EXPERIMENT 4

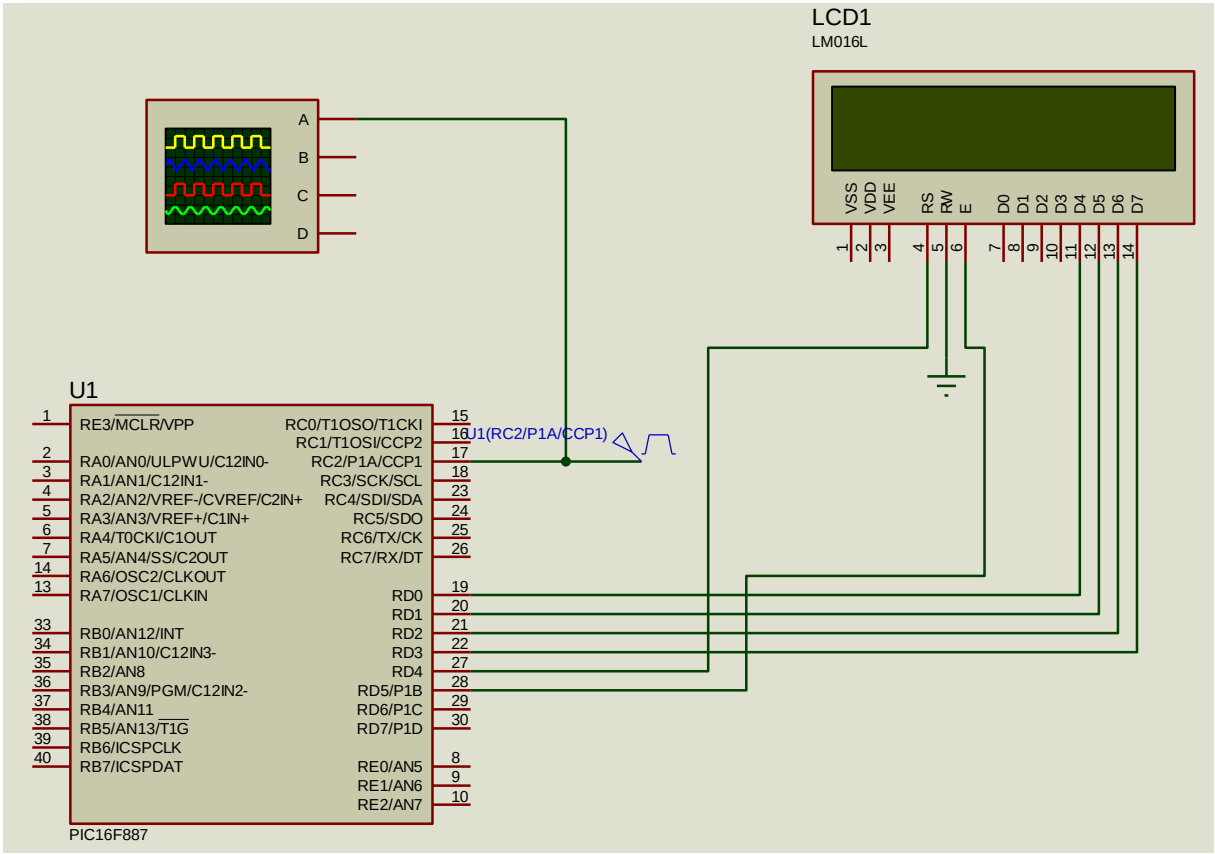
DESIGN OF A FREQUENCY METER

USING THE CAPTURE AND INTERRUPT PROPERTIES OF PIC 16F887

Preliminary Work:

1) Study the information given at the end of this document to learn how to configure the SFR's to be able to use the interrupt feature of the PIC16F887 microcontroller and also to learn how to use the Capture feature of the CCP1 module.

2) Draw the following circuit diagram using the Proteus Simulator



3) Open a new project called CaptureInterrupt and write the following MikroC code using the MikroC pro for PIC IDE.

```
sbit LCD_RS at RD4_bit;
sbit LCD_EN at RD5_bit;
sbit LCD_D4 at RD0_bit;
sbit LCD_D5 at RD1_bit;
sbit LCD_D6 at RD2_bit;
sbit LCD_D7 at RD3_bit;
sbit LCD_RS_Direction at TRISD4_bit;
sbit LCD_EN_Direction at TRISD5_bit;
sbit LCD_D4_Direction at TRISD0_bit;
```

```
sbit LCD_D5_Direction at TRISD1_bit;
sbit LCD_D6_Direction at TRISD2_bit;
sbit LCD_D7_Direction at TRISD3_bit;
 char txt[7];
 unsigned int period, frequency;
void main() {
   ANSEL = 0x00;
   ANSELH=0;
   CCP1IF_bit=0;    //capture1 interrupt flag is reset
   TRISC2_bit=1;    // RC2 is assigned as  input for CCP1
   TRISD=0x00;      // PortD is assigned a output to drive the LCD
   PORTD=0x00;
   CCP1CON=0x05;    // CCP1 module is configured to capture every rising edge at the RC2 input
   INTCON.GIE=INTCON.PEIE=1;  //GIE & PEIE are enabled to generate interrupt at rising edges
   PIE1.CCP1IE=1;   //CCP1 interrupt is enabled
   TMR1L=0x00;   //TMR1 is reset
   TMR1H=0x00;
   T1CON=0x01;   //Timer1 is started
    Lcd_Init();
    Lcd_Cmd(_LCD_CURSOR_OFF);
    Lcd_Out(1,1,"Period=");  // Print  Period = starting from the first column of the first line
    Delay_ms(200);
  while(1){
   IntToStr(period, txt);     //Period is converted from int to string to dislay on the LCD
   Lcd_Out(2,1,txt); // Print the value of the Period startingfrom the first column of the second row
   }
}
void interrupt(){
   CCP1IF_bit=0;   // Reset CCP1 interrupt flag
   period = CCPR1;   //current value of Timer1 when therising edge arrives is saved in period
   TMR1L=0x00;     //Timer1 is reset to strt timing for the period of the next pulse
   TMR1H=0x00;
 }
```

4) Read the period of the square waveform on the LCD, by changing the input frequency of the pulse generatur using its edit menu. Compare the value of the period displayed on the LCD with the period that you measure on the oscilloscope.

5) Draw the circuit diagram of Example 4 on page 241 of the pdf book using the proteus simulator, write the Timer0 interrupt programs given in this example and run the programs on your simulator.

Conclusions:

1) Which instructions are used to configure the Capture interrupt on CCP1?
2) Why do you have to reset the CCP1IF interrupt flag in the interrupt routine?
3) Why do we use the capture feature to measure the period of the pulses?
4) Explain what you have learned from this experiment.
5) In which applications can you use the circuits and techniques that you have learned during this experiment?
6) What were the problems that you have faced with during this experimental work?

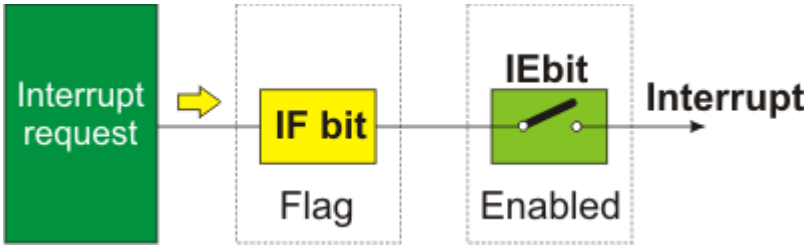
INTERRUPT SYSTEM REGISTERS

Most programs use interrupts in their regular execution. The purpose of the microcontroller is mainly to respond to changes in its surrounding. In other words, when an event takes place, the microcontroller does something... For example, when you push a button on a remote controller, the microcontroller will register it and respond by changing a channel, turn the volume up or down etc. If the microcontroller spent most of its time endlessly checking a few buttons for hours or days, it would not be practical at all.

This is why the microcontroller has learnt a trick during its evolution. Instead of checking each pin or bit constantly, the microcontroller delegates the 'wait issue' to a 'specialist' which will respond only when something attention worthy happens.

The signal which informs the central processor unit about such an event is called an INTERRUPT.

When an interrupt request arrives, it doesn't mean that an interrupt will automatically occur, because it must also be enabled by the user (from within the program). Because of this, there are special bits used to enable or disable interrupts. It is easy to recognize them by the letters IE contained in their names (stands for *Interrupt Enable*). Besides, each interrupt is associated with another bit called the flag which indicates that an interrupt request has arrived regardless of whether it is enabled or not. They are also easily recognizable by the last two letters contained in their names- IF (*Interrupt Flag*).



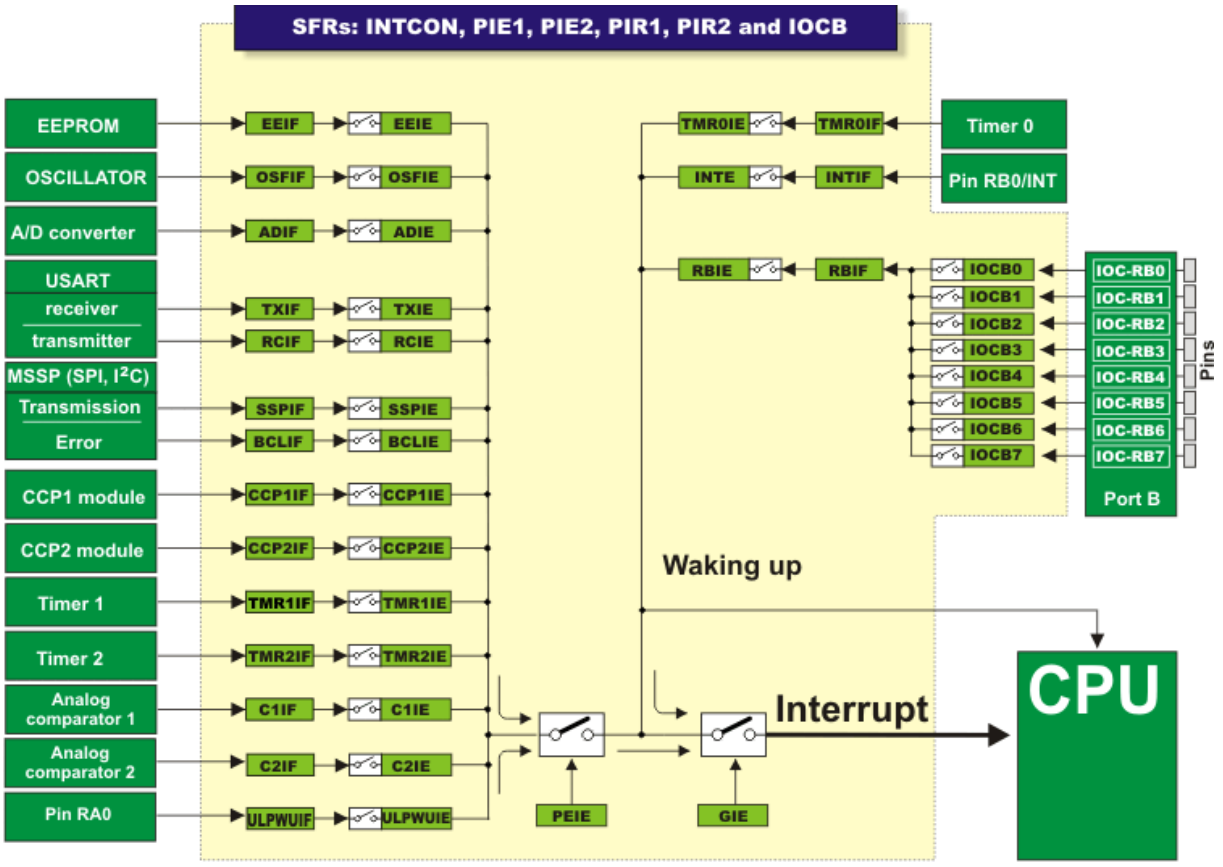
As seen, everything is based on a simple and efficient idea. When an interrupt request arrives, the flag bit is set first.

If the appropriate IE bit is not set (0), this condition will be completely ignored. Otherwise, an interrupt occurs! If several interrupt sources are enabled, it is necessary to detect the active one before the interrupt routine starts execution. Source detection is performed by checking flag bits.

It is important to know that the flag bits are not automatically cleared, but by software while the interrupt routine execution is in progress. If we neglect this detail, another interrupt will occur immediately after returning to the main program, even though there are no more requests for its execution. Simply put, the flag, as well as the IE bit, remain set.

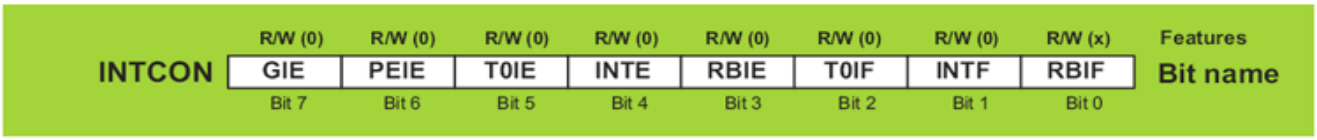
All interrupt sources typical of the PIC16F887 microcontroller are shown on the next page. Note several things:

The **GIE** bit enables all unmasked interrupts and disables all interrupts simultaneously. The **PEIE bit** enables all unmasked peripheral interrupts and disables all peripheral interrupts.



INTCON Register

The INTCON register contains various enable and flag bits for TMR0 register overflow, PORTB change and external INT pin interrupts.



- **GIE - Global Interrupt Enable bit** - controls all possible interrupt sources simultaneously.
  - **1** - Enables all unmasked interrupts.
  - **0** - Disables all interrupts.

- **PEIE - Peripheral Interrupt Enable bit** acts similar to the GIE it, but controls interrupts enabled by peripherals. It means that it has no impact on interrupts triggered by the timer TMR0 or by changing the state of PORTB or the RB0/INT pin.
  - **1** - Enables all unmasked peripheral interrupts.
  - **0** - Disables all peripheral interrupts.

PIE1 Register

The PIE1 register contains peripheral interrupt enable bits

- **CCP1IE - CCP1 Interrupt Enable bit** enables an interrupt request to be generated in CCP1 module used for PWM signal processing.
    - **1** - Enables the CCP1 interrupt.
    - **0** - Disables the CCP1 interrupt.

PIR1 Register

- The PIR1 register contains the interrupt flag bits.

- **CCP1IF - CCP1 Interrupt Flag bit.**
    - **1** - CCP1 interrupt condition has occurred (CCP1 is unit for capturing, comparing and generating PWM signal). Depending on operating mode, capture or compare match has occurred. In both cases, bit must be cleared in software. This bit is not used in PWM mode.
    - **0** - No CCP1 interrupt condition has occurred.

3.7 CCP MODULES

The CCP module (*Capture/Compare/PWM*) is a peripheral which allows the user to time and control different events.

Capture Mode provides access to the current state of a register which constantly changes its value. In this case, it is the timer TMR1 register.

Compare Mode constantly compares values of two registers. One of them is the timer TMR1 register. This circuit also allows the user to trigger an external event when a predetermined amount of time has expired.

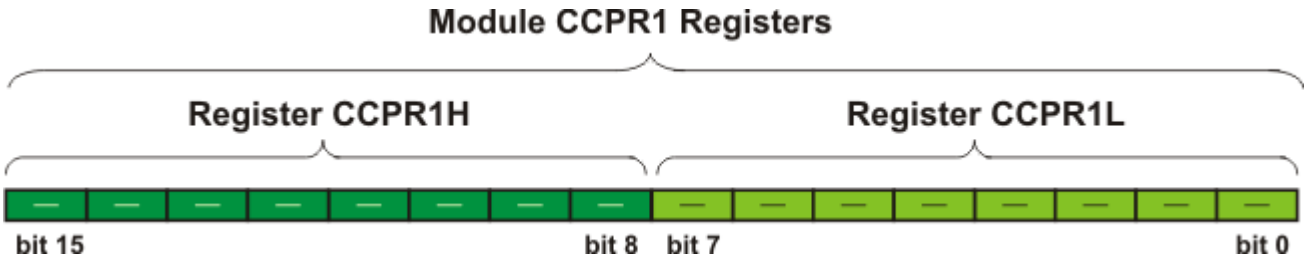
PWM (*Pulse Width Modulation*) can generate signals of varying frequency and duty cycle on one or more output pins.

The PIC16F887 microcontroller has two CCP modules- CCP1 and CCP2.

Both of them are identical in normal mode of operation, while the Enhanced PWM features are available on CCP1 only. This is why this chapter gives a detailed description of the CCP1 module. Concerning CCP2, only the features distinguishing it from CCP1 will be covered.

CCP1 MODULE

A central part of this circuit is a 16-bit register CCPR1 which consists of the CCPR1L and CCPR1H registers. It is used for capturing or comparing with binary numbers stored in the timer register TMR1 (TMR1H and TMR1L).



If enabled by software, the timer TMR1 reset may occur on match in Compare mode. Besides, the CCP1 module can generate PWM signals of varying frequency and duty cycle.

Bits of the CCP1CON register are in control of the CCP1 module.

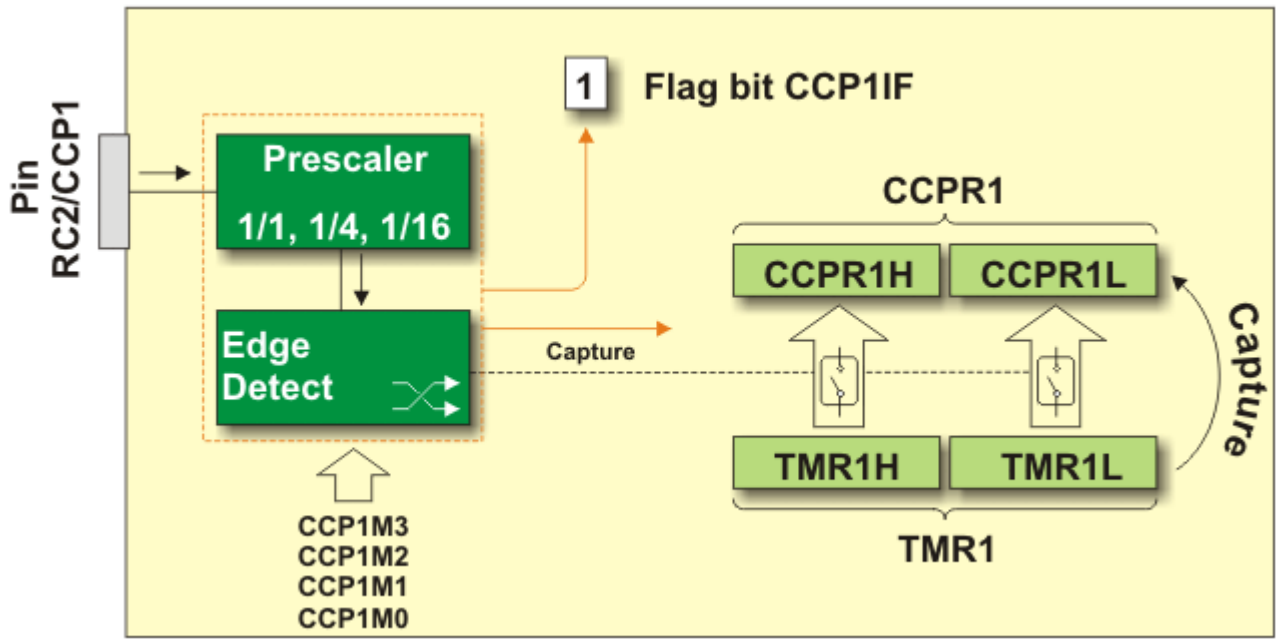
CCP1 IN CAPTURE MODE

In this mode, the timer register TMR1 (consisting of TMR1H and TMR1L) is copied to the CCP1 register (consisting of CCPR1H and CCPR1L) in the following situations:

- Every falling edge (1 -> 0) on the RC2/CCP1 pin;
- Every rising edge (0 -> 1) on the RC2/CCP1 pin;
- Every 4th rising edge (0 -> 1) on the RC2/CCP1 pin; and
- Every 16th rising edge (0 -> 1) on the RC2/CCP1 pin.

A combination of the four bits (CCP1M3 - CCP1M0) of the control register determines which of these events will cause 16-bit data to be transfered. In addition, the following conditions must be met:

- The RC2/CCP1 pin must be configured as an input; and
- TMR1 module must operate as timer or synchronous counter.



The flag bit CCP1IF is set when capture is made. If the CCP1IE bit of the PIE1 register is set when it happens, an interrupt occurs.

When the CCP1 module exits the capture mode, an unwanted capture interrupt may be generated. In order to avoid this, both a bit enabling CCP1IE interrupt and flag bit CCP1IF should be cleared prior to any change occurs in the control register.

CCP1CON 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)	Features
CCP1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Legend	
R/W	Readable/Writable bit
(0)	After reset, bit is cleared

P1M1, P1M0 - PWM Output Configuration bits - In all modes, except for PWM, the P1A pin is *Capture/Compare* module input. P1B, P1C and P1D pins act as input/output port D pins. In PWM mode, these bits affect the operation of the CCP1 module as shown in table below:

P1M1	P1M0	Mode
		PWM with single output
0	0	Pin P1A outputs modulated signal. Pins P1B, P1C and P1D are port D input/output
		Full Bridge - Forward configuration
0	1	Pin P1D outputs modulated signal Pin P1A is active

		Pins P1B and P1C are inactive
		Half Bridge configuration
1	0	Pins P1A and P1B output modulated signal PinsP1C and P1D are port D input/output
		Full Bridge - Reverse configuration
1	1	Pin P1B outputs modulated signal Pin P1C is active Pins P1A and P1D are inactive

DC1B1, DC1B0 - PWM Duty Cycle Least Significant bits - are only used in PWM mode in which they represent two least significant bits of a 10-bit number. This number determines duty cycle of the PWM signal. The rest of bits (8 in total) are stored in the CCPR1L register.

CCP1M3 - CCP1M0 - CCP1 Mode Select bits determine the mode of the CCP1 module.

CCP1M3	CCP1M2	CCP1M1	CCP1M0	Mode
0	0	0	0	Module is disabled (reset)
0	0	0	1	Unused
0	0	1	0	**Compare mode** CCP1IF bit is set on match
0	0	1	1	Unused
0	1	0	0	**Capture mode** Every falling edge on the CCP1 pin
0	1	0	1	**Capture mode** Every rising edge on the CCP1 pin
0	1	1	0	**Capture mode** Every 4th rising edge on the CCP1 pin
0	1	1	1	**Capture mode** Every 16th rising edge on the CCP1 pin
1	0	0	0	**Compare mode** Output and CCP1IF bit are set on match
1	0	0	1	**Compare mode** Output is cleared and CCP1IF bit is set on match
1	0	1	0	**Compare mode** Interrupt request arrives and bit CCP1IF is set on match
1	0	1	1	**Compare mode** Bit CCP1IF is set and timers 1 or 2 registers are cleared
1	1	0	0	**PWM mode** Pins P1A and P1C are active-high Pins P1B and P1D are active-high
1	1	0	1	**PWM mode**

				Pins P1A and P1C are active-high Pins P1B and P1D are active-low
				PWM mode
1	1	1	0	Pins P1A and P1C are active-low Pins P1B and P1D are active-high
				PWM mode
1	1	1	1	Pins P1A and P1C are active-low Pins P1B and P1D are active-low