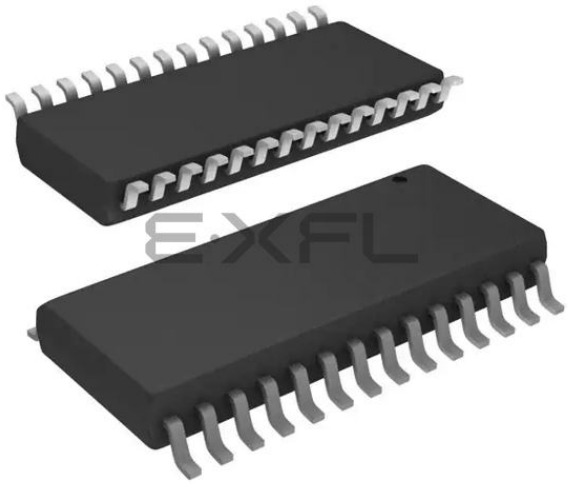


Welcome to E-XFL.COM

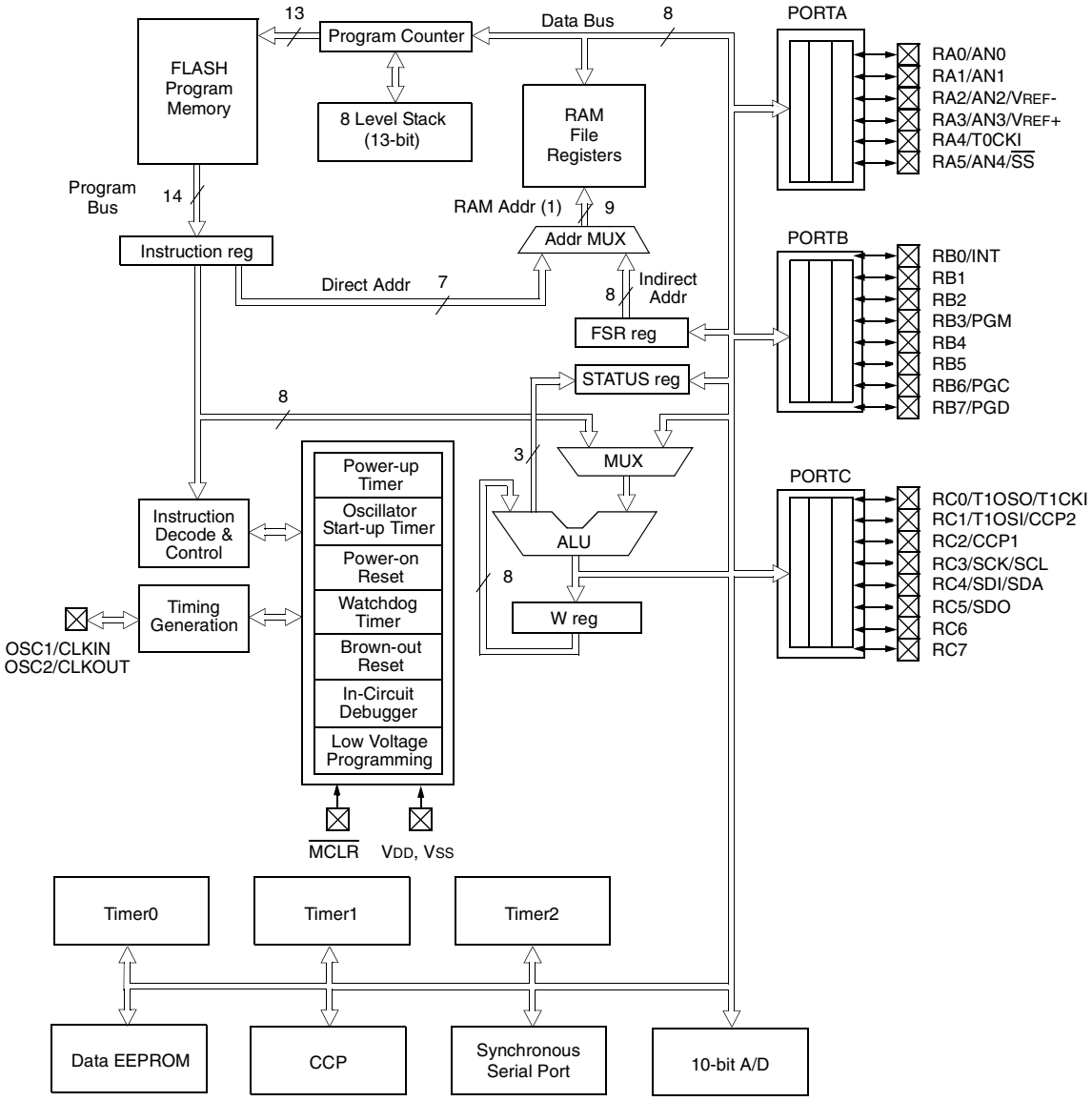
What is "Embedded - Microcontrollers"?

"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	20MHz
Connectivity	I²C, SPI
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	22
Program Memory Size	3.5KB (2K x 14)
Program Memory Type	FLASH
EEPROM Size	64 x 8
RAM Size	128 x 8
Voltage - Supply (Vcc/Vdd)	2.2V ~ 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/pic16lf872t-i-so

FIGURE 1-1: PIC16F872 BLOCK DIAGRAM



Note 1: Higher order bits are from the STATUS register.

2.5 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.

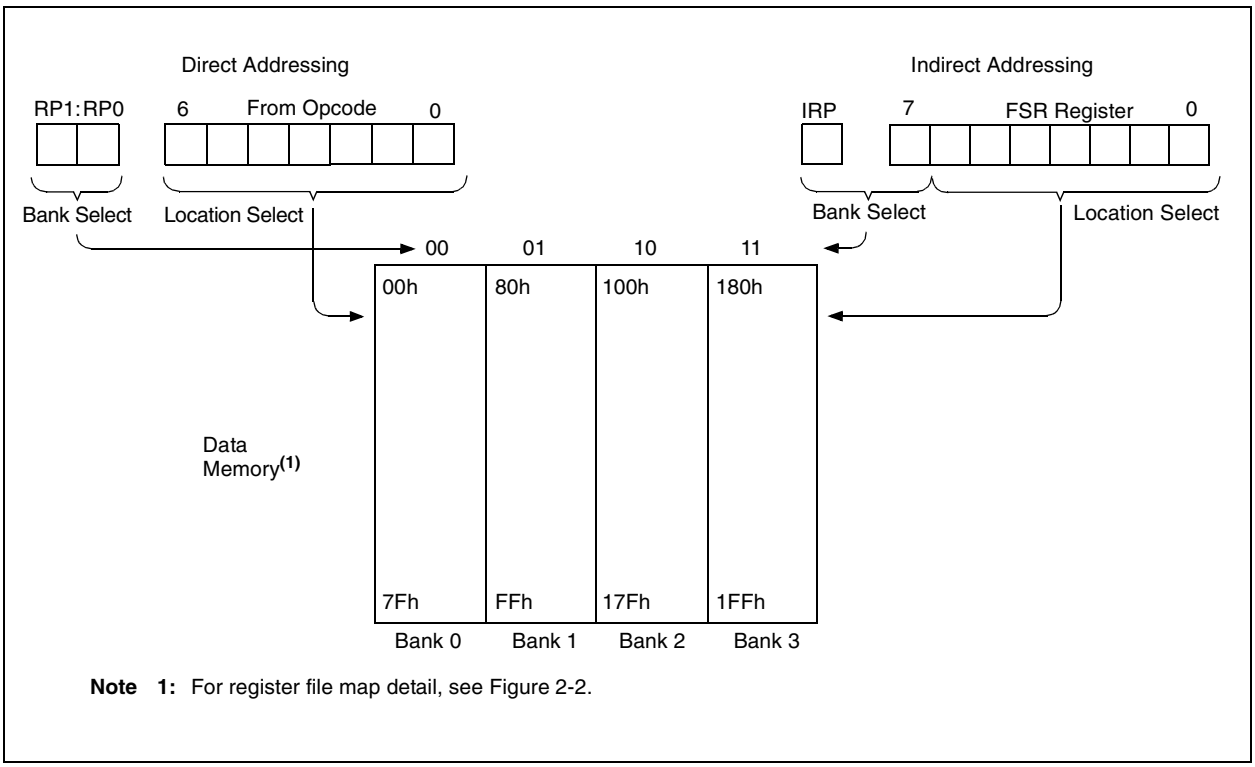
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself indirectly (FSR = '0'), will read 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-4.

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-1.

EXAMPLE 2-1: INDIRECT ADDRESSING

```
        MOVLW  0x20    ;initialize pointer
        MOVWF  FSR     ;to RAM
NEXT    CLRF   INDF    ;clear INDF register
        INCF   FSR,F   ;inc pointer
        BTFSS  FSR,4   ;all done?
        GOTO   NEXT    ;no clear next
CONTINUE
        :              ;yes continue
```

FIGURE 2-4: DIRECT/INDIRECT ADDRESSING



Note 1: For register file map detail, see Figure 2-2.

NOTES:

3.0 DATA EEPROM AND FLASH PROGRAM MEMORY

The Data EEPROM and FLASH Program Memory are readable and writable during normal operation over the entire VDD range. These operations take place on a single byte for Data EEPROM memory and a single word for Program memory. A write operation causes an erase-then-write operation to take place on the specified byte or word. A bulk erase operation may not be issued from user code (which includes removing code protection).

Access to program memory allows for checksum calculation. The values written to Program memory do not need to be valid instructions. Therefore, numbers of up to 14 bits can be stored in memory for use as calibration parameters, serial numbers, packed 7-bit ASCII, etc. Executing a program memory location, containing data that forms an invalid instruction, results in the execution of a NOP instruction.

The EEPROM Data memory is rated for high erase/write cycles (specification #D120). The FLASH Program memory is rated much lower (specification #D130) because EEPROM Data memory can be used to store frequently updated values. An on-chip timer controls the write time and it will vary with voltage and temperature, as well as from chip to chip. Please refer to the specifications for exact limits (specifications #D122 and #D133).

A byte or word write automatically erases the location and writes the new value (erase before write). Writing to EEPROM Data memory does not impact the operation of the device. Writing to Program memory will cease the execution of instructions until the write is complete. The program memory cannot be accessed during the write. During the write operation, the oscillator continues to run, the peripherals continue to function and interrupt events will be detected and essentially “queued” until the write is complete. When the write completes, the next instruction in the pipeline is executed and the branch to the interrupt vector will take place if the interrupt is enabled and occurred during the write.

Read and write access to both memories take place indirectly through a set of Special Function Registers (SFR). The six SFRs used are:

- EEDATA
- EEDATH
- EEADR
- EEADRH
- EECON1
- EECON2

The EEPROM Data memory allows byte read and write operations without interfering with the normal operation of the microcontroller. When interfacing to EEPROM Data memory, the EEADR register holds the address to be accessed. Depending on the operation, the EEDATA register holds the data to be written or the data read at the address in EEADR. The PIC16F872 has 64 bytes of EEPROM Data memory and therefore, requires that the two Most Significant bits of EEADR remain clear. EEPROM Data memory on these devices wraps around to 0 (i.e., 40h in the EEADR maps to 00h).

The FLASH Program memory allows non-intrusive read access, but write operations cause the device to stop executing instructions until the write completes. When interfacing to the Program memory, the EEADRH:EEADR registers pair forms a two-byte word which holds the 13-bit address of the memory location being accessed. The EEDATH:EEDATA register pair holds the 14-bit data for writes or reflects the value of program memory after a read operation. Just as in EEPROM Data memory accesses, the value of the EEADRH:EEADR registers must be within the valid range of program memory, depending on the device (0000h to 07FFh). Addresses outside of this range wrap around to 0000h (i.e., 0800h maps to 0000h).

3.1 EECON1 and EECON2 Registers

The EECON1 register is the control register for configuring and initiating the access. The EECON2 register is not a physically implemented register, but is used exclusively in the memory write sequence to prevent inadvertent writes.

There are many bits used to control the read and write operations to EEPROM Data and FLASH Program memory. The EEPGD bit determines if the access will be a program or data memory access. When clear, any subsequent operations will work on the EEPROM Data memory. When set, all subsequent operations will operate in the Program memory.

Read operations only use one additional bit, RD, which initiates the read operation from the desired memory location. Once this bit is set, the value of the desired memory location will be available in the data registers. This bit cannot be cleared by firmware. It is automatically cleared at the end of the read operation. For EEPROM Data memory reads, the data will be available in the EEDATA register in the very next instruction cycle after the RD bit is set. For program memory reads, the data will be loaded into the EEDATH:EEDATA registers, following the second instruction after the RD bit is set.

The steps to write to program memory are:

1. Write the address to EEADRH:EEADR. Make sure that the address is not larger than the memory size of the device.
2. Write the 14-bit data value to be programmed in the EEDATH:EEDATA registers.
3. Set the EEPGD bit to point to FLASH Program memory.
4. Set the WREN bit to enable program operations.
5. Disable interrupts (if enabled).
6. Execute the special five instruction sequence:
   - Write 55h to EECON2 in two steps (first to W, then to EECON2)
   - Write AAh to EECON2 in two steps (first to W, then to EECON2)
   - Set the WR bit
7. Execute two NOP instructions to allow the microcontroller to setup for write operation.
8. Enable interrupts (if using interrupts).
9. Clear the WREN bit to disable program operations.

At the completion of the write cycle, the WR bit is cleared and the EEIF interrupt flag bit is set. (EEIF must be cleared by firmware). Since the microcontroller does not execute instructions during the write cycle, the firmware does not necessarily have to check either EEIF or WR to determine if the write had finished.

EXAMPLE 3-4: FLASH PROGRAM WRITE

```
        BSF     STATUS, RP1     ;
        BCF     STATUS, RP0     ;Bank 2
        MOVF    ADDRL, W        ;Write address
        MOVWF   EEADR           ;of desired
        MOVF    ADDRH, W        ;program memory
        MOVWF   EEADRH          ;location
        MOVF    VALUEL, W       ;Write value to
        MOVWF   EEDATA          ;program at
        MOVF    VALUEH, W       ;desired memory
        MOVWF   EEDATH          ;location
        BSF     STATUS, RP0     ;Bank 3
        BSF     EECON1, EEPGD   ;Point to Program memory
        BSF     EECON1, WREN    ;Enable writes
                                ;Only disable interrupts
        BCF     INTCON, GIE     ;if already enabled,
                                ;otherwise discard
Required
Sequence
        MOVLW   0x55            ;Write 55h to
        MOVWF   EECON2          ;EECON2
        MOVLW   0xAA            ;Write AAh to
        MOVWF   EECON2          ;EECON2
        BSF     EECON1, WR      ;Start write operation
        NOP                     ;Two NOPs to allow micro
        NOP                     ;to setup for write
                                ;Only enable interrupts
        BSF     INTCON, GIE     ;if using interrupts,
                                ;otherwise discard
        BCF     EECON1, WREN    ;Disable writes
```

3.6 Write Verify

The PIC16F87X devices do not automatically verify the value written during a write operation. Depending on the application, good programming practice may dictate that the value written to memory be verified against the original value. This should be used in applications where excessive writes can stress bits near the specified endurance limits.

3.7 Protection Against Spurious Writes

There are conditions when the device may not want to write to the EEPROM Data memory or FLASH program memory. To protect against these spurious write conditions various mechanisms have been built into the device. On power-up, the WREN bit is cleared and the Power-up Timer (if enabled) prevents writes.

The write initiate sequence and the WREN bit together help prevent any accidental writes during brown-out, power glitches or firmware malfunction.

TABLE 4-5: PORTC FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC1/T1OSI/CCP2	bit1	ST	Input/output port pin or Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output.
RC2/CCP1	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/ PWM output.
RC3/SCK/SCL	bit3	ST	RC3 can also be the synchronous serial clock for both SPI and I^2C modes.
RC4/SDI/SDA	bit4	ST	RC4 can also be the SPI Data In (SPI mode) or Data I/O (I^2C mode).
RC5/SDO	bit5	ST	Input/output port pin or Synchronous Serial Port data output (SPI mode).
RC6	bit6	ST	Input/output port pin.
RC7	bit7	ST	Input/output port pin.

Legend: ST = Schmitt Trigger input

TABLE 4-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

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
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged

TABLE 6-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

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
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	**(3)**	ADIF	**(3)**	**(3)**	SSPIF	CCP1IF	TMR2IF	TMR1IF	r0rr 0000	0000 0000
8Ch	PIE1	**(3)**	ADIE	**(3)**	**(3)**	SSPIE	CCP1IE	TMR2IE	TMR1IE	r0rr 0000	0000 0000
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

7.1 Timer2 Prescaler and Postscaler

The prescaler and postscaler counters are cleared when any of the following occurs:

- a write to the TMR2 register
- a write to the T2CON register
- any device RESET (POR, $\overline{\text{MCLR}}$ Reset, WDT Reset or BOR)

TMR2 is not cleared when T2CON is written.

7.2 Output of TMR2

The output of TMR2 (before the postscaler) is fed to the SSP module, which optionally uses it to generate shift clock.

TABLE 7-1: REGISTERS ASSOCIATED WITH TIMER2 AS A TIMER/COUNTER

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
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	**(3)**	ADIF	**(3)**	**(3)**	SSPIF	CCP1IF	TMR2IF	TMR1IF	r0rr 0000	0000 0000
8Ch	PIE1	**(3)**	ADIE	**(3)**	**(3)**	SSPIE	CCP1IE	TMR2IE	TMR1IE	r0rr 0000	0000 0000
11h	TMR2	Timer2 Module Register								0000 0000	0000 0000
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
92h	PR2	Timer2 Period Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer2 module.

9.1 SPI Mode

The SPI mode allows 8 bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

- Serial Data Out (SDO)
- Serial Data In (SDI)
- Serial Clock (SCK)

Additionally, a fourth pin may be used when in a Slave mode of operation:

- Slave Select ($\overline{SS}$)

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits (SSPCON<5:0> and SSPSTAT<7:6>). These control bits allow the following to be specified:

- Master mode (SCK is the clock output)
- Slave mode (SCK is the clock input)
- Clock Polarity (IDLE state of SCK)
- Data input sample phase (middle or end of data output time)
- Clock edge (output data on rising/falling edge of SCK)
- Clock Rate (Master mode only)
- Slave Select mode (Slave mode only)

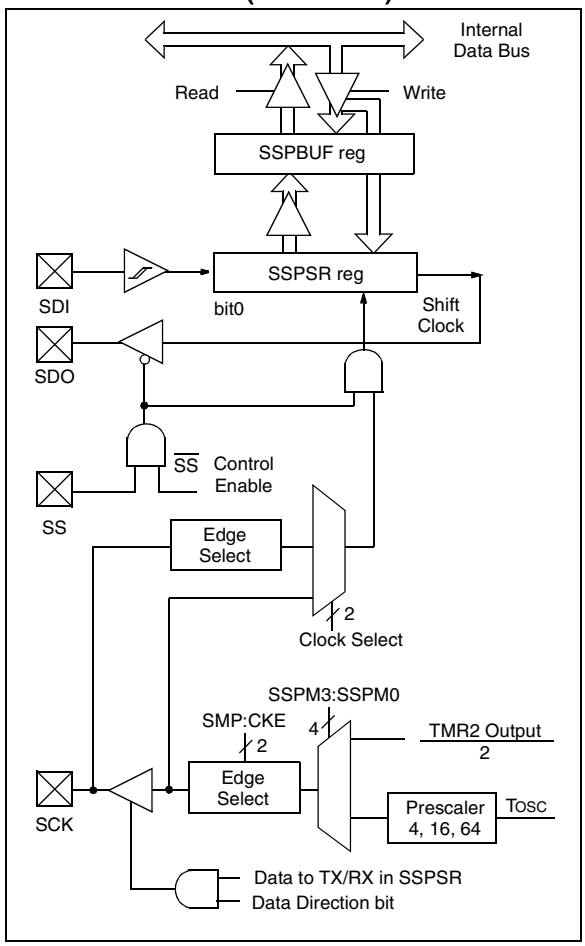
Figure 9-4 shows the block diagram of the MSSP module when in SPI mode.

To enable the serial port, MSSP Enable bit, SSPEN (SSPCON<5>) must be set. To reset or reconfigure SPI mode, clear bit SSPEN, re-initialize the SSPCON registers, and then set bit SSPEN. This configures the SDI, SDO, SCK and $\overline{SS}$ pins as serial port pins. For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS register) appropriately programmed. That is:

- SDI is automatically controlled by the SPI module
- SDO must have TRISC<5> cleared
- SCK (Master mode) must have TRISC<3> cleared
- SCK (Slave mode) must have TRISC<3> set
- $\overline{SS}$ must have TRISA<5> set, and
- Register ADCON1 must be set in a way that pin RA5 is configured as a digital I/O

Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

FIGURE 9-1: MSSP BLOCK DIAGRAM (SPI MODE)



9.1.1 MASTER MODE

The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave (Processor 2, Figure 9-5) is to broadcast data by the software protocol.

In Master mode, the data is transmitted/received as soon as the SSPBUF register is written to. If the SPI module is only going to receive, the SDO output could be disabled (programmed as an input). The SSPSR register will continue to shift in the signal present on the SDI pin at the programmed clock rate. As each byte is received, it will be loaded into the SSPBUF register as if a normal received byte (interrupts and status bits appropriately set). This could be useful in receiver applications as a “line activity monitor”.

11.3 Reset

The PIC16F872 differentiates between various kinds of RESET:

- Power-on Reset (POR)
- $\overline{MCLR}$ Reset during normal operation
- $\overline{MCLR}$ Reset during SLEEP
- WDT Reset (during normal operation)
- WDT Wake-up (during SLEEP)
- Brown-out Reset (BOR)

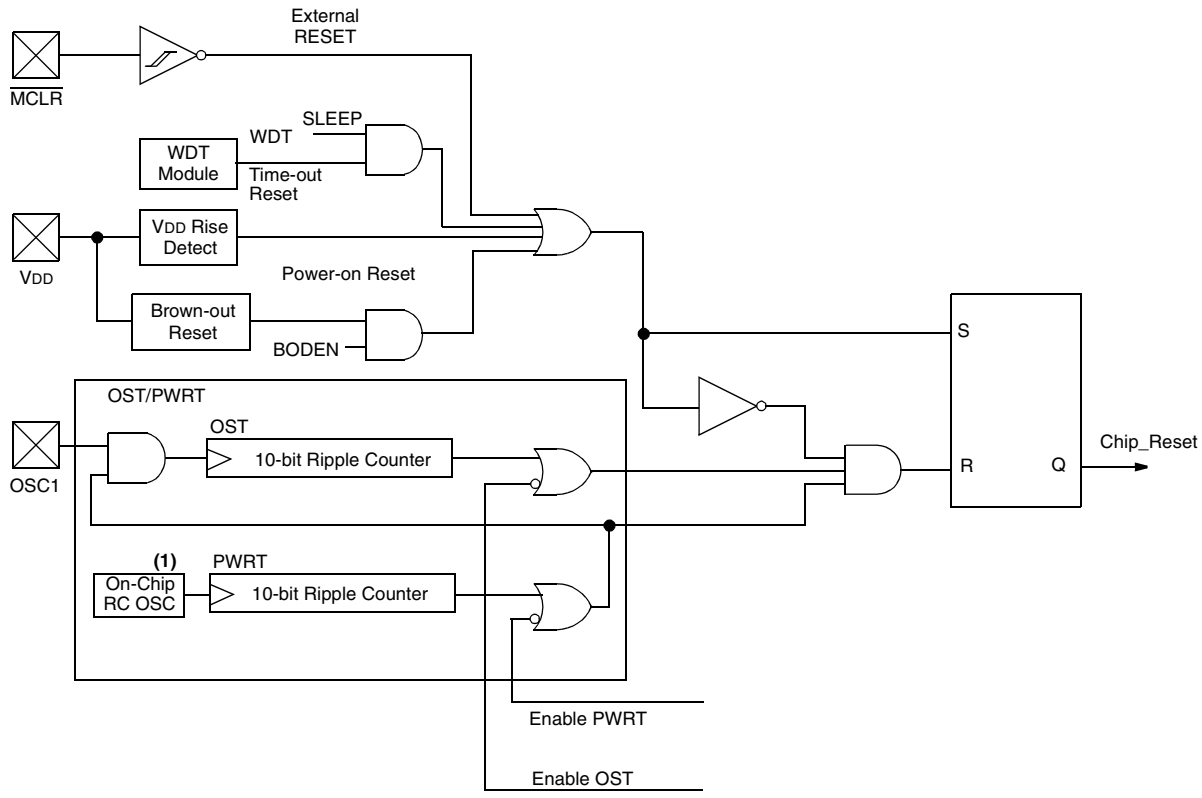
Some registers are not affected in any RESET condition. Their status is unknown on POR and unchanged in any other RESET. Most other registers are reset to a "RESET state" on Power-on Reset (POR), on the $\overline{MCLR}$ and WDT Reset, on $\overline{MCLR}$ Reset during SLEEP, and Brown-out Reset (BOR). They are not affected by a WDT Wake-up, which is viewed as the resumption of normal operation. The $\overline{TO}$ and $\overline{PD}$ bits are set or cleared differently in different RESET situations, as indicated in Table 11-4. These bits are used in software to determine the nature of the RESET. See Table 11-6 for a full description of RESET states of all registers.

A simplified block diagram of the On-Chip Reset circuit is shown in Figure 11-4.

These devices have a $\overline{MCLR}$ noise filter in the $\overline{MCLR}$ Reset path. The filter will detect and ignore small pulses.

It should be noted that a WDT Reset does not drive $\overline{MCLR}$ pin low.

FIGURE 11-4: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT



Note 1: This is a separate oscillator from the RC oscillator of the CLKIN pin.

FIGURE 11-5: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ TIED TO V_{DD} VIA RC NETWORK)

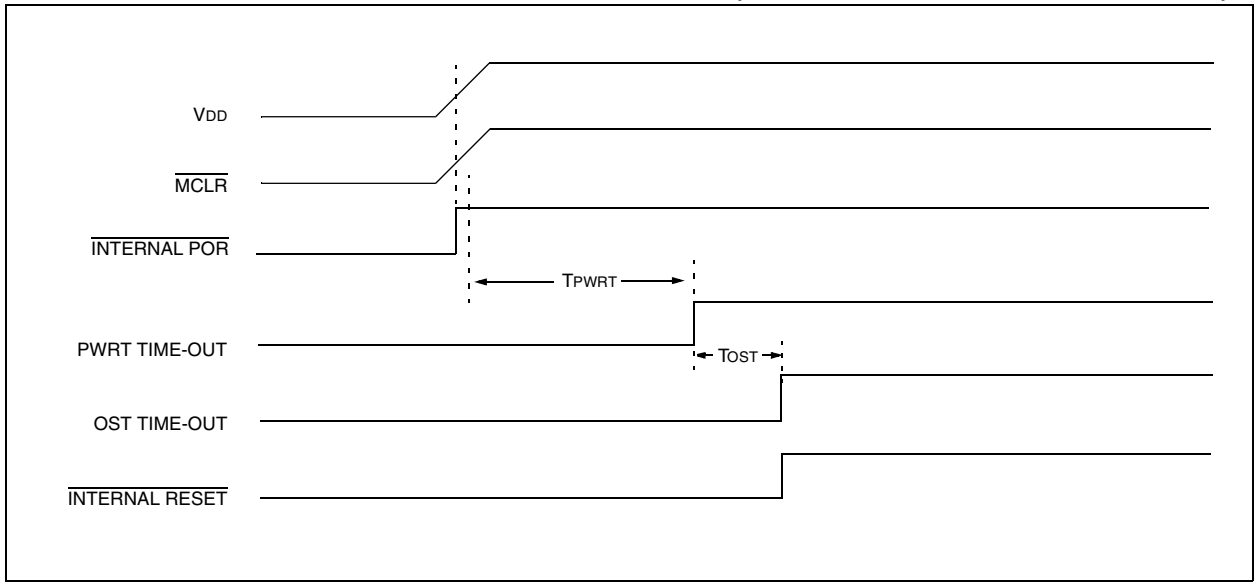
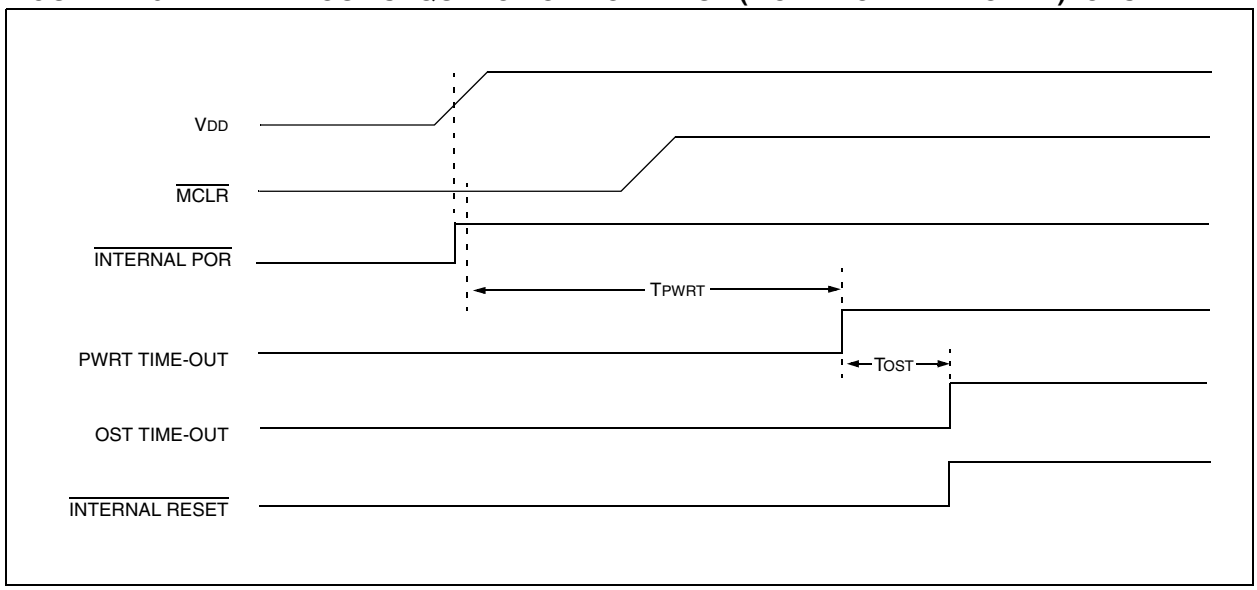


FIGURE 11-6: TIME-OUT SEQUENCE ON POWER-UP ($\overline{MCLR}$ NOT TIED TO V_{DD}): CASE 1



11.10.1 INT INTERRUPT

External interrupt on the RB0/INT pin is edge triggered, either rising if bit INTEDG (OPTION_REG<6>) is set, or falling if the INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, flag bit INTF (INTCON<1>) is set. This interrupt can be disabled by clearing enable bit INTE (INTCON<4>). Flag bit INTF must be cleared in software in the Interrupt Service Routine before re-enabling this interrupt. The INT interrupt can wake-up the processor from SLEEP, if bit INTE was set prior to going into SLEEP. The status of global interrupt enable bit GIE, decides whether or not the processor branches to the interrupt vector following wake-up. See Section 11.13 for details on SLEEP mode.

11.10.2 TMR0 INTERRUPT

An overflow (FFh → 00h) in the TMR0 register will set flag bit TMR0IF (INTCON<2>). The interrupt can be enabled/disabled by setting/clearing enable bit TMR0IE (INTCON<5>), see Section 5.0.

11.10.3 PORTB INTCON 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<4>), see Section 4.2.

11.11 Context Saving During Interrupts

During an interrupt, only the return PC value is saved on the stack. Typically, users may wish to save key registers during an interrupt, (i.e., W register and STATUS register). This will have to be implemented in software.

Since the upper 16 bytes of each bank are common in PIC16F872 devices, temporary holding registers, W_TEMP, STATUS_TEMP and PCLATH_TEMP, should be placed in here. These 16 locations don't require banking and therefore, make it easier for context save and restore. The same code shown in Example 11-1 can be used.

EXAMPLE 11-1: SAVING STATUS, W, AND PCLATH REGISTERS IN RAM

```
MOVWF    W_TEMP           ;Copy W to TEMP register
SWAPF    STATUS,W         ;Swap status to be saved into W
CLRF     STATUS           ;bank 0, regardless of current bank, Clears IRP,RP1,RP0
MOVWF    STATUS_TEMP      ;Save status to bank zero STATUS_TEMP register
MOVF     PCLATH, W        ;Only required if using pages 1, 2 and/or 3
MOVWF    PCLATH_TEMP      ;Save PCLATH into W
CLRF     PCLATH           ;Page zero, regardless of current page
:
:(ISR)                    ;(Insert user code here)
:
MOVF     PCLATH_TEMP, W   ;Restore PCLATH
MOVWF    PCLATH           ;Move W into PCLATH
SWAPF    STATUS_TEMP,W    ;Swap STATUS_TEMP register into W
                          ;(sets bank to original state)
MOVWF    STATUS           ;Move W into STATUS register
SWAPF    W_TEMP,F         ;Swap W_TEMP
SWAPF    W_TEMP,W         ;Swap W_TEMP into W
```

12.2 Instruction Descriptions

ADDLW **Add Literal and W**

Syntax:	[*label*] ADDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) + k → (W)
Status Affected:	C, DC, Z
Description:	The contents of the W register are added to the eight-bit literal 'k' and the result is placed in the W register.

ADDWF **Add W and f**

Syntax:	[*label*] ADDWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) + (f) → (destination)
Status Affected:	C, DC, Z
Description:	Add the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

ANDLW **AND Literal with W**

Syntax:	[*label*] ANDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .AND. (k) → (W)
Status Affected:	Z
Description:	The contents of W register are AND'ed with the eight-bit literal 'k'. The result is placed in the W register.

ANDWF **AND W with f**

Syntax:	[*label*] ANDWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) .AND. (f) → (destination)
Status Affected:	Z
Description:	AND the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

BCF **Bit Clear f**

Syntax:	[*label*] BCF f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	0 → (f<b>)
Status Affected:	None
Description:	Bit 'b' in register 'f' is cleared.

BSF **Bit Set f**

Syntax:	[*label*] BSF f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	1 → (f<b>)
Status Affected:	None
Description:	Bit 'b' in register 'f' is set.

BTFSS **Bit Test f, Skip if Set**

Syntax:	[*label*] BTFSS f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b < 7
Operation:	skip if (f<b>) = 1
Status Affected:	None
Description:	If bit 'b' in register 'f' is '0', the next instruction is executed. If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead, making this a 2TCY instruction.

BTFSC **Bit Test, Skip if Clear**

Syntax:	[*label*] BTFSC f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	skip if (f<b>) = 0
Status Affected:	None
Description:	If bit 'b' in register 'f' is '1', the next instruction is executed. If bit 'b', in register 'f', is '0', the next instruction is discarded, and a NOP is executed instead, making this a 2TCY instruction.

SWAPF	Swap Nibbles in f
Syntax:	[*label*] SWAPF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f<3:0>) → (destination<7:4>), (f<7:4>) → (destination<3:0>)
Status Affected:	None
Description:	The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed in register 'f'.

XORLW	Exclusive OR Literal with W
Syntax:	[*label*] XORLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .XOR. k → (W)
Status Affected:	Z
Description:	The contents of the W register are XOR'ed with the eight-bit literal 'k'. The result is placed in the W register.

XORWF	Exclusive OR W with f
Syntax:	[*label*] XORWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) .XOR. (f) → (destination)
Status Affected:	Z
Description:	Exclusive OR the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

13.4 MPLINK Object Linker/ MPLIB Object Librarian

The MPLINK object linker combines relocatable objects created by the MPASM assembler and the MPLAB C17 and MPLAB C18 C compilers. It can also link relocatable objects from pre-compiled libraries, using directives from a linker script.

The MPLIB object librarian is a librarian for pre-compiled code to be used with the MPLINK object linker. When a routine from a library is called from another source file, only the modules that contain that routine will be linked in with the application. This allows large libraries to be used efficiently in many different applications. The MPLIB object librarian manages the creation and modification of library files.

The MPLINK object linker features include:

- Integration with MPASM assembler and MPLAB C17 and MPLAB C18 C compilers.
- Allows all memory areas to be defined as sections to provide link-time flexibility.

The MPLIB object librarian features include:

- Easier linking because single libraries can be included instead of many smaller files.
- Helps keep code maintainable by grouping related modules together.
- Allows libraries to be created and modules to be added, listed, replaced, deleted or extracted.

13.5 MPLAB SIM Software Simulator

The MPLAB SIM software simulator allows code development in a PC-hosted environment by simulating the PICmicro series microcontrollers on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a file, or user-defined key press, to any of the pins. The execution can be performed in single step, execute until break, or trace mode.

The MPLAB SIM simulator fully supports symbolic debugging using the MPLAB C17 and the MPLAB C18 C compilers and the MPASM assembler. The software simulator offers the flexibility to develop and debug code outside of the laboratory environment, making it an excellent multi-project software development tool.

13.6 MPLAB ICE High Performance Universal In-Circuit Emulator with MPLAB IDE

The MPLAB ICE universal in-circuit emulator is intended to provide the product development engineer with a complete microcontroller design tool set for PICmicro microcontrollers (MCUs). Software control of the MPLAB ICE in-circuit emulator is provided by the MPLAB Integrated Development Environment (IDE), which allows editing, building, downloading and source debugging from a single environment.

The MPLAB ICE 2000 is a full-featured emulator system with enhanced trace, trigger and data monitoring features. Interchangeable processor modules allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the MPLAB ICE in-circuit emulator allows expansion to support new PICmicro microcontrollers.

The MPLAB ICE in-circuit emulator system has been designed as a real-time emulation system, with advanced features that are generally found on more expensive development tools. The PC platform and Microsoft® Windows environment were chosen to best make these features available to you, the end user.

13.7 ICEPIC In-Circuit Emulator

The ICEPIC low cost, in-circuit emulator is a solution for the Microchip Technology PIC16C5X, PIC16C6X, PIC16C7X and PIC16CXXX families of 8-bit One-Time-Programmable (OTP) microcontrollers. The modular system can support different subsets of PIC16C5X or PIC16CXXX products through the use of interchangeable personality modules, or daughter boards. The emulator is capable of emulating without target application circuitry being present.

FIGURE 14-1: PIC16F872 VOLTAGE-FREQUENCY GRAPH

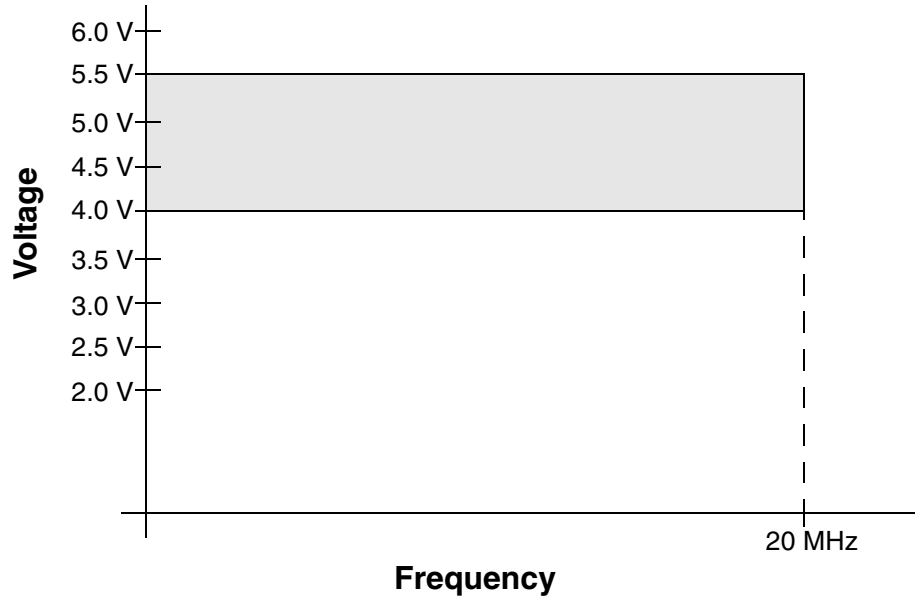
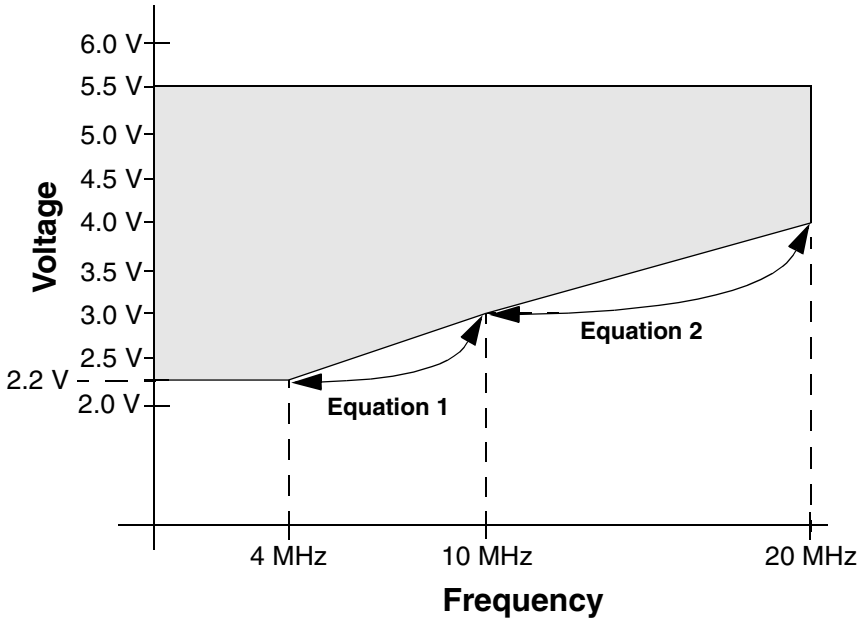


FIGURE 14-2: PIC16LF872 VOLTAGE-FREQUENCY GRAPH



Equation 1: FMAX = (6.0 MHz/V) (VDDAPPMIN - 2.0V) + 4 MHz; VDDAPPMIN = 2.2V - 3.0V

Equation 2: FMAX = (10.0 MHz/V) (VDDAPPMIN - 3.0V) + 10 MHz; VDDAPPMIN = 3.0V - 4.0V

Note 1: VDDAPPMIN is the minimum voltage of the PICmicro® device in the application.

2: FMAX has a maximum frequency of 10 MHz.

FIGURE 14-4: EXTERNAL CLOCK TIMING

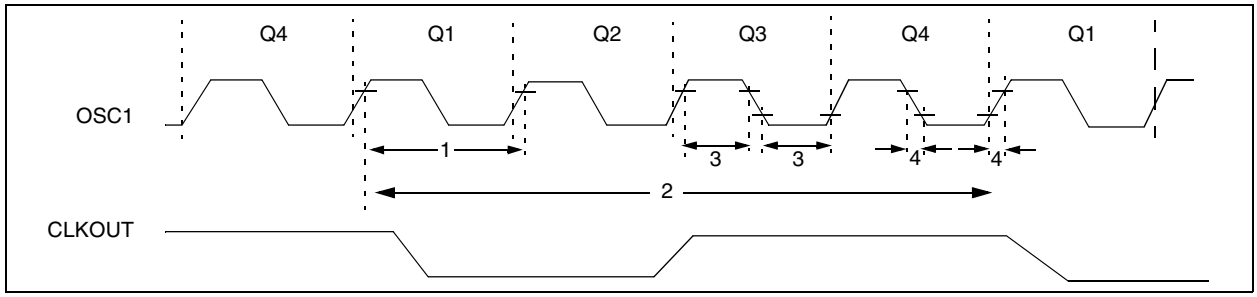


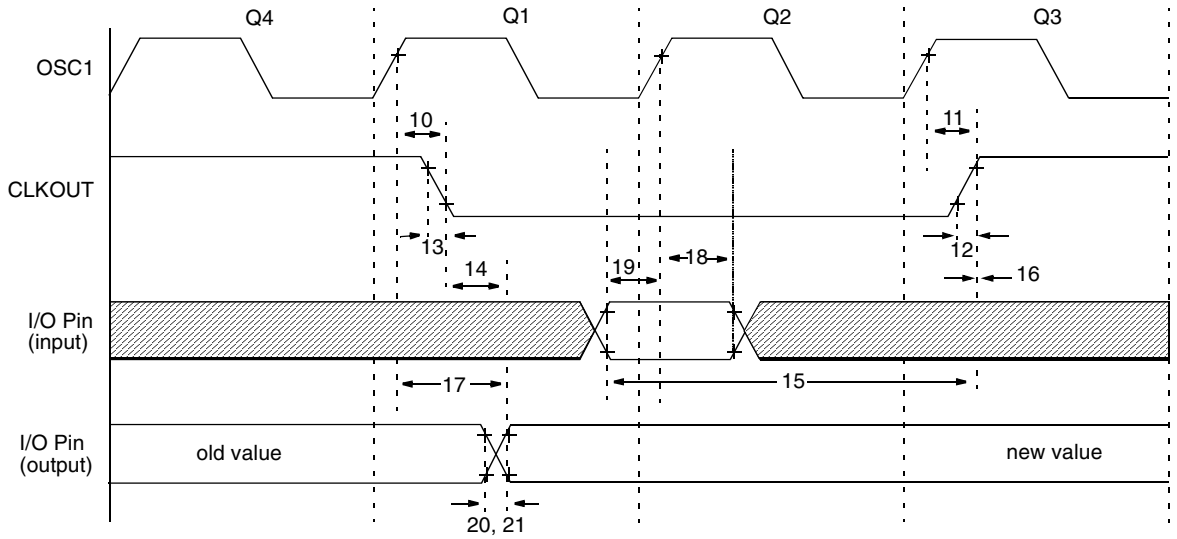
TABLE 14-1: EXTERNAL CLOCK TIMING REQUIREMENTS

Parameter No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
	FOSC	External CLKIN Frequency **(Note 1)**	DC	—	4	MHz	XT and RC osc mode
			DC	—	4	MHz	HS osc mode (-04)
			DC	—	20	MHz	HS osc mode (-20)
			DC	—	200	kHz	LP osc mode
		Oscillator Frequency **(Note 1)**	DC	—	4	MHz	RC osc mode
			0.1	—	4	MHz	XT osc mode
			4	—	20	MHz	HS osc mode
			5	—	200	kHz	LP osc mode
1	TOSC	External CLKIN Period **(Note 1)**	250	—	—	ns	XT and RC osc mode
			250	—	—	ns	HS osc mode (-04)
			50	—	—	ns	HS osc mode (-20)
			5	—	—	µs	LP osc mode
		Oscillator Period **(Note 1)**	250	—	—	ns	RC osc mode
			250	—	10,000	ns	XT osc mode
			250	—	250	ns	HS osc mode (-04)
			50	—	250	ns	HS osc mode (-20)
			5	—	—	µs	LP osc mode
2	TCY	Instruction Cycle Time **(Note 1)**	200	TCY	DC	ns	TCY = 4/FOSC
3	TosL, TosH	External Clock in (OSC1) High or Low Time	100	—	—	ns	XT oscillator
			2.5	—	—	µs	LP oscillator
			15	—	—	ns	HS oscillator
4	TosR, TosF	External Clock in (OSC1) Rise or Fall Time	—	—	25	ns	XT oscillator
			—	—	50	ns	LP oscillator
			—	—	15	ns	HS oscillator

† Data in "Typ" column is at 5V, 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 the OSC1/CLKIN pin. When an external clock input is used, the "Max." cycle time limit is "DC" (no clock) for all devices.

FIGURE 14-5: CLKOUT AND I/O TIMING



Note: Refer to Figure 14-3 for load conditions.

TABLE 14-2: CLKOUT AND I/O TIMING REQUIREMENTS

Param No.	Symbol	Characteristic		Min	Typ†	Max	Units	Conditions
10*	TosH2ckL	OSC1↑ to CLKOUT↓		—	75	200	ns	**(Note 1)**
11*	TosH2ckH	OSC1↑ to CLKOUT↑		—	75	200	ns	**(Note 1)**
12*	TckR	CLKOUT rise time		—	35	100	ns	**(Note 1)**
13*	TckF	CLKOUT fall time		—	35	100	ns	**(Note 1)**
14*	TckL2ioV	CLKOUT↓ to Port out valid		—	—	0.5TCY + 20	ns	**(Note 1)**
15*	TioV2ckH	Port in valid before CLKOUT↑		TOSC + 200	—	—	ns	**(Note 1)**
16*	TckH2ioI	Port in hold after CLKOUT↑		0	—	—	ns	**(Note 1)**
17*	TosH2ioV	OSC1↑ (Q1 cycle) to Port out valid		—	100	255	ns	
18*	TosH2ioI	OSC1↑ (Q2 cycle) to Port input invalid (I/O in hold time)	Standard (**F**)	100	—	—	ns	
			Extended (**LF**)	200	—	—	ns	
19*	TioV2osH	Port input valid to OSC1↑ (I/O in setup time)		0	—	—	ns	
20*	TIOR	Port output rise time	Standard (**F**)	—	10	40	ns	
			Extended (**LF**)	—	—	145	ns	
21*	TIOF	Port output fall time	Standard (**F**)	—	10	40	ns	
			Extended (**LF**)	—	—	145	ns	
22††*	TINP	INT pin high or low time		TCY	—	—	ns	
23††*	TRBP	RB7:RB4 change INT high or low time		TCY	—	—	ns	

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

†† These parameters are asynchronous events not related to any internal clock edges.

Note 1: Measurements are taken in RC mode, where CLKOUT output is 4 x TOSC.

FIGURE 15-11: ΔIBOR vs. VDD OVER TEMPERATURE

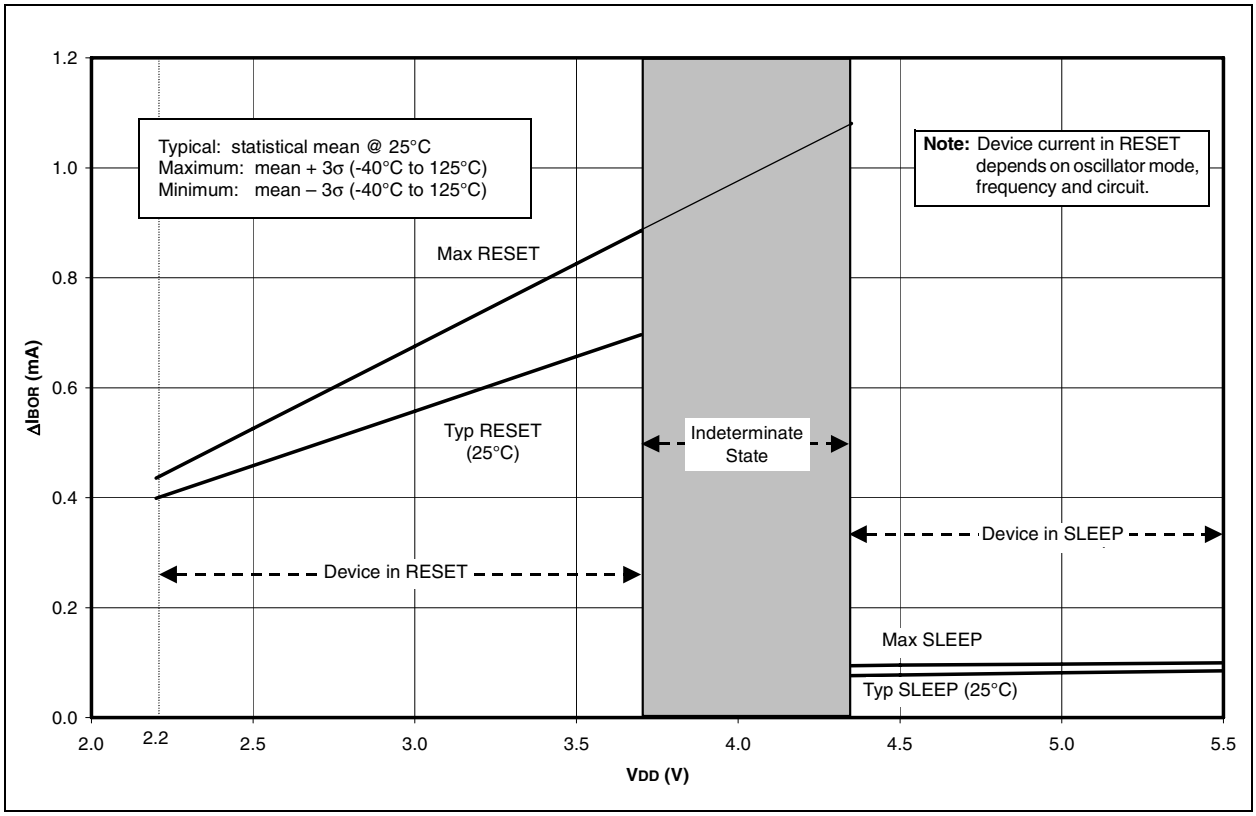


FIGURE 15-12: TYPICAL AND MAXIMUM ΔITMR1 vs. VDD OVER TEMPERATURE (-10°C TO +70°C, TIMER1 WITH OSCILLATOR, XTAL=32 kHZ, C1 AND C2=50 pF)

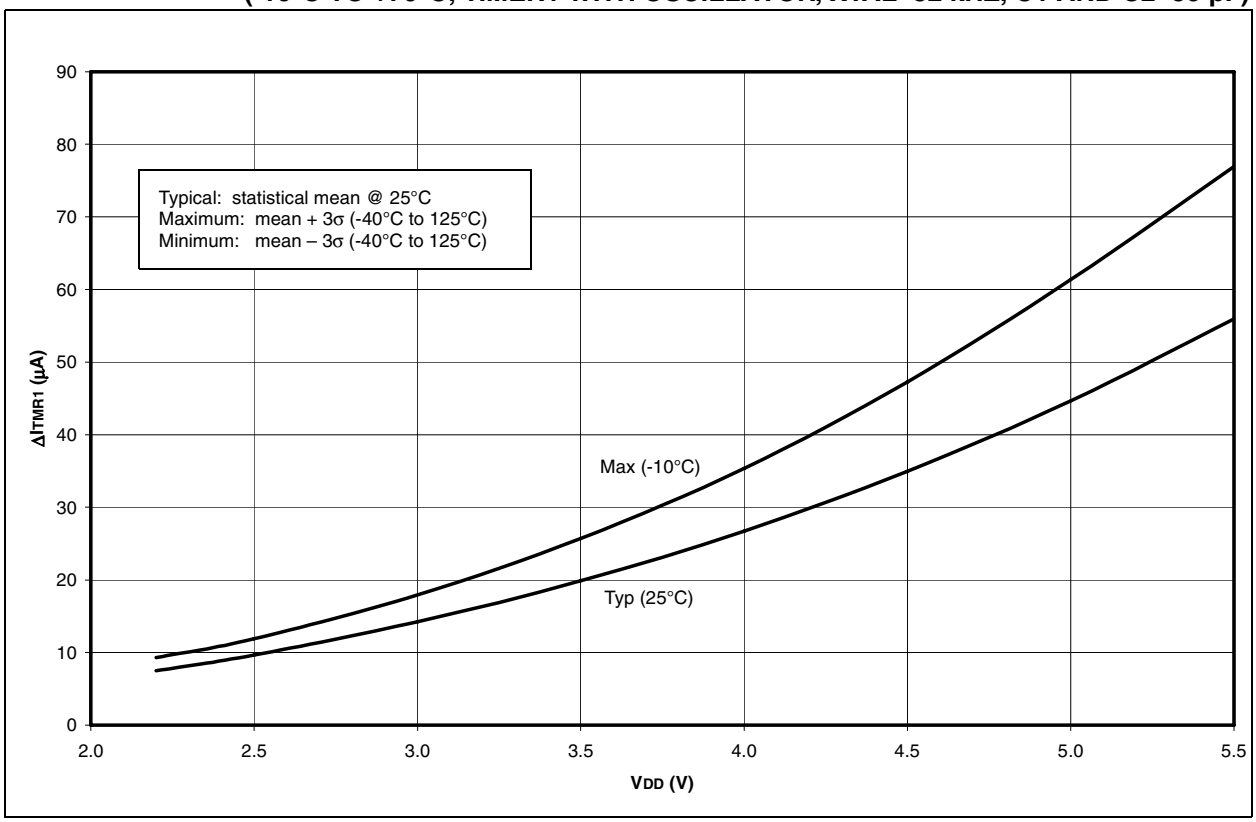


FIGURE 15-21: MINIMUM AND MAXIMUM VIN vs. VDD (ST INPUT, -40°C TO +125°C)

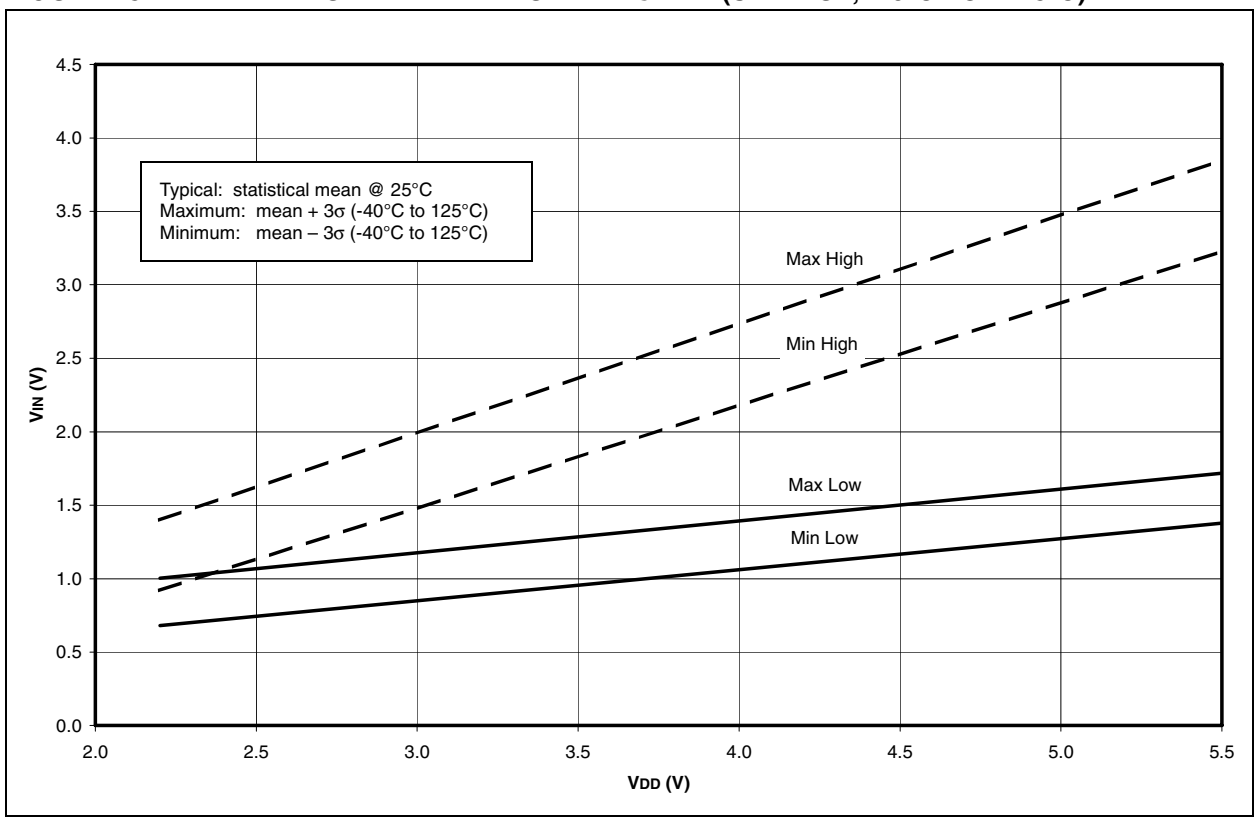


FIGURE 15-22: MINIMUM AND MAXIMUM VIN vs. VDD (I^2C INPUT, -40°C TO +125°C)