ENBA SOPANRAO MOZE COLLEGE OF ENGINEERING, Balewadi, Pune.

INDEX

Expt. No.	Perform. Date	Name of Experiment	Page No	Date	Remarks	Sign
01	10/3/23	Study of Addition of two no.s	01	10/3/23		
02	17/3/23	Study of Array Addition.	02	17/3/23		
03	17/3/23	Study of multiplication of two nos & study of division of two no.s.	03	17/3/23		
04	31/3/23	Interfacing LED's to PORTD of PIC18 µcontroller.	05	31/3/23		
05	28/04/23	· Buzzer ON & OFF using TMR0 interrupt & Relay ON & off using external interrupt	10	28/4/23		
06	28/4/23	Study of LCD interfacing to PIC18 µcontroller.	22	28/4/23		
07	12/5/23	Study of PWM generation using PIC18F4520 to vary speed of DC motor.	26	12/5/23		
08	12/5/23	Study of serial comm" with PC to PC µcontroller.	32	12/5/23		
09	19/5/23	Study of Raspberry Pi / Arduino Board / Beagle Board & operating systems for the same.	37	19/5/23		
10	19/5/23	Study of open-source prototype platform - Raspberry Pi / Beagle Board.	41	19/5/23		

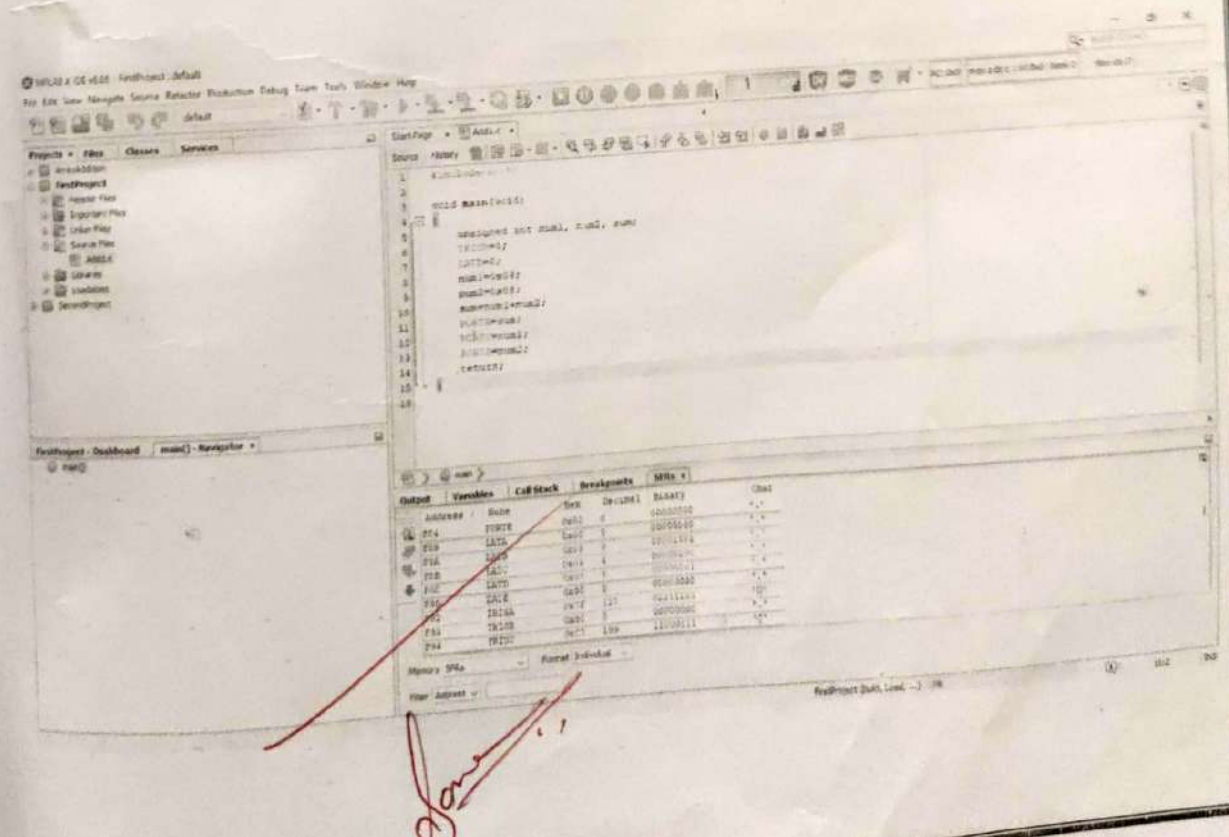
Programming Skill Development Lab

Practical No. 1

Addition of Two Numbers:

```
#include<xc.h>
void main(void)
{  unsigned int num1, num2, sum;
   TRISB=0;
   LATB=0;
   num1=0x04;
   num2=0x05;
   sum=num1+num2;
   PORTB=sum;
   PORTC=num1;
   PORTD=num2;
   return;
}
```

OUTPUT



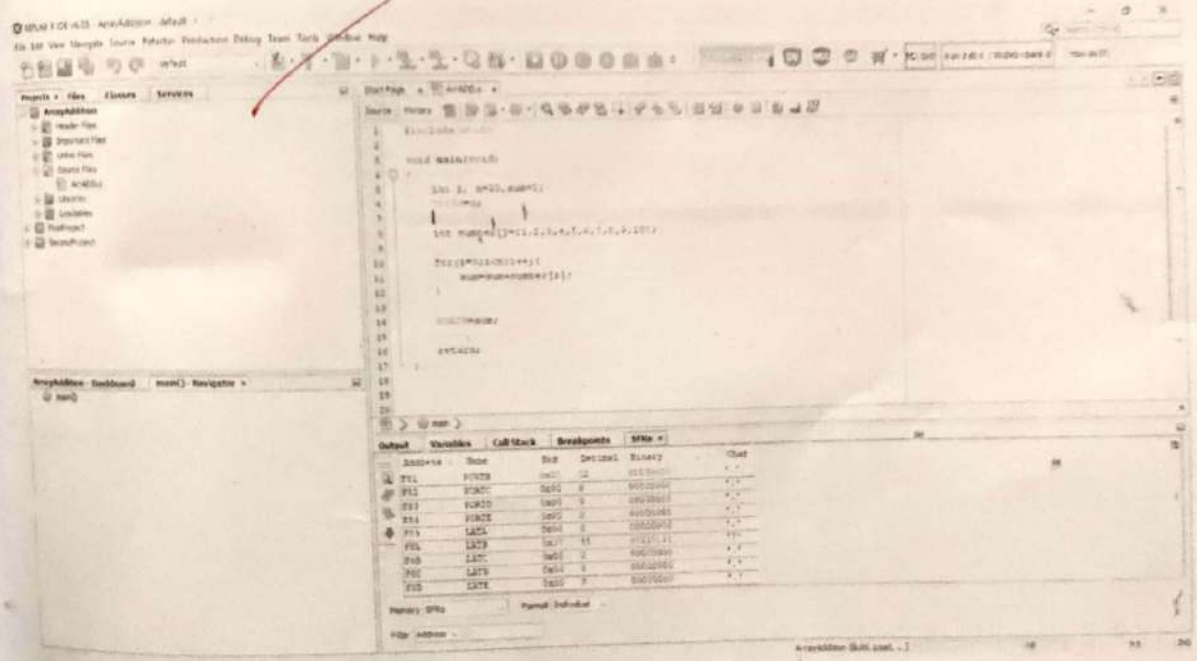
Programming Skill Development Lab

Practical No. 2

Add Array of n Numbers:

```
#include<xc.h>
void main(void)
{  int i, n=10,sum=0;
   TRISB=0;
   int number[]={1,2,3,4,5,6,7,8,9,10};
   for(i=0;i<n;i++){
      sum=sum+number[i];
   }
   PORTB=sum;
   return;
}
```

OUTPUT



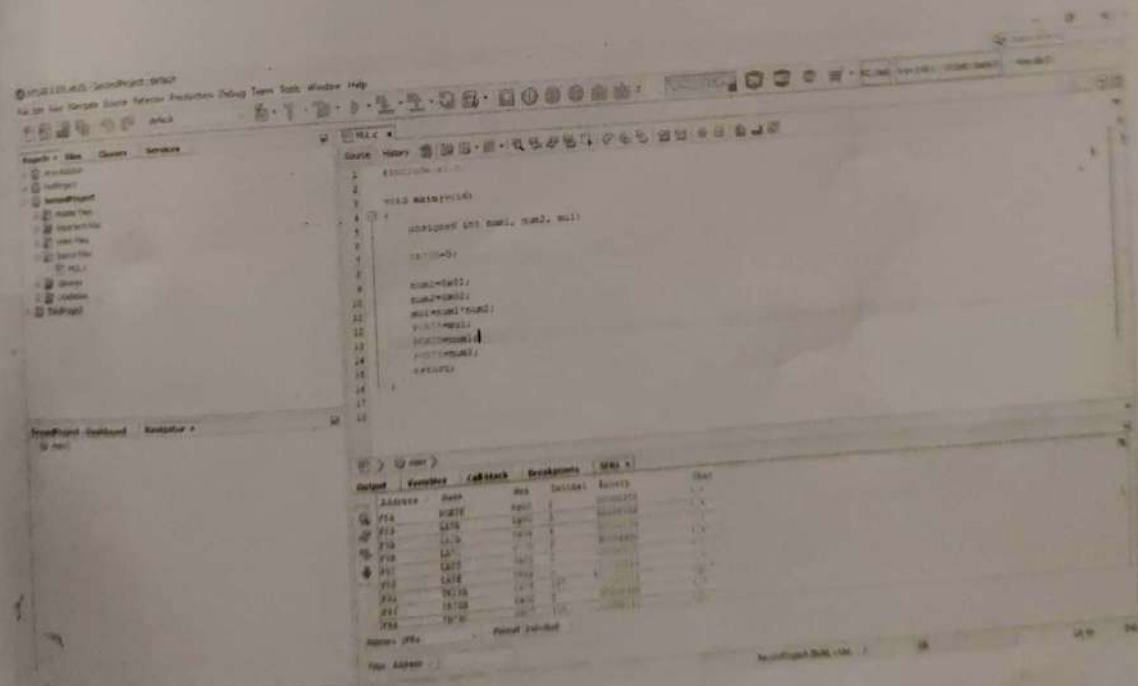
Programming Skill Development Lab

Practical No. 3

1. Multiplication of Two Numbers:

```
#include<xc.h>
void main(void)
{ unsigned int num1, num2, mul;
  TRISB=0;
  num1=0x02;
  num2=0x02;
  mul=num1*num2;
  PORTB=mul;
  PORTD=num1;
  PORTE=num2;
  return;
}
```

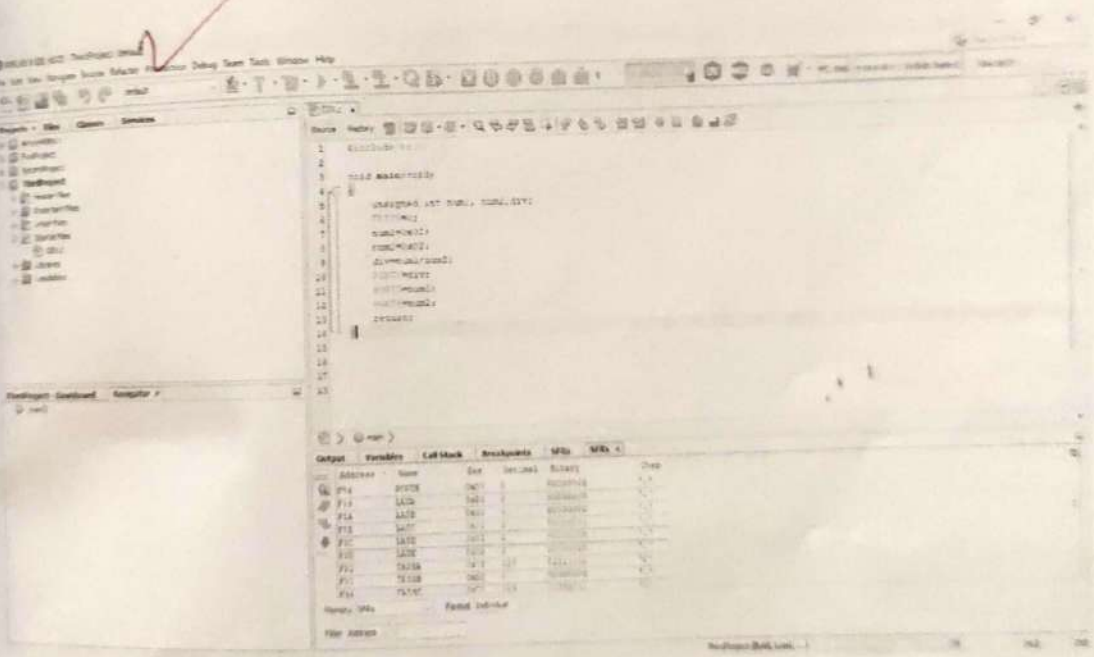
OUTPUT



2. Division of Two Numbers:

```
#include<xc.h>
void main(void)
{  unsigned int num1, num2,div;
   TRISB=0;
   num1=0x02;
   num2=0x02;
   div=num1/num2;
   PORTC=div;
   PORTD=num1;
   PORTE=num2;
   return;
}
```

OUTPUT



Date :
Pg No :

*Experiment no:- 04 *

Title: Interfacing LED's to PIC microcontroller.

Problem statement:
Interface LED's to PIC μcontroller. Write an embedded C program to blink the LED's at specific interval.

Objective:
a) To understand the PORT structure of PIC μcontroller.
b) To study the SFRs to control the PORT pins.
c) To interface common peripherals like LEDs, Relay, Buzzers.
d) To understand the use of MPLAB IDE and C18 compiler.
e) To write a simple program in Embedded C.

S/W Packages & H/W used:
MPLABX IDE / MPLAB IDE, X8 / C18 compiler, PIC Development Board.

Theory:
Depending on the device selected, there are up to five ~~peripherals~~ general purpose I/O ports available in PIC18F μcontroller devices. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

1) Some common features of the I/O ports
- Up to 70 bi-directional I/O pins
  - some multiplexed with peripheral f^ns.
- High drive capability

- 25 mA source/sink capability
- Direct single cycle bit manipula
- 4KV ESD protection diodes.
  - Based on human body model.
- After reset:
  - Digital I/O Default to Input (H
  - Analog capable pins default to

2.) SFR associated with I/O port:

Each port has three registers for its ope and figure @ & ⓑ shows the function of each registers.

① TRIS register : (Data direction reg To select PORT pin as input or output. All p pins are input by default. Whenever a bit TRISx register is 0, the corresponding PORTx is an output. If the bit in TRISx 1 the corresponding bit in PORTx is an

② PORT register :-

Reads the levels on the pins of the dev

③ LAT register : (O/p latch)

The data latch (LAT register) is useful read-modify-write operations on the v that the I/O pins are driving.

All the ports of PIC 18 are bi-direc and identical. They all have the followi components in their structure as show

ⓐ Data latch

Date : Page No.

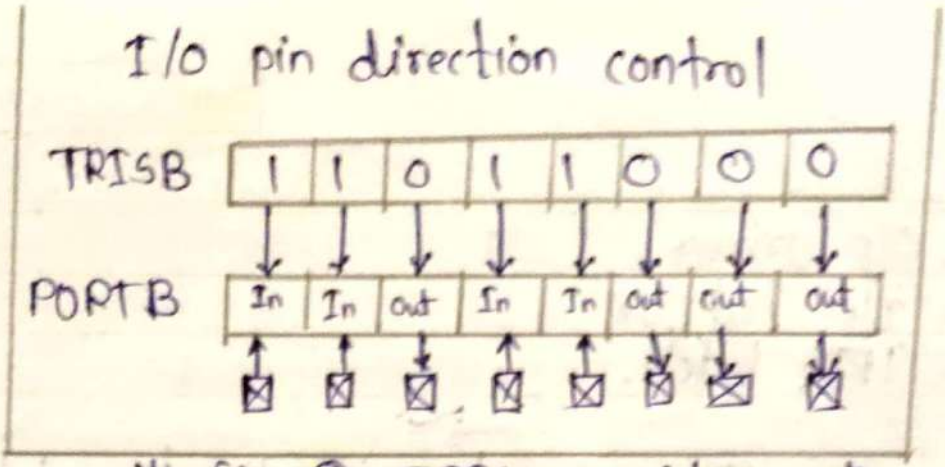
(b) O/p drivers
(c) I/p buffer
(d) TRIS Latch.

The PIC 18 ports have both the latch and buffers. ∴ when reading the ports there are two possibilities.

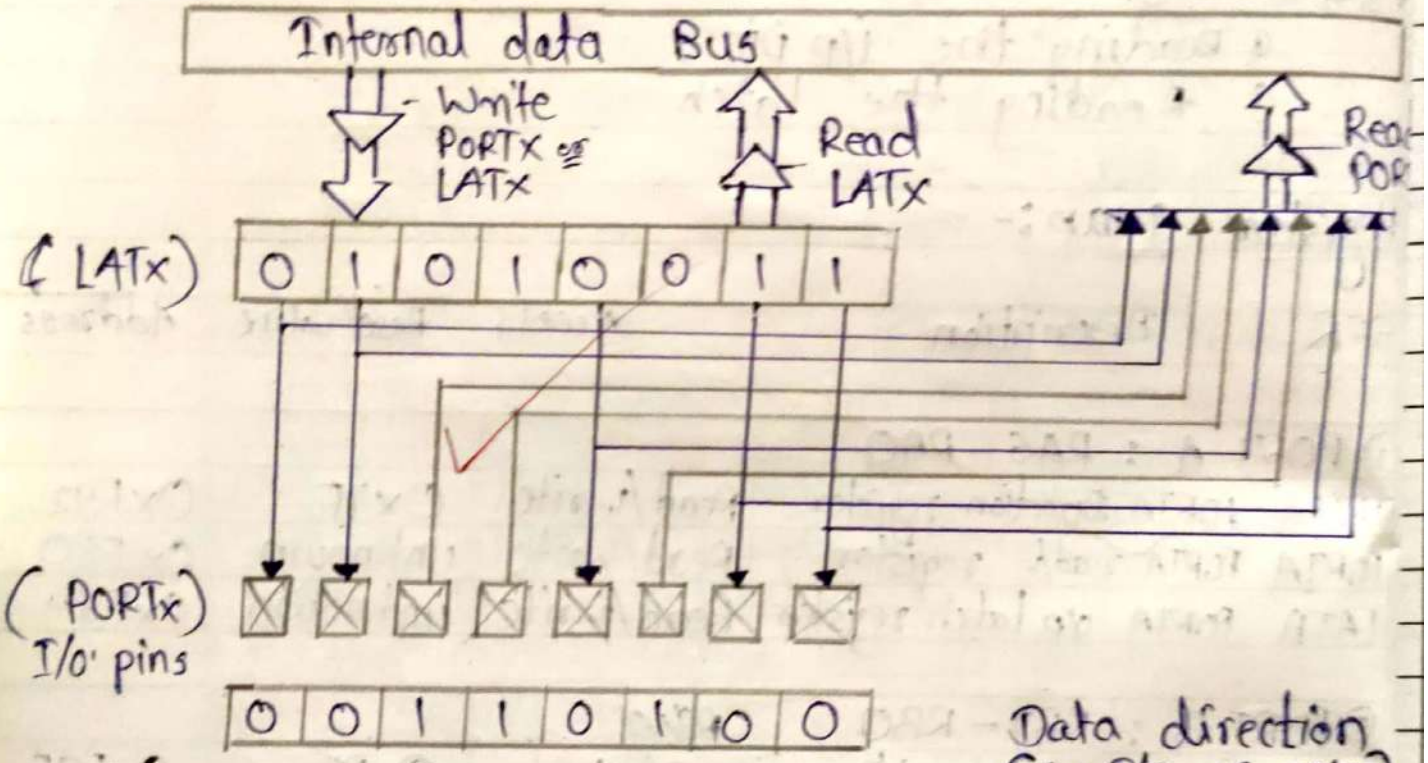
(a) Reading the i/p pin
(b) Reading the latch.

Register Map :-

SFR	Description	Access	Reset value	Address
① PORT A : RA6 - RA0				
TRISA	PORTA Direction register	Read/write	0x7F	0xF92
PORTA	PORTA read/w register	Read/write	unknown	0x F80
LATA	PORTA o/p latch register	Read/write	unknown	0xF89
② PORT B : RB7 - RB0		R/W'		
TRISB	PORTB Direct^n register.	R/W'	0xff	0xF93
PORTB	PORTB r/w register	R/W'	unknown	0xF81
LATB	PORTB o/p latch register.	R/W'	unknown	0xF8A
③ PORT C : RC7 - RC0		R		
TRISC	PORTC Direct^n register	R/W'	0xFF	0xF94
PORTC	PORTc r/w register	R/W'	unknown	0xF82
LATC	PORTC o/p latch register.	R/W'	unknown	0xF8B
④ PORT D : RD7 - RD0				
TRISD	PORTD direct^n register	Read/write	0xFF	0xf95
PORTD	PORTD r/w register	R/W'	unknown	0xF83



* fig. ⓐ TRISx register. *

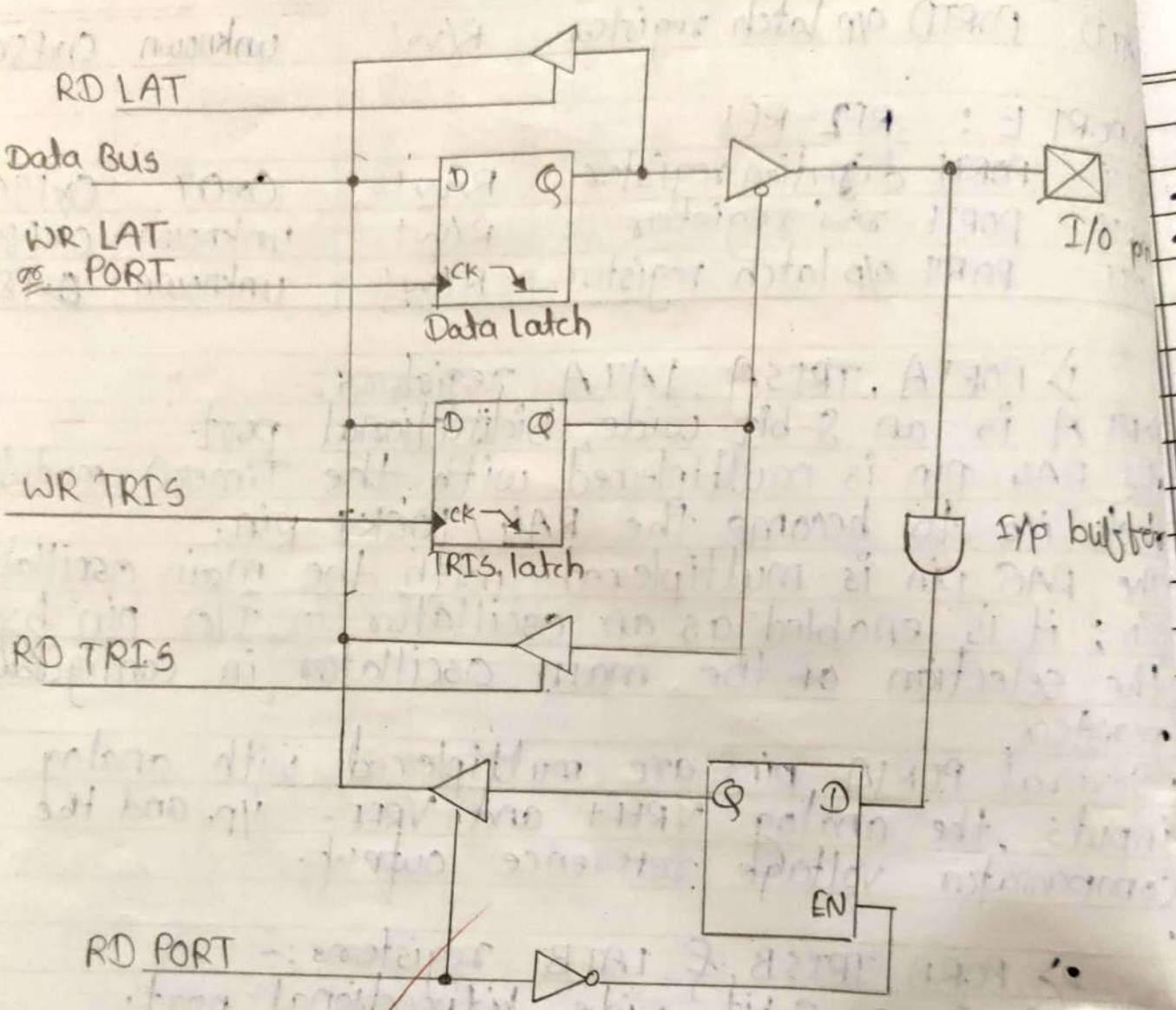


* fig. ⓑ SFRs associated with I/O port

LATD	PORTD o/p latch register	R/W'	unknown	0xF8C
(5) PORT E : RE2-RE1				
TRISE	PORTE direction register	R/W'	0x07	0xF96
PORTE	PORTE r/w' register	R/W'	unknown	0xF84
LATE	PORTE o/p latch register	R/W'	unknown	0xF8D

1> PORTA, TRISA, LATA registers:-
- PORT A is an 8-bit wide, bidirectional port.
- The RA4 pin is multiplexed with the Timer0 module clock i/p to become the RA4/T0CKI pin.
- The RA6 pin is multiplexed with the main oscillator pin; it is enabled as an oscillator or I/O pin by the selection of the main oscillator in configuration register.
- Several PORTA pins are multiplexed with analog inputs, the analog VREF+ and VREF- i/ps and the comparator voltage reference output.

2> PORB, TRISB & LATB registers:-
- PORTB is an 8-bit wide, bidirectional port.
- Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit RBPU (INTCON2 register). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.
- On a Power-on reset, RB4:RB0 are configured as analog i/ps by default and read as '0'; RB7:RB5 are configured as digital i/ps.



* Generic I/O port Operation *

3) PORTC, TRISC & LATC registers:

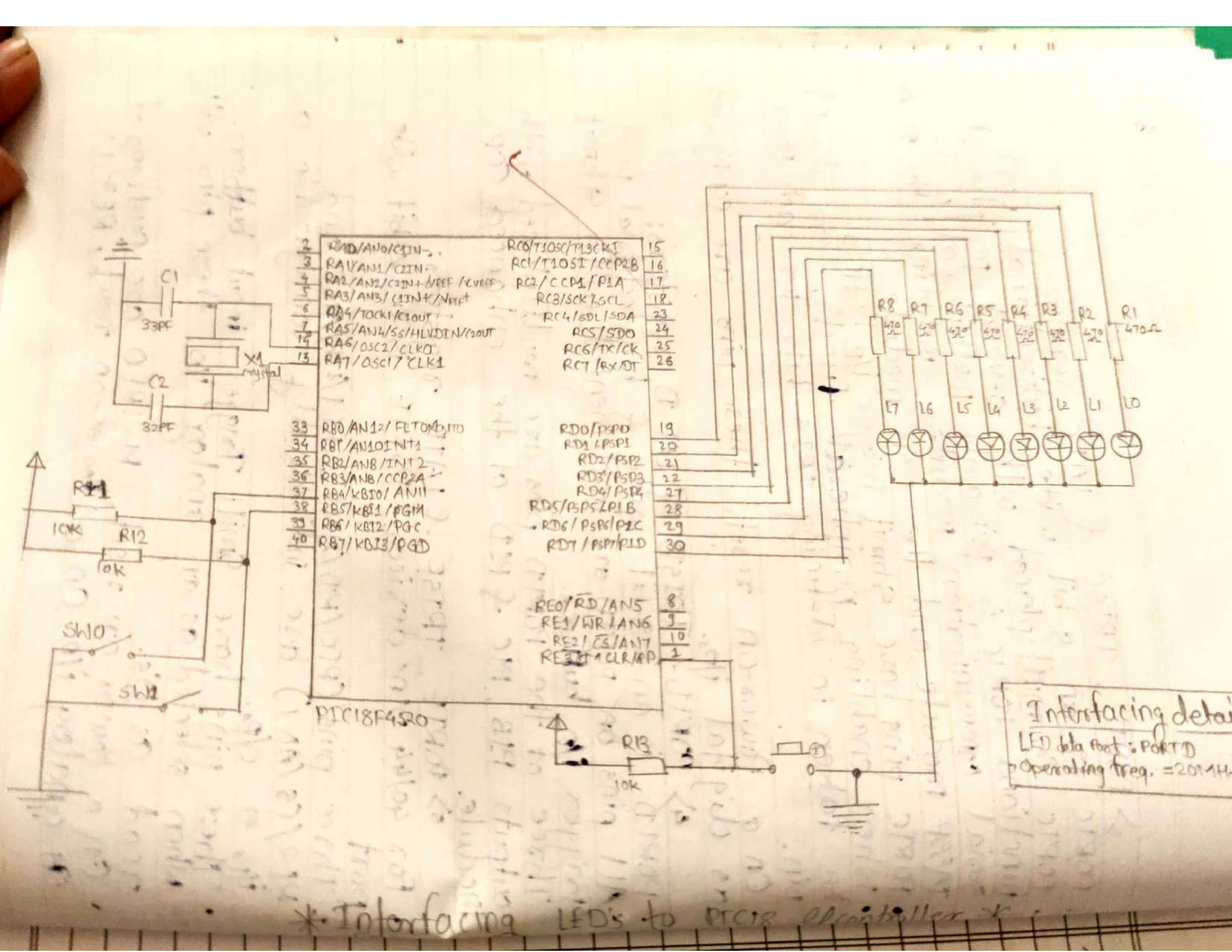
- PORTC is an 8-bit wide, bidirectional port.
- PORTC is multiplexed with several peripheral functions. PORTC is primarily multiplexed with serial communication modules, including the EUSART, MSSP module and the USB module.
- PORTC pins have Schmit Trigger i/p buffers.
- When enabling peripheral funct's, care should be taken in defining TRIS bits for each PORTC pin.
- On a Power-on reset, these pins are configured as digital i/p's.

4) PORTD, TRISD & LATD registers:

- PORTD is an 8-bit, wide & bidirectional port.
- All pins on PORTD are implemented with Schmit Triggers i/p buffers.
- Three of the PORTD pins are multiplexed with outputs P1B, P1C & P1D of the enhanced CCP module.

5) PORTE, TRISE & LATE registers:

- For 40/44 pin devices, PORTE is a 4-bit wide port.
- Three pins (RE0/RD/AN5, RE1/WR/AN6 and RE2/CS/AN7) are individually configurable as i/p's or o/p's.
- These pins have Smitt Trigger input buffers. When selected as an analog i/p, these pins will read as 0's.
- On a Power-on reset, RE2:RE0 are configured as analog i/p. On a Power-on reset, RE3 is



* Interfacing LED's to PIC18 Microcontroller *

is enabled as a digital i/p only if master clear functionality is disabled.

*Algorithm :-

1) Configure PORTD as output port writting 0x00 to the TRISD register since LED's are interfaced to PORTD.
2) Turn ON the LEDs.
3) Call delay routine.
4) Repeat step 3 to 4.

Conclusion :-

Thus we studied that the concept of Interfacing LED's to PIC μcontroller.

Expt.: Interfacing LED's to PORTD of PIC18 Microcontroller

```
#include <xc.h>//Include Controller specific .h
//Configuration bit settings
#pragma config OSC = HS //Oscillator Selection
#pragma config WDT = OFF//Disable Watchdog timer
#pragma config LVP = OFF //Disable Low Voltage Programming
#pragma config PBADEN = OFF //Disable PORTB Analog inputs
//Function Prototypes
void msdelay (unsigned int time);//Function for delay
//Start of Program Code
void main()
{
 INTCON2bits.RBPU=0;
ADCON1 = 0x0F;//To disable the all analog inputs
TRISD = 0x00;//To configure PORTD as output
while (1)//While loop for repeated operation
{
PORTD = 0xFF;   //Turn ON the all LED's
msdelay(250);// Delay
PORTD = 0x00;   //Turn OFF the all LED's
msdelay(250);// Delay
}
} //End of the Program
//Function Definition for delay degeneration
void msdelay (unsigned int time)
{
unsigned int i, j;
for (i = 0; i< time; i++)
for (j = 0; j < 710; j++); //Calibrated for a 1 ms delay in MPLAB
}
```

Code :

```
/*simple program to glow led when switch is pressed*/
/*demonstrates the use of digital read and write */
/* for pinout refer to the manual*/
/*LEDS - pins 0, 1, 2, 4*/
/*LED will be ON when pin is LOW - inverse logic*/
/*SWITCH - 10,11,12,13 */

void setup() {
  // put your setup code here, to run once:
pinMode(0,OUTPUT);  /*set LED pin as OUTPUT*/
pinMode(1,OUTPUT);  /*set LED pin as OUTPUT*/
pinMode(2,OUTPUT);  /*set LED pin as OUTPUT*/
pinMode(4,OUTPUT);  /*set LED pin as OUTPUT*/
pinMode(10,INPUT);  /*set switch pin as INPUT*/
pinMode(11,INPUT);  /*set switch pin as INPUT*/
pinMode(12,INPUT);  /*set switch pin as INPUT*/
pinMode(13,INPUT);  /*set switch pin as INPUT*/
}
void loop() {
  // put your main code here, to run repeatedly:
digitalWrite(0,HIGH);
digitalWrite(1,HIGH);
digitalWrite(2,HIGH);
digitalWrite(4,HIGH);
 /*check for key press and then make the LED ON*/
if(digitalRead(10)== 0)
{
digitalWrite(0,LOW);
delay(1000);
}
if(digitalRead(11)== 0)
{
digitalWrite(1,LOW);
delay(1000);
}
if(digitalRead(12)== 0)
{
digitalWrite(2,LOW);
delay(1000);
}
if(digitalRead(13)== 0)
{
digitalWrite(4,LOW);
delay(1000);
}}
```

*Experiment no: 05 *

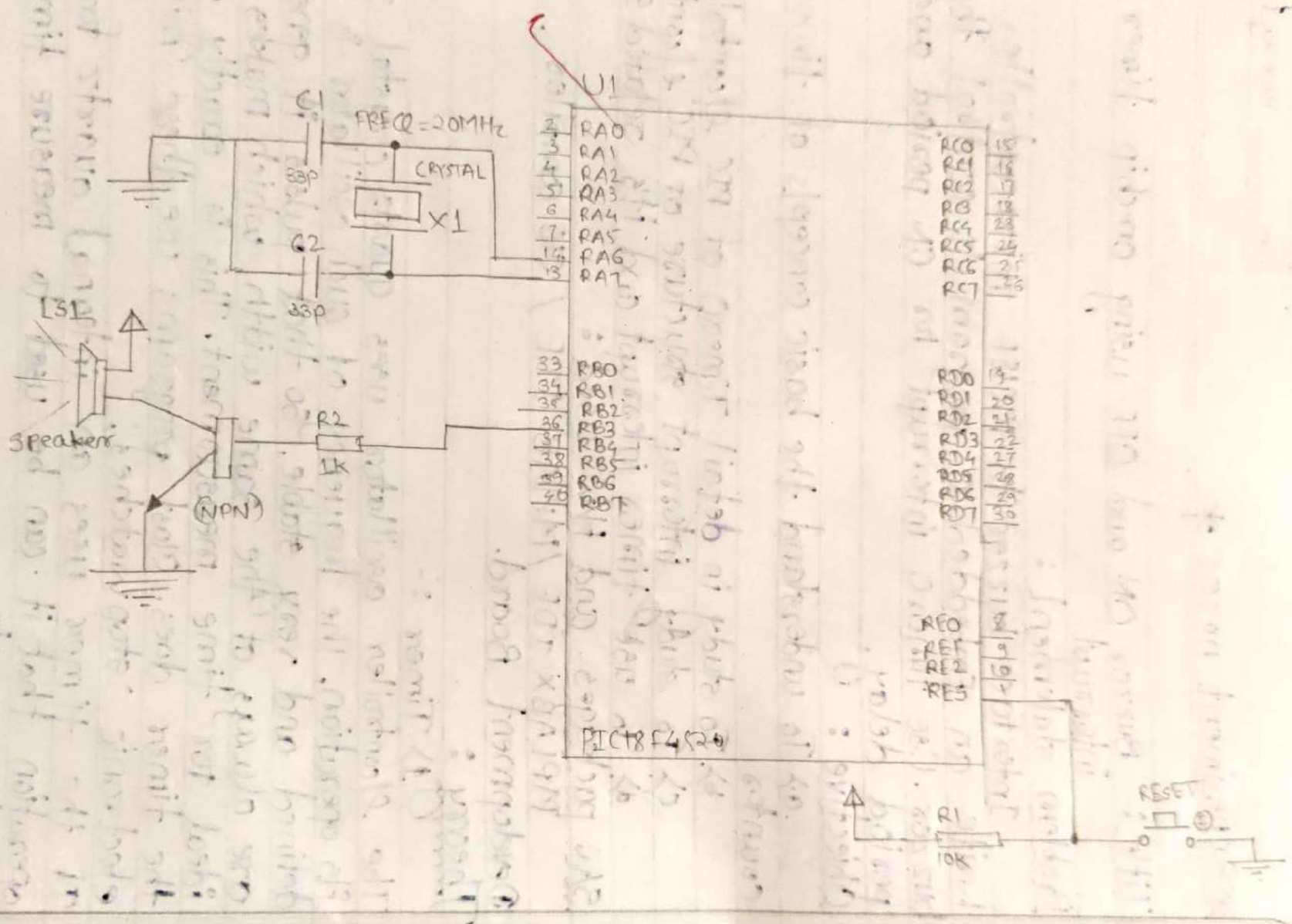
Title : Buzzer ON and OFF using on-chip timer interrupt.

Problem statement :
Interface buzzer to PIC18F microcontroller. Write an Embedded C program to control the buzzer. Use timer0 interrupt for ON period and OFF period delay.

Objective :
a) To understand the basic concepts of timer & counter.
b) To study in detail Timer0 of PIC μcontroller.
c) To study interrupt structure of PIC μcontroller.
d) To use timer interrupt and it's related SFR.

S/W packages and H/W used :
MPLABX IDE / MPLAB IDE / C18 compiler, PIC Development Board.

Theory :
1) Timer :-
The μcontroller oscillator uses quartz crystal for its operation. The frequency of such oscillator is precisely defined and very stable, so that pulses it generates are always of the same width, which makes them ideal for time measurement. This is exactly what the timer does. Most programs use these miniature electronic 'stopwatches'.
If the timer uses an internal quartz for its operation that it can be used to measure time betⁿ two events.
ⓐ How does the timer operate ?
In practice, pulses generated by the quartz



* Buzzer Interfacing to PIC Microcontroller. *

oscillator are once per each machine cycle, directly or via a prescalar, bought to the circuit which increments the number stored in the timer register. It is easy to measure short time intervals, upto 256 μsec. in the way described above because it is the largest number that one register can store.

⑥ Using a prescalar in timers operation:
A prescaler is an electronic device used to reduce freq. by a predetermined factor. In order to generate a pulse on its output, it is necessary to bring 1, 2, 4 or more pulses on its input. If one prescaler is shared by timer and watchdog timer, it cannot be used by both simultaneously.

⑥ User Using interrupt in timer operation:
If the timers register consist of 8-bits, the largest number it can store is 255. As for 16-bit registers it is the number 65535. This condition is called an overflow. If enabled from within the program, the overflow can cause an interrupt, which gives completely new possibilities. Delays of arbitary duration, having almost no influence on the main program execution by assigning the prescaler to the timer.

2) Counters:
If the timer recieves pulses from the μcontroller input pin, then it turns into a counter. Obviously, it is the same electronic circuit able to operate in two different modes. The only diff. is that in this case pulses to be counted come over the microcontroller input pin and their duration (width) is more undefined.

3) Timers / Counters in PIC elcontroller:
There are 3-5 timers on board in PIC elcontroller. These timers can also be used as counters when external pulses are applied. The timers are programmable and sometimes share with other peripheral devices. These are named as TMR0, TMR1, TMR2, & TMR4.

4) Timer0 module in PIC elcontroller:
The Timer0 module incorporates the following features:

- Software selectable operation as a timer or counter in both 8-bit or 16-bit modes.
- Readable and writable registers.
- Prescaler
- Edge select for external clock.
- Interrupt on overflow.

The T0CON register (Register 11-1): controls all aspects of the module's operation, including the prescale selection. It is both readable and writable.

(a) Timer0 operation:
Timer0 can operate as either a timer or a counter; the mode is selected by clearing the T0CS bit (T0CON <5>). In timer mode the module increments on every clock by default means a different prescalar value is selected. If the TMR0 register is written to, the increment is inhibited for the following two instruction cycles.

(b) Prescaler:
An 8-bit counter is available as a prescaler

Bit No.	Control Bit	Description
Bit 7	TMR0 ON	Timer0 On/Off Control Bit 1 = Enables ⇒ Timer0 , 0 = Stops Timer0
Bit6	T08BIT	Timer0 8-bit /16-bit Control bit 1 = Timer 0 is configured as an 8-bit timer/counter 0 = Timer0 is configured as an 16-bit timer/coun
Bit5	T0CS	Timer0 source select bit 1 = Transition on T0CKI pin 0 = Internal instruction cycle clock (CLKO)
Bit4	T0SE	Timer0 Prescaler Assignment bit 1 = Timer0 prescaler is NOT assigned. 0 = Timer0 prescaler is assigned.
Bit3	PSA	Timer0 source edge select bit 1 = Timer0 prescaler is NOT assigned. 0 = Timer0 prescaler is assigned.
Bit 2:0	T0PS2 : T0PS0	Timer0 prescaler select bits 111 = 1:256 prescaler value 110 = 1:128 Prescaler value 101 = 1:64 100 = 1:32 011 = 1:16 010 = 1:8 001 = 1:4 001 = 1:2 } Prescaler value

for the Timer0 module. The prescaler is not directly readable or writeable; its value is set by the PSA and T0PS2:T0PS0 bits (T0CON<3:0>) which determine the prescaler assignment and prescaler ratio. Clearing the PSA bit assign the prescaler to the timer0 module.

c) Timer0 interrupt:

The TMR0 interrupt is generated when the TMR0 register overflow from FFh to 00h in 8-bit mode, or from FFFh to 0000h in 16-bit mode. This overflow sets the TMR0IF flag bit. The interrupt can be masked by clearing the TMR0IF bit (INTCON<5>). Before reenabling the interrupt the TMR0IF bit must be cleared in software by the Interrupt Service Routine (ISR).

d) Timer0 Register Map:

SFR	Description	Access	Reset value	Address
T0CON	Timer0 control register	Read/write	0xFF	0xFD5
TMR0L	Timer0 Register Lower rate	R/W'	unknown	0xFD6
TMR0H	Timer0 Register Higher rate	R/W'	0x00	0xfD7
INTCON	Interrupt control register	R/W'	0x00	0xFD2
INTCON2	Interrupt Control register 2	R/W'	0xFF	0xFD1

i) TMR0L:

Used 8-bit and 16-bit mode. The register holds the current count value which is updated by clock source. User must write initial value.

ii) TMR0H:

Used only in 16-bit mode. The register holds the current count of higher byte which is updated by clock source.

iii) TOCON :
(Timer0 control register)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T0BBIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

iv) INTCON : (Interrupt Control 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
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

5) Time delay generation using timer :
The following steps are taken to generate a time delay using interrupt method.

- Load the proper value in TOCON indicating which timer mode, clock source, prescaler assigment.
- Load the registers TMR0H first and then TMR0L with initial count values.
- Enable the timer0 interrupt and global interrupt
- Start the timer by setting TMR0 ON bit in TOCON.
- Write the ISR at Interrupt vector 0x0018
  1. In ISR clear the TMR0IF flg for the next round.
  2. In ISR reload TMR0H and TMR0L values.

The size of the time delay depends on two factors :
① The crystal freq. & ② The timer's 16-bit register ③ the prescalar value. The largest delay

is achieved by making both TMROH & TMROL zero and using maximum prescaler value. i.e. 1:256.

* Without prescaler

Timer delay (T_d) = (65536 - NNNNN) x (Time period T_F x Prescaler value)

∴ Maximum delay = (65536-00000) x (0.2μs x 256) = 3.36 sec.

* With Prescaler (1:256)

Timer delay (T_d) = (65536-NNNNN) x (Time period T_F x Prescaler value)

∴ Max. delay = (65536-00000) x (0.2μs x 256) = 3.36 sec.

ⓐ Finding the value to be loaded into timer for desired delay :

1. Without prescalar.
   - Divide the desired value (Time delay) by T_F to get n ($n = T_d / 0.2\mu s$)
   - Perform 65536 - n. Where n is the decimal value from step 1
   - Set TMROL = XX and TMROH = YY
2. With Prescalar.
   - Multiply Timer period T_F by prescalar value to T_{eq}.

     $T_{eq} = T_F$ * Prescaler value
   - Divide the desired time delay T_d by T_{eq} to get 'n'

     $n = T_d / T_{eq}$
   - Perform (65536 - n). Where n is the decimal value for above.
   - Set TMROL = XX & TMROH = YY

(b) Calculation for generating square wave of 10Hz.

- Desired frequency of square wave $F_s = 10$ Hz
- $T_s = 1/F_s = 1/10 = 0.1$ sec.
- On period & Off period of square wave $= T_s/2 = 0.05$ s $= 50$ ms.
- To generate square wave the value of port pin RB0 should be toggled after every 50ms.

i) for operating freq. $f_{osc} = 20$ MHz.

- selecting Prescaler 1:8, timer clock source $= f_{osc}/4$
- ∴ Timer clock freq. $= F_t = f_{osc}/4 = 20\times10^6/4 = 5$ MHz
- $T_{eq} = T_t \times$ Prescaler value $= 0.2$ els $\times 8 = 1.6$ els
- $(YYXX)_{10} = 65536 - n = 65536 - 31250 = 34286$
- $(34286)_{10} = (85EE)_{16}$ should be loaded in TMROH:TMROL.

ii) for operating freq. $f_{osc} = 48$ MHz

- selecting Prescaler 1:16, Timer clock source $= f_{osc}/4$
- ∴ Timer clock freq. $= F_t = f_{osc}/4 = 48\times10^6/4 = 12$ MHz
- $T_{eq} = T_t \times$ Prescaler value $= 0.0833$ els $\times 16 = 1.333$ els
- To get n, $n = T_d/T_{eq} \Rightarrow 50\text{ms}/1.33\text{els} = 37501$
- $(YYXX)_{10} = 65536 - n = 65536 - 37501 = 28034$
- $(28034)_{10} = (6D82)_{16}$ should be loaded in TMROH: TMROL.

Algorithm:-

→

1. Configure port pin RB3 as output & initial value 0.
2. Load the value 0x02 in T0CON indicating which timer in 16-bit mode, prescaler assignment with 1:8
3. Load the registers TMR0H first then TMR0L with calculated initial count values.
4. Enable the Timer0 Interrupt & Global Interrupt
5. Start the time by setting TMR0N bit in T0CON.
6. Write the ISR at Interrupt vector 0x0018 which contain:
   a) Toggle the PORT pin RB3
   b) In ISR clear the TMR0IF flag for the next round.
   c) In ISR reload TMR0H & TMR0L values.

Conclusion :-

Thus we studied that Buzzer ON & OFF using on-chip timer interrupt.

* Experiment no: *

Title: Operate the relay based on External Interrupt.

Problem statement:
Interface relay and pushbutton to PIC18F µcontroller. Write an Embedded C program to generate the external interrupt to operate the relay whenever pushbutton is pressed.

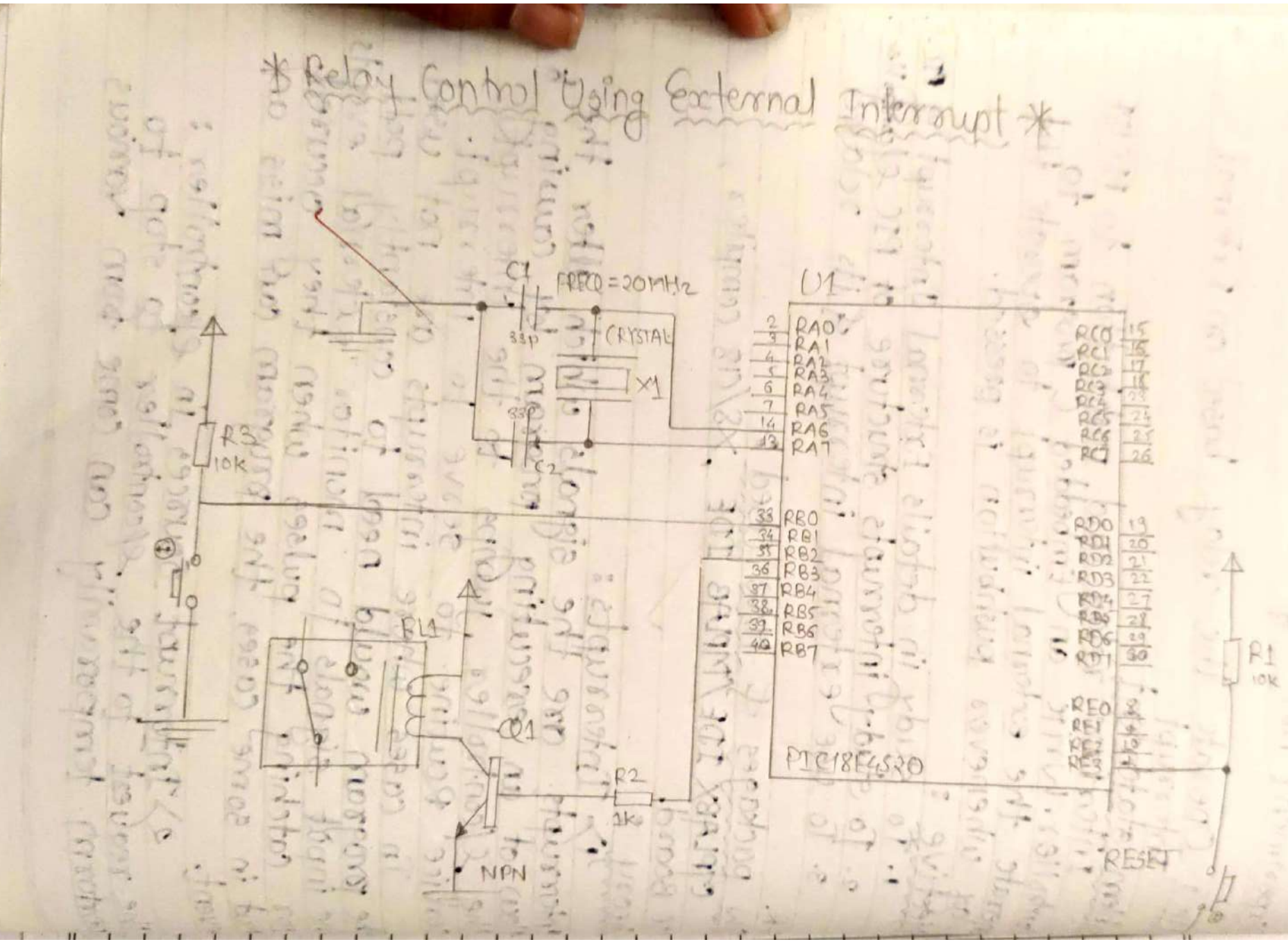
Objective:
1. To study in details External Interrupt of PIC.
2. To study interrupts structure of PIC µcontroller
3. To use external interrupt & its related SFR.

S/W packages & HW used:-
MPLABX IDE/MPLAB IDE, X8/C18 compiler, PIC Board.

Theory:
1) Interrupts:
Interrupts are the signals which alter the flow of an executing program by causing the µcontroller jumps to the interrupt Service Routine to serve to interrupt.
In cases where interrupts are not used, the program would need to constantly poll the input signals to monitor external events for catching the pulses when they occured. But in some cases, the program can miss an event.

2) Interrupt sources in µcontroller:
The request to the µcontroller to stop to perform temporarily can come from various

* Relay Control Using External Interrupt *



sources:

① Through external hardware devices like pressing specific key on the keyboard.

② During execution of the program, the µcontroller can also send interrupts to itself an error in code.

③ In the multi-processor system, the µcontrollers can send interrupts to each other to communicate.

3) Interrupt Types in µcontrollers:

There are 2 types of interrupts for PIC µcontroller
- Software interrupt
- Hardware interrupt.

4) Interrupt Sources in PIC18F4520

Following interrupts sources are present in PIC18F4520
- External H/w interrupts (INT0, INT1, INT2)
- Timer over interrupt.
- Parallel Port Read/Write interrupt.
- Master synchronous serial Port Interrupt.
- Data EEPROM Write complete Interrupt.

5) Legacy Interrupts Structure of PIC18F µcontroller:

6) Interrupt Registers:

These are the 9 registers for interrupt operation & minimum 1 register in PIC18F458 which are:
- RCON (Reset Control Register)
- INTCON, INTCON2, INTCON3 (Interrupt Control Register).
- PIE1, PIE2 (Peripheral Interrupt Req, Register)

① RCON register :
- Reset control register
- IPEN bit to enable interrupt priority scheme,
1 = Enable priority level on Interrupts.

② INTCON register :
- 3 Interrupts control registers INTCON1, INTCON2, INTCON3.
- Readable and writeable register which contains various enable and flag bits.
- Interrupt flag bits get set when an interrupt occurs.

③ PIE register :
- Peripheral Interrupt Enable register.
- May be multiple register (PIE1, PIE2) depending on the number of peripheral interrupt sources.

7) INTCON : Interrupt Control Register for configuring External Interrupt.

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	PIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

④ INTCON2 :
Interrupt control register 2 for configuring External Interrupt.

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RAIP
bit 7							bit 0

Bit No.	Control Bit	Description
Bit 7	RBPU	PORTB Pull-up Enable bit 1 = All PORTB pull-ups are disabled 0 = PORTB pull-ups are enabled by individual port
Bit 6	INTDG0	External Interrupt 0 Edge select bit 1 = Interrupt on rising edge 0 = Interrupt on falling edge.
Bit 5	INTDG1	External Interrupt 1 Edge select bit 1 = Interrupt on rising edge 0 = Interrupt on falling edge
Bit 4	INTEG1	TMR0 overflow Interrupt priority bit 1 = High priority 0 = low priority.
Bit 3	-	-
Bit 2	TMR0IP	TMR0 overflow Interrupt Priority bit.
Bit 1	-	-
Bit 0	RBIP	RB PORT change Interrupt priority bit 1 = High priority 0 = Low priority.

Algorithm :-

① Configure Port pin RB2 as o/p & initial value 0.
② Configure Port pin RB0 as i/p for external interrupt
③ Configure INTCON2 register for edge trigger i.e. negative edge.
④ Enable the External Interrupt & Global Interrupt.

Conclusion :

Thus we studied how to operate the relay based on external Interrupt.

Main Program:-

Expt:Interfacing LEDs, Switches, Buzzer and Relay

```
//Includes
#include <xc.h>                         //Include Controller specific .h

//Configuration bit settings
#pragma config OSC = HS //Oscillator Selection
#pragma config WDT = OFF //Disable Watchdog timer
#pragma config LVP = OFF //Disable Low Voltage Programming
#pragma config PBADEN = OFF //Disable PORTB Analog inputs

//Declarations
#define lrbit    PORTBbits.RB0    //SW1 interfaced to RB0
#define rlbit    PORTBbits.RB1        //SW2 interfaced to RB1
#define relay   PORTBbits.RB2        //Relay interfaced to RB2
#define buzzer          PORTBbits.RB3       //Buzzer interfaced to RB3


//Function Prototypes
void msdelay (unsigned int time);//Function for delay

//Start of Program Code
                                                //Main Program
void main()
{
        unsigned char val=0;            //Variable to latch the switch condition
```

```
INTCON2bits.RBPU=0;          //To Activate the internal pull on PORTB
    ADCON1 = 0x0F;                  //To disable the all analog inputs

    TRISBbits.TRISB0=1;      //To configure RB4 as input for sensing SW0
    TRISBbits.TRISB1=1;      //To configure RB5 as input for sensing SW1

    TRISBbits.TRISB2=0;      //To configure RC1 (relay) as output
    TRISBbits.TRISB3=0;             //To configure RC2 (buzzer) as output
    TRISD = 0x00;                   //To configure PORTD (LED) as output

    PORTD = 0x00;                   //Initial Value for LED
    buzzer = 0;                     //Initial Value for Buzzer
    relay = 0;                      //Initial Value for Relay

while (1)                           //While loop for repeated operation
    {
    if (!(lrbit))               //To check whether SW0 is pressed
        val = 1;                    // Latch the status of switch SW0
    if (!(rlbit))               //To check whether SW1 is pressed
        val = 2;                    // Latch the status of switch SW1

    if (val == 1)
      {
        buzzer = 1;
        relay = 1;
        PORTD = PORTD >>1;      //Shift left by 1 bit
```

```
            if (PORTD == 0x00)
                PORTD = 0x80;                // Make the MSB bit equal to 1
            msdelay(250);
        }
    if (val == 2)
        {
            buzzer = 0;
            relay = 0;
            PORTD = PORTD<<1;        //Shift right by 1 bit
            if (PORTD == 0x00)
                PORTD = 0x01;                // Make the LSB bit eqaul to 1
            msdelay(250);
        }
    }
}                                                    //End of the Program

//Function Definitions
void msdelay (unsigned int time)//Function for delay
{
unsigned int i, j;
 for (i = 0; i < time; i++)
        for (j = 0; j < 275; j++);       //Calibrated for a 1 ms delay in MPLAB
}
```

*Experiment no: 06 *

Title : Interfacing LCD to Display Message.

Problem statement:

Interface 16x2 LCD to PIC μcontroller in 8-bit mode. Write an Embedded C program to display user defined message on LCD without using standard library function.

Objective :

1) To understand the working of liquid Crystal Display (LCD)
2) To study the LCD interfacing modes and Timing diagram.
3) To study and use of the LCD commands to drive LCD.
4) To interface LCD in 8-bit mode to PIC μcontroller.

S/W packages & H/W used:

MPLABX IDE / MPLAB IDE, X8/C18 compiler, PIC development board.

Theory :-

1) LCD Interfacing :

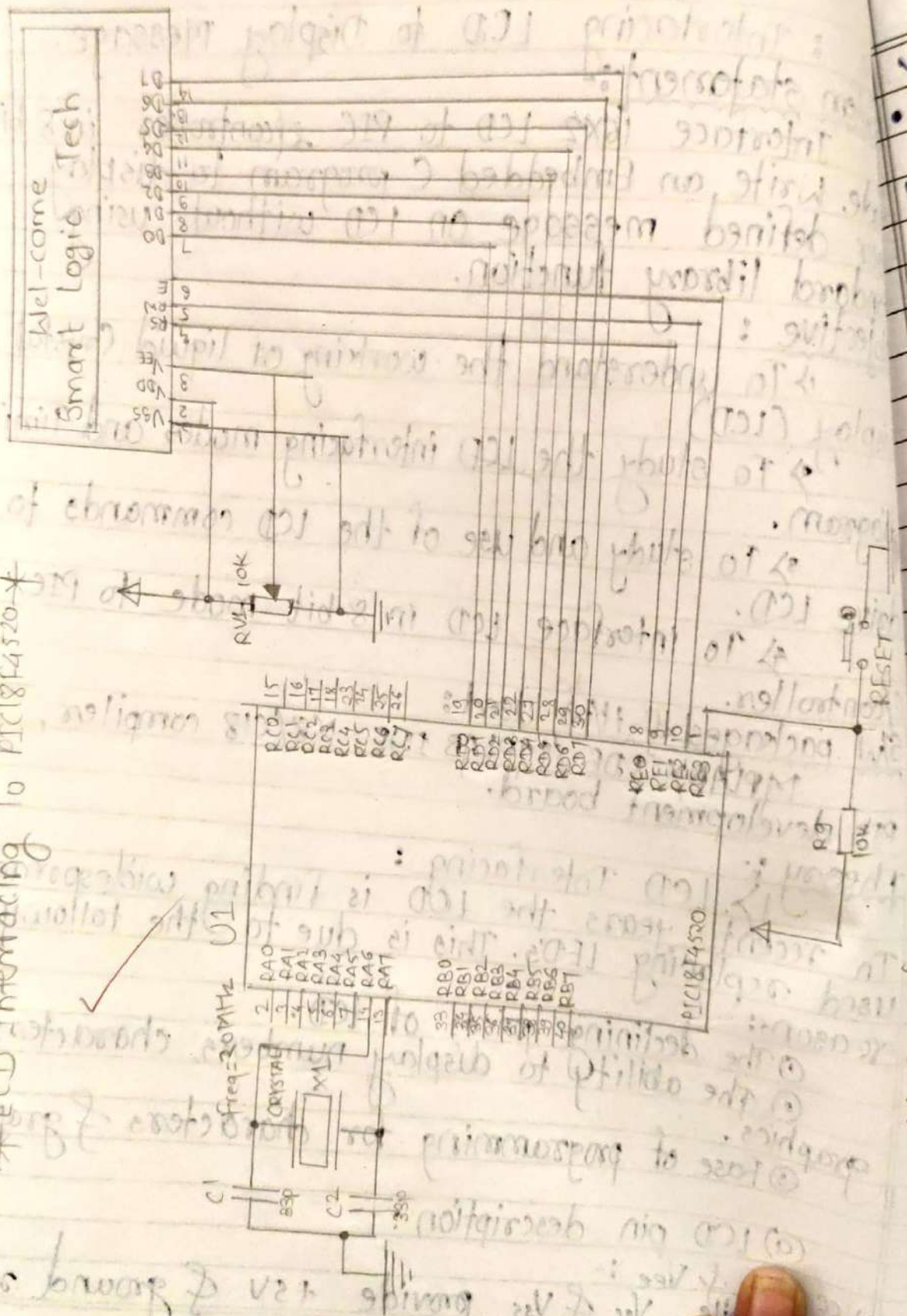
In recent years the LCD is finding widespread used replacing LED's. This is due to the following reasons:

① The declining prices of LCD.
② The ability to display numbers, characters & graphics.
③ Ease of programming for characters & graphics.

(a) LCD pin description:

•Vcc, Vss & Vee :-

While Vcc & Vss provide +5V & ground resp.


LCD Interfacing to PIC18F4520
Wel-come
Smart Logic Tech
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
VEE
VDD
VSS
RV1
10K
R9
10K
RESET
U1
PIC18F4520
RA0
RA1
RA2
RA3
RA4
RA5
RA6
RA7
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
RC0
RC1
RC2
RC4
RC5
RC6
RC7
RD0
RD1
RD2
RD3
RD4
RD5
RD6
RD7
RE0
RE1
RE2
RE3
Freq=20MHz
CRYSTAL
X1
C1
C2
33p
33p

Vee is used for controlling LCD contrast.

- Register select (RS):

There are two very important register inside LCD.

a) RS=0 the instruction command code register is selected, allowing user to send command.

b) RS=1, the data register is selected, allowing the user to send data on LCD.

- Read/Write (R/W):-

R/W i/p allows the user to write to LCD or read info from it. R/W=1 when read,
R/W=0 when write.

- Enable (EN):

The enable pin is used by the LCD to latch info presented to its data pins. This pulse must be a minimum of 450ns wide.

- Data bus (D0-D7):

The 8-bit data pins, D0-D7 are used to send the information to the LCD or read the contents of the LCD's internal registers.

(b) LCD command codes:

Sr. No.	Command to LCD instruction	Code (Hex)
1	Clear display screen	01
2	Return Home	02
3.	Decrement cursor (shift cursor to left)	04
4.	Increment cursor (shift cursor to right)	06
5.	shift display right	05
6.	shift display left	07
7.	Display off, cursor off.	08
8.	Display on, cursor on.	0A
9	Display on, cursor off	0C

10	Display on, cursor blinking	OE
11	shift cursor position to left	10
12	shift cursor position to right	14
13	shift the entire display to left	18
14	shift the entire display to right	1C
15	Force cursor to begining of 1st line	80
16	Force cursor to beginning of 2nd line	C0
17	2 lines and 5X7 matrixes	38

Algorithm:

① As LCD data bus is connected to PORTD & handshaking signal EN-RE0, RS-RS1, E-RE2, o/p writing 0x00 to the TRIS register.

② The RE0, RE1, RE2 this pins have dual funct reset it works on analog input.

③ Initialize LCD.

a) Write a commands & 2 lines & 5X7 matrix.

④ Display message on first line

a) Write a commad 0x80 to LCD.

b) Load ASCII character from the given string.

c) Write Data to LCD.

⑤ Display message on second line.

a) Write a command 0xC0 to LCD.

b) load ASCII character from the given string.

c) Write data to LCD.

⑥ Endless loop.

Conclusion :-

Thus we studied that, Interfacing LCD to Display message.

Expt.: LCD Interfacing to PIC18 Microcontroller

```
#include <xc.h>//Include Controller specific .h
//Configuration bit settings
#pragma config OSC = HS //Oscillator Selection
#pragma config WDT = OFF //Disable Watchdog timer
#pragma config LVP = OFF //Disable Low Voltage Programming
#pragma config PBADEN = OFF //Disable PORTB Analog inputs
//Declarations
#define LCD_DATA PORTD//LCD data port to PORTD
#define ctrl PORTE//LCD control port to PORTE
#define rsPORTEbits.RE0//register select signal to RE0
#define rwPORTEbits.RE1//read/write signal to RE1
#define enPORTEbits.RE2//enable signal to RE2
//Function Prototypes
void init_LCD(void);//Function to initialize the LCD
void LCD_command(unsigned char cmd);//Function to pass command to LCD
void LCD_data(unsigned char data); //Function to write char to LCD
void LCD_write_string(static char *str);//Function to write string
void msdelay (unsigned int time);
//Function to generate delay
//Start of Main Program
void main(void)
{
 char var1[] = " Wel-Come";//Declare message to be displayed
 char var2[] = "Smart Logic Tech";
 ADCON1 = 0x0F;//Configuring the PORTE pins as digital I/O
 TRISD = 0x00; //Configuring PORTD as output
 TRISE = 0x00;//Configuring PORTE as output
 init_LCD();// call function to initialize of LCD
 msdelay(50); // delay of 50 milliseconds
```

```
LCD_write_string(var1);//Display message on first line
msdelay(15);
LCD_command(0xC0); // initiate cursor to second line
LCD_write_string(var2);//Display message on second line
while (1);              //Loop here
                        //End of main

}
//Function Definitions
void msdelay (unsigned int time) //Function to generate delay
{
unsigned int i, j;
for (i = 0; i< time; i++)
for (j = 0; j < 710; j++);//Calibrated for a 1 ms delay in MPLAB
}
void init_LCD(void)// Function to initialize the LCD
{
LCD_command(0x38); // initialization of 16X2 LCD in 8bit mode
msdelay(15);
LCD_command(0x01); // clear LCD
msdelay(15);
LCD_command(0x0C); // cursor off
msdelay(15);
LCD_command(0x80); // go to first line and 0th position
msdelay(15);
}
void LCD_command(unsigned char cmd) //Function to pass command to LCD
{
 LCD_DATA = cmd;//Send data on LCD data bus
 rs = 0;//RS = 0 since command to LCD
 rw = 0;//RW = 0 since writing to LCD
```

```
en = 1;//Generate High to low pulse on EN
msdelay(15);
en = 0;
}
void LCD_data(unsigned char data)//Function to write data to the LCD
{
LCD_DATA = data;//Send data on LCD data bus
                                    //RS = 1 since data to LCD
rs = 1;
rw = 0;//RW = 0 since writing to LCD
en = 1;//Generate High to low pulse on EN
msdelay(15);
en = 0;
}
//Function to write string to LCD
void LCD_write_string(static char *str)
{
int i = 0;
while (str[i] != '\0') //Check for end of the string
{
LCD_data(str[i]); // sending data on LCD byte by byte
msdelay(15);
i++;
}
}
```

Date: Page No. 26

* Experiment no: 07 *

Title: Generation of PWM signal to vary the speed of DC motor.

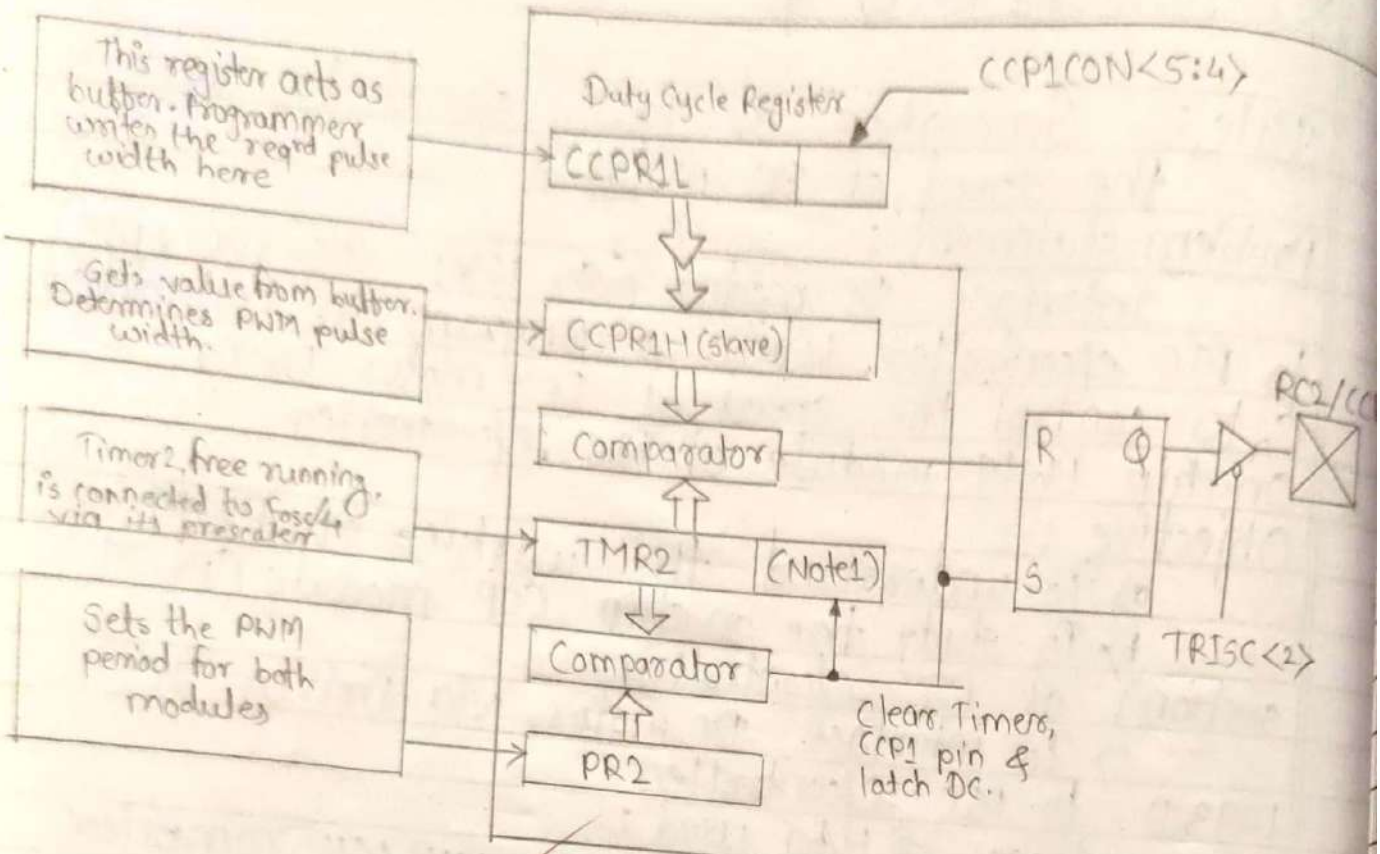
Problem statement:-
Interface DC motor (9V-12V; 200-300 rpm) to PIC μcontroller. Write a program in Embedded C to control the speed of DC motor using on-chip PWM module of PIC μcontroller.

Objective:
a) To understand the working of PWM
b) To study the on-chip CCP module (PWM section) of PIC μcontroller.
c) To interface DC motor, via Driver IC L293D, to PIC μcontroller.

S/W packages & H/W used:
MPLABX IDE / MPLAB IDE, XC8 / C18 compiler, PIC development Board.

Theory:
i) Pulse Width Modulation:
PWM is a way to encode data such that it corresponds to the width of the pulse given a fixed frequency. It is also a way to control motors, power circuits, etc.

a) Period (T) & PWM freq. (f).
Period (T) is the time required for a new pulse to arrive. It is basically the sum of ON time (T-ON) and OFF time (T-OFF) of a PWM cycle. PWM freq., the pulse is repeated. It also called as 'Repeatition Rate'.



* PWM Block diagram *

(b) T-ON, T-OFF & Duty cycle :-
Each period of PWM signal is divided into T-ON & T-OFF where T-ON is the time required or taken for the pulse to remain ON. & vice versa.

(c) Duty cycle as ratio :

$$\text{Duty cycle} = \frac{T_{ON}}{T_{ON} + T_{OFF}}$$

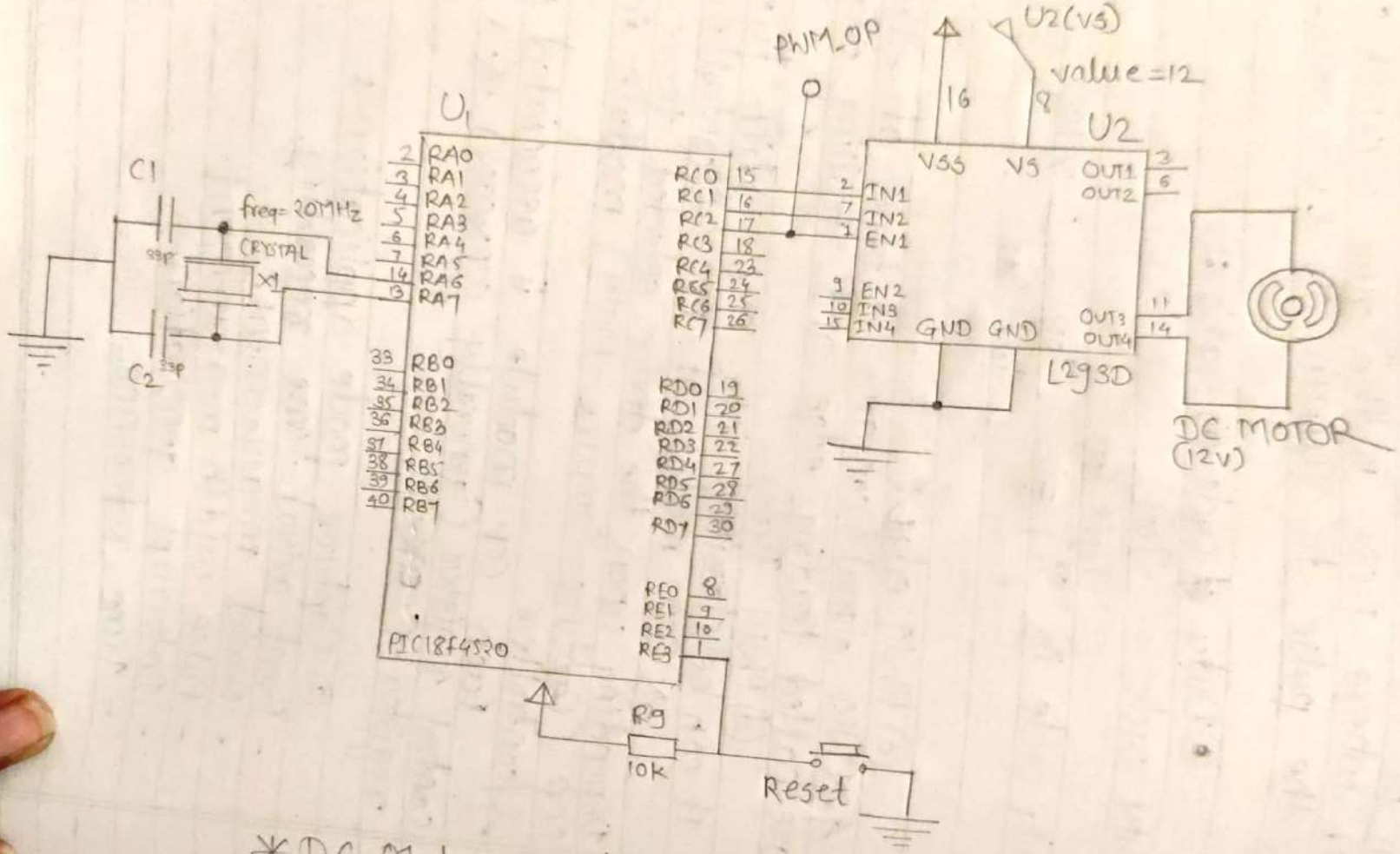
$$(\text{Duty cycle in } \%) = \frac{T_{ON}}{T_{ON} + T_{OFF}} \times 100$$

(c) PWM edge :
A PWM signal contains 2 types of edges which are called leading edge and Trailing edge.

(d) PWM Types :
It can be classified into many diff. types.

2) CCP module of PIC µcontroller :
Depending upon the device selected there are 1-4 CCP (Capture-Compare-PWM) module in PIC µcontrollers.
Each CCP module is associated with a control register (generally, CCPxCON) & a data register (CCPRx)

(a) Capture mode application :
- Event arival time recording
- Period measurement.
- Pulse width measurement.
- Interrupt generation.
- Time reference..



*DC Motor speed control using PWM *

Date: Page No. 28

(b) Compare mode Applications:

- Generate .....
  - Single pulse
  - Train of Pulses.
  - Periodic waveform.
- Trigger Start ADC conversion.
- Time reference.

3) PWM mode of CCP module:

In Pulse-width modulation (PWM) mode the CCPx pin produces upto a 10-bit resolution PWM output. Since the CCP2 pin is multiplexed with a PORTB or PORTC data latch, the appropriate TRIS bit must be cleared to make the CCP2 pin an o/p pin.

The simplified block diagram of the CCP1 configured to generated PWM is shown in figure.(a)

Having established the PWM period, let us consider how the pulse width is determined. A second compare register arrangement is introduced to do this. It is the comparator that determines the pulse width. This increases resolution.

(a) PWM period and Duty cycle:

The PWM period (T) is determined by the interaction of the PR2 register and the 8-bits of Timer2. It may be calculated as,

PWM period = T_{pwm} = (PR2+1) x (Timer 2 i/p clock)

T_{pwm} = (PR2+1) x (4 x T_{osc} x TMR2 Prescaler value)

∴ To find PR2 for desired PWM period.

$$PR2 = \left[\left(\frac{T_{pwm}}{4 \times T_{osc} \times \text{TMR2 prescaler value}} \right) - 1 \right]$$

above eqn can be represented in the terms of freq. as below,

$$PR2 = \left[\left(\frac{Fosc}{4 \times fpwm \times (TMR2\ prescaler)}\right) - 1\right]$$

(b) Pin Description :-

Signal	Pin no.	Symbol.
CCP2	16	RC1 / T1OSI / CCP2 / UOE
CCP1	17	RC2 / CCP1 / P1A
CCP	36	RB3 / AN9 / CCP2 / VPO.

RB3 is the alternate pin for CCP2 multiplexing.

(c) PWM Register Map :

SFR	Description	Access	Reset value	Address
CCP1CON	Standard CCP1 control reg.	R/W'	0x00	0xFBD
CCPR1L	CCP1 register low byte	R/W'	unknown	0xFBE
CCPR1H	CCP1 register high byte	No access	unknown	0xFBF
CCP2CON	Standard CCP2 control	R/W'	0x00	0x FBA
CCPR2L	CCP2 register low byte	R/W'	unknown	0xFBB
CCPR2H	CCP2 register high byte	No access	unknown	0x FBC
T2CON	Timer2 control register	R/W'	0x00	0x FCA
TMR2	Timer2 register	R/W'	0x00	0xFCC
PR2	Timer2 Reader register.	R/W'	0xFF	0xFCB

4) Timer 2 module :

The Timer 2 module timer incorporates the following :

- 8-bit timer & period register (TMR2 & PR2 resp.)
- Readable and writable (both registers).
- Software programmable prescaler (1:1, 1:4 & 1:16).

Date: Page No. 30

- Interrupt on TMR2 to PR2 match.
The module is controlled through the T2CON register which enables or disables timer & configure the prescaler & postscaler.

5) Calculation for PWM operation:

① Desired PWM freq. = 4kHz

∴ PWM Period = $\frac{1}{\text{PWM freq}} = \frac{1}{4\times10^3} = 250\times10^{-6}$ sec

PWM period = 250 cls.

a) For Fosc = 20MHz

$T_{osc} = \frac{1}{F_{osc}} = \frac{1}{20\times10^6} = 0.05$ cls

TMR2 prescalar value = 16.

$PR2 = \left\{\left[\frac{T_{pwm}}{4\times T_{osc}\times(\text{TMR2 Prescaler})}\right]-1\right\}$

6) Quadruple half H drivers (L293D):

The L293D is quadruple high-current half-H drivers. The L293D is designed to provide bidirectional drive currents of upto 600 mA at voltages from 4.5V to 3.6V. The device is designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2 EN & driver 3 & 4 enabled by 3,4 EN as shown in pin dig. →

Algorithm:

The following steps should be taken while configuring the CCP module for PWM operation:

① Set the PWM period by writing to the PR2 register.
② Set the PWM duty cycle by writing to the CCPR1L register & CCP1CON<5:4> bits.
③ Make the CCP1 pin an output by clearing the TRIS bit.
④ Make the RC6 & RC7 as o/p to Turn ON the motor.
⑤ Set the TMR2 prescaler value, then enable Timer2.
⑥ Configure the CCP1 module for PWM operation.

Conclusion:-

Thus we studied that, how we can generate the PWM signal to vary the speed of DC motor.

Expt. : PWM Generation using PIC18F4520 to vary speed of DC motor.

```
#include <xc.h>//Include Controller specific .h
#pragma config OSC = HS//Oscillator Selection
#pragma config WDT = OFF//Disable Watchdog timer
#pragma config LVP = OFF//Disable Low Voltage Programming
#pragma config PBADEN = OFF//Disable PORTB Analog inputs
void myMsDelay (unsigned int time) // Definition of delay subroutine

{
unsigned int i, j;
for (i = 0; i< time; i++)// Loop for time
for (j = 0; j < 710; j++);// Calibrated for a 1 ms delay in MPLAB

}
void main()

{
TRISCbits.TRISC0 = 0;// Set PORTC, RC6 as output (DCM IN1)
TRISCbits.TRISC1 = 0;// Set PORTC, RC6 as output (DCM IN2)
TRISCbits.TRISC2 = 0;// Set PORTC, RC2 as output (CCP1)
PR2 = 0x4E; // set PWM Frequency 4KHz
CCP1CON = 0x0C; // Configure CCP1CON as PWM mode.
T2CON = 0x07;// Start timer 2 with prescaler 1:16
PORTCbits.RC0 = 1;// Turn ON the Motor
PORTCbits.RC1 = 0;
while(1)// Endless Loop
{
// --------Duty Cycle 80%----------
CCP1CONbits.DC1B0 = 0;
CCP1CONbits.DC1B1 = 1;
CCPR1L = 0x3E;
myMsDelay(2000);
// --------Duty Cycle 60%----------
```

```
CCP1CONbits.DC1B0 = 1;
CCP1CONbits.DC1B1 = 1;
CCPR1L = 0x2E;
myMsDelay(2000);
//---------Duty Cycle 40%----------
CCP1CONbits.DC1B0 = 1;
CCP1CONbits.DC1B1 = 0;
CCPR1L = 0x1F;
myMsDelay(2000);
//---------Duty Cycle 20%----------
CCP1CONbits.DC1B0 = 0;
CCP1CONbits.DC1B1 = 1;
CCPR1L = 0x0F;
myMsDelay(2000);
}
}
```

Date: Page No. 32

* Experiment no: 08 *

Title: Serial Communication bet^n PC to PC & PIC e/controller.

Problem statement:
Write a program in Embedded C to transfer the message and recieve serially at 9600 baud, 8 bit data & 1 stop bit.

Objective:
a) To study the RS232 standard for serial communication.
b) To study the on-chip USART of PIC e/controller.
c) To calculate the desired baud rate.
d) To interface MAX232 with PIC e/controller.

S/W packages & H/w used:
MPLAB IDE, C18 compiler.

Theory:
1) Serial communication:
The e/controller is parallel device that transfers 8-bits of data simultaneously over 8 data lines to parallel I/O devices.
In serial data comm^n, 8-bit data is converted to serial bits using a parallel in to serial out-shift register and then it is transmitted over a single data line.

a) Comm^n links:
1) Simplex comm^n links:
In simplex transmission, the line is dedicated for txn. The transmitter sends and the reciever receives the data.
2) Half duplex comm^n links:
In half duplex, the comm^n link can be used

Date: Page No. 34

Receiver Transmitter (EUSART) module is one of the 2 serial I/O modules. (Generally, USART is also known as SCI.).

(a) PIN Description:

signal	Pin No.	Symbol.
TxD	25	RC6/Tx/CK
RxD	26	RC7/Rx/DT/SDO

(b) EUSART Register Map:

The operation of the EUSART is controlled through 3 registers.

- TxSTA
- RCSTA
- BAUDCON.

(c) EUSART Register Description.:

1) Transmit status & control Register:

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	Tx9	TxEN	SYNC	SENDB	BRGH	TRMT	Tx9D
bit7							bit0

(d) Baud Rate Generator (BRG)

The BRG is a dedicated 8-bit or 16-bit generator that supports both the Asynchronous and synchronous modes of the EUSART. By default, the BRG operates in 8-bit mode. Setting the BRG16 bit (BAUDCON<3>) select 16-bit mode.

Configuration bits			BRG mode / EUSART mode	Baud rate formula [n = SPBRGH : SPBRG]
SYNC	BRG16	BRGH		
0	0	0	8bit/Asynchronous	BR = Fosc/[64(n+1)]
0	0	1	8bit/Asynchronous	
0	1	0	16bit/Synchronous	BR = Fosc/[16(n+1)]
0	1	1	16bit/Synchronous	
0	1	x	8bit/Synchronous	BR = Fosc/[4(n+1)]
1	0	x	8bit/Synchronous	
			16bit/Synchronous	

Date :
Pg No : 35

(e) Baud rate calculation :

Mode selection.

a) BRG mode = 16-bit by setting BRG16 bit in BAUDCON register.

b) EUSART mode = Asynchronous by clearing sync. bit in TxSTA

c) Formulae desired Baud rate = fosc/4[SPBRGH:SPBRG] selected by setting BRG16 bit in TxSTA register.

(f) Interfacing diagram :

Algorithm :

① Calculate the SPBGRH:SPBRG value for baud rate.

② Configure Port pin RC6 as o/p and port pin RC7 as i/p.

③ Set BRG mode = 8-bit by setting BRG16 bit in BAUDCON.

④ Set EUSART mode = Asynchronous, by clearing sync bit in TxSTA.

⑤ Enable the serial port by setting SPEN-bit in RCSTA.

⑥ Enable the transmission by setting TXEN- bit in TxSTA.

⑦ Check TXART TRMT- bit in TxSTA register to TxREG empty.

⑧ Repeat the step ⑥ until complete message transmitted.

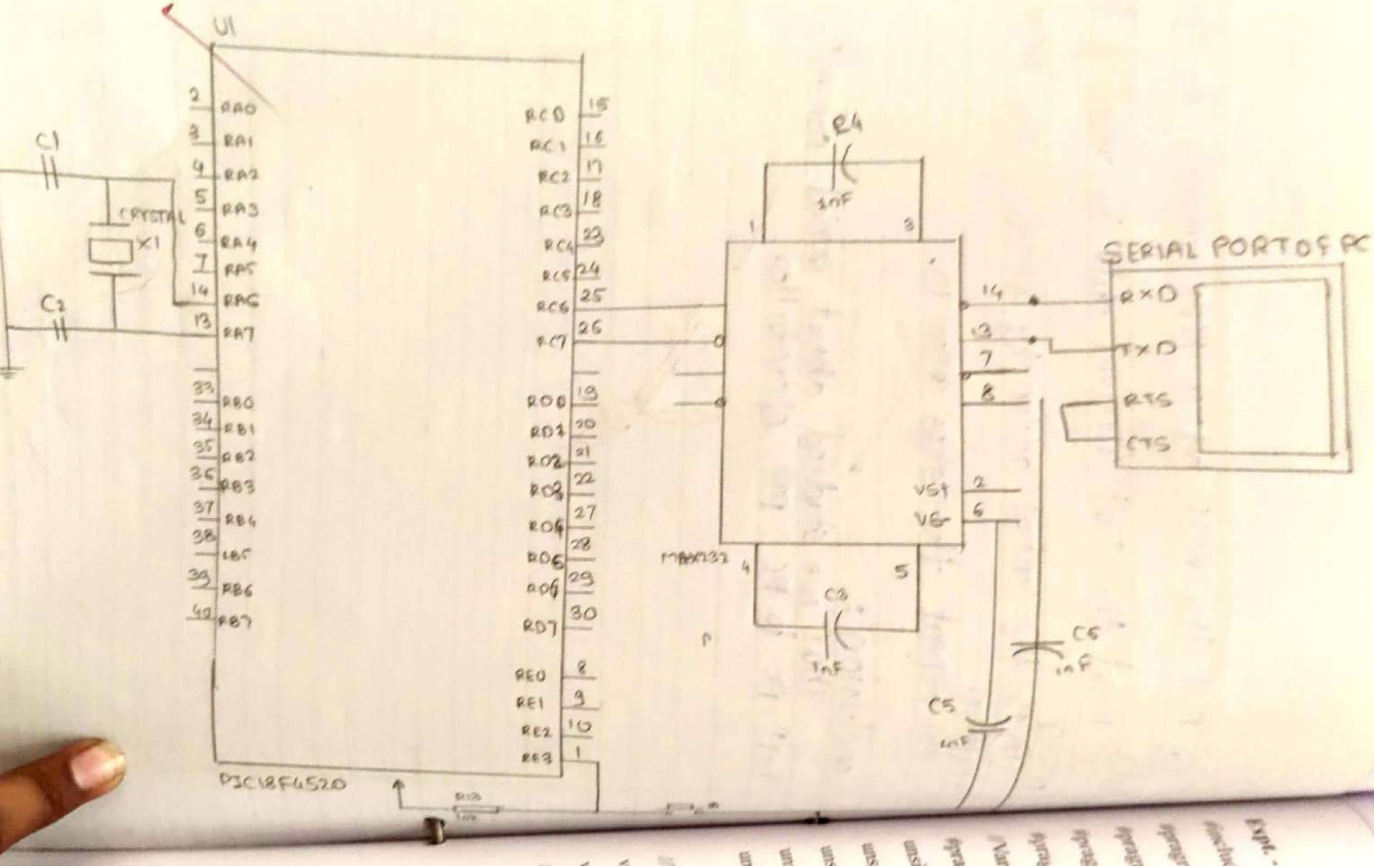
⑨ Enable the continuous ~~repetition~~. reception by setting CREN bit in RCSTA.

(10) Poll the RCIF bit in PIR1 register to check data.
(11) Read the 8-bit received data by reading the RCREG.
(12) Do the control action by reading the RCREG register.
(13) Repeat the steps from 10.

Conclusion:

Thus we studied about serial comm^n bet^n pc to PC PIC μcontroller.

Serial Communication



Expt. - Serial Communication with PC

```
#include <xc.h>//Include Controller specific .h
#pragma config OSC = HS//Oscillator Selection
#pragma config WDT = OFF//Disable Watchdog timer
#pragma config LVP = OFF//Disable Low Voltage Programming
#pragma config PBADEN = OFF//Disable PORTB Analog inputs
//Variables
#pragma idata
unsigned char string1[]={"\n\rSmart Logic Technologies"};\
unsigned char string2[]={"\n\rUSART Test Code"};
unsigned char string3[]={"\n\rSend 10 character to uC\n\r"};
unsigned char string4[]={"\n\rTransmitted Characters are:"};
unsigned char string5[]={"\n\rRx Tx test complete\n\r"};

//Function Prorotypes
void TXbyte(unsigned char data); //To transmit single character
void TXstring (unsigned char *string); //To transmit string
//Start of Main Program
void main()
 {
unsigned char i=0;
unsigned char rx_data [20]; // Buffer to store received data
TRISCbits.TRISC7=1;// RXD line as input
TRISCbits.TRISC6=0; // TXD line as output
SPBRG = 0x08;
SPBRGH = 0x02; // 0x0208 for 9600 baud
TXSTA = 0x24; // TX enable BRGH=1, SPEN=1
RCSTA = 0x90; // SPEN= 1, continuous RX = 1
BAUDCON = 0x08; // BRG16 = 1
```

```
TXstring (string1); // Transmit string 1
TXstring (string2); // Transmit string 2
TXstring (string3); // Transmit string 3
for (i=0; i<10; i++)
{
while (PIR1bits.RCIF==0); // Wait until data received
rx_data [i]= RCREG;// Read the received data
}
rx_data [10]= 0;//To make ASCIIZ string
TXstring (string4);// Transmit string 4
TXstring (rx_data);// Transmit received data
TXstring (string5);// Transmit string 5
while(1);                                   // loop forever
                                            // End of the program
}
void TXbyte(unsigned char data)
{
while(TXSTAbits.TRMT==0);//wait till transmit buffer is not empty
TXREG = data; // Transmit Data
}
void TXstring(unsigned char *string)
{
unsigned char i=0;
for(i=0;string[i]!='\0';i++) //loop till end of the string
TXbyte(string[i]);//Send single character
}
```

* Experiment no: 09 *

Title : Study for Raspberry Pi-4 / Arduino board / Beagle board and operating systems for the same. Understand the process of OS installation on the Raspberry Pi / Arduino board / Beagle Board.

S/W packages & H/W Used :
Arduino IDE, Noobs OS, Debian OS, Arduino Board, Raspberry Pi Board, Beagle Board.

Theory :

Internet of Things (IOT) :
IOT is a short term for Internet of things. The internet of things refers to the ever-growing network of physical objects that feature an IP address for Internet connectivity, and the comm^n that occurs and systems.

Examples of IOT :
1. Apple watch and Home kit
2. Smart cars.
3. Smart Refrigerators.
4. Google Glass.

Arduino :
An Arduino board consists of an atmol 8-bit AVR µcontroller.

Specifications of Arduino UNO :
Complementary components that facilitate programming and incorporation into other circuits. It's an open-source physical computing platform based on a simple µcontroller board.

- µcontroller : AT mega 328P
- Operating voltage : 5V.

- Input voltage (recommended) : 7-12 V
- Input voltage (limit) : 6-20V
- Digital I/O pins : 14 (of which 6 provide PWM o/p)
- Analog I/P pins : 6
- DC current per I/O pin : 20 mA
- SRAM : 2kB (A-Tmege 328P)

Raspberry-Pi :

A Raspberry-Pi is a series of small single board computers developed in the united kingdom by the Raspberry-Pi foundation to promote the teaching of basic computer science in school & in developing countries.

Features :

- All models features a Broad COM system on a chip (SoC) which includes on ARM compatible central processing unit (CPU) & on-chip graphics processing unit.
- CPU speed ranges from 700MHz to 1.5 GHz for the Pi 4 & on board memory range from 256MB to 4GB RAM

Beagle Board :

The Beagle Board is a low-power open-source single-board computer produced by Texas Instruments in Association with Digi-key & network element 14. The Beagle Board was also designed with open source s/w development in mind, and as a way of demonstrating the Texas Instruments OMAP3530 system-on-a-chip.

Components on Beagle Board:
- Processor : AM335X 1GHz ARM® Cortex-A8
- 512 MB DDR3 RAM
- 4GB 8-bit eMMC on-board flash storage
- 3D graphics accelator.
- Neon floating-point accelator.
- 2x PRU 32-bit µcontrollers.
- connectivity
- USB client for power & comm.

• Operating systems:
The foundation provides Raspbian, a Debian based Linux distribution for download, as well as third party Ubuntu, windows 10 IOT core, RISC OS & specialized media center distributions.
It promotes Python & scratch as the main programming language with support for other language.
Download it from official website:
https://www.raspberrypi.org/downloads/noobs/

• BeagleBone back:
The BeagleBone back includes a 2GB & 4GB on-board eMMC flash memory chip.

• OS which install on BeagleBone Black:
Angstrom, Android, Debian, Fedora, Buildroot Gentoo, Nerves Erlang/OTP, Sabayon, Ubuntu, Yocto, MINIX3.

• How to install Debian on BeagleBone Black:
① Download Debian imgxz file (iso file).
② Unzip the file.
③ Insert your (microSD (µSD) card into the proper slot.
④ Now open Win32 Disc imager, click the blue

folder icon, navigate to the debian img location.
(5) Remove the SD adaptor from the card slot.
(6) Remove the eMMC card from the adaptor.
(7) Plug the USB cable in & wait some time.
(8) If you are not seeing the LEDs swing back and fourth you will need to unplug the USB cable, then plug in the USB cable.
(9) Remove the eMMC card and reboot your BBB. You can reboot the BBB by removing & reconnecting the USB cable.
(10) Now the using putty, or your SSH flavor of choice, connect to the BBB using the IP address 192.168.172.0

Arduino:

The Arduino itself has no real operating system. You develop code for the Arduino using the Arduino IDE which you can download from Arduino-Home.

Arduino consists of both a physical programmable circuit Board (often refered as a µcontroller) & a piece of S/w, or IDE code to the physical board.

Conclusion:

Thus we studied about the Raspberry-Pi / Arduino Board / Beagle board.

Date : Page No. 41

* Experiment no 10 *

Title : Open-source prototype platform- Raspberry Pi/ Beagle board/Arduino.
2A - simple program digital read/write using LED and switch.
2B - Analog read/write using sensor and actuators.

Aim : (Objective)
To blink the LED when the switch is pressed using Arduino

S/W packages & H/W used :
Arduino IDE, Arduino Board, LED, Push Button.

Theory :
LED : (Light Emitting diode)
A LED is a semiconductor device light source that emits light when current flows through it. Electrons in the semiconductor recombine the e⁻ holes, releasing energy in the form of photons.

Push Button :
A push-button also called spellbutton or simply button is a simple switch mechanism to control some aspect of a machine or a process. Buttons are typically made out of a hard materials, usually plastic or metal.

Procedure :
① Connect the Arduino board to the Micro-IOT sensor board using the FRC cable provided with board.
② Connect the power supply adaptor and power on the circuit.

③ Open Arduino IDE and create a new sketch (program) for LED blinking using the above this.

④ In the Arduino IDE go to tools part & select the appropriate COM port.

⑤ In the Arduino IDE, click on the upload button ○ to compile & download the code into the Arduino Uno. When successfully downloaded the code will start running and you can observe the LED's blinking on the board.

Observation:

You can observe the following when you press a switch the LED will turn ON & after 1 sec. it will turn off.

Conclusion:

Thus we studied ~~that~~ the concept of Open-source prototype platform - Raspberry pi.

Title: Simple program for Analog read using sensor.

Aim: Write a simple program using Arduino to read analog values for LDR.

Apparatus:
Arduino Uno board, el-IOT sensor actuator board, Power adaptor.

Interface:

Peripheral	Arduino pin
Light Depend Register	A0 (CN11)

Procedure:

(1) Connect the Arduino board to the el-IOT sensor board using the FRC cable provided with the board.

(2) Connect the power supply adaptors & power on the circuit

(3) Open Arduino IDE and create a new sketch (program) using the above pins.

(4) In the Arduino IDE go to tools → Port and select the appropriate COM port.

(5) In the Arduino IDE click on the upload button to compile and download the code into the Arduino UNO. When successfully downloaded, the code will start running.

(6) Connect the LDR sensor to CN11 (pin A0). Open the serial Monitor from tools → serial monitor and observe the analog values of the LDR.

Observation :

The analog values from the LDR change, the LDR is exposed to light as against when it is shielded from light in darkness.

Conclusion :

Thus we studied that how to write a simple program to read analog using sensor.