PIC Microcontrollers

PIC stands for Peripheral Interface Controller coined by Microchip Technology to identify its single-chip microcontrollers. These devices have been phenomenally successful in 8-bit microcontroller market. The main reason is that Microchip Technology has constantly upgraded the device architecture and added needed peripherals to the microcontroller to 'suit customers’ requirements. The development tools such as assembler and simulator are freely available on the internet at www.microchip.com

Low-end Architectures

Microchip PIC microcontrollers are available in various types. When PIC – Micro®MCU first became available from General Instruments in early 1980’s, the microcontroller consisted of a very simple processor executing 12-bit wide instructions with basic I/O functions. These devices are known as low-end architectures.

Some of the low-end device past numbers are 12C5XX, 16C5X, and 16C505

Mid-range Architectures

Mid-range Architectures are built by upgrading low-end architecture with more number of peripherals, more numbers of register and more data memory. Some of the mid-range devices are 16C6X, 16C7X, 16F87X

↑Program memory type
C = EPROM
F = Flash
RC = Mask ROM

Popularity of PIC microcontrollers is due to the following factors:

1. Speed: Harvard Architecture, RISC Architecture
   1 instruction Cycle = 4 clock cycles.
   For 20 MHz clock, most of the instructions are executed in 0.2\(\mu\)s or five instructions per microsecond.

2. Instruction Set Simplicity:
   The instruction set consists of just 35 instructions (as opposed to 111 instructions for 8051)

3. Power on reset
   Power-out reset
   Watch-dog timer
   Oscillator Options
   - low-power Crystal
   - Mid-range Crystal
   - High-range Crystal
   - RC Oscillator

4. Programmable timer options on chip ADC
5. Up to 12 independent interrupt sources

6. Powerful output pin control
   25mA (max.) current sourcing capability.

7. EPROM/OTP/ROM/Flash memory options.

8. Free assembler and simulator support from microchip at http://www.microchip.com

CPU Architecture and Instruction Set

Pipelining of instruction fetch successive addressing
Introduction of extra cycle for a jump/goto instruction

Fetch of $n^{th}$ instruction from address $n$

Execution of $n^{th}$ instruction

Fetch of jump instruction from address $n+1$

Change the program count to new address

Fetch of $(n+2)^{th}$ instruction

Ignore $(n+2)^{th}$ instruction

Fetch instruction from new address
Register File Structure and Addressing Modes

Register file → locations that an instruction can access via an address. Register file consists of two components.

1. General purpose register file (same as RAM)
2. Special purpose register file

RPO bit in the Status register detects the bank. 7 bit of direct address TRPO determines the absolute address of the register.
Indirect addressing mode
FSR contains the 8-bit address of the data/register.

CPU Registers

W, the working register, is used by many instructions as the source of an operand. It may also serve as the destination for the result of the instructions execution. It works as the accumulator.

W working register

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RPO</td>
<td>NOT_TO</td>
<td>NOT_PD</td>
<td>Z</td>
<td>DC</td>
<td>C</td>
</tr>
</tbody>
</table>

STATUS
(address 03H,83H)
C = Carry bit
DC = Digit Carry (same as AC, Auxiliary Carry)
Z = Zero bit
NOT-TO, NOT-PD → Used in conjunction with PIC’s sleep mode
RPO → register bank select bit used in conjunction with the direct addressing mode.
FSR
(address 04H, 84H)
Indirect data memory address points.
FSR is the pointer used for indirect addressing.
The program is supported by an eight-level stack. When an interrupt occurs, the program counter is automatically pushed on to the stack. Since PIC microcontrollers programs are normally designed for handling one interrupt at a time, further

Basic Architecture of PIC Microcontroller

W → Temporary holding register, often called as an accumulator, cannot be accessed directly. Instead, contents must be moved to other registers that can be accessed directly.
Bank Addressing

<table>
<thead>
<tr>
<th></th>
<th>Bank 0</th>
<th>Bank 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>INDF</td>
<td>INDF</td>
<td>80</td>
</tr>
<tr>
<td>01</td>
<td>TMRO</td>
<td>OPTION</td>
<td>81</td>
</tr>
<tr>
<td>02</td>
<td>PCL</td>
<td>PCL</td>
<td>82</td>
</tr>
<tr>
<td>03</td>
<td>STATUS</td>
<td>STATUS</td>
<td>83</td>
</tr>
<tr>
<td>04</td>
<td>FSR</td>
<td>FSR</td>
<td>84</td>
</tr>
<tr>
<td>05</td>
<td>PORTA</td>
<td>TRISA</td>
<td>85</td>
</tr>
<tr>
<td>06</td>
<td>PORTB</td>
<td>TRISB</td>
<td>86</td>
</tr>
<tr>
<td>07</td>
<td>PORTC</td>
<td>TRISC</td>
<td>87</td>
</tr>
<tr>
<td>08</td>
<td>PORTD</td>
<td>TRISD</td>
<td>88</td>
</tr>
<tr>
<td>09</td>
<td>PORTE</td>
<td>TRISE</td>
<td>89</td>
</tr>
<tr>
<td>0A</td>
<td>PCLATH</td>
<td>PCLATH</td>
<td>8A</td>
</tr>
<tr>
<td>0B</td>
<td>INTCON</td>
<td>INTCON</td>
<td>8B</td>
</tr>
<tr>
<td>0C</td>
<td></td>
<td></td>
<td>8C</td>
</tr>
<tr>
<td>0D</td>
<td></td>
<td></td>
<td>8D</td>
</tr>
<tr>
<td>0E</td>
<td></td>
<td></td>
<td>8E</td>
</tr>
<tr>
<td>0F</td>
<td></td>
<td></td>
<td>8F</td>
</tr>
<tr>
<td>1F</td>
<td></td>
<td></td>
<td>9F</td>
</tr>
<tr>
<td>07F</td>
<td></td>
<td></td>
<td>0FF</td>
</tr>
</tbody>
</table>

TRIS bit set → Post bit in I mode
Reset → Post bit in 0 mode.

EX: To set PORT B bit 0 as an output and loaded with a 1, PIC micro® MCU code would execute as:
Port B. Bit = 1
STATUS. RPO = 1
TRIS B. Bit 0 = 0
STATUS. RPO = 0

PIC 16C74A

- Program Memory (EPROM)×14: 4k
- Data Memory (Bytes)×8: 192
- I/O Pins: 33
- Parallel slave port: Yes
- A/D channel: 8
- Serial Comm: SPI/I²C, USART
- Interrupt sources: 12
Memory Organization

The PIC 16C7X family has a 13-bit program counter capable of addressing 8k×14 program memory. PIC16C74A has 4k×14 program memory. For those devices with less than 8k program memory, accessing a location above the physically implemented address will cause a wraparound.

Program memory map and stack

16C74A has 4k program memory. The address range is 0000H - 0FFFH. The reset vector is 0000H and the interrupt vector is 0004H.
LED Driver Example

PIC 16C74A has five ports. Each port is a bidirectional I/O port. In addition, they have the following alternative functions.

<table>
<thead>
<tr>
<th>Port</th>
<th>Alternative uses of I/O pins</th>
<th>I/O pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTA</td>
<td>A/D Converter inputs ( PIC 16C7X parts)</td>
<td>6  6</td>
</tr>
<tr>
<td>PORTB</td>
<td>External interrupt inouts</td>
<td>8  8</td>
</tr>
<tr>
<td>PORTC</td>
<td>Serial port, Timer I/O</td>
<td>8  8</td>
</tr>
<tr>
<td>PORTD</td>
<td>Parallel slave port</td>
<td>8  0</td>
</tr>
<tr>
<td>PORTE</td>
<td>A/D Convertor inputs ( PIC 16C 7X )</td>
<td>3  0</td>
</tr>
</tbody>
</table>

Total I/O pins: 33 22
Total pins: 40/44 28

Port D alternative function is parallel slave port which enables one PIC microcontroller to be connected to the data bus of another microprocessor. Since three LED’s are connected to three pins of Port D to be used as normal I/O operation, the special alternative function is ruled out.
Structure of a port - pin of PIC microcontroller

![Diagram of a port-pin of a PIC microcontroller]

TRIS register controls the direction of data flow.
TRIS = 1 Sets the pin in the input mode.
TRIS = 0 Sets the pin in the output mode.

; Toggle the green LED every half second.
List P = PIC16C74A, F = INHX8M, C = 160, N = 80
ST = FF, MM = OFF, R = DEC
include "C:\MPLAB\P16C74A.INC"
-config (_CP_OFF & _PWRTE_ON & _XT-OSC & _WDT_OFF & _BODEN_OFF)
error level -302

; Equates
Bank0 RAM equ 20H
MaxCount equ 50
Green equ 00000000H
TenMsH equ 13
TenMsL equ 250

; Variables
cblock Bank0RAM ; Variables are declared
BLNKCNT
COUNTH
COUNTL
endc

; Vectors
org 000H
goto Mainline
org 004H
Stop:
goto stop
Mainline:
call Initial ; Initialize

Main loop:
call Blink ; Blink LED
call TenMs ; Inset ten millisecond delay
indent goto Mainloop

;Initial Subroutine

Initial:
  movlw MaxCount 0.5 second
  movwf BLKCNT ← N
  movlw Green
  movwf PORTD ← W
  bsf STATUS,RPO Set register access to bank 1
  clrf TRISD Set PORTD as O/P port
  bcf STATUS,RPO Set register access to bank 0
  return

; Blink Subroutine. This subroutine blinks a green LED in every 0.5 sec

Blink:
  decfsz BLNKCNT,F ; decrement loop counter and return if not zero
goto BlinkEnd
  movlw MaxCount ; Reinitialize BLNKCNT
  movwf BLNKCNT
  movlw GREEN w ← Green Toggle green LED
  Xorwf PORTD,F w ← Green Toggle green LED

Blink End;
  return

; Ten Ms subroutine (delay of 10ms)

Ten Ms:
  nop
  movlw TenMsH
  movwf COUNTH ; COUNTH ← w
  movlw TenMsL
  movwf COUNTL

TEN,1:
  decfsz CountL.f
goto Ten,1
  decfsz COUNTH,f
goto Ten,1
  return

Ten Ms subroutine introduces a delay of 10ms by counting 10,000 instruction cycles. This is achieved by nested loops. The sequence of instructions executed from calling Ten Ms is listed and corresponding instruction cycles are mentioned against the instructions.
Instructions
Call ten Ms  2
nop  1
movlw 13 (TenMsH)  1
movwf COUNTH  1
movlw 250 (Ten MsL)  1
movwf COUNTL  1

decfsz COUNTL,F  1
  goto Ten_1

decfsz COUNTL,F  1
  COUNTL: 250 \rightarrow 249 \rightarrow \ldots \rightarrow 1  3 \times 249 = 747

decfsz COUNTL,F  1
  COUNTL: 1 \rightarrow 0  2

decfsz COUNTH,F  1
  COUNTH: 13 \rightarrow 12  1

goto Ten_1  2

decfsz COUNTL,F  1
  COUNTL: 0 \rightarrow 255 \rightarrow 254 \rightarrow \ldots \rightarrow 1

  255 \times 3 = 765

decfsz COUNTH,F  1
  COUNTH: 1 = 0  2

decfsz COUNTH,F  1
  COUNTH: 12 \rightarrow 11  1

goto Ten_1  2

Repeat this block 11 times as

COUNTH: 12 \rightarrow 11 \rightarrow \ldots \rightarrow 2 \rightarrow 1

decfsz COUNTL,F  1
  COUNTL: 0 \rightarrow 255 \rightarrow \ldots \rightarrow 2 \rightarrow 1  3 \times 255 = 765

goto Ten_1  2

decfsz COUNTL,F  1
  COUNTL: 1 \rightarrow 0  2

decfsz COUNTH,F  1
  COUNTH: 1 \rightarrow 0  2

goto Ten_1  2

Total = 10,000
**I²C Bus for Peripheral Chip Access**

Requires two open-drain I/O pins. 
Port-C of PIC IC can be used for I²C communication. 
SCL (Serial Clock)   RC3/SCK/SCL 
SDA (Serial Data)   RC4/SDI/SDA

Low output on SCL or SDA I/O pin set to be an output with ”0” written to it. 
High output on SCL or SDA I/O pin set to be an input. 
Transfers on the I²C bus take place a bit at a time. 

The clock line, SCL, is driven by the PIC chip, which server as *bus master*. The open drain feature of every chip’s bus driver can be used by the receiver to hold the clock line low, there by signalling the transmitter to pause until the clock line is released by the receiver. The open drain feature is also needed if this PIC will ever become an I²C slave to another PIC, in which it must relinquish control of the SCL line. 
The previous figure illustrates that the first eight bits on the SDA line are sent by the transmitter whereas the ninth bit is the acknowledgment bit which is sent by the receiver in response to the byte sent by the transmitter. For instance, when the PIC sends out a chip address, it is the transmitter,
while every other chip on the I\(^2\)C bus is a receiver. During the acknowledgment bit time, the addressed chip is the only one that drives the SDA line, pulling it low in response to the masters pulse on SCL, acknowledging the reception of its chip address.

When the data transfer direction is reversed that is form a peripheral chip to the PIC, which is the master, the peripheral chip drives the eight data bits in response to the clock pulse from PIC. In this case, the acknowledge bit is driven in a special way by the PIC, which is serving as receive but also as bus master. If the peripheral chip is one that can send the contents of successive internal address back to the PIC, then PIC completes the reception of each byte and signals a request for the next byte by pulling SDA line low in acknowledgment. After any number of bytes have been received by the master from the peripheral, the PIC can signal the peripheral to stop any further transfers by not pulling the SDA line low in acknowledgment.

SDA line should be stable during high period of the clock (SCL). When the slave peripheral is driving SDA line, either as transmitter or acknowledge, it initiates the new bit in response to the falling edge of SCL, after a specified time. It maintains that bit on SDA line until the next falling edge of SCL, again after a specified hold time.

I\(^2\)C bus transfers consist of a number of byte transfers framed between a START condition and either another START condition or a STOP condition. Both SDA and SCL lines are released by all drives and float high when bus transfers are not taking place. The PIC (I\(^2\)C bus controller) initiates a transfer with a START condition by first pulling SDA low and then pulling SCL as shown in the figure.

\[
\begin{array}{c}
SDA \\
SCL \\
\end{array}
\quad \text{START Condition}
\quad \begin{array}{c}
SDA \\
SCL \\
\end{array}
\quad \text{STOP Condition}
\]

Similarly, the PIC terminates a multiple byte transfer with the STOP condition. With both SDA and SCL initially low, it first releases SCL and then SDA. Both then occurrences are easily recognized by I\(^2\)C hardware in each peripheral chip since they both consist of a change in SDA line which SCL is high, a condition that never happens in the middle of a byte transfer.

**Data Communication protocol**

In I\(^2\)C communication standard, there is one bus master and several slaves. It can be assumed here that the PIC microcontroller is the bus master and several peripheral devices connected to SDA and SCL bus are slaves.

Following a start condition, the master sends a 7-bit address of the slave on SDA line. The MSB is sent first. After sending 7 bit address of the slave peripheral a R/W bit (8\(^{th}\) bit) is sent by the master. If R/W bit is 0 the following byte (after the acknowledgment) is written by the master to the addressed slave peripheral. If R/W bit is 1, the following byte after the acknowledgment bit has to be read from the slave by the master. After sending the 7-bit address of the slave, the master sends the address of the internal register of the salve where from the data has to be used or written to. The subsegment access is automatically directed to the next address of the internal register.
The following diagrams give the general format to write and read from several peripheral internal registers.

General format to write to several peripheral internal registers or addresses.

General format to from several peripheral internal registers or addresses.

The 1995 I²C bus specification includes the timing constraints for older chips designed for a maximum bit rate of 100kbits/s. It also includes constraints for newer fast-mode 400kbits/s parts.
**I²C Bus Subroutines:**

I²C bus fast-mode timing constraints.

STOP condition  

START condition  

STOP - to - START Constraints  

ACKNOWLEDGE bit  

START - to - START Constraints  

Data bit to data bit
Because the SCL pin must have an open pin output which the SDA pin must be either an input or have an open drain output, the I²C subroutines will repeatedly access TRISC, the data direction register for PORTC, However, TRISC is located at the bank 1 address 87H, which cannot be accessed by direct addressing without changing RPO bit to 1.

bsf STATUS, RPO
Then required bit of TRISC can be changed followed by clearing RPO and reveting back to Bank 0.

bsf STATUS, RPO
Instead of doing this, the indirect pointer FSR can be loaded with the address of TRISC and the bit setting and bit clearing of TRISC can be done indirectly.

For example, with the following definitions

SCL equ 3
SDA equ 4
bsf INDF, SDA
will release the SDA line, letting the external pull up register pull it high or some I²C chp pull it low.

When FSR is used for indirect addressing, care should be taken to restore FSR value when a subroutine is completed and the program returns to the mainline program.

I²C Subroutines

Freq equ 4
SDA equ 4
SCL equ 3

cblock.
.
.
DEVADD
INTADD
DATAOUT
DATAIN
TXBUFF
RXBUFF
.
.
endc
;DEVADD, INTADD, and DATAOUT
I2C out:
call start
movf DEVADD, W ; Send peripheral address with R/W=0 (write)
Call Tx
movf INTADD, W
Call Tx
movf DATAOUT, W
Call Tx
Call Stop ; Generate Stop condition
return
; The I2C in subroutine transfers out DEVADD (with R/W=0)
; and INTADD, restarts, transfers out DEVADD (with R/W=1)
; and read one byte back into DATAIN.

I2C in:
Call Start ; Generate start condition
movf DEVADD, W ; Send peripheral address R/W=0 (write)
Call Tx
movf INTADD, W ; Send peripheral’s internal address
Call Tx
Call ReStart ; Re START
movf DEVADD, W ; Send peripheral’s address.
iorlw 0000000.1 B ; with R/W=1 (read)
Call Tx
bsf TXBUFF, 7 ; NOACK the following reading of one byte
Call Rx ; Read byte
movwf DATAIN ; inte DATAIN
Call stop ; Generate stop condition
return
; The start subroutine initializes the I2C bus and then
; generates the START condition on the I2C bus
; The ReStart entry point bypadd the initialization of the
; I2C bus

Start:
movlw 00111011 ; Enable I2C Master mode.
movwf SSPCON
bcf PORTC, SDA ; DRIVE SDA low when it is an output
bcf PORTC, SCL ; DRIVE SCL low when it is an output
movlw TRISC ; Set indirect pointer to TRISC
movwf FSR

ReStart:
bsf INDF, SDA ; Make sure SDA is high - I/P mode
bsf INDF, SCL ; Make sure SCL is high - I/P mode
delay 0,1,2 not
bcf INDF, SDA ; Make SDA low
delay 0,1,2 nop
bcf INDF, SCL ; Make SCL low
return

Stop:
  bcf INDF, SDA ; Return SDA low
  bsf INDF, SCL ; Drive SCL high
delay 0,1,2
  bcf INDF, SDA ; and then drive SDA high
return

; The Tx subroutine sends out the byte passed to it in W.
; It returns with z = 1 if ACK occurs.
; It returns with z = 0 if NOACK occurs.

Tx:
  movwf TXBUFF
  bsf STATUS, C

Tx_1:
  rlf TXBUFF, F ; rotate TXBUFF left, through carry
  movf TXBUFF, F ; Set Z bit when all 8 bits have been transformed
  btfss STATUS, Z ; until z = 1
  Call Bitout ; Send carry bit then clear carry bit
  btfss STATUS, Z
  goto TX_1
  Call Bit In
  movlw 00000001 B
  End wf RXBUFF, W ; z = 1 if ACK z = 0 if NOACK
return

; The Rx subroutine receives a byte from I2C bus into W,
; using RXBUFF buffer
; Call Rx with bit 7 of TXBUFF clear for ACK
; Call Rx with bit 7 of TXBUFF set for NOACK

Rx:
  movlw 00000001 B
  movwf RXBUFF

Rx_1:
  rlf RXBUFF, F
  Call Bit In
  btfss STATUS, C
goto Rx_1
rlf TXBUFF, F
Call BitOut
movf RXBUFF, W
return
; The BitOut subroutine transmits, then clears, the carry bit
BitOut:
  bcf INDF, SDA ; copy carrybit to SDA
  btfsc STATUS, c
  bcf INDF, SDA
  bsf INDF, SCL ; pulse clockline
  delay 0,1,2 ; t: HIGH
  bcf INDF, SCL
  bcf STATUS, c
  return
; The bit In subroutine receives one bit into
; bit 0 of RXBUFF
BitIn:
  bsf INDF, SDA
  bsf INDF, SCL ; Drive clock line high
  bcf RXBUFF, 0 ; copy SDA to bit 0 of RXBUFF
  btfsc PORTC, SDA
  bsf RXBUFF, 0
  bcf INDF, SCL ; Drive clock low again
  return
Examples of I2C bus Interfacing

I. DAC Interfacing

Two digital-to-analog converter outputs are easily added to a PIC with MAX518 eight-pin DIP. Each output channel produces an output voltage that ranges from 0V to $\frac{255}{256} V_{DD}$ where $V_{DD}$ is the power supply to the DAC chip. If $V_{DD} = 5V$, an output of 2.5V will appear on the OUT0 pin if the following three bytes are sent to the chip.

'01011000', '00000000' '10000000'

An output of 2.5 V will appear on the OUT1 pin by sending the following three bytes

'01011000' '00000001' '100000000'

The MAX518 chip includes a power-on reset circuit that drives the two outputs to 0V initially. The two address inputs, AD1 and AD0, provide an adjustable part of the chip’s I2C address. With 5 bits fixed at 01011 and two adjustable bits, it is possible to connect four MAC518 chips to a PIC.
DAC Interfacing on I²C bus

Analog outputs

MAX 518
Dual 8-bit DAC

7-bit address

First byte of message string

Second byte

Third byte

Analog output voltage = $V_{DD} \frac{B}{256}$
II. Interfacing a Temperature Sensor

National Semiconductor’s LM 75 chip combines an analog temperature transducer, an analog-to-digital convertor (9-bit), and an I²C bus interface, all in a tiny S)-8 surface mount package. The temperature range covered is -25°C to +100°C with ±2°C accuracy. The two’s complement form of the temperature is available from the 9-bit ADC. The resolution of the ADC is about 0.5°C.

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Digital Output</th>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>125°C</td>
<td>25</td>
<td>01111 1010</td>
<td>250</td>
</tr>
<tr>
<td>25°C</td>
<td>50</td>
<td>00011 0010</td>
<td>50</td>
</tr>
<tr>
<td>0.5°C</td>
<td>1</td>
<td>000000001</td>
<td>1</td>
</tr>
<tr>
<td>0°C</td>
<td>0</td>
<td>00000 0000</td>
<td>0</td>
</tr>
<tr>
<td>-0.5°C</td>
<td>-25</td>
<td>11111 1111</td>
<td></td>
</tr>
<tr>
<td>-25°C</td>
<td>-55</td>
<td>11100 1110</td>
<td></td>
</tr>
<tr>
<td>-55°C</td>
<td></td>
<td>11001 0010</td>
<td></td>
</tr>
</tbody>
</table>

LM 75 chip also includes a thermal watch dog that can be setup to interrupt PIC on its RBO/INT edge-triggered interrupt input when the temperature rises above a programmable, $T_{OS}$. It also includes programmable hysteresis so that the temperature must dip down below the setpoints $T_{OS}$ threshold to a lower $T_{HYST}$ threshold before rising again past the $T_{OS}$ setpoint to generate another output edge.
O.S. stands for over temperature shutdown.
Register Structure

When a "write" message string is sent, the first byte selects the chip for a write and the second byte loads the pointer register. The write message string can stop there or it can continue with a 2-byte write of $T_{OS}$ (Over tem shutdown). Once the pointer has been set, any of their register can be read, reading two bytes for temperature, $T_{OS}$, or $T_{HYST}$ or reading just 1 byte for the configuration register.

<table>
<thead>
<tr>
<th>Pointer</th>
<th>0 0</th>
<th>Temperature (read only) default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1</td>
<td>Configuration (read / write)</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>$T_{HYST}$ (read / write)</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>$T_{OS}$ (read / write )</td>
</tr>
</tbody>
</table>

Configuration

| 0 0 0 0 0 0 0 0 |

1 : Low-power shut down (14A typical)
0 : Normal operation (default)

Other features not selected

Temp.

| b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | X  | X  | X  | X  | X  | X  | X  |

$T_{OS}$

| b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | X  | X  | X  | X  | X  | X  | X  | X  | X  | X  | X  | X  |

$T_{HYST}$


Synchronous Serial Port Module

Mid range PIC microcontroller includes a Synchronous Serial Port (SSP) module, which can be configured into either of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I²C)

Either of these modes can be used to interconnect two or more PIC chips to each other using a minimal number of wires for interconnection. Alternatively, either can be used to connect a PIC
chip to a peripheral chip. In this case of the I\(^2\)C mode, the peripheral chip must also include an I\(^2\)C interface. In contrast, the SPI mode provides the clock and serial data lines for direct connection to shift registers, adding an arbitrary number of I/O pins to a PIC chips.

**Serial Peripheral Interface**

![SPI block within PIC diagram]

Portc three pins RC5, RC4 and RC3 are used for Synchronous Serial Interface. These pins revert to their normal general purpose I/O pins if neither of the two SSP modes is selected. The SPI port requires the RC3/SCK pin to be an output that generates the clock signal used by the external shift registers. This output line characterizes the SPI’s master mode. In slave mode, RC3/SCK works as the input for the clock.

When a byte of data is written to SSPBUF register, it is shifted out the SDO pin in synchronism with the emitted pulses on the SCK pin. The MSB of SSPBUF is the first bit to appear on SDO pin. Simultaneously, the same write to SSPBUF also initiates the 8-bit data reception into SSPBUF of whatever appears on SDI pin at the time of rising edges of the clock on SCK pin.
A read or write of one of the PIC’s ports, such as PORTD takes one internal clock cycle to execute. In contrast, a read or write of an expansion port that is implemented with an SPI-connected octal shift register is slowed down by an order of magnitude by the eight clock pulses as seen before. If the SSPIF flag in the PIR1 register is cleared before the SPI transmission is initiated, then it will be automatically set at the completion of the transfer setting of SSSPIF flag indicated that the transferred data is in place and ready to be used.
Output port expansion

Port configurations

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit Placement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRISC</td>
<td>87H</td>
<td>Output for SCK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gen purpose o/p to drive latch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Output for SDO</td>
</tr>
<tr>
<td>TRISD</td>
<td>85H</td>
<td>General purpose o/p to drive latch</td>
</tr>
<tr>
<td>SSPCON</td>
<td>14H</td>
<td>SPI &quot;master&quot; mode with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SCK = osc / 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CKP = 1 : SCK will idle high</td>
</tr>
<tr>
<td>PIR1</td>
<td>OCH</td>
<td>SSPEN = 1 : Enable Synchronous Serial Port (SPI)</td>
</tr>
<tr>
<td>SSPBUF</td>
<td>13H</td>
<td>SSPIF = 1 When transfer is complete: clear before beginning of each transfer</td>
</tr>
</tbody>
</table>
Input port expansion

- PIC
- SPI
- RC4/SDI
- RC3/SCK
- 74HC165 Shift register
- Serial clock
- Load
- Data in
- Data out
- MSB first

Timing diagram

- RD7
- SSPIF
- SCK (CKP=0)
- SDI
- SPI reads input bit here
- Read SSPBUF

Write to SSBUF to initiate transfer

9 µs (for osc = 4MHz)
Port configurations

<table>
<thead>
<tr>
<th>TRISC 87H</th>
<th>X</th>
<th>X</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Output for SCK</td>
<td>Input for SDI</td>
<td>General purpose input (to preempt SD0 o/p)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRISD 88H</th>
<th>0</th>
<th>X</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gen. purpose o/p to drive load input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSPCON 14H</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPI “master” with SCK = ocs /4 CKP = 0 : SK0 will idle low SSPEN = 1 : enable Synchronous Serial Port (SPI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Analog-to-Digital Converter

Features (16C7X)

- Eight input channels
- An analog multiplexer
- A track and hold circuit for signal on the selected input channel
- Alternative clock sources for carrying out the conversion.
- An adjustable autonomous sampling rate.
- The choice of an internal or external ref. voltage.
- 8-bit conversion
- Interrupt response when conversion is completed.
Port A and Port E pins are used for analog inputs/Reference voltage for ADC.

**Port A pins**
- RA0/AN0 - Can be used as analog input-0
- RA1/AN1 - Can be used as analog input-1
- RA2/AN2 - Can be used as analog input-2
- RA3/AN3/VREF - RA3 can be used as analog input 3 or analog reference voltage
- RA4/TOCKI - RA4 can be used as clock input to Timer-0
- RA5/SS/AN4 - RA5 can be used as analog input 4 or the slave select for the sync serial port

**Port E pins**
- RE0/RD/AN5 - Can be used as analog input 5
- RE1/WR/AN6 - Can be used as analog input 6
- RE2/CS/AN7 - Can be used as analog input 7

PIC microcontroller has internal sample and hold circuit. The input signal should be stable across the capacitor before the conversion is initiated.
After waiting out the sampling time, a conversion can be initiated. The ADC circuit will open the sampling switch and carry out the conversion of the input voltage as it was at the moment of opening of the switch. Upon completion of the conversion, the sampling switch is closed and $V_{\text{HOLD}}$ again tracks $V_{\text{SOURCE}}$.

### Using the A/D Converter

Registers ADCON1, TRISA, and TRISE must be initialized to select the reference voltage and the input channels. The first step selects the ADC clock source from among four choices (OSC/2, OSC/8, OSC/32, and RC). The constraint for selecting clock frequency is that the ADC clock period must be 1.6 $\mu$s or greater.

The A/D modules have three registers. These registers are

- A/D Result Register (ADRES)
- A/D Control Register 0 (ADCON0)
- A/D Control Register 1 (ADCON1)

The ADCONO register as shown here, controls the operation of A/D module.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADCS1</td>
<td>ADCS0</td>
<td>CHS2</td>
<td>CHS1</td>
<td>CHS0</td>
<td>GO/DONE</td>
<td>-</td>
<td>ADON</td>
</tr>
</tbody>
</table>

**bit 7 - 6**

- ADCS1 : ADCS 0
  - 00 = $F_{\text{osc}}$/2
  - 01 = $F_{\text{osc}}$/8
  - 10 = $F_{\text{osc}}$/32
  - 11 = $F_{\text{RC}}$ (clock derived from an internal RC oscillator)

**bit 5 - 3**

- CHS2, CHS0
  - 000 - channel 0 - AN0
  - 001 - channel 1 - AN1
  - 010 - channel 2 - AN2
  - 011 - channel 3 - AN3
  - 100 - channel 4 - AN4
  - 101 - channel 5 - AN5
  - 110 - channel 6 - AN6
  - 111 - channel 7 - AN7

**bit 2**  

- A/D Conversion Status bit
  - GO/DONE
    - if ADON = 1
      - 1 = A/D conversion is in progress (setting this bit starts the A/D conversion)
      - 0 = A/D conversion is not in progress (this bit is automatically cleared by hardware when...
A/D conversion is complete.)

**bit 1**

Unimplemented

**bit 0**

ADON: A/D on bit
1 = A/D converter module is ON
0 = A/D converter module is OFF (not operating.)

**ADCON1 Register**

<table>
<thead>
<tr>
<th>bit</th>
<th>PCFG2</th>
<th>PCFG1</th>
<th>PCFG0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

PCFG2 : PCFG0 : A/D Port Configuration Control bits

<table>
<thead>
<tr>
<th>PCFG2 : PCFG0</th>
<th>RA0</th>
<th>RA1</th>
<th>RA2</th>
<th>RA5</th>
<th>RA3</th>
<th>RE0</th>
<th>RE1</th>
<th>RE2</th>
<th>V_{REF}</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>V_{DD}</td>
</tr>
<tr>
<td>001</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>V_{REF}</td>
<td>A</td>
<td>A</td>
<td>RA3</td>
</tr>
<tr>
<td>010</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>V_{DD}</td>
</tr>
<tr>
<td>011</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>V_{REF}</td>
<td>D</td>
<td>D</td>
<td>RA3</td>
</tr>
<tr>
<td>100</td>
<td>A</td>
<td>A</td>
<td>D</td>
<td>D</td>
<td>A</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>V_{DD}</td>
</tr>
<tr>
<td>101</td>
<td>A</td>
<td>A</td>
<td>D</td>
<td>D</td>
<td>V_{REF}</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>RA3</td>
</tr>
<tr>
<td>11X</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>-</td>
</tr>
</tbody>
</table>

**TRISA**

Analog I/P
1: Digital I
0: Digital 0
1: D/A
0: Digital

**TRISE**

Disable Port E alternate function
1: Analog / Digital input
0: Digital output
1. Configure A/D module
   - Configure analog pins/ voltage reference/ and digital I/O (ADCON1)
   - Select A/D channel (ADCON0)
   - Select A/D conversion clock (ADCONO)
   - Turn on A/D module (ADCONO)

2. Configure A/D interrupt (if required)
   - Clear AD—F bit in PIR 1 reg
   - Set AD—E bit in PIE 1 reg
   - Set G—E bit

3. Wait for required acquisition time

4. Start conversion
   - Set GO/\textit{DONE}

5. Wait for A/D conversion to complete by either
   - polling for GO/\textit{DONE} bit to be cleared
   - waiting for the A/D interrupt

6. Read A/D result register (ADRES) Clear AD—F if required.

Example Program

A/D Conversion with Interrupt

bsf STATUS, RPO ; Select Bank1
clr ADCON 1 ; Configure A/D input
bsf PIE1, ADIE ; Enable A/D interrupt
bcf STATUS, RPO ; Select Bank 0
movlw 0811+ ; Select fosc/32, channel 0, A/D on movwf ADCONO
bcf PIR1, ADIF
bsf INTCON, PEIE
bsf INTCON, GIE

; Ensure that the required sampling time for the
; selected input channel has elapsed.
; Then the conversion may be started
bsf ADCONO, GO ; start A/D conversion
; AD| F bit will be set
; and GO/\textit{DONE} bit is cleared
; upon completion of A/D conversion
Code structure for large Programs

Memory paging is essential if the code exceeds 2k of program memory (2048). PIC 16C74A supports 4096 addresses and hence it is important to consider memory paging for this processor.

**PCL and PCLATH**

The program counter (PC) is 13-bit wide. The low byte comes from the PCL register, which is a readable and writable register. The upper bits (PC<12:8>) are not readable, but are indirectly writable through the PCLATH register. On any reset, the upper bits of the PC will be cleared. PCL ← 0 and PCLATH ← 0. Two situations for loading the PC following any reset are given here.

1. Any write to PCL register load the content of PCL to lower 8 bit of PC and content of PCLATH to higher 5 bits.

   \[
   \text{mov wf PCL}
   \]

   ![Diagram showing loading of PC and PCLATH](image)

2. PC is also loaded during a call or goto instruction

   \[0 \leq k \leq 2047\]

   **Operation:**

   \[k \rightarrow PC < 10 : 0 >\]

   \[PCLATH < 4 : 3 > \rightarrow PC < 12 : 11 >\]

   Goto is an unconditional branch. The eleven bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH <4:3>.

![Diagram showing goto instruction](image)

**STACK**

The PIC16CXX family has an 8 level deep X 13-bit wide hardware stack. The stack space is not part of either program or data memory and the stack pointer is not readable or writable. The PC is pushed onto the stack when a CALL instruction is executed or an interrupt causes a branch.
stack os POPed in the event of a RETURN, RETLW or a RETIE instruction execution. PCLATH is not affected by a PUSH or a POP operation. The stack operates on a circular buffer.

**Paging:**
Following any reset PCL and PCLATH are cleared to 0. For a 4k program memory, the address range is from 0000H to 0FFFH. Hence each call and goto instruction will actually reach the desired address only if bit 3 of PCLATH is set or cleared correctly. However even for 4k PIC controllers, there is no need to take care of PCLATH bit 3, if the code size fits into 2k address space. Bit 3 of PCLATH will come out of reset in the zero state and there will never be a need to change it. Consequently, every call and goto instruction will go to the correct place.

For large programs, it is helpful to break out blocks of code that are reached by a single call instruction and that terminates in a single return instruction. Such a block of code can be placed on program memory’s page 1. Then, before executing the call instruction to reach the block, the following instruction can be executed.

```assembly
bsf PCLATH, 3 ; Switch to program memory’s Page 1.
```

When it is finally time to exit from the block to return to the mainline program in Page 0, the return instruction is preceded by the instruction

```assembly
bcf PCLATH, 3
```

---

**Program memory allocation for large programs**

<table>
<thead>
<tr>
<th>Hex address</th>
<th>Page 0</th>
<th>Hex address</th>
<th>Page 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
<td>800</td>
<td></td>
</tr>
<tr>
<td>004</td>
<td>Store W, STATUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCLATH</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bsf PCLATH , 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>goto Int.Service</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main line</td>
<td></td>
<td>Int Service</td>
<td></td>
</tr>
<tr>
<td>7FF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bsf PCLATH , 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Call Block 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bsf PCLATH , 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Call Block 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overview of Timer Modules

Timer-0 Overview

The Timer 0 module is a simple 8-bit overflow counter. The clock source can be either the internal clock \( f_{\text{osc}}/4 \) or an external clock. When the clock source is an external clock, the Timer-0 module can be selected to increment on either the rising or falling edge.

Timer-0 module also has a programmable prescalar option. This prescalar can be assigned either to Timer 0 or the watchdog Timer.

The counter sets a flag TOIF when it overflows and can cause an interrupt at that time if that interrupt source has been enabled \( \text{(TOIF}=1) \). Timer 0 can be assigned an 8-bit prescalar that can divide the input by 2, 4, 8, 16, ..., 256. Writing to TMRO resets the prescalar assigned to it.

Timer-0, or its prescalar can be connected to either of two input sources.

1. \( f_{\text{osc}}/4 \)

2. RA4/ TOCKI, the input connected to bit 4 of PORTA.

<table>
<thead>
<tr>
<th>RBPU</th>
<th>INTEDG</th>
<th>0</th>
<th>TOSE</th>
<th>1</th>
<th>PS2</th>
<th>PS1</th>
<th>PS0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 1: Prescaler assigned to watchdog timer
- 0: Prescaler assigned to Timer 0

\[\text{TMRO} \quad 01H\]

- 8-bit counter
- 2-cycle delay
- \( f_{\text{osc}}/4 \)

- Interrupt CPU
- INTCON 0BH, 8BH
- Global int. enable bit
- TOIE : Timer 0 overflow int. enable bit
- TOIF : TMRO overflow int. flag

- Set
- Other Int.
Timer-0 use with prescalar

TOCS = 0, Timer 0 clock is $f_{osc}/4$

PCA = 0, Prescaler assigned to Timer0

Prescaler:
- 0 0 0 2
- 0 0 1 4
- 0 1 0 8
- 0 1 1 16
- 1 0 0 32
- 1 0 1 64
- 1 1 0 128
- 1 1 1 256

Overflow

8-bit counter

2-cycle delay

Prescaler

$\frac{f_{osc}}{4}$
External clock synchronization

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 register pair (TMR1H: TMR1L) increments from 0000H to FFFFH and rolls over to 0000H. The TMR1 interrupt, if enabled, is generated on overflow which sets the interrupt flag bit TMR1IF-(PIR< 0 >). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit TMR1IE-(PIE < 0 >)

The operating and control modes of Timer 1 is determined by the special purpose register T1CON. T1CON (10H)
Timer 1 can operate in one of the two modes.

- As a timer. \( (\text{TMR1CS} = 0) \)
  In timer mode, Timer 1 increments in every instruction cycle. The Timer 1 clock source is \( \frac{f_{\text{osc}}}{4} \). Since the internal clock is selected, the timer is always synchronized and there is no further need of synchronization.

- As a counter \( (\text{TMR1CS} = 1) \)
In counter mode, external clock input from the pin RCO/T1OSC/T1CKI is selected.

**Use of Timer-2**

Timer 0: 8-bit timer/counter with 8-bit prescalar  
Timer 1: 16-bit timer/counter with prescalar, can be incremented during sleep via external crystal/clock.  
Timer 2: 8-bit timer/counter with 8-bit period register, prescalar, post scalar.

**Timer 2 Circuitry**

Timer 2 is an 8-bit timer with a prescalar and a port scalar. It can be used on the PWM mode of CCP modules. The TMR2 register is readable and writable and is cleared on any device reset. The input clock \( f_{osc}/4 \) has a prescalar option of 1:1, 1:4 or 1:16 selected by bits 0 and 1 of T2CON register.

The timer 2 module has a 8-bit period register (PR2). timer 2 increments from 00H until it matches PR2 and then resets to 00H on the next increment cycle. PR2 is a readable and a writable register. PR2 is initialized to FFH on reset.

The output of TMR2 goes through a 4-bit post scalar (1:1, 1:2 to 1:16) to generate a TMR2 interrupt by setting TMR2IF flag.
CCP overview

The CCP module(s) can operate in one of the three modes: 16-bit capture, 16-bit compare, or upto 1-bit Pulse Width Modulation (PWM).

Capture mode captures the 16-bit value of TMR1 into CCPRxH: CCPRxL register pair. The capture event can be programmed for either the falling edge, rising edge, fourth rising edge, or the sixteenth rising edge of the CCPx pair.

Compare mode compares the TMR1H: TMR1L register pair to the CCPRxH: CCPRxL register pair. When a match occurs an interrupt can be generated, and the output pin CCPx can be forced to given state (High or Low), TMR1 can be reset (CCP1) or TMR1 reset and start A/D conversion (CCP2). This depends on the control bits < CCPxM3 : CCPxM0>

PWM mode compares the TMR2 register to a 10 bit duty cycle register (CCPRxH : CCPRxL<5:4>) as well as an 8-bit period register (PR2). When the TMR2 register= Duty cycle register, the CCPx pin will be forced low. When TMR2=PR2, TM2 is cleared to 00H, an interrupt can be generated, and the CCPx pin, if programmed in the O/P mode, will be forced high.

Compare Mode

Timer 1 is a 16-bit counter which can be used with CCP (Capture/compare/PWM) module to drive a pin high or low at precisely controlled time, independent of what the CPU is doing at that time. The pins are Port-C RC1/CCP2 and RC2/CCP1 pins.

Which Timer1 includes a prescalar to divide the internal clock by 1,2,4 or 8, the choice of divide-by-one gives the finest resolution in setting the time of an output edge.
Capture/Compare/PWM modules

Each CCP (Capture/compare/PWM) module contains a 16-bit register which can operate as a 16-bit capture register, as a 16-bit compare register or as a PWM master/slave duty cycle register. Both CCP1 and CCP2 are identical in operation, with the exception of the operation of the special event trigger.

The following shows the CCP mode timer resources.

<table>
<thead>
<tr>
<th>CCP Mode</th>
<th>Timer Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capture</td>
<td>Timer 1</td>
</tr>
<tr>
<td>Compare</td>
<td>Timer 1</td>
</tr>
<tr>
<td>PWM</td>
<td>Timer 2</td>
</tr>
</tbody>
</table>

**CCP1 Module:**

Capture/Compare/PWM Register 1 consists of two 8-bit register: CCPR1L (low byte) and CCPR2H (high byte). The CCP1CON register controls the operation of CCP1. All are readable and writable.

**CCP2 Module:**

Capture/Compare/PWM Register 2 consists of two 8-bit registers: CCPR2L (low byte) and CCPR2H (high byte). The CCP2CON register controls the operation of CCP2. All are readable and writable.

**CCP1CON Register / CCP2CON Register**

<table>
<thead>
<tr>
<th>bit 7</th>
<th>bit 6</th>
<th>bit 5-4</th>
<th>bit 3-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>—</td>
<td>CCPxX</td>
<td>CCPxY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCPxM3</td>
<td>CCPxM2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCPxM1</td>
<td>CCPxM0</td>
</tr>
</tbody>
</table>

- bit 5-4: CCPxX : CCPxY : PWM Least Significant bits.
  - Capture mode: Unused
  - Compare mode: Unused
  - PWM mode: These bits are the two LSBs of the PWM duty cycle. The eight MSBs are found in CCPRxL.

- bit 3-0: CCPxM3 : CCPxM0 : CCPx Mode select bits.

*Capture Mode*
**Compare Mode**

**PWM Mode**

In Pulse Width Modulation (PWM) mode, the CCPx pin produced upto a 10-bit resolution PWM output. Since CCP1 pin is multiplexed with PORT C data latch, the TRISC < 2 > pin must be cleared to make CCP1 pin an output.

*Simplified PWM Block Diagram*
A PWM output as shown has a time period. The time for which the output stays high is called duty cycle.

**PWM Period**

The PWM period is specified by writing to PR2 register. The PWM period can be calculated using the following formula:

\[
\text{PWM period} = \left( (PR2) + 1 \right) \times 4 \times T_{osc} \times (TMR2 \text{ prescale value})
\]

PWM frequency = \(1/\text{PWM period}\)

When TMR2 is equal to PR2, the following events occur on the next increment cycle.

- TMR2 is cleared
- the CCP1 pin is set (if PWM duty cycle is 0)
- The PWM duty cycle is latched from CCPR1L into CCPR1H

**PWM duty cycle**

The PWM duty cycle is specified by writing to the CCPR1L register and to CCP1CON < 5 : 4 > bits. Up to 10-bit resolution is available where CCPR1L contains the eight MSBs and CCP1CON < 5 : 4 > contains the two LSB’s. The 10-bit value is represented by CCPR1L : CCP1CON < 5 : 4 >. The PWM duty cycle is given by

\[
\text{PWM duty cycle} = (\text{CCPR1L : CCP1CON < 5 : 4 >}) \times T_{osc} \times (TMR2 \text{ prescale value})
\]

Although CCPR1L and CCP1CON < 5 : 4 > can be written to at anytime, the duty cycle value is not latched.

**Counting mechanism in Timer 2**

**PWM Mode**
into CCPR1H until a match between PR2 and TMR2 occurs. In PWM mode, CCPR1H is a read-only register.

The CCPR1H register and a 2-bit internal latch are used to double buffer the PWM duty cycle. This double buffering is essential for glitchless PWM operation. When the CCPR1H and 2-bit latch match TMR2 concatenated with an internal 2-bit Q clock or 2-bits of prescalar, the CCP1 pin is cleared.

Maximum PWM resolution (bits) for a given PWM frequency can be calculated as

$$\frac{\log(f_{osc})}{\log(f_{PNM})}$$

If the PWM duty cycle is longer than the PWM period, then the CCP1 pin will not be cleared.

**PWM Period and duty cycle calculation**

Example Desired PWM frequency = 78.125 kHz

$$f_{osc} = 20\text{MHz}$$

TMR2 Prescalar = 1

$$\frac{1}{78.125 \times 10^3} = (PR2 + 1)4 \times \frac{1}{20 \times 10^6} \quad PR2 = 63$$

Find the maximum resolution of duty cycle that can be used with a 78.124 kHz frequency and 20 MHz oscillator.

$$\frac{1}{78.125 \times 10^3} = 2^{\text{PWM Resolution}} \cdot \frac{1}{20 \times 10^6} \cdot 1$$

$$256 = 2^{\text{PWM Resolution}}$$

PWM Resolution = 8

At most, an 8-bit resolution duty cycle can be obtained from a 78.125 kHz frequency and 20 MHz oscillator, ie, 0 ≤ CCPR1L : CCP1CON < 5 : 4 >≤ 255. Any value greater than 255 will result in a 100% duty cycle. The following table gives the PWM frequency $$f_{PWM}$$ if $$f_{osc}=20\text{MHz}$$.
<table>
<thead>
<tr>
<th>Duty cycle resolution</th>
<th>10-bit counter scale</th>
<th>PR2 value</th>
<th>Prescalar 1</th>
<th>Prescalar 4</th>
<th>Prescalar 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 bit</td>
<td>1024</td>
<td>255</td>
<td>19.53 KHz</td>
<td>4.88 kHz</td>
<td>1.22 kHz</td>
</tr>
<tr>
<td>≈ 10 bit</td>
<td>1000</td>
<td>249</td>
<td>20k Hz</td>
<td>5kHz</td>
<td>1.25kHz</td>
</tr>
<tr>
<td>8 bit</td>
<td>256</td>
<td>63</td>
<td>78.125kHz</td>
<td>19.53kHz</td>
<td>4.88kHz</td>
</tr>
<tr>
<td>6 bit</td>
<td>64</td>
<td>15</td>
<td>312.5kHz</td>
<td>78.125kHz</td>
<td>19.53kHz</td>
</tr>
</tbody>
</table>

**Interrupt Logic**

Four of PORTB’s pins RB7 : RB4 have an interrupt on change feature. Only pins configured on inputs can cause this interrupt to occur. The input pins (of RB7 : RB4) are compared with the old values on the last read of Port B. the ”mismatch” outputs of RB7 : RB4 are used together to generate the RB port change interrupt flag bit RB1F.