**LAB 3 - PERFORM COMPARE, CAPTURE AND PWM OPERATIONS USING PIC16F877A**

**Aim:**

To perform the Compare, Capture and PWM mode operations using CCP module in PIC16F877A

**Requirements:**

1. **Hardware** – PIC16F877A, 20Mhz oscillator, capacitors -33pf, Oscilloscope, power supplies.
2. **Software** – Proteus (simulation software), MPLAB IDE

**Theory:**

The PIC16F877A Microcontroller consists of a special module called Compare Capture module (CCP Module). This is a multi-purpose module that we can switch between 3 different modes of operation. At each mode of operation, this module can perform a specific task that could be useful for many applications. Each of the Capture/Compare/PWM (CCP) Modules contains a 16-bit register which can operate as a:

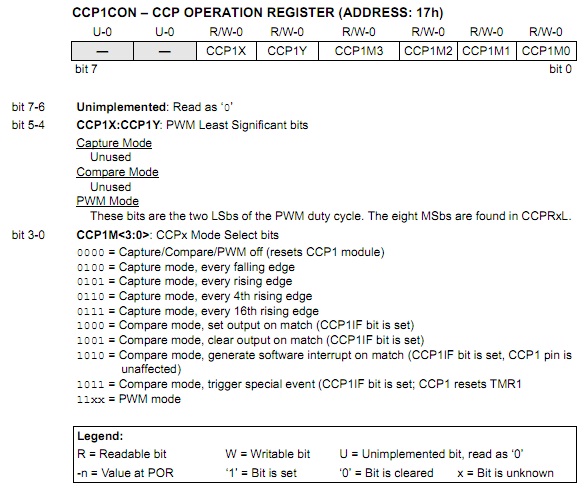
* 16-Bit Capture Register
* 16-Bit Compare Register
* PWM Duty Cycle Register

PIC16F877A Chip has two of identical CCP modules CCP1 & CCP2 , identical in structure and operation The 16-Bit Data Register for CCP modules is actually a couple of 8-Bit SFRs (CCPRxL-CCPRxH) where x maybe 0 or 1 for CCP1 & CCP2 modules.

Both the CCP1 & CCP2 module require a hardware timer as a resource for their operation. The hardware timer module being used for both of CCP modules is determined based on the mode of operation. The table below indicates which timer module is being used for each mode.

|  |  |
| --- | --- |
| **CCP Mode**  Capture  Compare  PWM | **TIMER Resource**  Timer 1  Timer 1  Timer 2 |

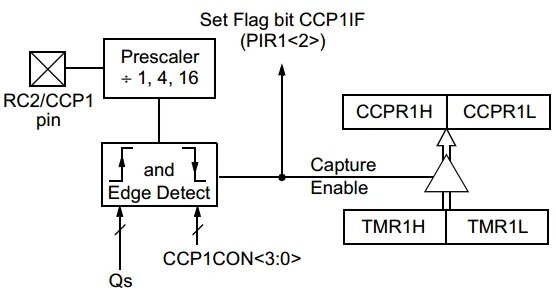
This diagram displays the register associated with the CCP module that is CCPCON registers, it states the different configuration bits used for Capture/ Compare/PWM operations. The operation of both CCP1 & CCP2 modules is controlled via the CCPxCON Registers which is a of 8-Bit SFRs (CCP1CON & CCP2CON) respectively.



**Capture:** In Capture mode, the CCP module captures the 16-Bit value of Timer1 module in the CCPRx register upon a specific user-defined event. While the Timer1 module is running in either timer-mode or synchronized counter-mode. The event that fires a capture signal can be :

* Every rising edge
* Every falling edge
* Every 4th rising edge
* Every 16th rising edge

The type of event that fires the capture signal is configured by the 4-control bits (CCP1M3:CCP1M0)



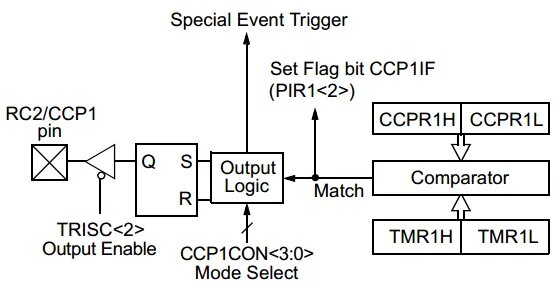
*Configuration Steps for Capture Operations*

1. Set Timer1 module to operate in timer/counter mode.
2. Turn ON Timer1 module
3. Configure the CCP1 module to operate in Capture Mode (using CCP1CON register).
4. Choose the event on which a capture occurs (using CCP1CON register – CCP1Mx Bits).
5. Configure the CCP1 interrupt
6. Write The ISR Handler for CCP-Capture Interrupt

**Compare:** In Compare Mode, the 16-Bit CCPR1 (CCPR1H:CCPR1L) register is constantly compared against the TMR1 register. When a match occurs, The CCP1 pin (RC2) is:

* Driven Low
* Driven High
* Remains Unchanged

The type of event that fires the capture signal is configured by the 4-control bits (CCP1M3:CCP1M0)



*Configuration Steps for Capture Operations*

1. Set Timer1 module to operate in timer or counter mode
2. Turn ON Timer1 module
3. Preload The CCPR1 Register with the desired value (from calculations)
4. Configure the CCP1 module to operate in Compare Mode (using CCP1CON register)
5. Choose the event on the RC2 pin when a match occurs (using CCP1CON register – CCP1Mx Bits)
6. Configure the CCP1 interrupt
7. Write The ISR Handler for CCP-Capture Interrupt

**Pulse Width Modulation (PWM)**: - PWM is a digital signal which is most commonly used in control circuitry. This signal is set high (5v) and low (0v) in a predefined time and speed. The time during which the signal stays high is called the “on time” and the time during which the signal stays low is called the “off time”. There are two important parameters for a PWM.

1. **Duty cycle of the PWM**

The percentage of time in which the PWM signal remains HIGH (on time) is called as duty cycle. If the signal is always ON it is in 100% duty cycle and if it is always off it is 0% duty cycle.

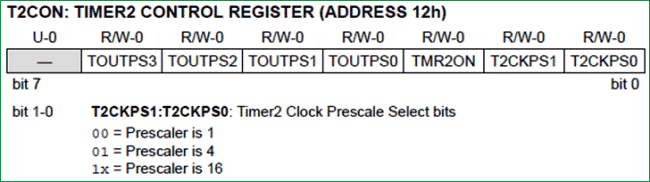
1. **Frequency of a PWM:**

The frequency of a PWM signal determines how fast a PWM completes one period. One Period is complete ON and OFF of a PWM signal as shown in the above figure.

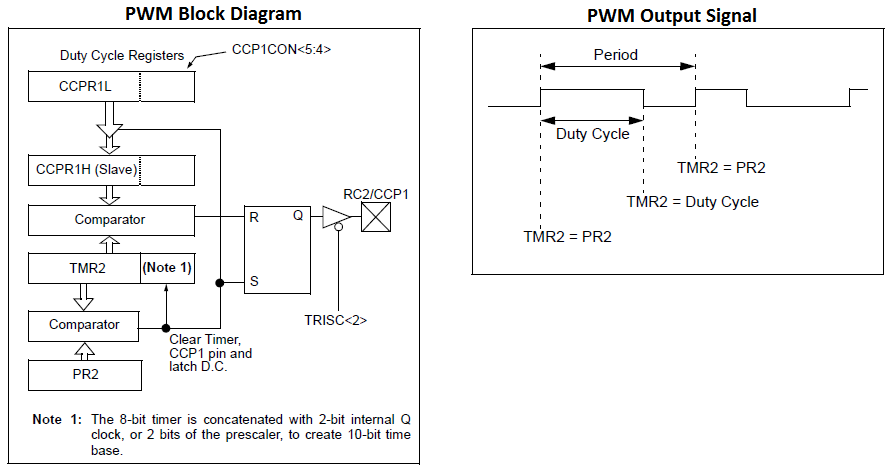
PWM signals can be generated in our PIC Microcontroller by using the CCP (Compare Capture PWM) module. The resolution of our PWM signal is 10-bit, that is for a value of 0 there will be a duty cycle of 0% and for a value of 1024 (2^10) there be a duty cycle of 100%. There are two CCP modules in the PIC MCU (CCP1 and CCP2), this means two PWM signals on two different pins (pin 17 and 16) can be generated. The registers associated with CCP module are,

* CCP1CON (CCP1 control Register)
* T2CON (Timer 2 Control Register)
* PR2 (Timer 2 modules Period Register)
* CCPR1L (CCP Register 1 Low)

Prescaler is set by making the bit T2CKPS0 as high and T2CKPS1 as low the bit TMR2ON is set to start the timer.



The desired frequency has to be set in the PR2 register, the formulae is , Here, \_XTAL\_Freq is 20Mhz



**Procedure:**

*Step 1: -*Open the proteus simulator and select the components required and connect the circuit as per the circuit diagram**.**

*Step 2: -*After successful connection of the circuit, open MPLab IDE to program the controller.

*Step 3: -*Select the configuration bit, desired operation Capture/Compare/PWM. From the configuration steps

*Step 4: -* Write the ISR handler for Compare and Capture

*Step 5: -*. Configure the CCPCON register from the data sheet for desired output

**CAPTURE MODE OPERATION**

In this experiment the Timer 1 is configured in counter mode and is incremented by input pulse for a push button. When the CCP1(RC2) is driven HIGH (Rising edge), the TMR1 counts are written to the Port B (blinks the led in the 8bit binary sequence)

**Code:**

/\*

\* File: capture.c

\* Author: Guga

\*/

// 'C' source line config statements

// CONFIG

#pragma config FOSC = HS

#pragma config WDTE = OFF

#pragma config PWRTE = OFF

#pragma config BOREN = ON

#pragma config LVP = OFF

#pragma config CPD = OFF

#pragma config WRT = OFF

#pragma config CP = OFF

#include <xc.h>

#define \_XTAL\_FREQ 20000000 //XTAL crystal FREQ

void main(void)

{

// PORTB as out for Capture Operation (CCPR1 register)

TRISB = 0x00; // Set output

PORTB = 0x00; // Initial State

// PORTD as out for TIMER Operation (TMR1 register)

TRISD = 0x00; // Set output

PORTD = 0x00; // Initial State

// Initializing Timer one in Counter mode

TMR1 = 0;

T1CKPS0 = 0;

T1CKPS1 = 0;

TMR1CS = 1;

T1OSCEN = 1;

T1SYNC = 0;

TMR1ON = 1;

// Initializing CCP1 module for capture operation

CCP1M0 = 1;

CCP1M1 = 0;

CCP1M2 = 1;

CCP1M3 = 0;

// Enable CCP1 Interrupt

CCP1IE = 1;

PEIE = 1;

GIE = 1;

// Create The Main Loop Of The System

while (1)

{

// Read & Print Out The TMR1 Counts

PORTD = TMR1;

// On rise of interrupt ISR will be executed here

}

return;

}

// ISR Handler

void \_\_interrupt() isr(void)

{

if (CCP1IF)

{

// when Capture event Occurs, the CCPR1 register's value is written to PORTB

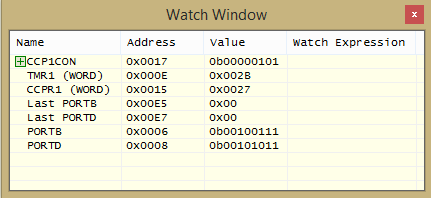
PORTB = CCPR1;

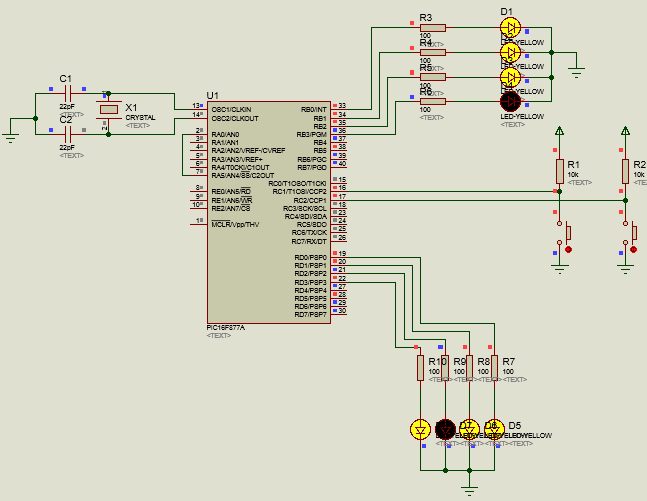
CCP1IF = 0;

}

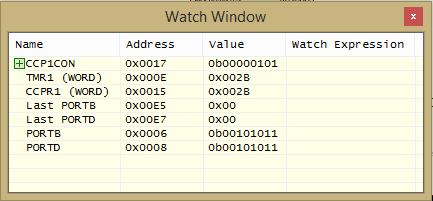
}

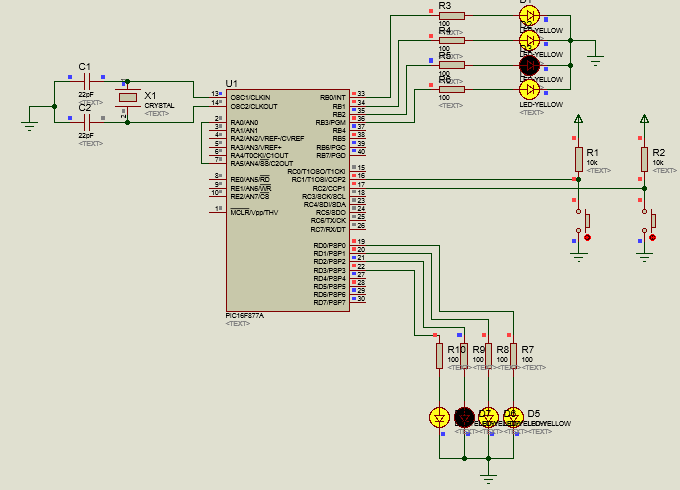
**OUTPUT**





**On rising edge is driven HIGH**





**COMPARE MODE OPERATION**

In this experiment, the timer1 is used to generate a square wave of 50Hz with 50% duty cycle. The Ton off the square wave is 10ms (which is approximated to 50,000 counts from 0) and the Toff is 10ms again. The CCPR1 register is written with the value 50000 (0xC350). An interrupt is raised whenever the Timer 1 value matches the CCPR1 register and the LED is toggled (ideally every 10ms the LED is toggled)

/\*

\* File: compare.c

\* Author: Gopika

\*

\* Created on 15 December, 2021, 5:36 PM

\*/

// CONFIG

#pragma config FOSC = HS

#pragma config WDTE = OFF

#pragma config PWRTE = OFF

#pragma config BOREN = ON

#pragma config LVP = OFF

#pragma config CPD = OFF

#pragma config WRT = OFF

#pragma config CP = OFF

#include <xc.h>

#define \_XTAL\_FREQ 20000000 //XTAL crystal FREQ

void main(void)

{

// PORTC as out for Compare Operation (CCPR1 register)

PORTC = 0;

TRISC = 0;

//Setting up CCPCON register and enabling interrupt

CCP1CON = 11;

CCP1IE = 1;

GIE = 1;

PEIE = 1;

// Compare Values written to CCPR1 register

CCPR1H = 0xC3;

CCPR1L = 0x50;

//CCPR1 = 0xC350 = 50000

// Initializing Timer 1

T1CON = 0x01; //Prescaler 1:1, start Timer 1

while(1){

//Do whatever else is required

}

}

void \_\_interrupt() isr(void)

{

if (CCP1IF)

{

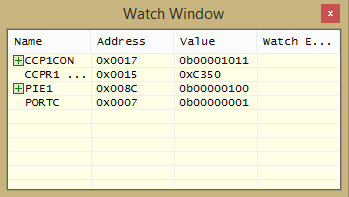
RC0 = ~RC0; // Toggle Port RC0 on match event

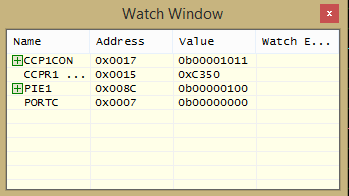
CCP1IF = 0; // Clearing the Register Flag

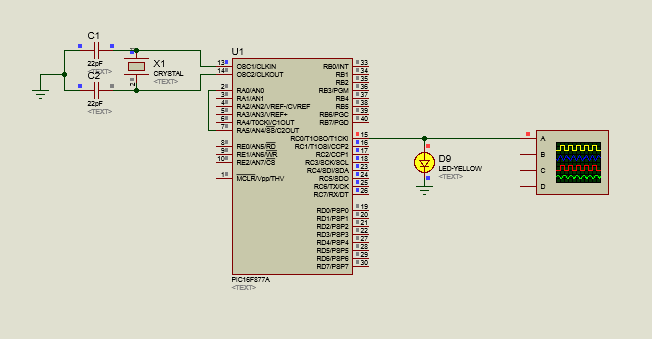
}

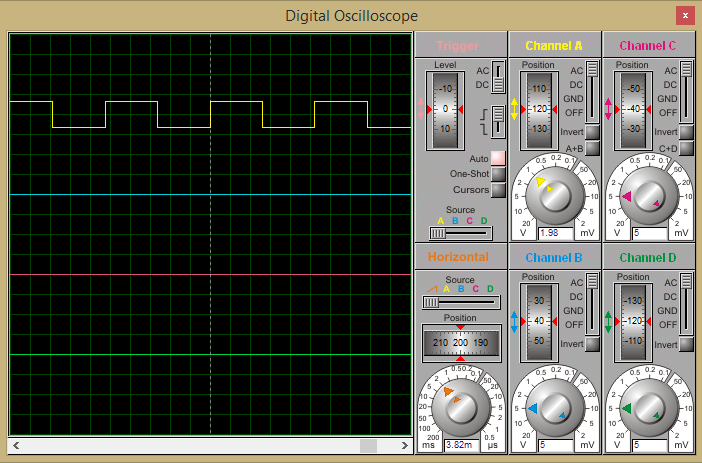
}

**OUTPUT**









**PWM MODE OPERATION**

In this experiment CCP module is configured to operate in PWM mode, Analog voltage of 0-5v from a potentiometer which is connected to ADC port of the controller is read and mapped to 0-1024 using our ADC module. PWM signal with frequency 5000Hz is generated and its duty cycle is varied based on the input Analog voltage. That is 0-1024 will be converted to 0%-100% Duty cycle by writing the value to the CCP1 register.

**Code:**

// CONFIG

#pragma config FOSC = HS

#pragma config WDTE = OFF

#pragma config PWRTE = ON

#pragma config BOREN = OFF

#pragma config LVP = ON

#pragma config CPD = OFF

#pragma config WRT = OFF

#pragma config CP = OFF

#define \_XTAL\_FREQ 20000000

#define TMR2PRESCALE 4

#include <xc.h>

long PWM\_freq = 5000;

PWM\_Initialize()

{

PR2 = (\_XTAL\_FREQ/(PWM\_freq\*4\*TMR2PRESCALE)) - 1;

CCP1M3 = 1; CCP1M2 = 1; //Configure the CCP1 module

T2CKPS0 = 1;T2CKPS1 = 0; TMR2ON = 1;

//Configure the Timer module

TRISC2 = 0; // make port pin on C as output

}

PWM\_Duty(unsigned int duty)

{

if(duty<1023)

{

duty = ((float)duty/1023)\*(\_XTAL\_FREQ/(PWM\_freq\*TMR2PRESCALE)); // On reducing //duty = (((float)duty/1023)\*(1/PWM\_freq)) / ((1/\_XTAL\_FREQ) \* TMR2PRESCALE);

CCP1X = duty & 1; //Store the 1st bit

CCP1Y = duty & 2; //Store the 0th bit

CCPR1L = duty>>2;// Store the remining 8 bit

}

}

void ADC\_Initialize()

{

ADCON0 = 0b01000001; //ADC ON and Fosc/16 is selected

ADCON1 = 0b11000000; // Internal reference voltage is selected

}

unsigned int ADC\_Read(unsigned char channel)

{

ADCON0 &= 0x11000101;

ADCON0 |= channel<<3;

\_\_delay\_ms(2);

GO\_nDONE = 1;

while(GO\_nDONE); //Wait for A/D Conversion to complete

return ((ADRESH<<8)+ADRESL); //Returns Result

}

void main()

{

int adc\_value;

TRISC = 0x00; //PORTC as output

TRISA = 0xFF; //PORTA as input

TRISD = 0x00;

ADC\_Initialize(); //Initializes ADC Module

PWM\_Initialize(); //This sets the PWM frequency of PWM1

do

{

adc\_value = ADC\_Read(4); //Reading Analog Channel 0

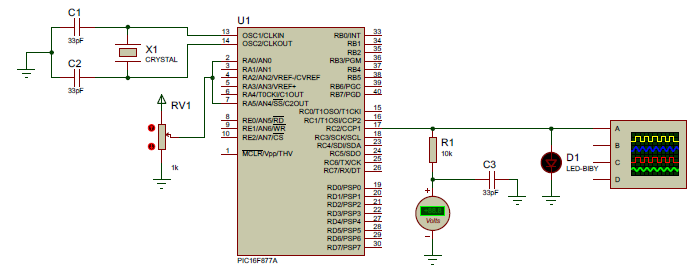
PWM\_Duty(adc\_value);

\_\_delay\_ms(50);

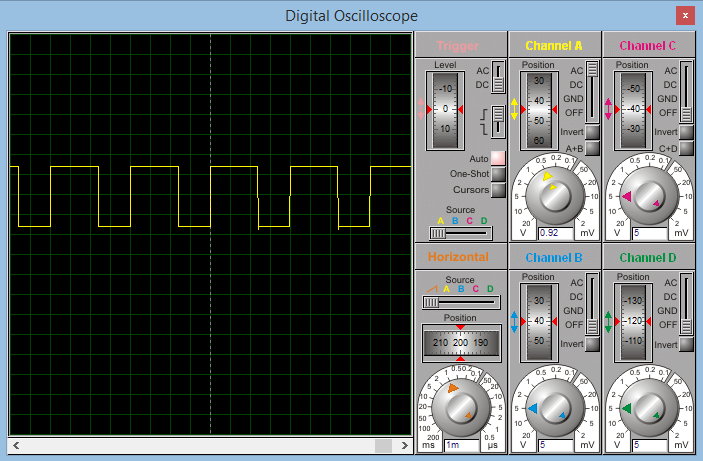
}while(1); //Infinite Loop

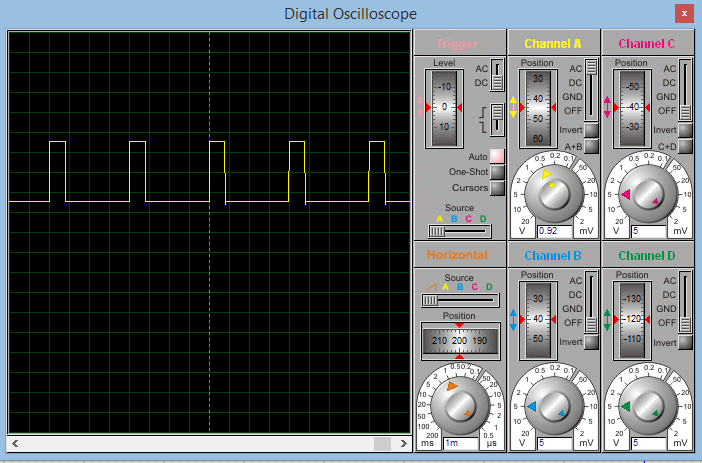
}

**Circuit Diagram:**



**PWM Waveform:**





**RESULT:**

The CCP module of PIC16F877A is studied and verified for the operations of Compare/ Capture/ PWM modes.