



Welcome to [E-XFL.COM](https://www.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.

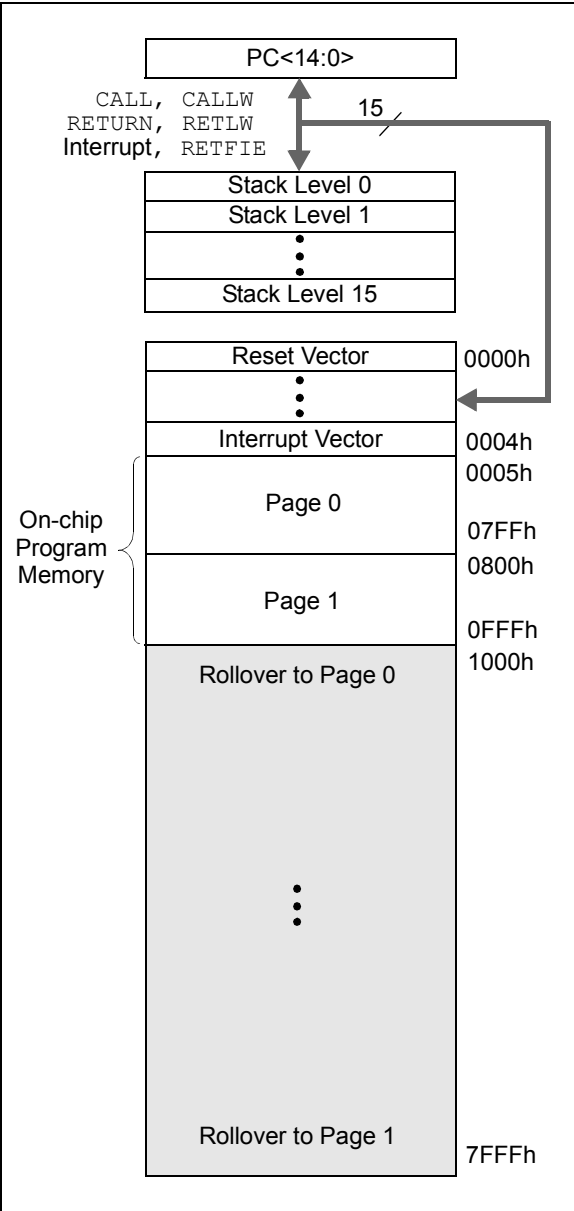
Applications of "[Embedded - Microcontrollers](#)"

Details

Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	32MHz
Connectivity	I ² C, LINbus, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LCD, POR, PWM, WDT
Number of I/O	25
Program Memory Size	7KB (4K x 14)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	256 x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 5.5V
Data Converters	A/D 11x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	28-VQFN Exposed Pad
Supplier Device Package	28-QFN (6x6)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic16f1933t-i-ml

PIC16(L)F1933

FIGURE 3-1: PROGRAM MEMORY MAP AND STACK FOR 4 KW PARTS



3.1.1 READING PROGRAM MEMORY AS DATA

There are two methods of accessing constants in program memory. The first method is to use tables of RETLW instructions. The second method is to set an FSR to point to the program memory.

3.1.1.1 RETLW Instruction

The RETLW instruction can be used to provide access to tables of constants. The recommended way to create such a table is shown in [Example 3-1](#).

EXAMPLE 3-1: RETLW INSTRUCTION

```
constants
    BRW                ;Add Index in W to
                        ;program counter to
                        ;select data

    RETLW DATA0        ;Index0 data
    RETLW DATA1        ;Index1 data
    RETLW DATA2
    RETLW DATA3

my_function
    ;... LOTS OF CODE...
    MOVLW DATA_INDEX
    call constants
    ;... THE CONSTANT IS IN W
```

The BRW instruction makes this type of table very simple to implement. If your code must remain portable with previous generations of microcontrollers, then the BRW instruction is not available so the older table read method must be used.

3.5 Stack

All devices have a 16-level x 15-bit wide hardware stack (refer to Figures 3-4 through 3-7). The stack space is not part of either program or data space. The PC is PUSHed onto the stack when `CALL` or `CALLW` instructions are executed or an interrupt causes a branch. The stack is POPed in the event of a `RETURN`, `RETLW` or a `RETFIE` instruction execution. PCLATH is not affected by a PUSH or POP operation.

The stack operates as a circular buffer if the STVREN bit is programmed to '0' (Configuration Word 2). This means that after the stack has been PUSHed sixteen times, the seventeenth PUSH overwrites the value that was stored from the first PUSH. The eighteenth PUSH overwrites the second PUSH (and so on). The STKOVF and STKUNF flag bits will be set on an Overflow/Underflow, regardless of whether the Reset is enabled.

Note 1: There are no instructions/mnemonics called PUSH or POP. These are actions that occur from the execution of the `CALL`, `CALLW`, `RETURN`, `RETLW` and `RETFIE` instructions or the vectoring to an interrupt address.

3.5.1 ACCESSING THE STACK

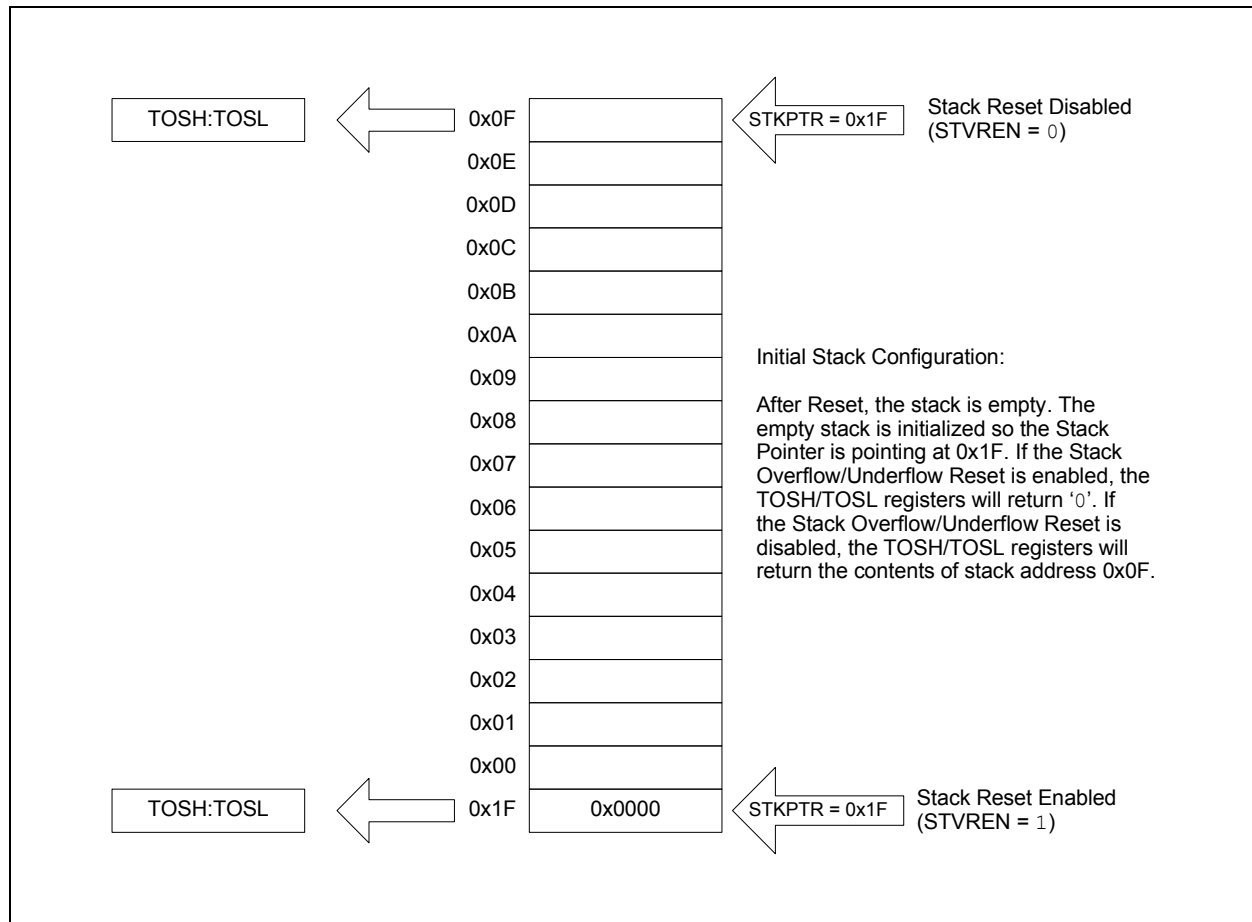
The stack is available through the TOSH, TOSL and STKPTR registers. STKPTR is the current value of the Stack Pointer. TOSH:TOSL register pair points to the TOP of the stack. Both registers are read/writable. TOS is split into TOSH and TOSL due to the 15-bit size of the PC. To access the stack, adjust the value of STKPTR, which will position TOSH:TOSL, then read/write to TOSH:TOSL. STKPTR is five bits to allow detection of overflow and underflow.

Note: Care should be taken when modifying the STKPTR while interrupts are enabled.

During normal program operation, `CALL`, `CALLW` and Interrupts will increment STKPTR while `RETLW`, `RETURN`, and `RETFIE` will decrement STKPTR. At any time, STKPTR can be inspected to see how much stack is left. The STKPTR always points at the currently used place on the stack. Therefore, a `CALL` or `CALLW` will increment the STKPTR and then write the PC, and a return will unload the PC and then decrement the STKPTR.

Reference Figure 3-4 through Figure 3-7 for examples of accessing the stack.

FIGURE 3-4: ACCESSING THE STACK EXAMPLE 1



PIC16(L)F1933

REGISTER 4-2: CONFIG2: CONFIGURATION WORD 2

R/P-1	R/P-1	U-1	R/P-1	R/P-1	R/P-1
LVP ⁽¹⁾	DEBUG ⁽³⁾	—	BORV	STVREN	PLLEN
bit 13					bit 8

U-1	U-1	R/P-1	R/P-1	U-1	U-1	R/P-1	R/P-1
—	—	VCAPEN<1:0> ⁽²⁾		—	—	WRT<1:0>	
bit 7							bit 0

Legend:

R = Readable bit

P = Programmable bit

U = Unimplemented bit, read as '1'

'0' = Bit is cleared

'1' = Bit is set

-n = Value when blank or after Bulk Erase

- bit 13 **LVP:** Low-Voltage Programming Enable bit⁽¹⁾
 1 = Low-voltage programming enabled
 0 = High-voltage on MCLR must be used for programming
- bit 12 **DEBUG:** In-Circuit Debugger Mode bit⁽³⁾
 1 = In-Circuit Debugger disabled, ICSPCLK and ICSPDAT are general purpose I/O pins
 0 = In-Circuit Debugger enabled, ICSPCLK and ICSPDAT are dedicated to the debugger
- bit 11 **Unimplemented:** Read as '1'
- bit 10 **BORV:** Brown-out Reset Voltage Selection bit⁽⁴⁾
 1 = Brown-out Reset voltage (*Vbor*), low trip point selected.
 0 = Brown-out Reset voltage (*Vbor*), high trip point selected.
- bit 9 **STVREN:** Stack Overflow/Underflow Reset Enable bit
 1 = Stack Overflow or Underflow will cause a Reset
 0 = Stack Overflow or Underflow will not cause a Reset
- bit 8 **PLLEN:** PLL Enable bit
 1 = 4xPLL enabled
 0 = 4xPLL disabled
- bit 7-5 **Unimplemented:** Read as '1'
- bit 4 **VCAPEN:** Voltage Regulator Capacitor Enable bits⁽²⁾
 00 = VCAP functionality is enabled on RA0
 01 = VCAP functionality is enabled on RA5
 10 = VCAP functionality is enabled on RA6
 11 = No capacitor on VCAP pin
- bit 3-2 **Unimplemented:** Read as '1'
- bit 1-0 **WRT<1:0>:** Flash Memory Self-Write Protection bits
 00 = VCAP functionality is enabled on RA0
 01 = VCAP functionality is enabled on RA5
 10 = VCAP functionality is enabled on RA6
 11 = No capacitor on VCAP pin

Note 1: The LVP bit cannot be programmed to '0' when Programming mode is entered via LVP.

Note 2: Reads as '11' on PIC16LF193X only.

Note 3: The **DEBUG** bit in Configuration Words is managed automatically by device development tools including debuggers and programmers. For normal device operation, this bit should be maintained as a '1'.

Note 4: See *Vbor* parameter for specific trip point voltages.

6.12 Power Control (PCON) Register

The Power Control (PCON) register contains flag bits to differentiate between a:

- Power-on Reset ($\overline{\text{POR}}$)
- Brown-out Reset ($\overline{\text{BOR}}$)
- Reset Instruction Reset ($\overline{\text{RI}}$)
- Stack Overflow Reset (STKOVF)
- Stack Underflow Reset (STKUNF)
- MCLR Reset ($\overline{\text{RMCLR}}$)

The PCON register bits are shown in [Register 6-2](#).

6.13 Register Definitions: Power Control

REGISTER 6-2: PCON: POWER CONTROL REGISTER

R/W/HS-0/q	R/W/HS-0/q	U-0	U-0	R/W/HC-1/q	R/W/HC-1/q	R/W/HC-q/u	R/W/HC-q/u
STKOVF	STKUNF	—	—	$\overline{\text{RMCLR}}$	$\overline{\text{RI}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

Legend:

HC = Bit is cleared by hardware

HS = Bit is set by hardware

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

u = Bit is unchanged

x = Bit is unknown

-m/n = Value at POR and BOR/Value at all other Resets

'1' = Bit is set

'0' = Bit is cleared

q = Value depends on condition

bit 7 **STKOVF:** Stack Overflow Flag bit

1 = A Stack Overflow occurred

0 = A Stack Overflow has not occurred or set to '0' by firmware

bit 6 **STKUNF:** Stack Underflow Flag bit

1 = A Stack Underflow occurred

0 = A Stack Underflow has not occurred or set to '0' by firmware

bit 5-4 **Unimplemented:** Read as '0'

bit 3 **$\overline{\text{RMCLR}}$:** MCLR Reset Flag bit

1 = A $\overline{\text{MCLR}}$ Reset has not occurred or set to '1' by firmware

0 = A MCLR Reset has occurred (set to '0' in hardware when a $\overline{\text{MCLR}}$ Reset occurs)

bit 2 **$\overline{\text{RI}}$:** RESET Instruction Flag bit

1 = A RESET instruction has not been executed or set to '1' by firmware

0 = A RESET instruction has been executed (set to '0' in hardware upon executing a RESET instruction)

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status bit

1 = No Power-on Reset occurred

0 = A Power-on Reset occurred (must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit

1 = No Brown-out Reset occurred

0 = A Brown-out Reset occurred (must be set in software after a Power-on Reset or Brown-out Reset occurs)

PIC16(L)F1933

TABLE 6-5: SUMMARY OF REGISTERS ASSOCIATED WITH RESETS

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on Page
BORCON	SBOREN	—	—	—	—	—	—	BORRDY	71
PCON	STKOVF	STKUNF	—	—	$\overline{\text{RMCLR}}$	$\overline{\text{RI}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$	75
STATUS	—	—	—	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C	18
WDTCON	—	—	WDTPS<4:0>					SWDTEN	97

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

Note 1: Other (non Power-up) Resets include $\overline{\text{MCLR}}$ Reset and Watchdog Timer Reset during normal operation.

7.3 Interrupts During Sleep

Some interrupts can be used to wake from Sleep. To wake from Sleep, the peripheral must be able to operate without the system clock. The interrupt source must have the appropriate Interrupt Enable bit(s) set prior to entering Sleep.

On waking from Sleep, if the GIE bit is also set, the processor will branch to the interrupt vector. Otherwise, the processor will continue executing instructions after the `SLEEP` instruction. The instruction directly after the `SLEEP` instruction will always be executed before branching to the ISR. Refer to the [Section 9.0 “Power-Down Mode \(Sleep\)”](#) for more details.

7.4 INT Pin

The INT pin can be used to generate an asynchronous edge-triggered interrupt. This interrupt is enabled by setting the INTE bit of the INTCON register. The INTEDG bit of the OPTION_REG register determines on which edge the interrupt will occur. When the INTEDG bit is set, the rising edge will cause the interrupt. When the INTEDG bit is clear, the falling edge will cause the interrupt. The INTF bit of the INTCON register will be set when a valid edge appears on the INT pin. If the GIE and INTE bits are also set, the processor will redirect program execution to the interrupt vector.

7.5 Automatic Context Saving

Upon entering an interrupt, the return PC address is saved on the stack. Additionally, the following registers are automatically saved in the Shadow registers:

- W register
- STATUS register (except for $\overline{\text{TO}}$ and $\overline{\text{PD}}$)
- BSR register
- FSR registers
- PCLATH register

Upon exiting the Interrupt Service Routine (ISR), these registers are automatically restored. Any modifications to these registers during the ISR will be lost. If modifications to any of these registers are desired, the corresponding Shadow register should be modified and the value will be restored when exiting the ISR. The Shadow registers are available in Bank 31 and are readable and writable. Depending on the user's application, other registers may also need to be saved.

EXAMPLE 11-5: WRITING TO FLASH PROGRAM MEMORY

```

; This write routine assumes the following:
; 1. The 16 bytes of data are loaded, starting at the address in DATA_ADDR
; 2. Each word of data to be written is made up of two adjacent bytes in DATA_ADDR,
;    stored in little endian format
; 3. A valid starting address (the least significant bits = 000) is loaded in ADDRH:ADDRL
; 4. ADDRH and ADDRL are located in shared data memory 0x70 - 0x7F (common RAM)
;
    BCF      INTCON,GIE      ; Disable ints so required sequences will execute properly
    BANKSEL  EEADRH          ; Bank 3
    MOVF     ADDRH,W         ; Load initial address
    MOVWF    EEADRH          ;
    MOVF     ADDRL,W         ;
    MOVWF    EEADRL          ;
    MOVLW    LOW DATA_ADDR  ; Load initial data address
    MOVWF    FSR0L           ;
    MOVLW    HIGH DATA_ADDR ; Load initial data address
    MOVWF    FSR0H           ;
    BSF      EECON1,EEPGD     ; Point to program memory
    BCF      EECON1,CFGSR     ; Not configuration space
    BSF      EECON1,WREN      ; Enable writes
    BSF      EECON1,LWLO     ; Only Load Write Latches

LOOP
    MOVIW    FSR0++          ; Load first data byte into lower
    MOVWF    EEDATL          ;
    MOVIW    FSR0++          ; Load second data byte into upper
    MOVWF    EEDATH          ;

    MOVF     EEADRL,W         ; Check if lower bits of address are '000'
    XORLW    0x07            ; Check if we're on the last of 8 addresses
    ANDLW    0x07            ;
    BTFSC    STATUS,Z         ; Exit if last of eight words,
    GOTO     START_WRITE     ;

    Required Sequence
    MOVLW    55h              ; Start of required write sequence:
    MOVWF    EECON2           ; Write 55h
    MOVLW    0AAh            ;
    MOVWF    EECON2           ; Write AAh
    BSF      EECON1,WR        ; Set WR bit to begin write
    NOP      ; Any instructions here are ignored as processor
    NOP      ; halts to begin write sequence
    NOP      ; Processor will stop here and wait for write to complete.

    ; After write processor continues with 3rd instruction.

    INCF     EEADRL,F         ; Still loading latches Increment address
    GOTO     LOOP            ; Write next latches

START_WRITE
    BCF      EECON1,LWLO     ; No more loading latches - Actually start Flash program
    ; memory write

    Required Sequence
    MOVLW    55h              ; Start of required write sequence:
    MOVWF    EECON2           ; Write 55h
    MOVLW    0AAh            ;
    MOVWF    EECON2           ; Write AAh
    BSF      EECON1,WR        ; Set WR bit to begin write
    NOP      ; Any instructions here are ignored as processor
    NOP      ; halts to begin write sequence
    NOP      ; Processor will stop here and wait for write complete.

    ; after write processor continues with 3rd instruction

    BCF      EECON1,WREN      ; Disable writes
    BSF      INTCON,GIE      ; Enable interrupts

```


PIC16(L)F1933

23.3.6 PWM RESOLUTION

The resolution determines the number of available duty cycles for a given period. For example, a 10-bit resolution will result in 1024 discrete duty cycles, whereas an 8-bit resolution will result in 256 discrete duty cycles.

The maximum PWM resolution is 10 bits when PRx is 255. The resolution is a function of the PRx register value as shown by [Equation 23-4](#).

EQUATION 23-4: PWM RESOLUTION

$$Resolution = \frac{\log[4(PR_x + 1)]}{\log(2)} \text{ bits}$$

Note: If the pulse width value is greater than the period the assigned PWM pin(s) will remain unchanged.

TABLE 23-5: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (Fosc = 32 MHz)

PWM Frequency	1.95 kHz	7.81 kHz	31.25 kHz	125 kHz	250 kHz	333.3 kHz
Timer Prescale	16	4	1	1	1	1
PRx Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

TABLE 23-6: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (Fosc = 20 MHz)

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescale	16	4	1	1	1	1
PRx Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

TABLE 23-7: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (Fosc = 8 MHz)

PWM Frequency	1.22 kHz	4.90 kHz	19.61 kHz	76.92 kHz	153.85 kHz	200.0 kHz
Timer Prescale	16	4	1	1	1	1
PRx Value	0x65	0x65	0x65	0x19	0x0C	0x09
Maximum Resolution (bits)	8	8	8	6	5	5

24.2 SPI Mode Overview

The Serial Peripheral Interface (SPI) bus is a synchronous serial data communication bus that operates in Full-Duplex mode. Devices communicate in a master/slave environment where the master device initiates the communication. A slave device is controlled through a Chip Select known as Slave Select.

The SPI bus specifies four signal connections:

- Serial Clock (SCK)
- Serial Data Out (SDO)
- Serial Data In (SDI)
- Slave Select (\overline{SS})

Figure 24-1 shows the block diagram of the MSSP module when operating in SPI Mode.

The SPI bus operates with a single master device and one or more slave devices. When multiple slave devices are used, an independent Slave Select connection is required from the master device to each slave device.

Figure 24-4 shows a typical connection between a master device and multiple slave devices.

The master selects only one slave at a time. Most slave devices have tri-state outputs so their output signal appears disconnected from the bus when they are not selected.

Transmissions involve two shift registers, eight bits in size, one in the master and one in the slave. With either the master or the slave device, data is always shifted out one bit at a time, with the Most Significant bit (MSb) shifted out first. At the same time, a new Least Significant bit (LSb) is shifted into the same register.

Figure 24-5 shows a typical connection between two processors configured as master and slave devices.

Data is shifted out of both shift registers on the programmed clock edge and latched on the opposite edge of the clock.

The master device transmits information out on its SDO output pin which is connected to, and received by, the slave's SDI input pin. The slave device transmits information out on its SDO output pin, which is connected to, and received by, the master's SDI input pin.

To begin communication, the master device first sends out the clock signal. Both the master and the slave devices should be configured for the same clock polarity.

The master device starts a transmission by sending out the MSb from its shift register. The slave device reads this bit from that same line and saves it into the LSb position of its shift register.

During each SPI clock cycle, a full-duplex data transmission occurs. This means that while the master device is sending out the MSb from its shift register (on its SDO pin) and the slave device is reading this bit and saving it as the LSb of its shift register, that the slave device is also sending out the MSb from its shift register (on its SDO pin) and the master device is reading this bit and saving it as the LSb of its shift register.

After eight bits have been shifted out, the master and slave have exchanged register values.

If there is more data to exchange, the shift registers are loaded with new data and the process repeats itself.

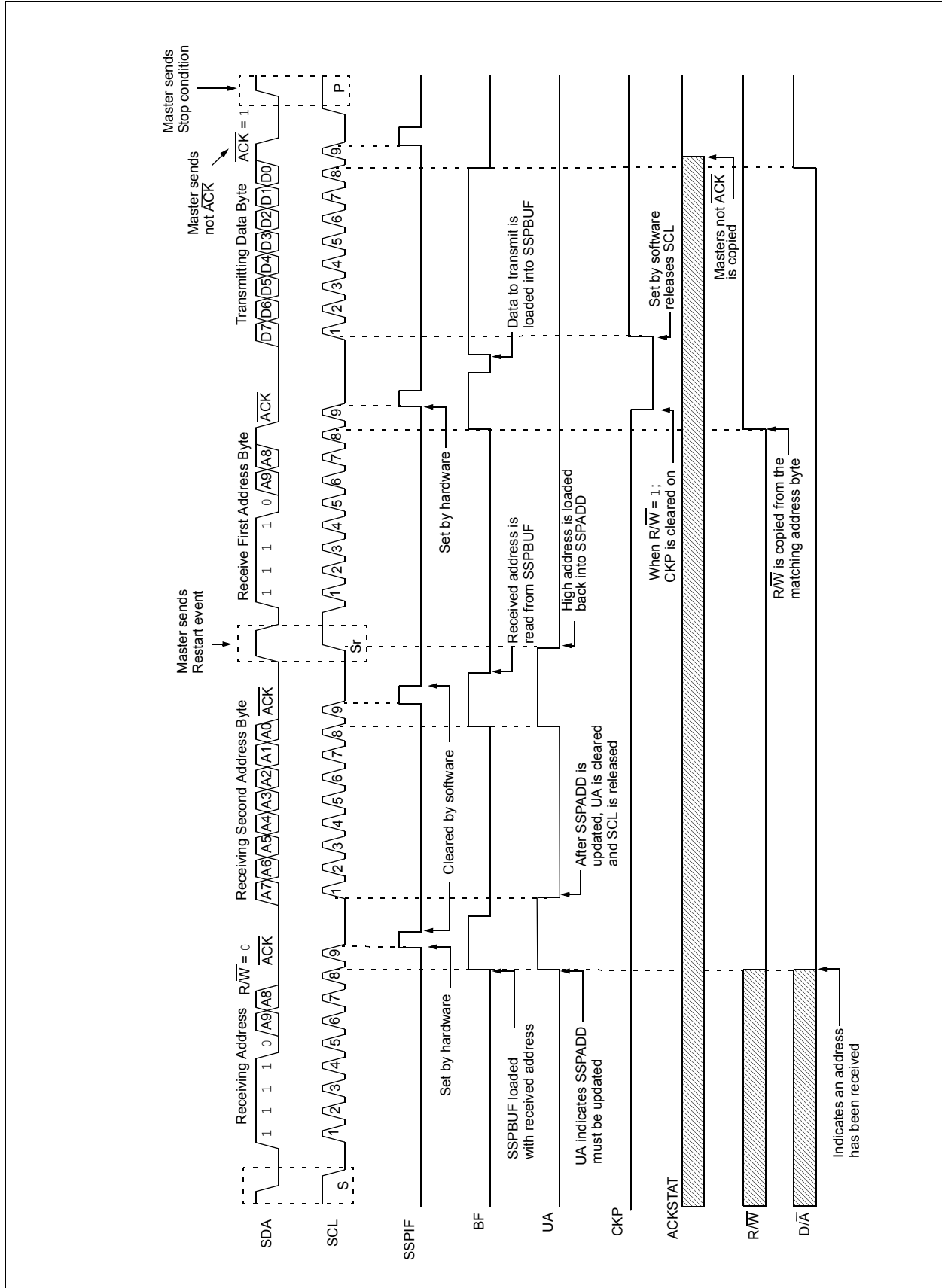
Whether the data is meaningful or not (dummy data), depends on the application software. This leads to three scenarios for data transmission:

- Master sends useful data and slave sends dummy data.
- Master sends useful data and slave sends useful data.
- Master sends dummy data and slave sends useful data.

Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops sending the clock signal and it deselects the slave.

Every slave device connected to the bus that has not been selected through its slave select line must disregard the clock and transmission signals and must not transmit out any data of its own.

FIGURE 24-22: I²C SLAVE, 10-BIT ADDRESS, TRANSMISSION (SEN = 0, AHEN = 0, DHEN = 0)



24.6.4 I²C MASTER MODE START CONDITION TIMING

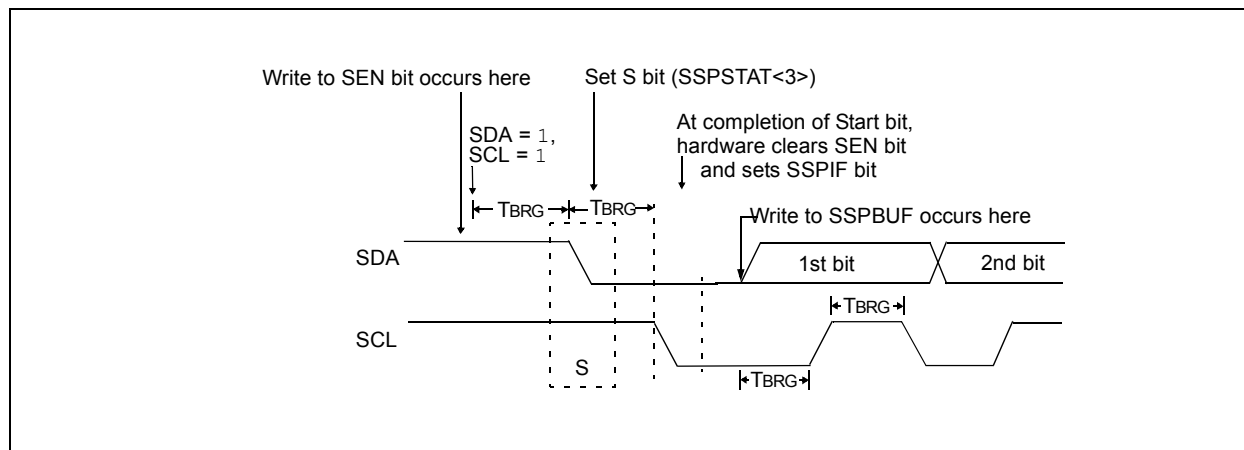
To initiate a Start condition (Figure 24-26), the user sets the Start Enable bit, SEN bit of the SSPCON2 register. If the SDA and SCL pins are sampled high, the Baud Rate Generator is reloaded with the contents of SSPADD<7:0> and starts its count. If SCL and SDA are both sampled high when the Baud Rate Generator times out (TBRG), the SDA pin is driven low. The action of the SDA being driven low while SCL is high is the Start condition and causes the S bit of the SSPSTAT1 register to be set. Following this, the Baud Rate Generator is reloaded with the contents of SSPADD<7:0> and resumes its count. When the Baud Rate Generator times out (TBRG), the SEN bit of the SSPCON2 register will be automatically cleared by

hardware; the Baud Rate Generator is suspended, leaving the SDA line held low and the Start condition is complete.

Note 1: If at the beginning of the Start condition, the SDA and SCL pins are already sampled low, or if during the Start condition, the SCL line is sampled low before the SDA line is driven low, a bus collision occurs, the Bus Collision Interrupt Flag, BCLIF, is set, the Start condition is aborted and the I²C module is reset into its Idle state.

2: The Philips I²C specification states that a bus collision cannot occur on a Start.

FIGURE 24-26: FIRST START BIT TIMING



25.4 EUSART Baud Rate Generator (BRG)

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

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

Table 25-3 contains the formulas for determining the baud rate. Example 25-1 provides a sample calculation for determining the baud rate and baud rate error.

Typical baud rates and error values for various asynchronous modes have been computed for your convenience and are shown in Table 25-3. It may be advantageous to use the high baud rate (BRGH = 1), or the 16-bit BRG (BRG16 = 1) to reduce the baud rate error. The 16-bit BRG mode is used to achieve slow baud rates for fast oscillator frequencies.

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

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

EXAMPLE 25-1: CALCULATING BAUD RATE ERROR

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

$$\text{Desired Baud Rate} = \frac{F_{OSC}}{64([SPBRGH:SPBRGL] + 1)}$$

Solving for SPBRGH:SPBRGL:

$$X = \frac{\frac{F_{OSC}}{\text{Desired Baud Rate}}}{64} - 1$$

$$= \frac{\frac{16000000}{9600}}{64} - 1$$

$$= [25.042] = 25$$

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

$$= 9615$$

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

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

27.0 LIQUID CRYSTAL DISPLAY (LCD) DRIVER MODULE

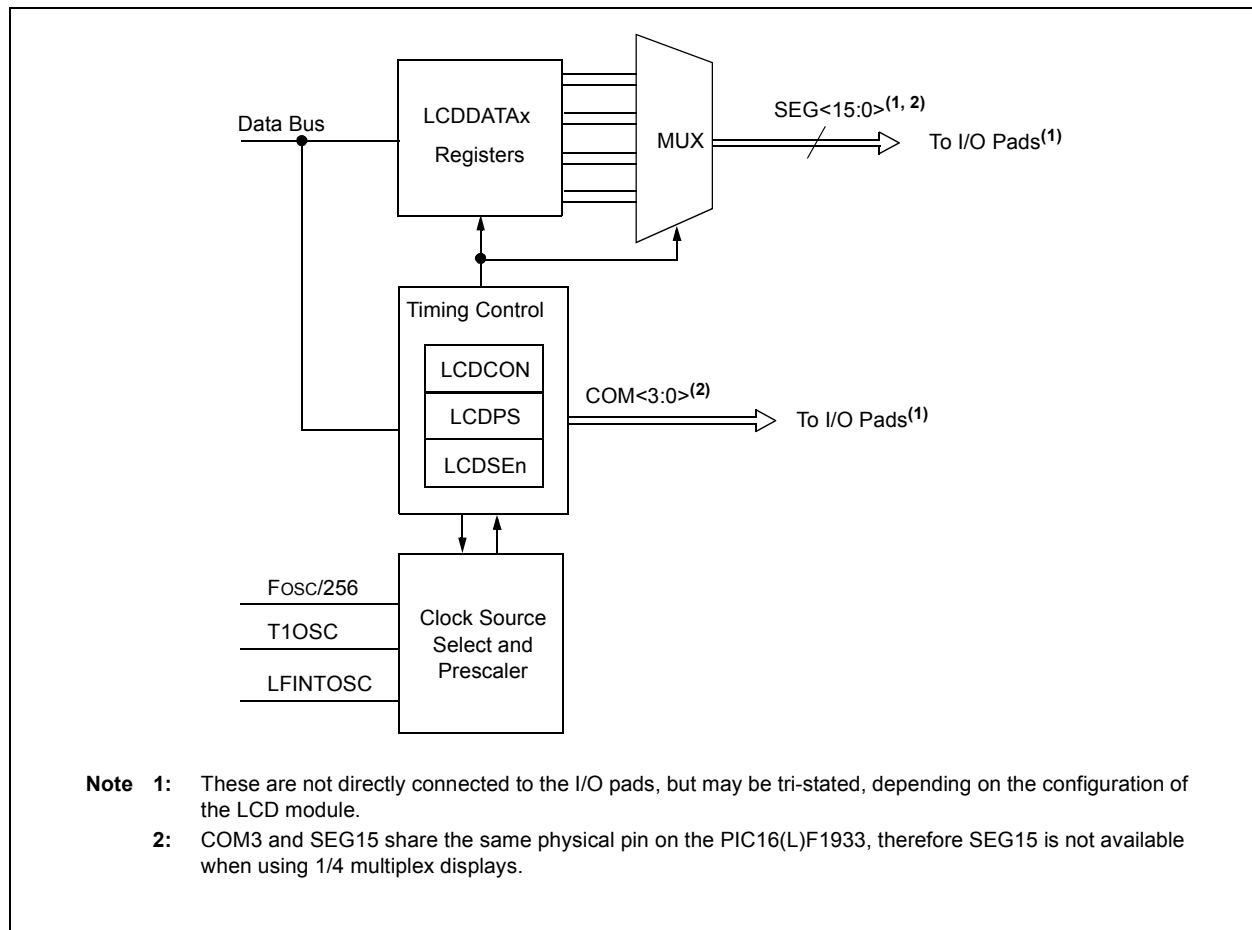
The Liquid Crystal Display (LCD) driver module generates the timing control to drive a static or multiplexed LCD panel. In the PIC16(L)F1933 device, the module drives the panels of up to four commons and up to 24 segments. The LCD module also provides control of the LCD pixel data.

The LCD driver module supports:

- Direct driving of LCD panel
- Three LCD clock sources with selectable prescaler
- Up to four common pins:
 - Static (1 common)
 - 1/2 multiplex (2 commons)
 - 1/3 multiplex (3 commons)
 - 1/4 multiplex (4 commons)
- Segment pins up to:
 - 16 (PIC16(L)F1933)
- Static, 1/2 or 1/3 LCD Bias

Note: COM3 and SEG15 share the same physical pin on the PIC16(L)F1933, therefore SEG15 is not available when using 1/4 multiplex displays.

FIGURE 27-1: LCD DRIVER MODULE BLOCK DIAGRAM



27.12 Operation During Sleep

The LCD module can operate during Sleep. The selection is controlled by bit SLPEN of the LCDCON register. Setting the SLPEN bit allows the LCD module to go to Sleep. Clearing the SLPEN bit allows the module to continue to operate during Sleep.

If a **SLEEP** instruction is executed and SLPEN = 1, the LCD module will cease all functions and go into a very low-current consumption mode. The module will stop operation immediately and drive the minimum LCD voltage on both segment and common lines. Figure 27-20 shows this operation.

The LCD module can be configured to operate during Sleep. The selection is controlled by bit SLPEN of the LCDCON register. Clearing SLPEN and correctly configuring the LCD module clock will allow the LCD module to operate during Sleep. Setting SLPEN and correctly executing the LCD module shutdown will disable the LCD module during Sleep and save power.

If a **SLEEP** instruction is executed and SLPEN = 1, the LCD module will immediately cease all functions, drive the outputs to Vss and go into a very low-current mode. The **SLEEP** instruction should only be executed after the LCD module has been disabled and the current cycle completed, thus ensuring that there are no DC voltages on the glass. To disable the LCD module, clear the LCDEN bit. The LCD module will complete the disabling process after the current frame, clear the LCDA bit and optionally cause an interrupt.

The steps required to properly enter Sleep with the LCD disabled are:

- Clear LCDEN
- Wait for LCDA = 0 either by polling or by interrupt
- Execute **SLEEP**

If SLPEN = 0 and **SLEEP** is executed while the LCD module clock source is Fosc/4, then the LCD module will halt with the pin driving the last LCD voltage pattern. Prolonged exposure to a fixed LCD voltage pattern will cause damage to the LCD glass. To prevent LCD glass damage, either perform the proper LCD module shutdown prior to Sleep, or change the LCD module clock to allow the LCD module to continue operation during Sleep.

If a **SLEEP** instruction is executed and SLPEN = 0 and the LCD module clock is either T1OSC or LFINTOSC, the module will continue to display the current contents of the LCDDATA registers. While in Sleep, the LCD data cannot be changed. If the LCDIE bit is set, the device will wake from Sleep on the next LCD frame boundary. The LCD module current consumption will not decrease in this mode; however, the overall device power consumption will be lower due to the shutdown of the CPU and other peripherals.

Table 27-8 shows the status of the LCD module during a Sleep while using each of the three available clock sources.

Note: When the LCDEN bit is cleared, the LCD module will be disabled at the completion of frame. At this time, the port pins will revert to digital functionality. To minimize power consumption due to floating digital inputs, the LCD pins should be driven low using the PORT and TRIS registers.

If a **SLEEP** instruction is executed and SLPEN = 0, the module will continue to display the current contents of the LCDDATA registers. To allow the module to continue operation while in Sleep, the clock source must be either the LFINTOSC or T1OSC external oscillator. While in Sleep, the LCD data cannot be changed. The LCD module current consumption will not decrease in this mode; however, the overall consumption of the device will be lower due to shutdown of the core and other peripheral functions.

Table 27-8 shows the status of the LCD module during Sleep while using each of the three available clock sources:

TABLE 27-8: LCD MODULE STATUS DURING SLEEP

Clock Source	SLPEN	Operational During Sleep
T1OSC	0	Yes
	1	No
LFINTOSC	0	Yes
	1	No
Fosc/4	0	No
	1	No

Note: The LFINTOSC or external T1OSC oscillator must be used to operate the LCD module during Sleep.

If LCD interrupts are being generated (Type-B waveform with a multiplex mode not static) and LCDIE = 1, the device will awaken from Sleep on the next frame boundary.

PIC16(L)F1933

FIGURE 30-1: PIC16F1933 VOLTAGE FREQUENCY GRAPH, $-40^{\circ}\text{C} \leq T_A \leq +125^{\circ}\text{C}$

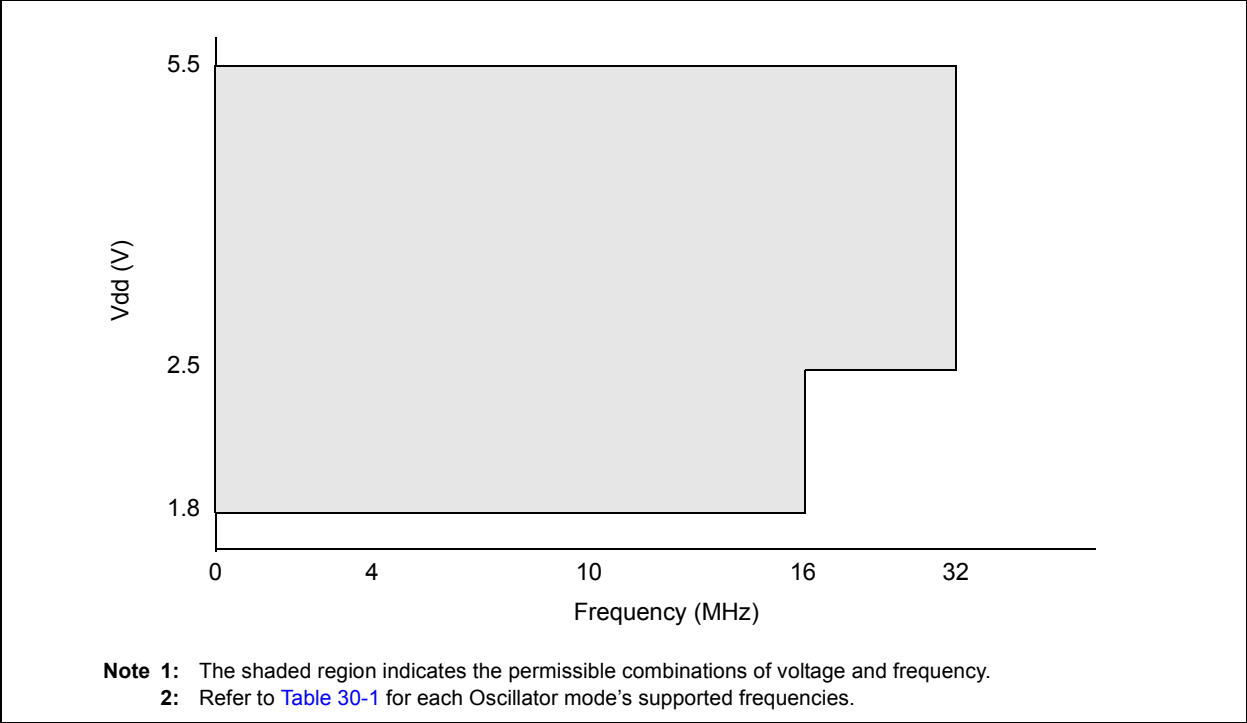


FIGURE 30-2: PIC16LF1933 VOLTAGE FREQUENCY GRAPH, $-40^{\circ}\text{C} \leq T_A \leq +125^{\circ}\text{C}$

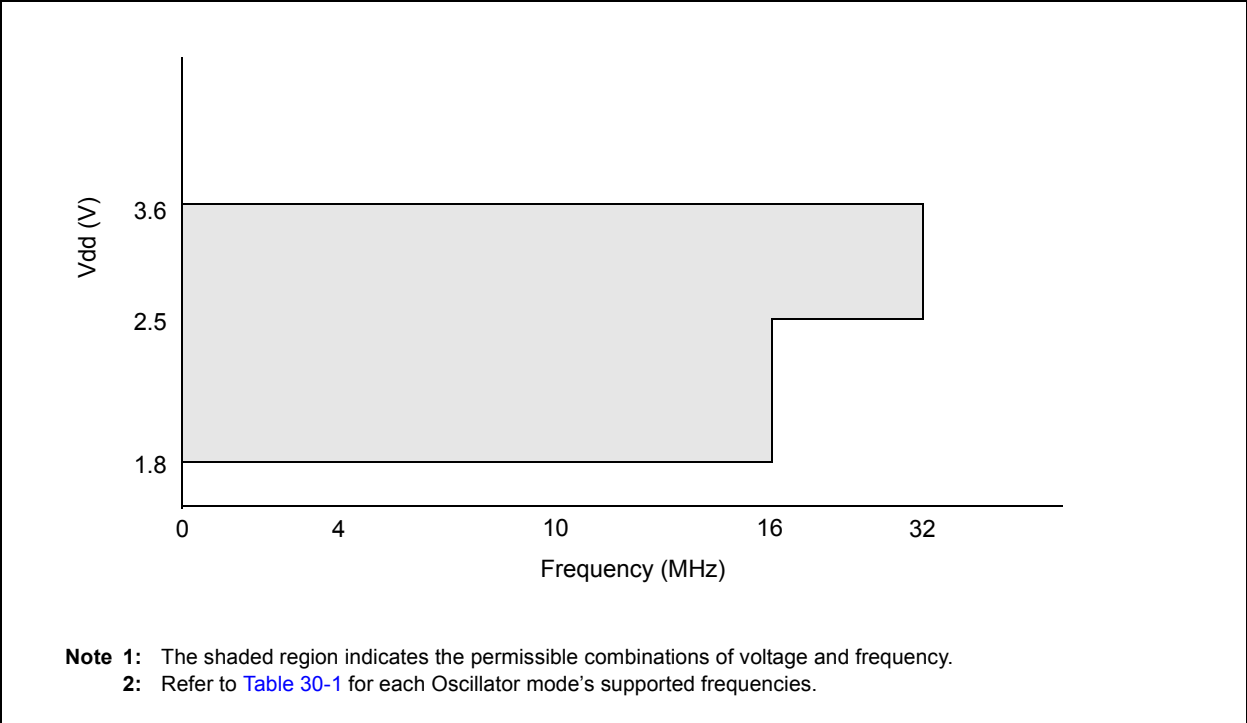


FIGURE 31-3: I_{DD} TYPICAL, XT AND EXTRC OSCILLATOR MODE, PIC16LF1933 ONLY

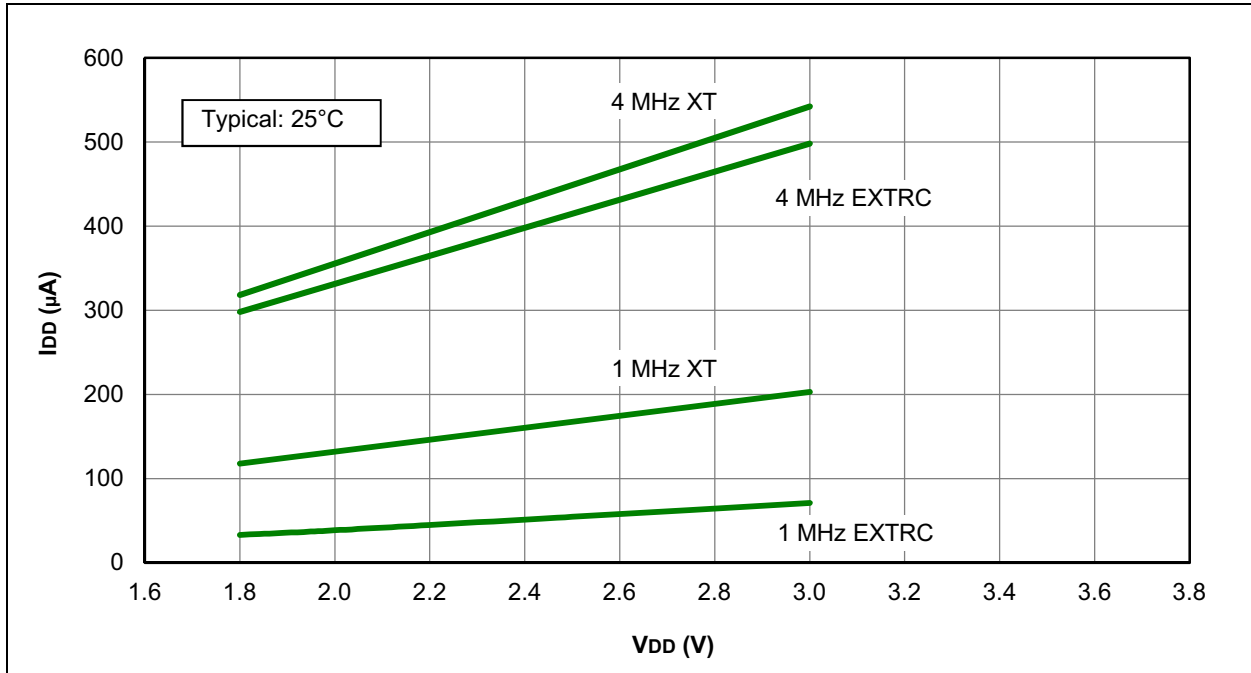
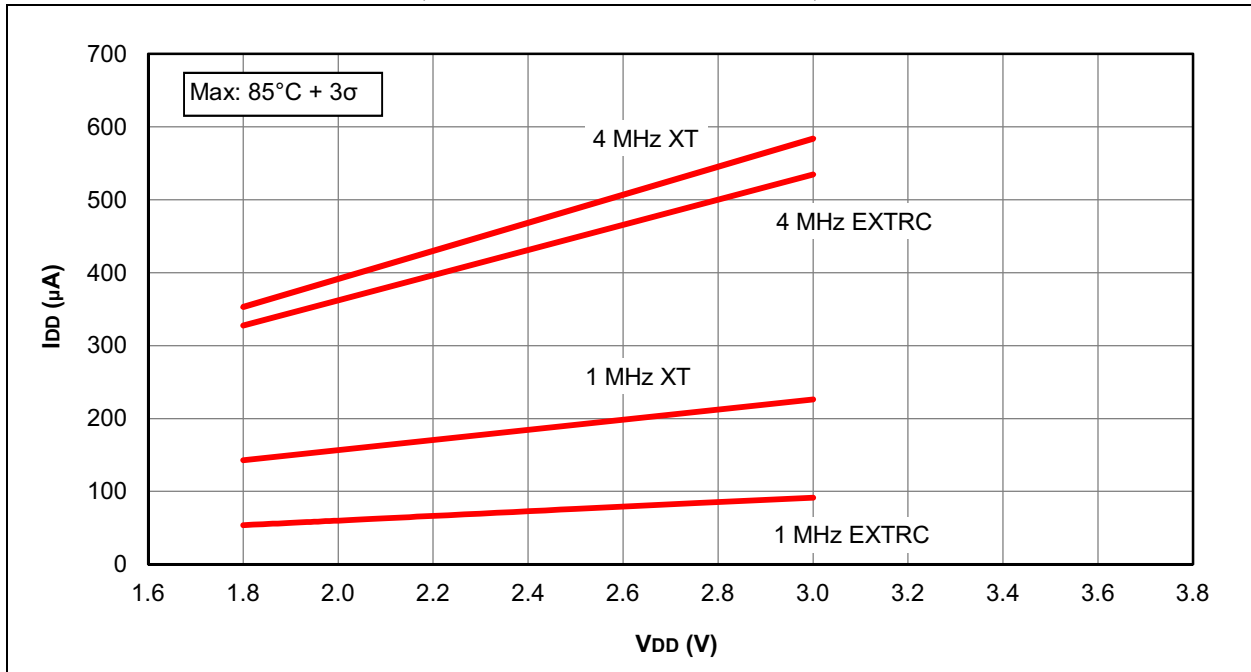


FIGURE 31-4: I_{DD} MAXIMUM, XT AND EXTRC OSCILLATOR, PIC16LF1933 ONLY



PIC16(L)F1933

FIGURE 31-13: I_{DD} TYPICAL, EC OSCILLATOR, HIGH-POWER MODE, PIC16LF1933 ONLY

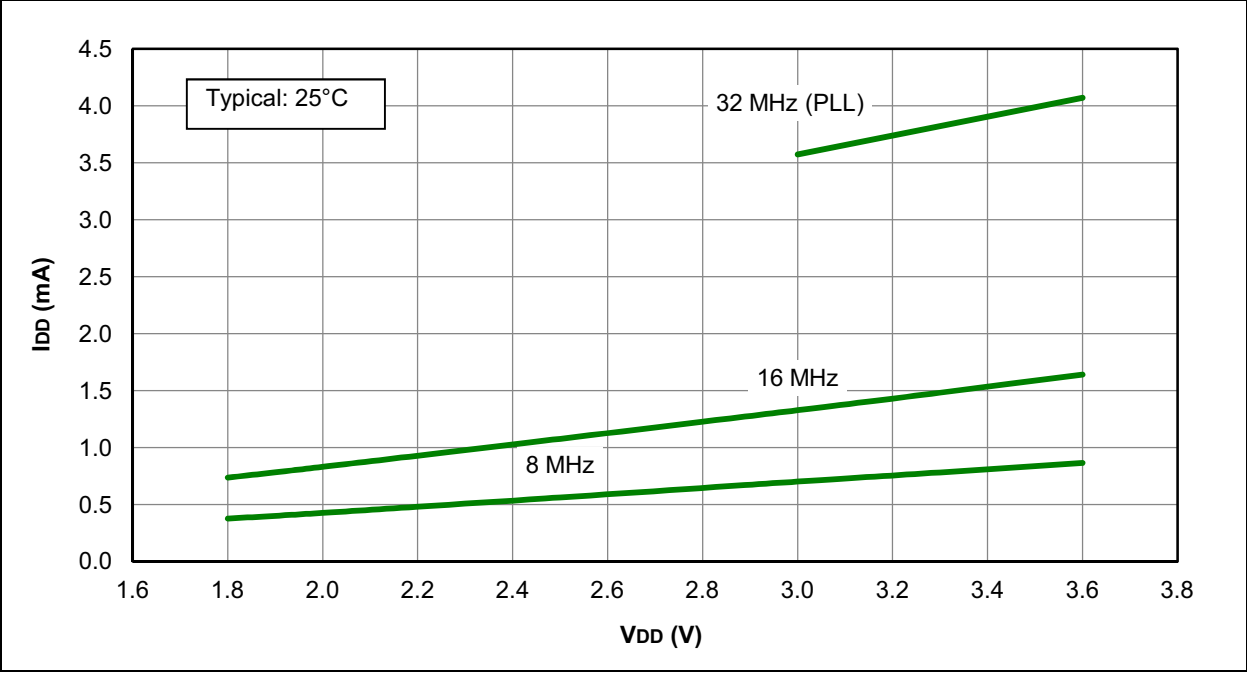
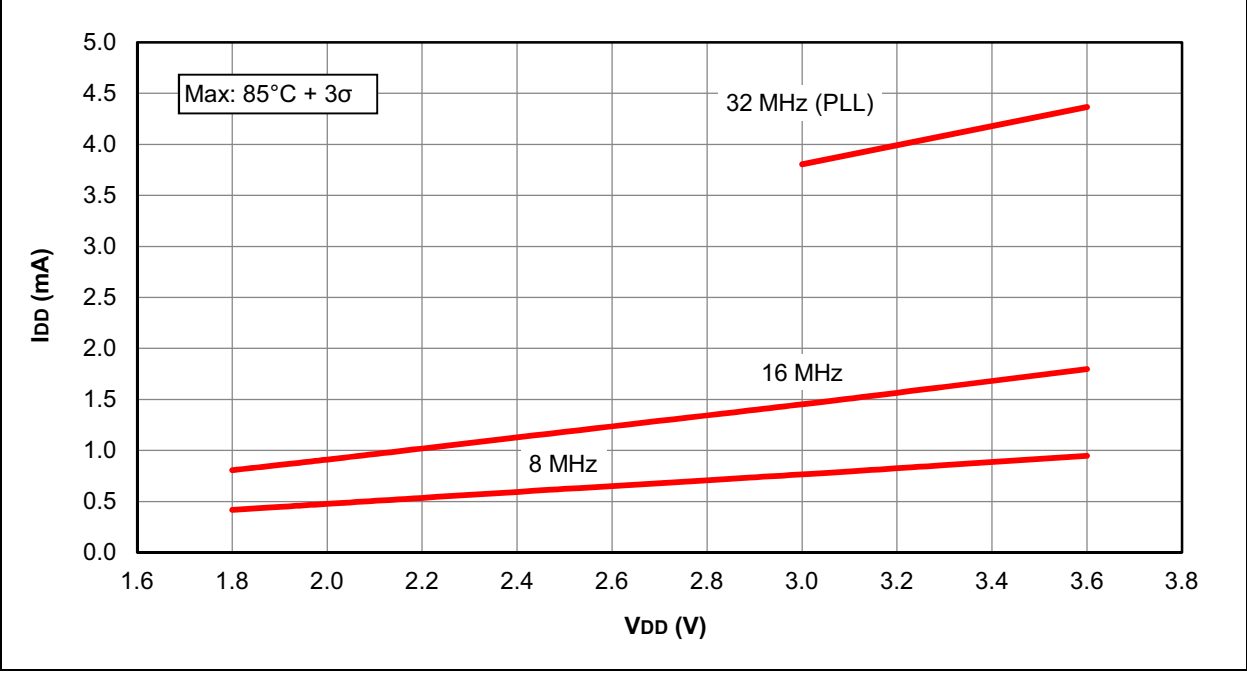


FIGURE 31-14: I_{DD} MAXIMUM, EC OSCILLATOR, HIGH-POWER MODE, PIC16LF1933 ONLY



PIC16(L)F1933

FIGURE 31-21: I_{DD} TYPICAL, HFINTOSC MODE, PIC16LF1933 ONLY

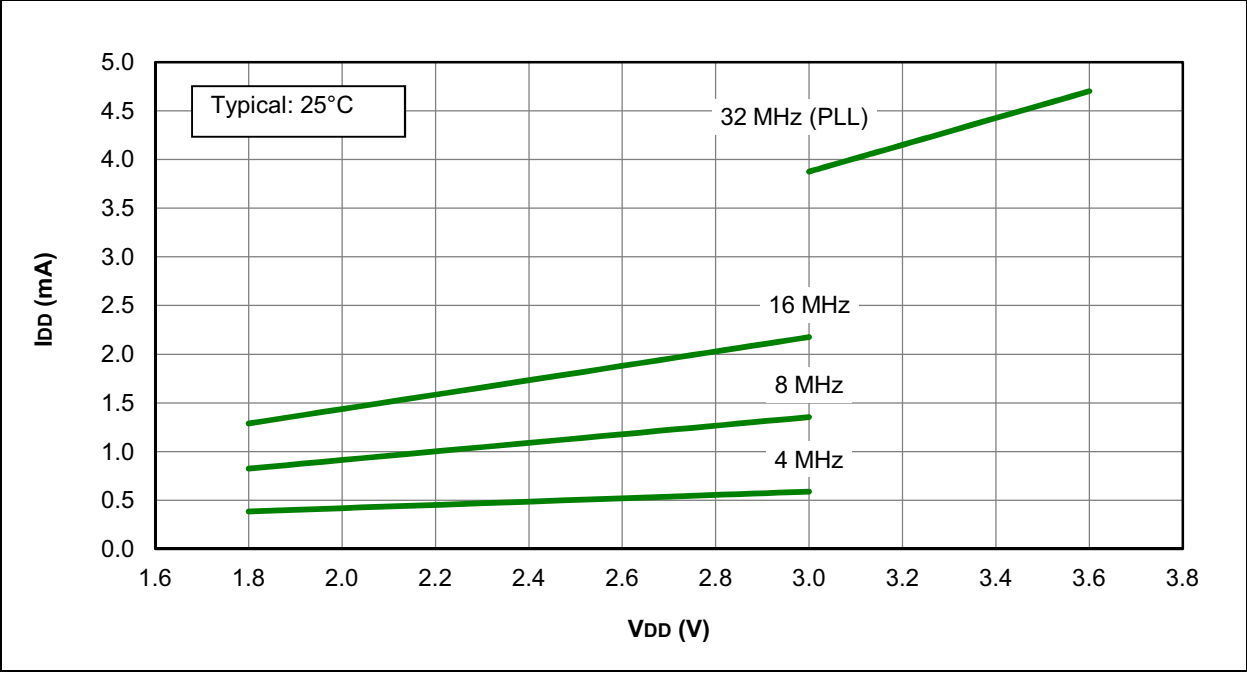
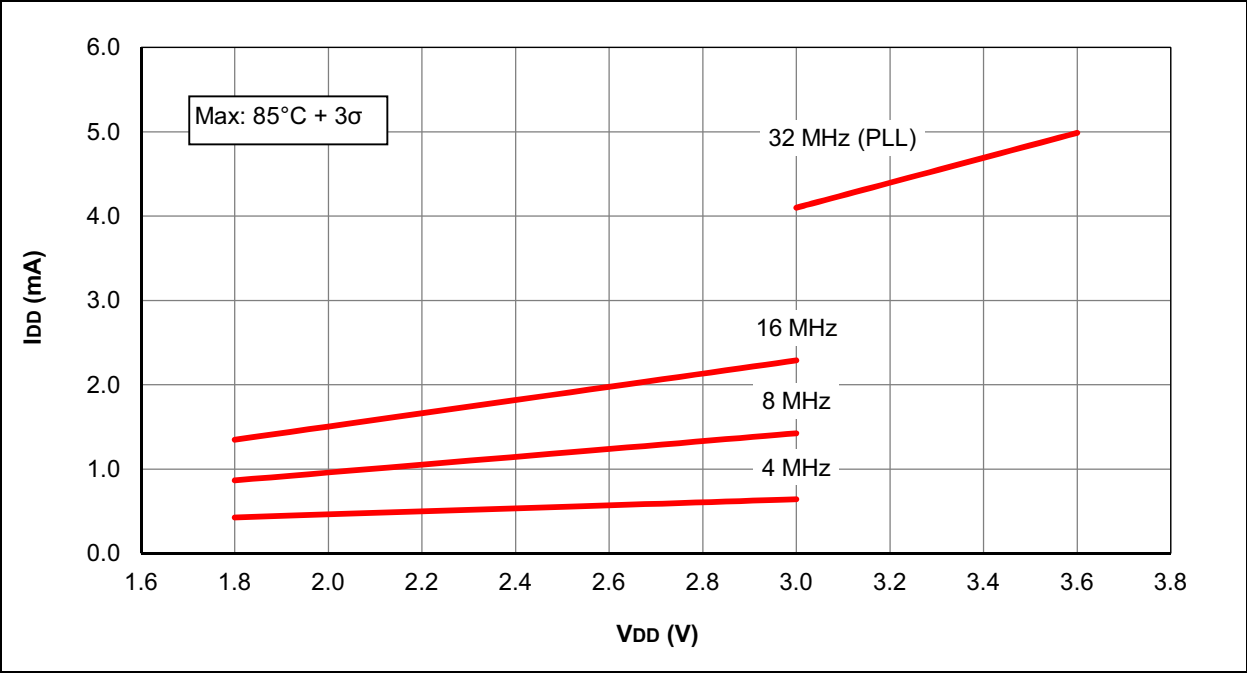


FIGURE 31-22: I_{DD} MAXIMUM, HFINTOSC MODE, PIC16LF1933 ONLY



PIC16(L)F1933

FIGURE 31-53: V_{OH} vs. I_{OH} , OVER TEMPERATURE, $V_{DD} = 1.8V$

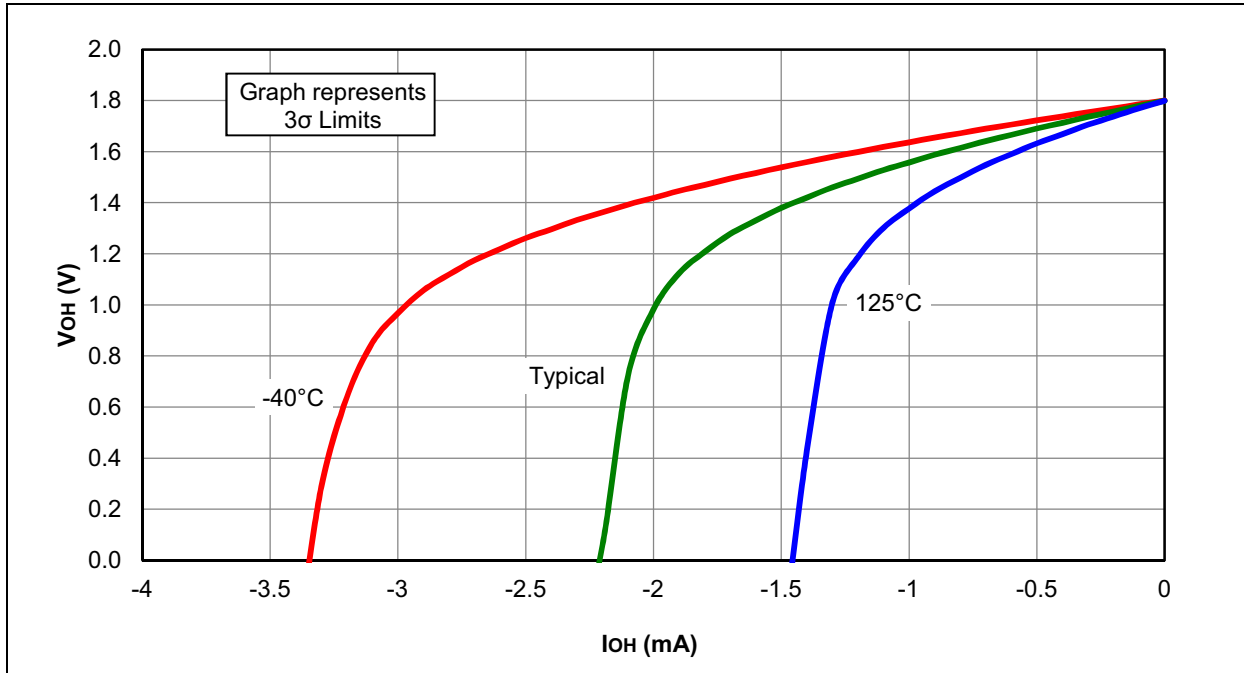
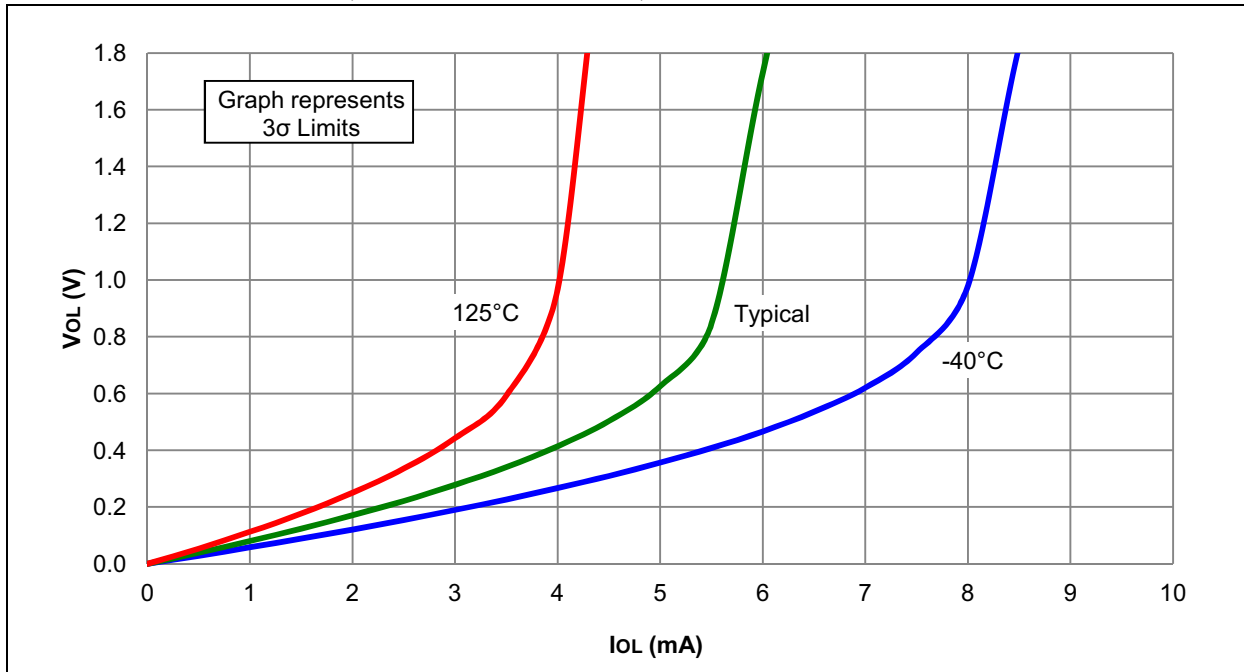


FIGURE 31-54: V_{OL} vs. I_{OL} , OVER TEMPERATURE, $V_{DD} = 1.8V$



PIC16(L)F1933

FIGURE 31-61: WDT TIME-OUT PERIOD

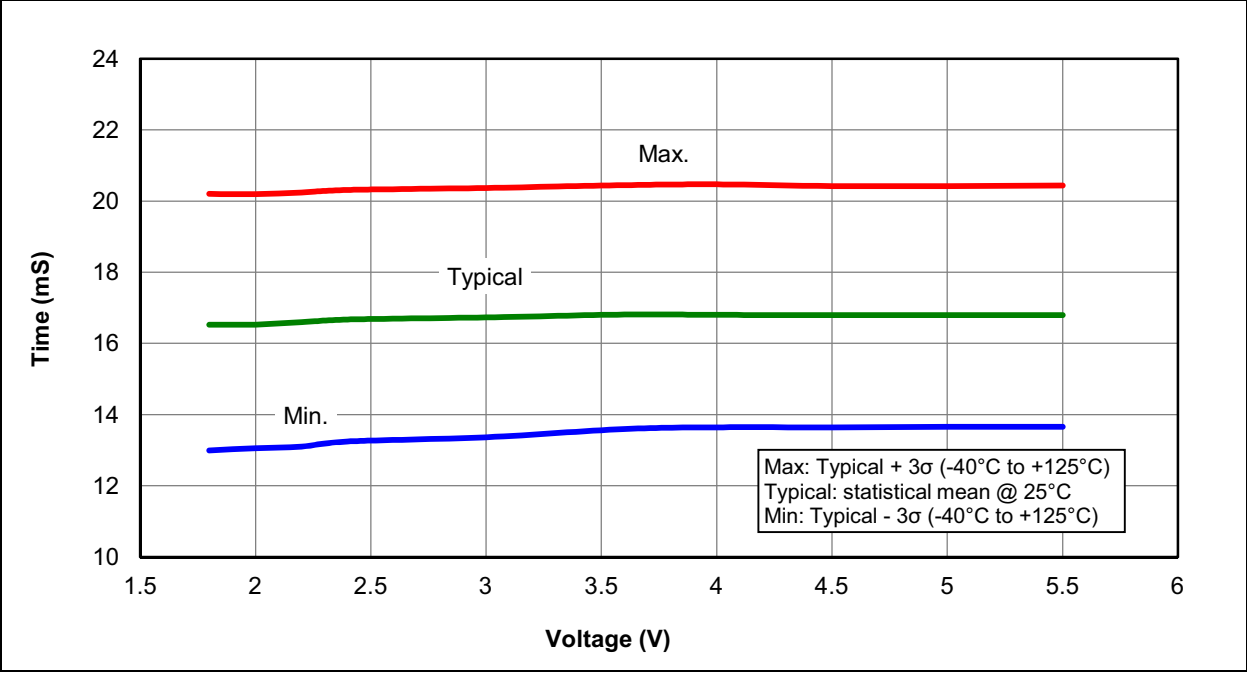


FIGURE 31-62: PWRT PERIOD

