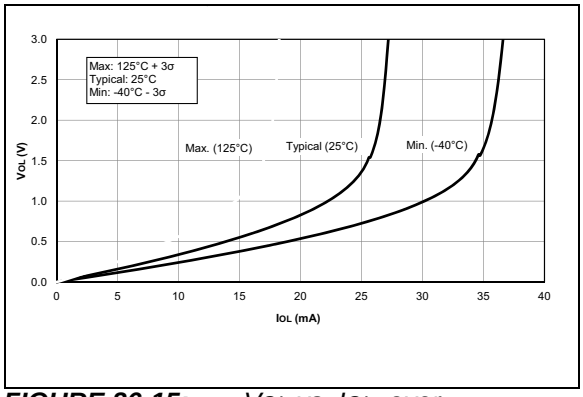



***FIGURE 26-15:*** *VOL vs. IOL, over Temperature, VDD=3.0V*

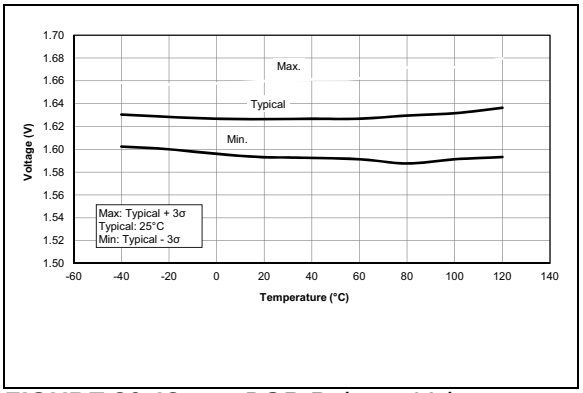
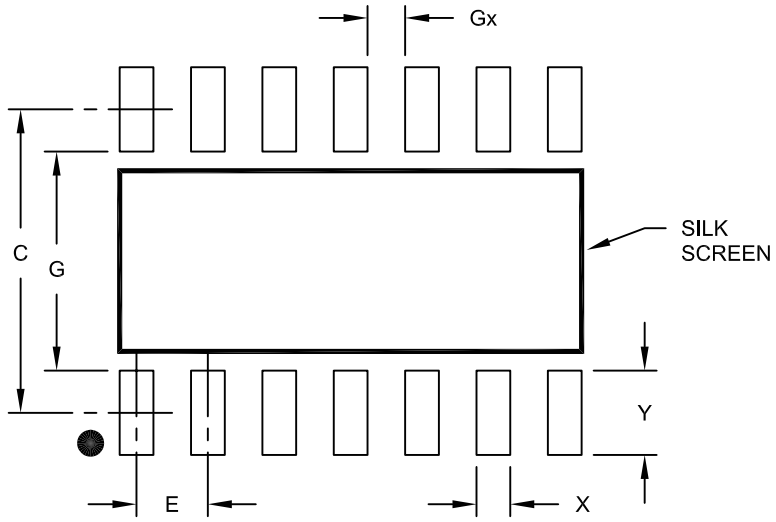



***FIGURE 26-18:*** *POR Release Voltage*

14-Lead Plastic Small Outline (SL) - Narrow, 3.90 mm Body [SOIC]

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




RECOMMENDED LAND PATTERN

| Units | | MILLIMETERS | | |
|---|---|---|---|---|
| Dimension Limits | | MIN | NOM | MAX |
| Contact Pitch | E | | 1.27 BSC | |
| Contact Pad Spacing | C | | 5.40 | |
| Contact Pad Width | X | | | 0.60 |
| Contact Pad Length | Y | | | 1.50 |
| Distance Between Pads | Gx | 0.67 | | |
| Distance Between Pads | G | 3.90 | | |

Notes:

1. Dimensioning and tolerancing per ASME Y14.5M

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

Microchip Technology Drawing No. C04-2065A