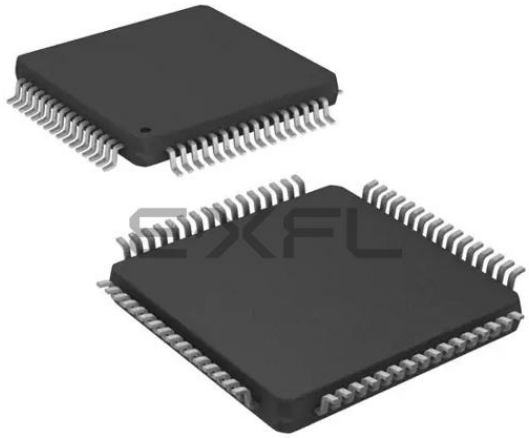


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

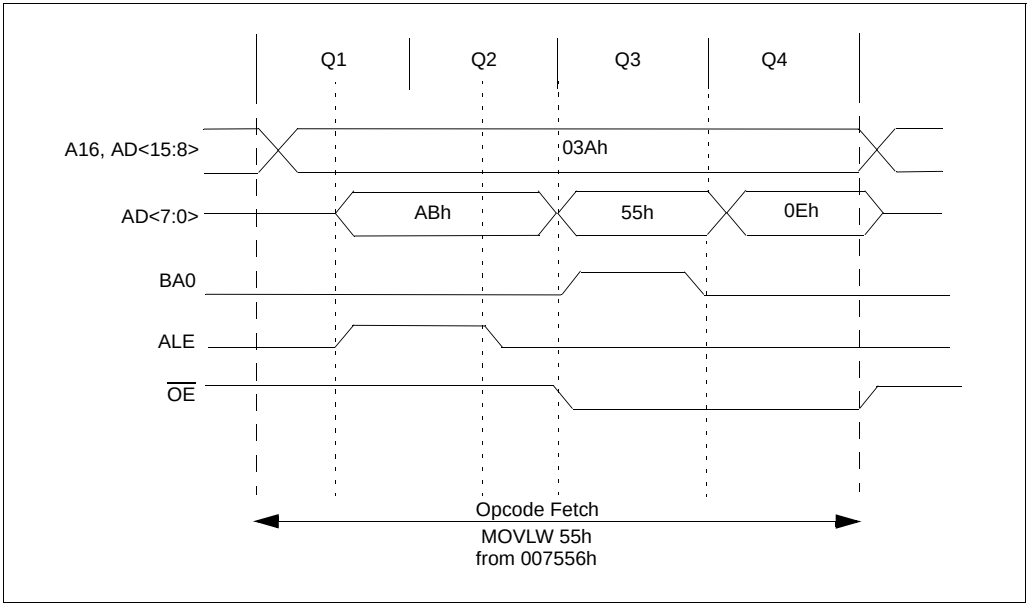
Details	
Product Status	Obsolete
Core Processor	PIC
Core Size	8-Bit
Speed	25MHz
Connectivity	EBI/EMI, I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, LVD, POR, PWM, WDT
Number of I/O	26
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	-
RAM Size	1.5K x 8
Voltage - Supply (Vcc/Vdd)	2V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	64-TQFP
Supplier Device Package	64-TQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18lc601t-i-pt

TABLE 1-2: PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number				Pin Type	Buffer Type	Description
	PIC18C601		PIC18C801				
	TQFP	PLCC	TQFP	PLCC			
							PORTA is a bi-directional I/O port.
RA0/AN0	24	34	30	42			
RA0					I/O	TTL	Digital I/O.
AN0					I	Analog	Analog input 0.
RA1/AN1	23	33	29	41			
RA1					I/O	TTL	Digital I/O.
AN1					I	Analog	Analog input 1.
RA2/AN2/VREF-	22	32	28	40			
RA2					I/O	TTL	Digital I/O.
AN2					I	Analog	Analog input 2.
VREF-					I	Analog	A/D reference voltage (Low) input.
RA3/AN3/VREF+	21	31	27	39			
RA3					I/O	TTL	Digital I/O.
AN3					I	Analog	Analog input 3.
VREF+					I	Analog	A/D reference voltage (High) input.
RA4/T0CKI	28	39	34	47			
RA4					I/O	ST/OD	Digital I/O – Open drain when configured as output.
T0CKI					I	ST	Timer0 external clock input.
RA5/AN4/$\overline{SS}$/LVDIN	27	38	33	46			
RA5					I/O	TTL	Digital I/O.
AN4					I	Analog	Analog input 4.
$\overline{SS}$					I	ST	SPI slave select input.
LVDIN					I	Analog	Low voltage detect input.

Legend: TTL = TTL compatible input
ST = Schmitt Trigger input with CMOS levels
I = Input
P = Power
CMOS = CMOS compatible input or output
Analog = Analog input
O = Output
OD = Open Drain (no P diode to VDD)

FIGURE 5-2: 8-BIT MULTIPLEXED MODE TIMING



5.2.2 8-BIT DE-MULTIPLEXED MODE

The 8-bit De-Multiplexed mode applies only to the PIC18C801. Data and address lines are available separately. External components are not necessary in this mode.

For 8-bit De-Multiplexed mode on the PIC18C801, the instructions are fetched as two 8-bit bytes on a dedicated data bus (PORTJ). The address will be presented for the entire duration of the fetch cycle on a separate address bus. The two instruction bytes are sequentially fetched within one instruction cycle (TCY). Therefore, the designer must choose external memory devices according to timing calculations, based on 1/2 TCY (2 times instruction rate). For proper memory speed selection, setup and hold times must be considered.

The Address Latch Enable (ALE) pin is left unconnected, since glue logic is not necessary. The $\overline{OE}$ output enable signal will enable one byte of program memory for a portion of the instruction cycle, then BA0 will change and the second byte will be enabled to form the 16-bit instruction word. The least significant bit of the address, BA0, must be connected to the memory devices in this mode. Figure 5-3 shows an example of 8-bit De-Multiplexed mode on the PIC18C801. The control signals used in 8-bit De-Multiplexed mode are outlined in Register 5-2. Register 5-4 describes 8-bit De-Multiplexed mode timing.

6.3.3 EXTERNAL TABLE WRITE IN 16-BIT WORD WRITE MODE

This mode allows Table Writes to any type of word-wide external memories.

This method makes a distinction between TBLWT cycles to even or odd addresses.

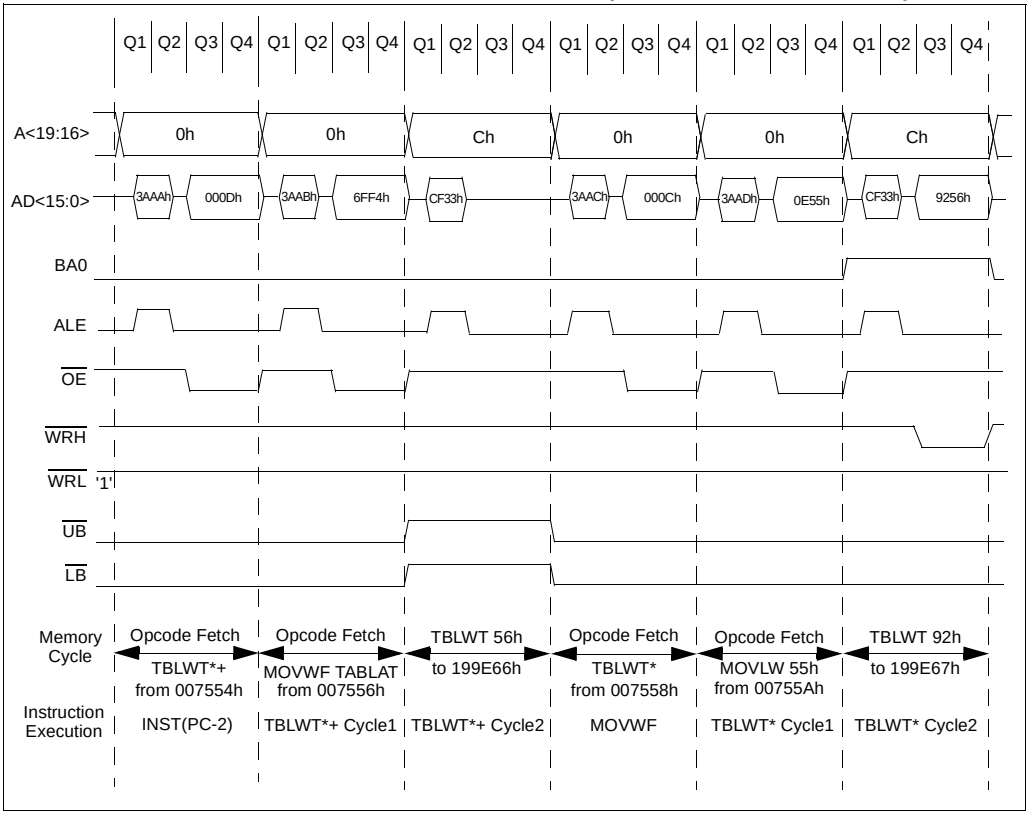
During a TBLWT cycle to an even address, where TBLPTR<0> = 0, the TABLAT data is transferred to a holding latch and the external address data bus is tri-stated for the data portion of the bus cycle. No write signals are activated.

During a TBLWT cycle to an odd address, where TBLPTR<0> = 1, the TABLAT data is presented on the upper byte of the AD<15:0> bus. The contents of the holding latch are presented on the lower byte of the AD<15:0> bus. The $\overline{WRH}$ line is strobed for each write cycle and the $\overline{WRL}$ line is unused. The BA0 line indicates the LSb of TBLPTR, but it is unnecessary. The $\overline{UB}$ and $\overline{LB}$ lines are active to select both bytes.

The obvious limitation to this method is that the TBLWT must be done in pairs on a specific word boundary to correctly write a word location.

Figure 6-9 shows the timing associated with this mode.

FIGURE 6-9: TBLWT EXTERNAL INTERFACE TIMING (16-BIT WORD WRITE MODE)



8.1 Control Registers

This section contains the control and status registers.

8.1.1 INTCON REGISTERS

The INTCON Registers are readable and writable registers, which contain various enable, priority, and flag bits.

REGISTER 8-1: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all high priority interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding enable bit, or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows software polling.

REGISTER 8-3: INTCON3 REGISTER

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
1 = High priority
0 = Low priority

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

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '0'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: Interrupt flag bits get set when an interrupt condition occurs, regardless of the state of its corresponding enable bit, or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows software polling.

TABLE 9-5: PORTC FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T13CKI	bit0	ST	Input/output port pin or Timer1 oscillator output or Timer1/Timer3 clock input.
RC1/T1OSI	bit1	ST	Input/output port pin, Timer1 oscillator input.
RC2/CCP1	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	bit3	ST	Input/output port pin or synchronous serial clock for SPI/I^2C.
RC4/SDI/SDA	bit4	ST	Input/output port pin or 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.
RC6/TX/CK	bit6	ST	Input/output port pin, Addressable USART Asynchronous Transmit, or Addressable USART Synchronous Clock.
RC7/RX/DT	bit7	ST	Input/output port pin, Addressable USART Asynchronous Receive, or Addressable USART Synchronous Data.

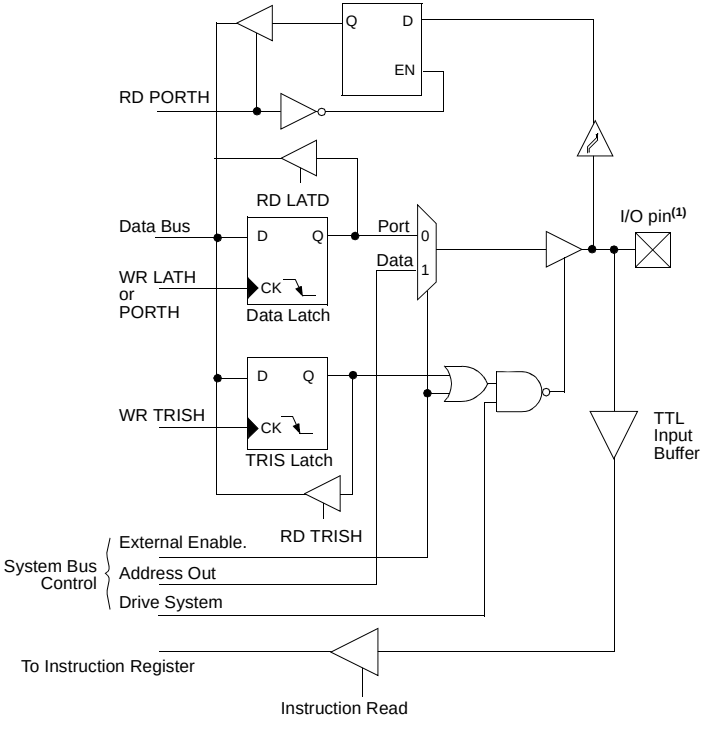
Legend: ST = Schmitt Trigger input

TABLE 9-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

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
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
LATC	LATC Data Output Register								xxxx xxxx	uuuu uuuu
TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged

FIGURE 9-18: RH3:RH0 PINS BLOCK DIAGRAM IN SYSTEM BUS MODE



Note 1: I/O pins have diode protection to VDD and VSS.

13.0 TIMER3 MODULE

The Timer3 module timer/counter has the following features:

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

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

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

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

Note:	Timer3 is disabled on POR.

REGISTER 13-1: T3CON REGISTER

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

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

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

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

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

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

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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

15.3 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) - RC5/SDO
- Serial Data In (SDI) - RC4/SDI/SDA
- Serial Clock (SCK) - RC3/SCK/SCL/LVOIN

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

- Slave Select ($\overline{SS}$) - RA5/$\overline{SS}$/AN4

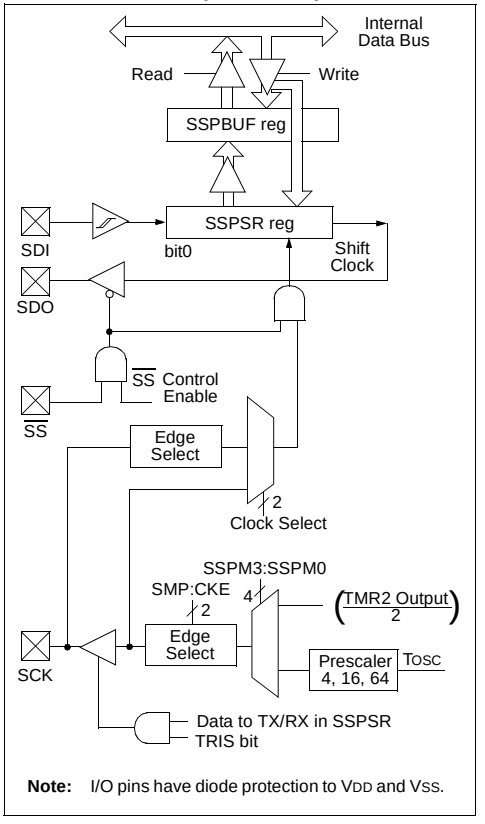
15.3.1 OPERATION

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits SSPCON1<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 15-1 shows the block diagram of the MSSP module, when in SPI mode.

FIGURE 15-1: MSSP BLOCK DIAGRAM (SPI MODE)



Note: I/O pins have diode protection to VDD and VSS.

REGISTER 17-3: ADCON2 REGISTER

R/W-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	—	ADCS2	ADCS1	ADCS0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified
0 = Left justified

bit 6-3 **Unimplemented:** Read as '0'

bit 2-0 **ADCS2:ADCS0:** A/D Conversion Clock Select bits
000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC (clock derived from an internal RC oscillator = 1 MHz max)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC (clock derived from an internal RC oscillator = 1 MHz max)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

The analog reference voltage is software selectable to either the device's positive and negative supply voltage (V_{DD} and V_{SS}), or the voltage level on the RA3/AN3/V_{REF}+ pin and RA2/AN2/V_{REF}-.

The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D conversion clock must be derived from the A/D's internal RC oscillator.

The output of the sample and hold is the input into the converter, which generates the result via successive approximation.

A device RESET forces all registers to their RESET state. This forces the A/D module to be turned off and any conversion is aborted.

Each port pin associated with the A/D converter can be configured as an analog input (RA3 can also be a voltage reference), or as a digital I/O.

The ADRESH and ADRESL registers contain the result of the A/D conversion. When the A/D conversion is complete, the result is loaded into the ADRESH/ADRESL registers, the GO/$\overline{\text{DONE}}$ bit (ADCON0 register) is cleared, and A/D interrupt flag bit ADIF is set. The block diagram of the A/D module is shown in Figure 17-1.

18.2 Operation

Depending on the power source for the device voltage, the voltage normally decreases relatively slowly. This means that the LVD module does not need to be constantly operating. To decrease current consumption, the LVD circuitry only needs to be enabled for short periods, where the voltage is checked. After doing the check, the LVD module may be disabled.

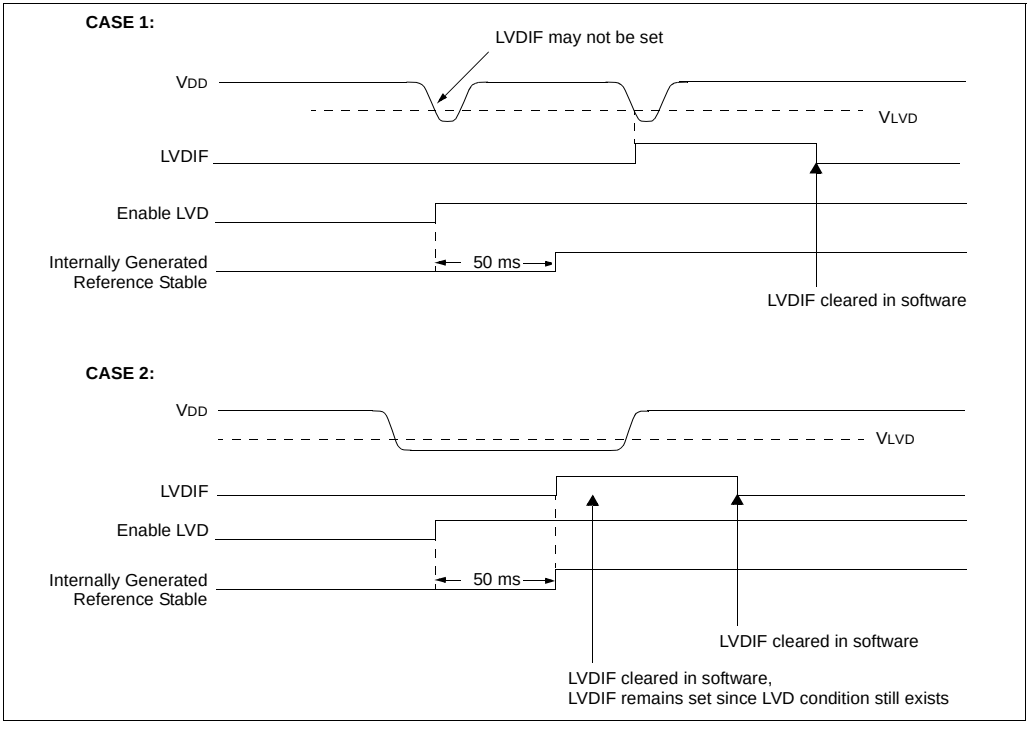
Each time that the LVD module is enabled, the circuitry requires some time to stabilize. After the circuitry has stabilized, all status flags may be cleared. The module will then indicate the proper state of the system.

The following steps are needed to setup the LVD module:

1. Write the value to the LVDL3:LVDL0 bits (LVDCON register), which selects the desired LVD trip point.
2. Ensure that LVD interrupts are disabled (the LVDIE bit is cleared or the GIE bit is cleared).
3. Enable the LVD module (set the LVDEN bit in the LVDCON register).
4. Wait for the LVD module to stabilize (the IRVST bit to become set).
5. Clear the LVD interrupt flag, which may have falsely become set, until the LVD module has stabilized (clear the LVDIF bit).
6. Enable the LVD interrupt (set the LVDIE and the GIE bits).

Figure 18-3 shows typical waveforms that the LVD module may be used to detect.

FIGURE 18-3: LOW VOLTAGE DETECT WAVEFORMS



BTG **Bit Toggle f**

Syntax: [*label*] BTG f, b [,a]

Operands: $0 \le f \le 255$
$0 \le b < 7$
$a \in [0,1]$

Operation: $(\overline{f<b>}) \rightarrow f<b>$

Status Affected: None

Encoding:

0111	bbba	ffff	ffff

Description: Bit 'b' in data memory location 'f' is inverted. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `BTG PORTC, 4`

Before Instruction:
PORTC = 0111 0101 [75h]

After Instruction:
PORTC = 0110 0101 [65h]

BOV **Branch if Overflow**

Syntax: [*label*] BOV n

Operands: $-128 \le n \le 127$

Operation: if overflow bit is '1'
$(PC) + 2 + 2n \rightarrow PC$

Status Affected: None

Encoding:

1110	0100	nnnn	nnnn

Description: If the Overflow bit is '1', then the program will branch.
The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:
If Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BOV Jump`

Before Instruction
PC = address (HERE)

After Instruction
If Overflow = 1;
PC = address (Jump)
If Overflow = 0;
PC = address (HERE+2)

CPFSGT — **Compare f with WREG, skip if f > WREG**

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

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

Operation: (f) – (WREG),
skip if (f) > (WREG)
(unsigned comparison)

Status Affected: None

Encoding:

0110	010a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of the WREG by performing an unsigned subtraction.

If the contents of 'f' are greater than the contents of **,** then the fetched instruction is discarded and a NOP is executed instead, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE      CPFSGT REG
NGREATER  :
GREATER   :
```
Before Instruction
PC = Address (HERE)
WREG = ?
After Instruction
If REG > WREG;
PC = Address (GREATER)
If REG ≤ WREG;
PC = Address (NGREATER)

CPFSLT — **Compare f with WREG, skip if f < WREG**

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

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

Operation: (f) – (WREG),
skip if (f) < (WREG)
(unsigned comparison)

Status Affected: None

Encoding:

0110	000a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of WREG by performing an unsigned subtraction.

If the contents of 'f' are less than the contents of WREG, then the fetched instruction is discarded and a NOP is executed instead, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:
```
HERE    CPFSLT REG
NLESS   :
LESS    :
```
Before Instruction
PC = Address (HERE)
WREG = ?
After Instruction
If REG < WREG;
PC = Address (LESS)
If REG ≥ WREG;
PC = Address (NLESS)

DECFSZ **Decrement f, skip if 0**

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

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

Operation: (f) – 1 → dest,
skip if result = 0

Status Affected: None

Encoding:

0010	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default).
If the result is 0, the next instruction, which is already fetched, is discarded, and a NOP is executed instead, making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

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

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:

```
HERE    DECFSZ  CNT
        GOTO    LOOP
CONTINUE
```

Before Instruction
PC = Address (HERE)

After Instruction
CNT = CNT - 1
If CNT = 0;
PC = Address (CONTINUE)
If CNT ≠ 0;
PC = Address (HERE+2)

DCFSNZ **Decrement f, skip if not 0**

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

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

Operation: (f) – 1 → dest,
skip if result ≠ 0

Status Affected: None

Encoding:

0100	11da	ffff	ffff

Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default).
If the result is not 0, the next instruction, which is already fetched, is discarded, and a NOP is executed instead, making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1(2)
Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

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

If skip:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2-word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example:

```
HERE    DCFSNZ  TEMP
ZERO    :
NZERO   :
```

Before Instruction
TEMP = ?

After Instruction
TEMP = TEMP - 1,
If TEMP = 0;
PC = Address (ZERO)
If TEMP ≠ 0;
PC = Address (NZERO)

LFSR **Load FSR**

Syntax: [*label*] LFSR f,k

Operands: $0 \le f \le 2$
$0 \le k \le 4095$

Operation: $k \rightarrow FSRf$

Status Affected: None

Encoding:

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

Description: The 12-bit literal 'k' is loaded into the file select register pointed to by 'f'.

Words: 2

Cycles: 2

Q Cycle Activity:

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

Example: `LFSR FSR2, 3ABh`

After Instruction
FSR2H = 03h
FSR2L = 0ABh

MOVF **Move f**

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

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

Operation: $f \rightarrow dest$

Status Affected: N,Z

Encoding:

0101	00da	ffff	ffff

Description: The contents of register 'f' is moved to a destination dependent upon the status of 'd'. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed back in register 'f' (default). Location 'f' can be anywhere in the 256 byte Bank. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `MOVF REG, W`

Before Instruction
REG = 22h
WREG = 0FFh
N = ?
Z = ?

After Instruction
REG = 22h
WREG = 22h
N = 0
Z = 0

SUBWFB **Subtract WREG from f with Borrow**

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

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

Operation: (f) – (WREG) – ($\overline{C}$) → dest

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

Encoding:

0101	10da	ffff	ffff

Description: Subtract WREG and the carry flag (borrow) from register 'f' (2's complement method). If 'd' is 0, the result is stored in WREG. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

SUBWFB (Cont.)

Example 1: `SUBWFB REG`

Before Instruction
REG = 19h (0001 1001)
WREG = 0Dh (0000 1101)
C = 1

After Instruction
REG = 0Ch (0000 1011)
WREG = 0Dh (0000 1101)
C = 1
Z = 0
N = 0 ; result is positive

Example 2: `SUBWFB REG, W`

Before Instruction
REG = 1Bh (0001 1011)
WREG = 1Ah (0001 1010)
C = 0

After Instruction
REG = 1Bh (0001 1011)
WREG = 00h
C = 1
Z = 1 ; result is zero
N = 0

Example 3: `SUBWFB REG`

Before Instruction
REG = 03h (0000 0011)
WREG = 0Eh (0000 1101)
C = 1

After Instruction
REG = 0F5h (1111 0100) [2's comp]
WREG = 0Eh (0000 1101)
C = 0
Z = 0
N = 1 ; result is negative

SWAPF	Swap nibbles in f
Syntax:	[*label*] SWAPF f [,d [,a]]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f<3:0>) → dest<7:4>, (f<7:4>) → dest<3:0>
Status Affected:	None

Encoding:

0011	10da	ffff	ffff

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in WREG. If 'd' is 1, the result is placed in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, the Bank will be selected as per the BSR value.

Words: 1

Cycles: 1

Q Cycle Activity:

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

Example: `SWAPF REG`

Before Instruction
REG = 53h

After Instruction
REG = 35h

21.4 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.

21.5 MPLAB SIM Software Simulator

The MPLAB SIM software simulator allows code development in a PC-hosted environment by simulating the PIC 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.

21.6 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 PIC 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.

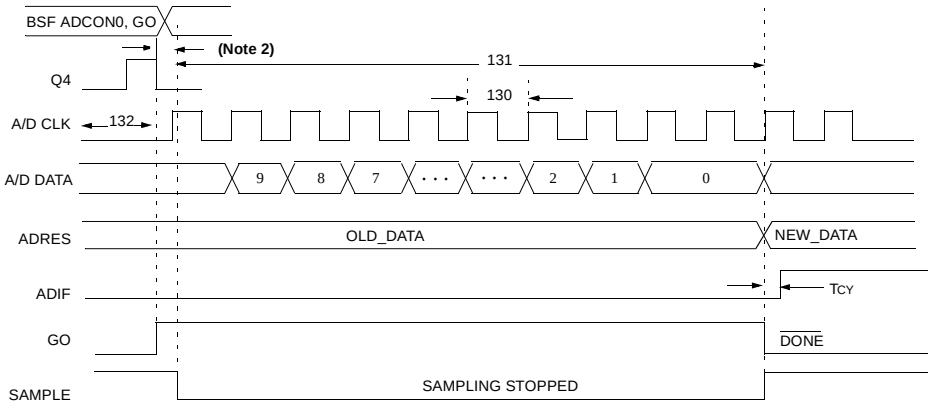
The MPLAB ICE 2000 is a full-featured emulator system with enhanced trace, trigger and data monitoring features. Interchangeable processor modules allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the MPLAB ICE in-circuit emulator allows expansion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

21.7 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 22-23: A/D CONVERSION TIMING



Note 1: If the A/D clock source is selected as RC, a time of TCY is added before the A/D clock starts.This allows the SLEEP instruction to be executed.

2: This is a minimal RC delay (typically 100 ns), which also disconnects the holding capacitor from the analog input.

TABLE 22-23: A/D CONVERSION REQUIREMENTS

Param No.	Symbol	Characteristic		Min	Max	Units	Conditions
130	TAD	A/D clock period	PIC18**C**601/801	1.6	20(5)	µs	TOSC based, VREF ≥ 3.0V
			PIC18**LC**601/801	3.0	20(5)	µs	TOSC based, VREF full range
			PIC18**C**601/801	2.0	6.0	µs	A/D RC mode
			PIC18**LC**601/801	3.0	9.0	µs	A/D RC mode
131	TCNV	Conversion time (not including acquisition time)(1)		11	12	TAD	
132	TACQ	Acquisition time(3)		15 10	— —	µs µs	-40°C ≤ Temp ≤ 125°C 0°C ≤ Temp ≤ 125°C
135	TSWC	Switching time from convert → sample		—	(Note 4)		
136	TAMP	Amplifier settling time(2)		1	—	µs	This may be used if the "new" input voltage has not changed by more than 1 LSb (i.e., 5 mV @ 5.12V) from the last sampled voltage (as stated on CHOLD).

Note 1: ADRES register may be read on the following TCY cycle.

2: See Section 17.0 for minimum conditions, when input voltage has changed more than 1 LSb.

3: The time for the holding capacitor to acquire the "New" input voltage, when the voltage changes full scale after the conversion (AVDD to AVSS, or AVSS to AVDD). The source impedance (*RS*) on the input channels is 50Ω.

4: On the next Q4 cycle of the device clock.

5: The time of the A/D clock period is dependent on the device frequency and the TAD clock divider.

NOTES:

