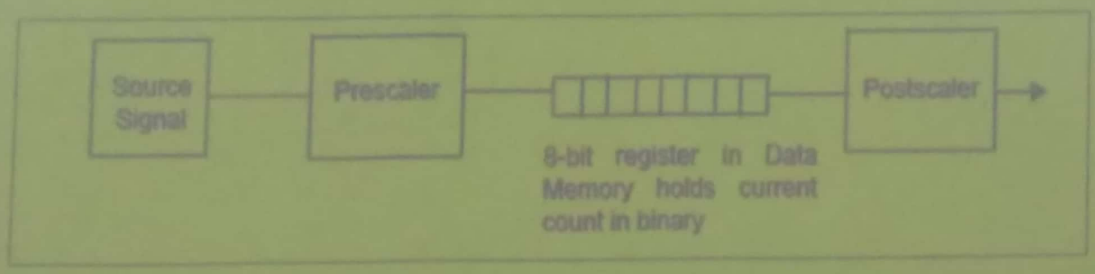
PIC 16F877A Timer Modules

Timer	Size	Control Register	Count Register	Min Delay	Max Delay
TIMER0	8-bit	OPTION_REG	TMR0	0.2usec	13.107ms
TIMER1	16-bit	T1CON	TMR1H,TMR1L	0.2usec	104.857ms
TIMER2	8-bit	T2CON	TMR2	0.2usec	819usec



Timer Calculation

$$Reg\_Value = TimerMax - \frac{Delay * F_{osc}}{Prescalr * 4}$$

Timer	Size	Formula for delay calculation
TIMER0	8-bit	Reg_Value = 256-((Delay * Fosc)/(Prescalar*4))
TIMER1	16-bit	Reg_Value = 65536-((Delay * Fosc)/(Prescalar*4))
TIMER2	8-bit	Reg_Value = 256-((Delay * Fosc)/(Prescalar*4))

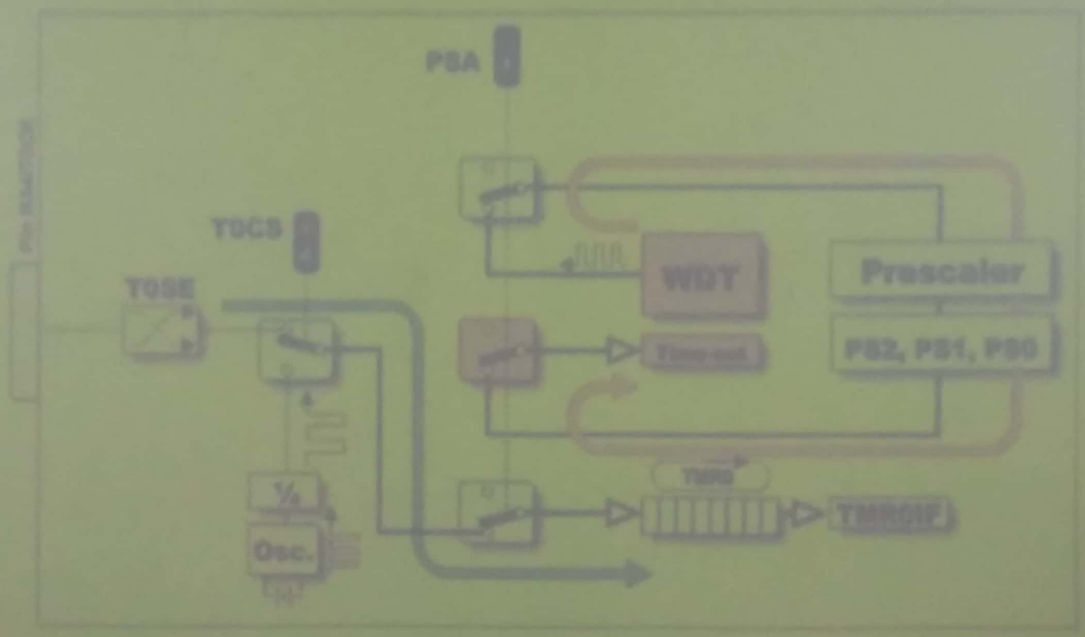
Watchdog Timer

What and Why is a Watchdog Timer?

Prescale for WDT

Bits (2,1,0)	Rate	WDT Time
0,0,0	1:1	18mS
0,0,1	1:2	36mS
0,1,0	1:4	72mS
0,1,1	1:8	144mS
1,0,0	1:16	288mS
1,0,1	1:32	576mS
1,1,0	1:64	1.1Seconds
1,1,1	1:128	2.3Seconds

When PSA bit is set, prescaler is assigned to watchdog timer as shown



TIMER1 MODULE

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 is latched in interrupt flag bit, TMR1IF (PIR1<0>).
- This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

- As a Timer
- As a Counter

- The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).
- In Timer mode, Timer1 increments every instruction cycle.
- In Counter mode, it increments on every rising edge of the external clock input.

- ✓ Timer1 can be enabled/disabled by setting/clearing control bit, TMR1ON (T1CON<0>).
- ✓ Timer1 also has an internal "Reset input".
- ✓ This Reset can be generated by either of the two CCP modules (**Capture/Compare/PWM Modules**)
- ✓ When the Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI/CCP2 and RC0/T1OSO/T1CKI pins become inputs.

Registers Associated With Timer1 as a Timer/Counter

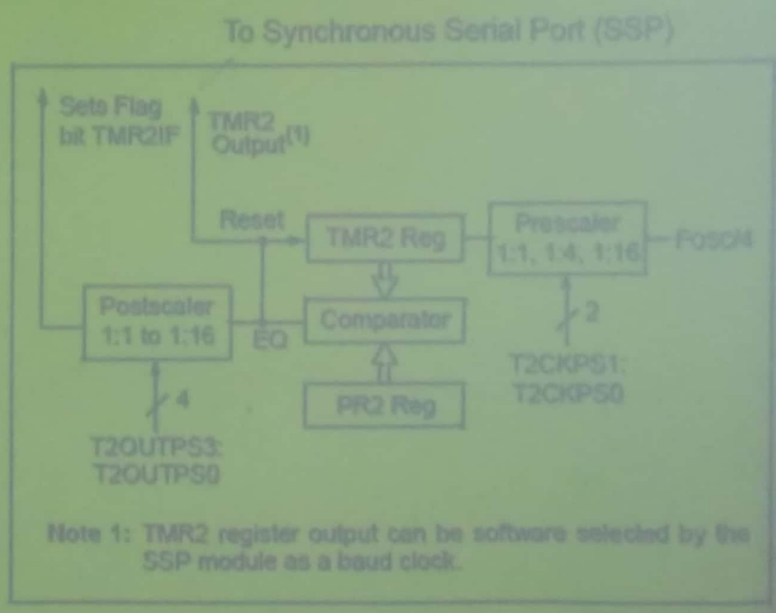
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
[illegible]	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
[illegible]	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
[illegible]	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu

[illegible]: x = unknown, u = unchanged, – = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.
Bits PSPIE and PSPIF are reserved on the 28-pin devices; always maintain these bits clear.

Timer2 Module

- Timer2 is an 8-bit timer with a prescaler and a postscaler.
- It can be used as the PWM time base for the PWM mode of the CCP module(s).
- TMR2 register is readable and writable and is cleared on any device Reset.
- The input clock (FOSC/4) has a prescale option of 1:1, 1:4 or 1:16, selected by control bits T2CKPS1:T2CKPS0 (T2CON<1:0>).
- The Timer2 module has an 8-bit period register, PR2.

- Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle.
- PR2 is a readable and writable register.
- The PR2 register is initialized to FFh upon Reset.
- The match output of TMR2 goes through a 4-bit postscaler (which gives a 1:1 to 1:16 scaling inclusive) to generate a TMR2 interrupt (latched in flag bit, MR2IF (PIR1<1>)).
- ner2 can be shut-off by clearing trol bit, TMR2ON (T2CON<2>



Timer2 Block Diagram

T2CON: TIMER2 Control Register (Address 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits

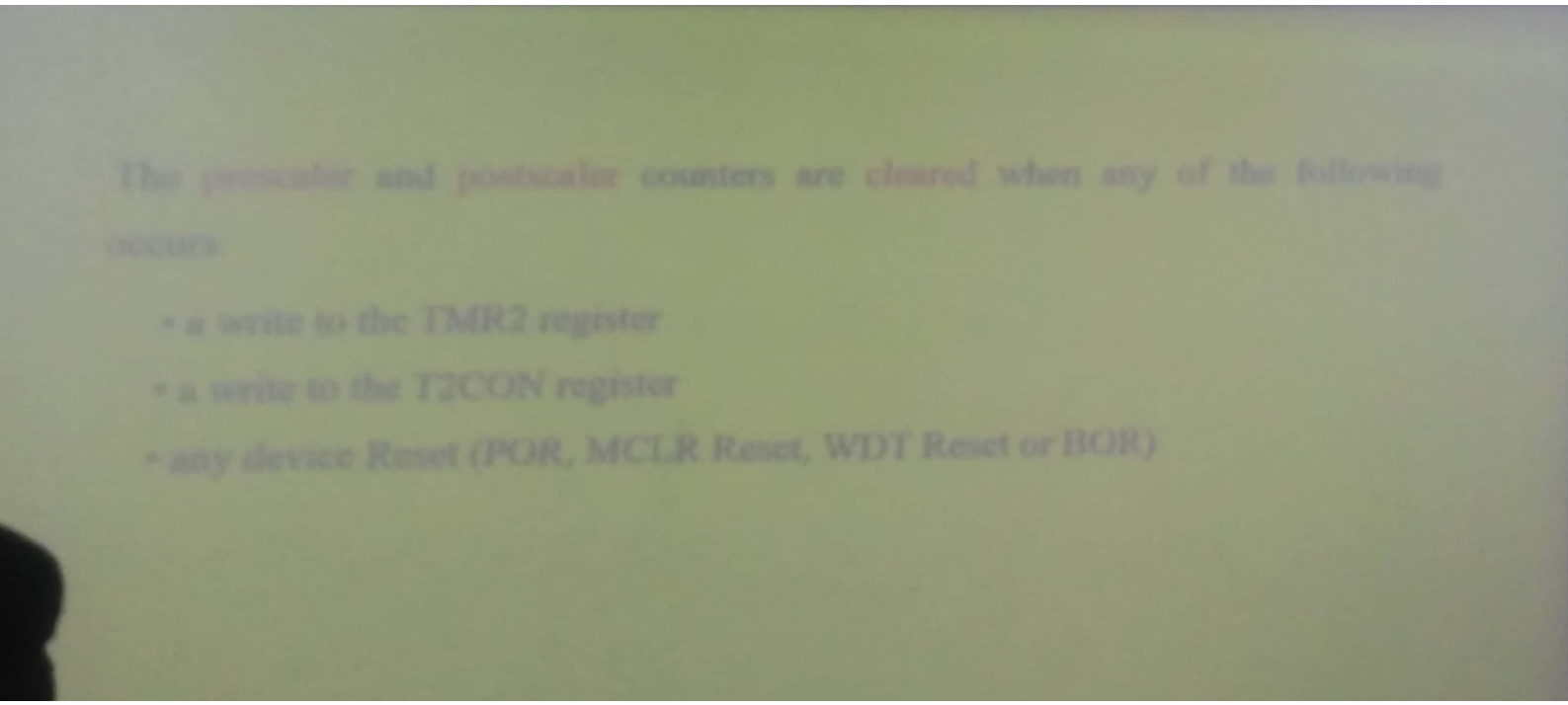
TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	Postscaler Ratio
0	0	0	0	1:1
0	0	0	1	1:2
0	0	1	0	1:3
0	0	1	1	1:4
0	1	0	0	1:5
0	1	0	1	1:6
0	1	1	0	1:7
0	1	1	1	1:8
1	0	0	0	1:9
1	0	0	1	1:10
1	0	1	0	1:11
1	0	1	1	1:12
1	1	0	0	1:13
1	1	0	1	1:14
1	1	1	0	1:15
1	1	1	1	1:16

bit 2 **TMR2ON**: Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

T2CKPS1	T2CKPS0	Prescaler Rate
0	0	1:1
0	1	1:4
1	1	1:16



Capture/Compare/PWM 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. 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 (Based on TMR2).

Each Capture/Compare/PWM (CCP) module contains a 16-bit register which can operate as a:

- 16-bit Capture register
- 16-bit Compare register
- PWM Master/Slave Duty Cycle register

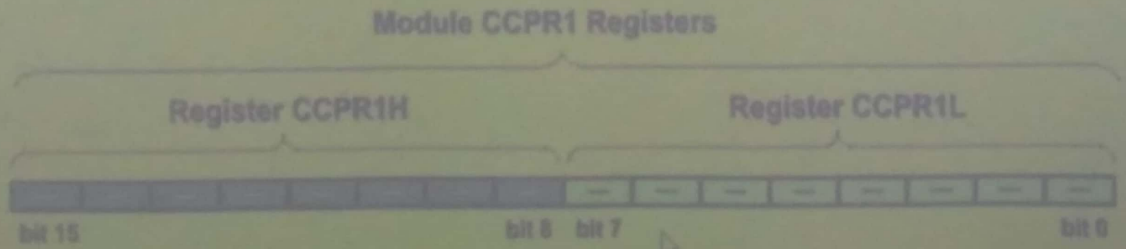
The PIC16F877 microcontroller has two CCP modules- CCP1 and CCP2.

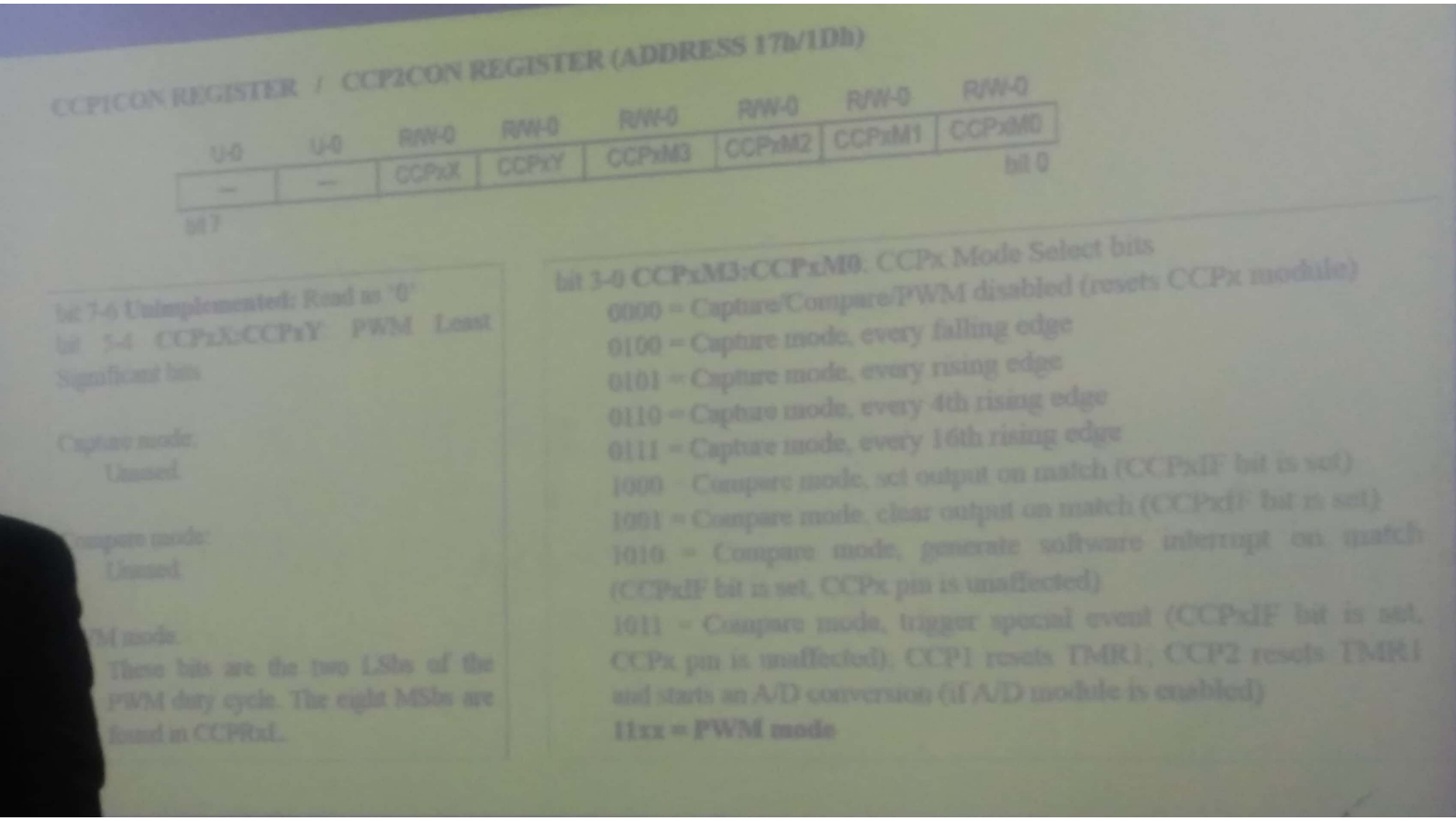
CCP Mode – Timer Resources Required

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

CCP1 Module:

Capture/Compare/PWM Register 1 (CCPR1) is comprised of two 8-bit registers: CCPR1L (low byte) and CCPR1H (high byte). The CCP1CON register controls the operation of CCP1. The special event trigger is generated by a compare match and will reset Timer1.





CCP1CON REGISTER / CCP2CON REGISTER (ADDRESS 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 Unimplemented: Read as '0'

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
0000 = Capture/Compare/PWM disabled (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

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:

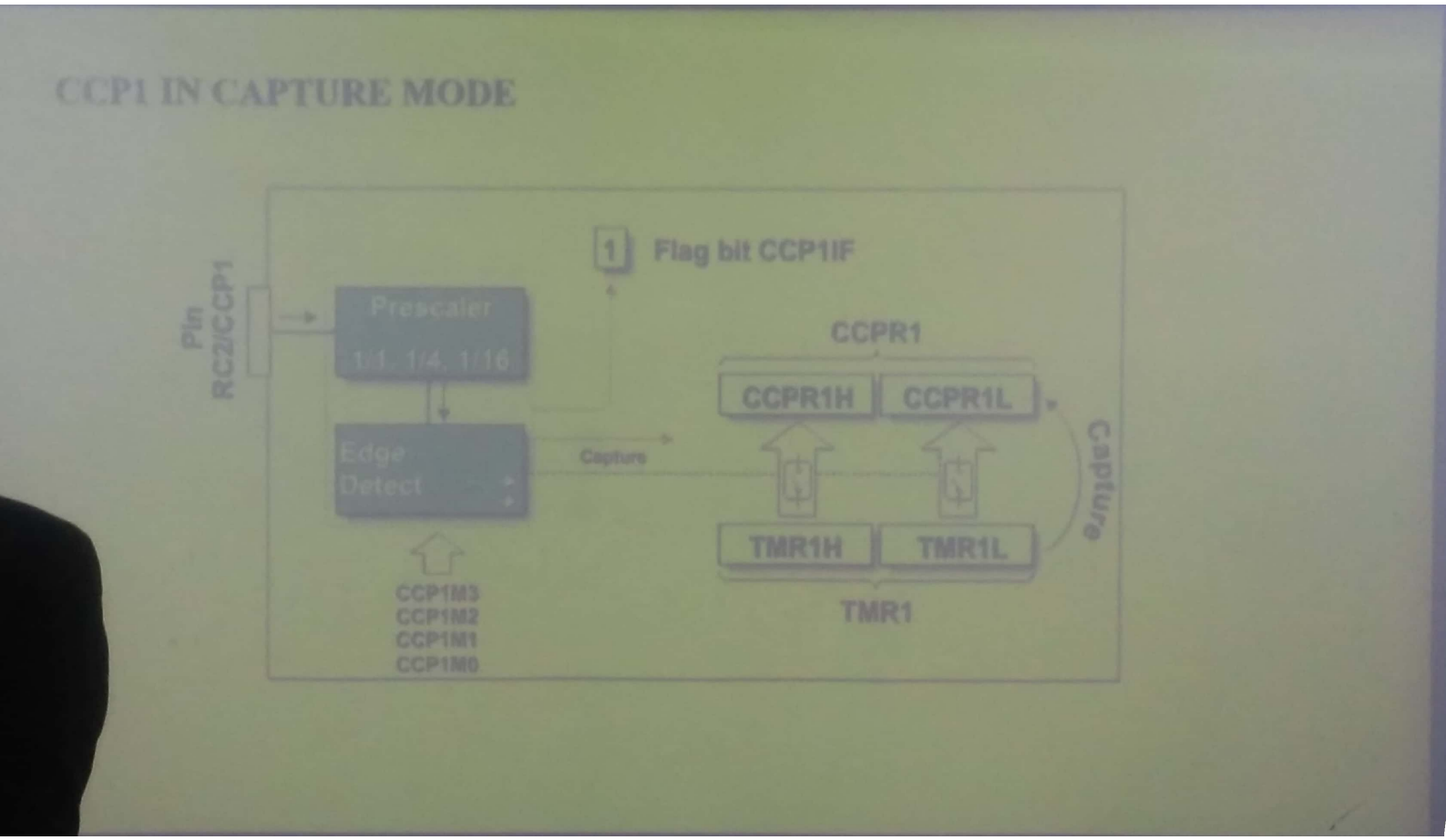
- Every falling edge (1 → 0) on the RC2/CCP1 pin;
- Every rising edge (0 → 1) on the RC2/CCP1 pin;
- Every 4th rising edge (0 → 1) on the RC2/CCP1 pin; and
- Every 16th rising edge (0 → 1) on the RC2/CCP1 pin.

A combination of the four bits (CCP1M3 - CCP1M0) of the control register determines which of these events will cause 16-bit data to be transferred.

In addition, the following conditions must be met:

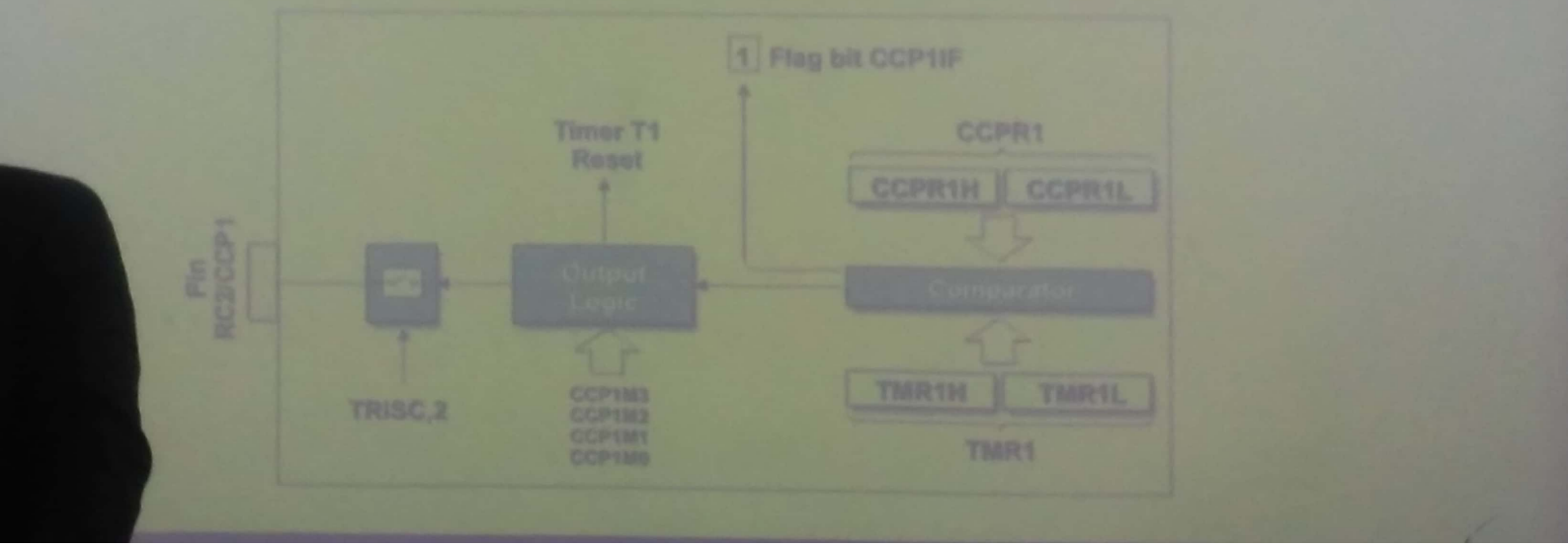
- The RC2/CCP1 pin must be configured as an input; and
- TMR1 module must operate as timer or synchronous counter.

CCP1 IN CAPTURE MODE



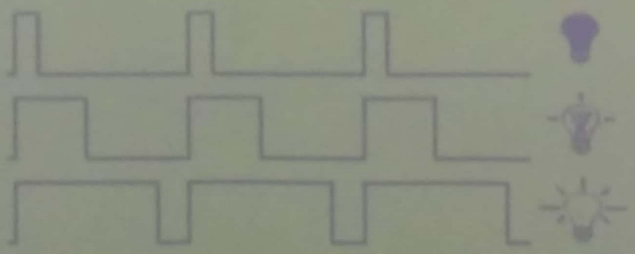
CCP1 IN COMPARE MODE

In this mode, the value stored in the CCP1 register is constantly compared to the value stored in the timer register TMR1. When a match occurs, the logic state of the RC2/CCP1 output pin may be changed, which depends on the state of bits in the control register (CCP1M3 - CCP1M0). The flag-bit CCP1IF will be simultaneously set.

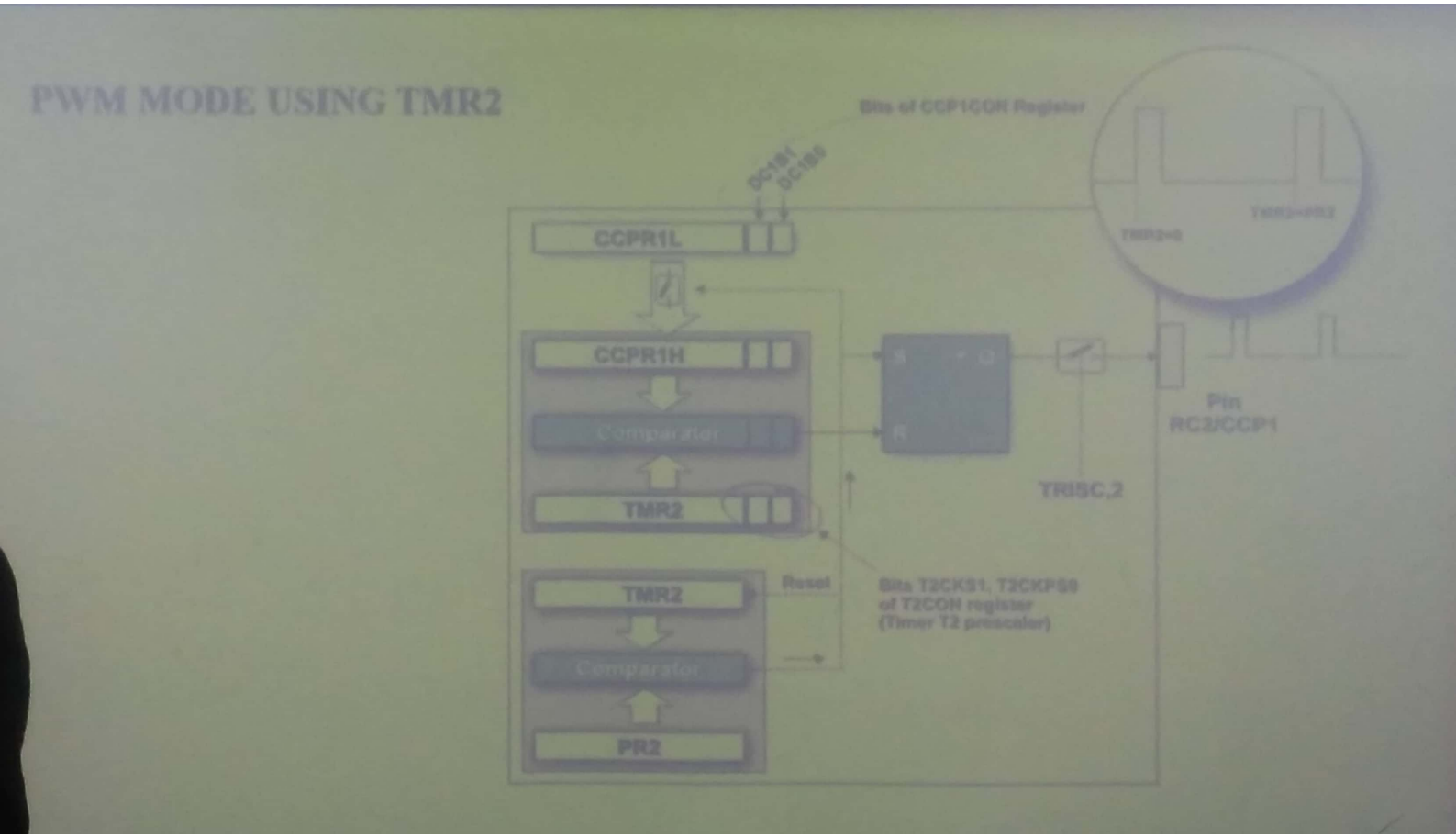


CCP1 IN PWM MODE

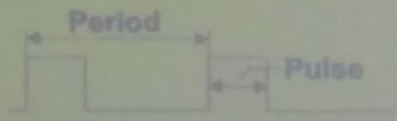
Signals of varying frequency and duty cycle have a wide range of application in automation. A typical example is a power control circuit. Refer to figure below. If a logic zero (0) indicates the switch-off and a logic one (1) indicates the switch-on, the electrical power that load consumers will be directly proportional to the pulse duration. This ratio is often called **Duty Cycle**.



PWM MODE USING TMR2



In order to generate a pulse of arbitrary form on its output pin, it is necessary to set pulse period (frequency) and pulse duration



PWM PERIOD

The output pulse period (T) is determined by the PR2 register of the timer TMR2. The PWM period can be calculated using the following equation:

PWM Period = (PR2 +1) * 4Tosc * TMR2 Prescale Value

If the PWM period (T) is known, then it is easy to determine the signal frequency F because these two values are related by equation F=1/T.

PWM DUTY CYCLE

CCPR1L register and two additional LSbs of the CCP1CON register (DC1B1 and DC1B0). The result is a 10-bit number contained in the formula.

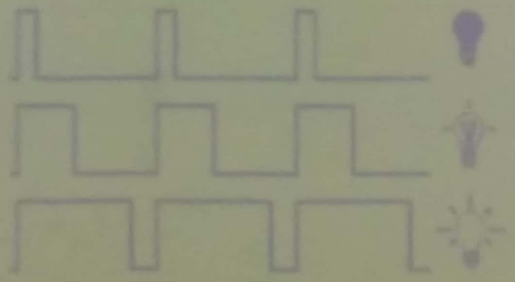
Pulse Width = (CCPR1L,DC1B1,DC1B0) * Tosc * TMR2 Prescale Value

The following table shows how to generate PWM signals of varying frequency if the microcontroller uses 20 MHz quartz-crystal (Tosc=50nS).

Frequency [KHz]	1.22	4.88	19.53	78.12	156.3	208.3
TMR2 Prescaler	16	4	1	1	1	1
PR2 Register	FFh	FFh	FFh	3Fh	1Fh	17h

How to use TIMER2 to Generate PWM (Pulse Width Modulation)

Signals of varying frequency and duty cycle have a wide range of application in automation. A typical example is a power control circuit. Refer to figure below.



There are two pins on PORTC that used to generate PWM signal (RC1 = (PWM2) ..... RC2 = (PWM1))

PWM Library in MikroC

Library Routines

- PWMx_Init : PWM1_Init(5000); // 5KHz Frequancy
- PWMx_Set_Duty : PWM1_Set_Duty(192); // Set duty ratio to 75% ((Percent*255)/100)
- PWMx_Start : PWM1_Start();
- PWMx_Stop : PWM1_Stop();

Example to generate a PWM on RC2 (PWM2)
Change Duty cycle from (0% : 100% with step 10/255

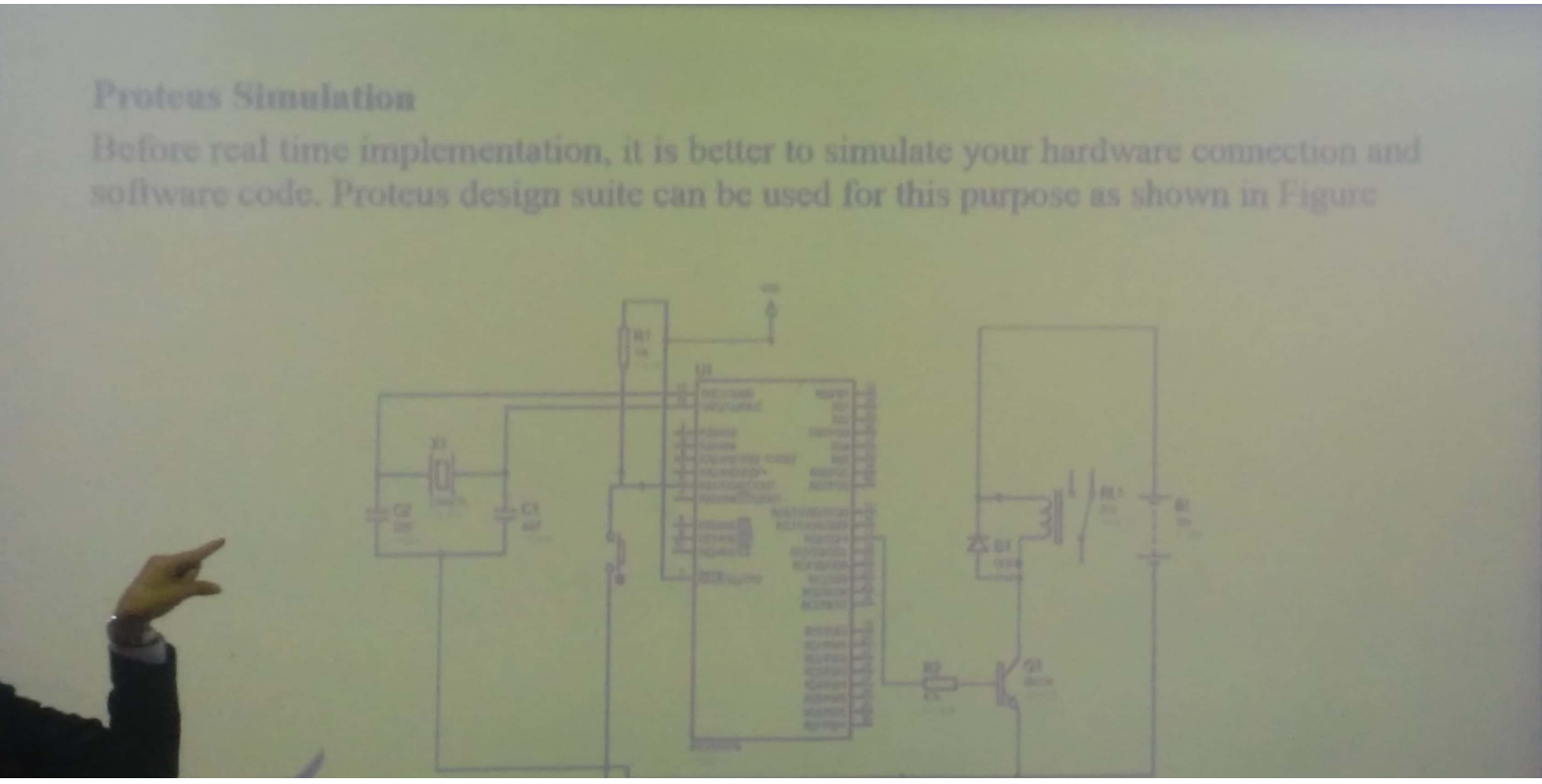
```
unsigned short int a;
void main()  {
 PORTC = 0;        // set PORTC to 0
 TRISC = 0;        // designate PORTC pins as output
 PWM2_Init (1000);  // Initialize PWM2 module at 1KHz

 PWM2_Start();          // start PWM2
 PWM2_Set_Duty (255);    // Set current duty cycle to 100%


 while (1) {                  // endless loop
 // 100% duty cycle:  duty ratio can be calculated as (Percent*255)/100
 //Change Duty cycle from (0% : 100% with step 10/255)

 for (a=0; a<=255; a=a+10)
{
 PWM2_Set_Duty (a);      // Change the duty cycle
 delay_ms(50);           // 50 ms delay
                  }
}
}
```

Example to generate a PWM on RC2 (PWM2)
Change Duty cycle from (0% : 100% with step 10/255

```
unsigned short int a;
void main()  {
 PORTC = 0;        // set PORT
 TRISC = 0;        // designate
 PWM2_Init (1000);  // initi

 PWM2_Start();          // st
 PWM2_Set_Duty (255);

 while (1) {                  //
 // 100% duty cycle:  duty ra
 //Change Duty cycle from (

 for (a=0; a<=255; a=a+10)
{
 PWM2_Set_Duty (a);      //
 delay_ms(50);           //
                }
}
}
```

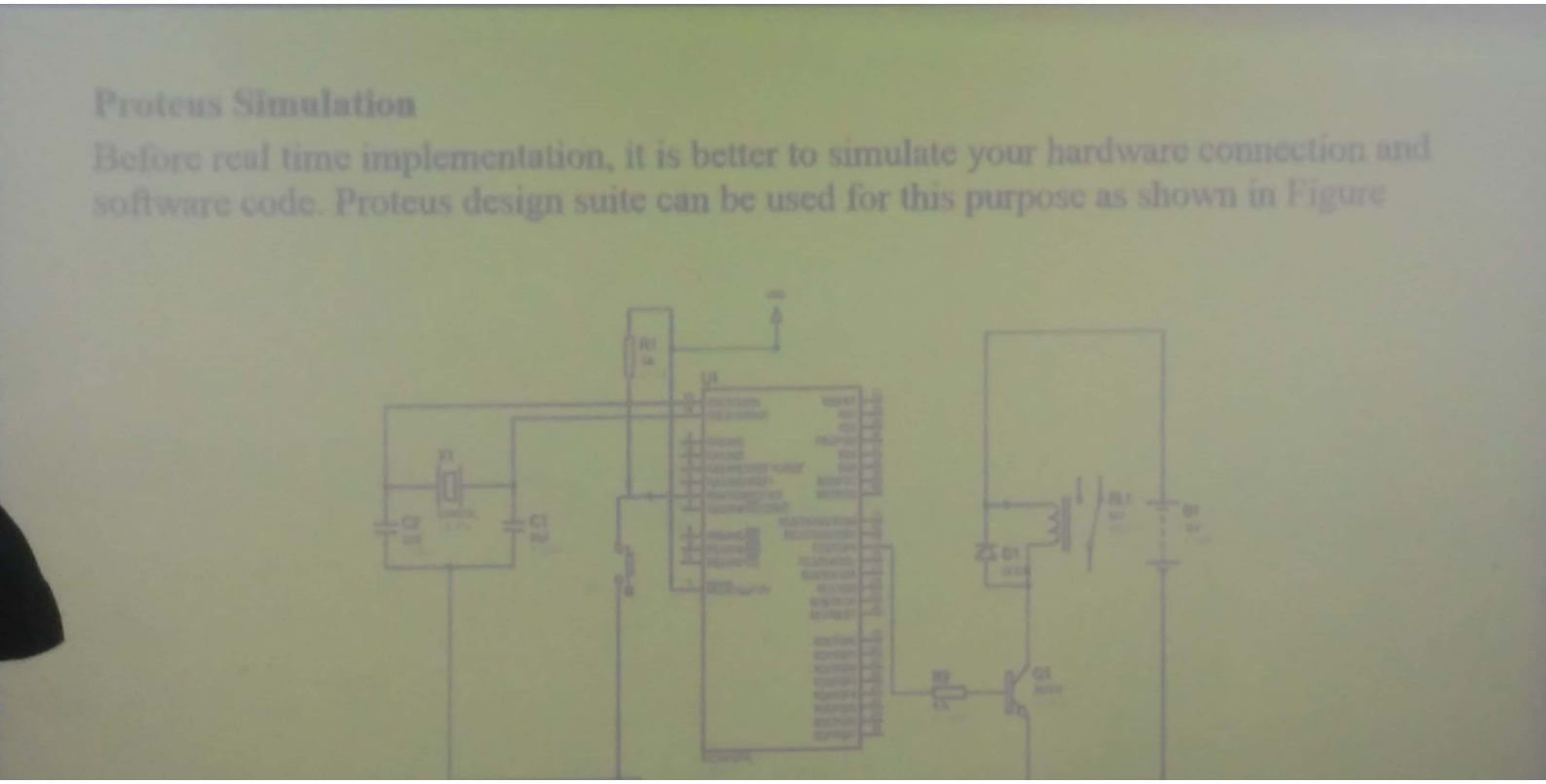
Proteus Simulation

Before real time implementation, it is better to simulate your hardware connection and software code. Proteus design suite can be used for this purpose as shown in Figure



Proteus Simulation

Before real time implementation, it is better to simulate your hardware connection and software code. Proteus design suite can be used for this purpose as shown in Figure



MikroC Code

```
#define TEST 5
void main()
{
TRISC = PORTC = 0;
TRISA4_bit = 1;                    // RA4/T0CKI input
OPTION_REG = 0b00111000;           // Prescaler (1:1), T0CS =1 for counter mode, Falling Edge
TMR0=0;
while (1)                          // Remain in endless loop
{
if(TMR0 == TEST)                   // Does the number in timer match constant TEST?
{
        PORTC.F2 = 1;              // Numbers match. Set the RC2 bit (output RELAY)
        TMR0=0;
        delay_ms(3000);            // Keep relay ON for 3 seconds
}
Else                               // Does the number in timer match constant TEST?
{
PORTC.F2 = 0;                      // Numbers do not match. clear the RC2 bit (output RELAY)
}   }
```