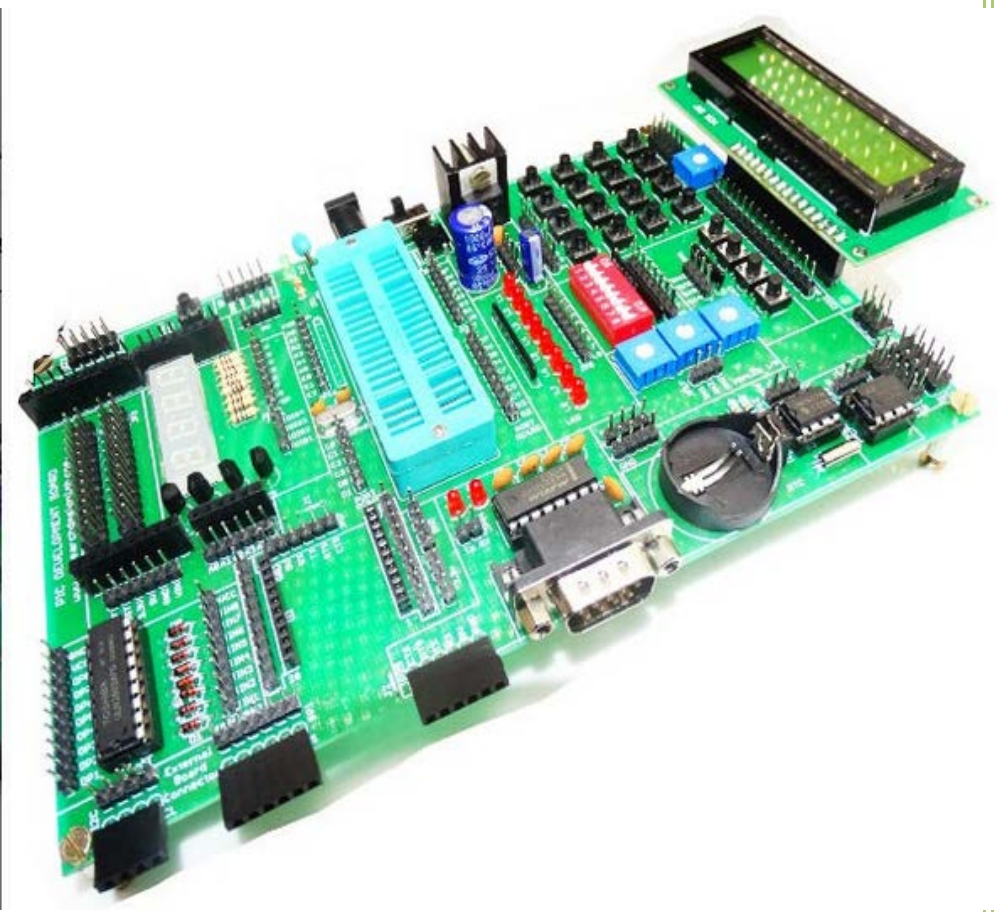
2014

Programming with PIC Microcontroller



Introduction of PIC

Getting started with MPLAB IDE

Interfacing

1) LED, LCD, KEYPAD

2) ADC, PWM, RELAYS

3) GSM

4)I2C, RTC

PIC Development Board

PIC CONTROLLER

Table of Contents

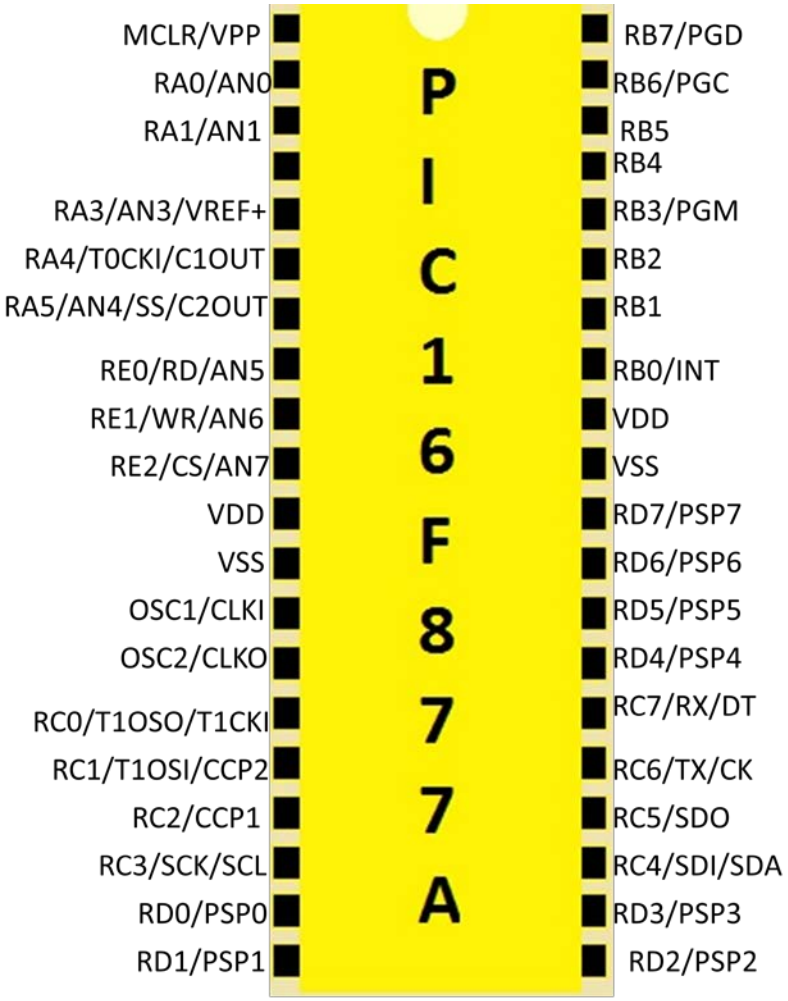
INTRODUCTION

EMBEDDED SYSTEMS

PIC16F877A

Overview:

The PIC 16F877A PIC microcontroller is one of the most popular general purpose microcontrollers. It is of 8-bit which means the most available operations are limited to 8-bits.It is a 40-pin IC.



Ports:

There is one 6-bit ports: A , 3 8-bit ports: B ,C,D and one 3 bit port:E.

PORTA (Pin 2 to 7)and TRISA register :PORTA is a 6-bit wide, bidirectional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1)
will make the corresponding. PORTA pin an input (i.e., put the corresponding output driver in a High-Impedance mode).Clearing a TRISAbit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).Reading the PORTA regiter reads the status of the pins,whereas writing to it will write to the port latch.All write operations are read-modify write operations.Therefore, a write to a port implies that the port pins are read, the value is modified and then written to the port data latch.

PORTB(Pin 33 to 40)and TRISB register: PORTB is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISB. Setting aTRISB bit (= 1)will make the corresponding PORTB pin an input(i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISB bit (= 0)will make the corresponding PORTB pin an output (i.e.,put the contents of the output latch on the selected pin).Three pins of PORTB are multiplexed with the In-Circuit.Debugger and Low-Voltage Programming function:RB3/PGM,RB6/PGC and RB7/PGD.

PORTC(pin 15 to 18 and pin 24 to 26)and TRISC register:PORTC is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISC bit (= 0)will make the corresponding PORTC pin an output (i.e.put the contents of the output latch on the selected pin).PORTC is multiplexed with several peripheral functions PORTC pins have Schmitt Trigger input buffers. When the I2C module is enabled, the PORTC<4:3>pins can be configured with normal I2C levels, or with SMBus levels, by using the CKE bit (SSPSTAT<6>).When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS

bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input. Since the TRIS bit override is in effect while the peripheral is enabled, read-modify write instructions (BSF, BCF, XORWF) with TRISC as the destination, should be avoided. The user should refer to corresponding peripheral section for the correct TRIS bit settings.

PORTD(Pin 19to22 and pin 27to30)**and TRISD register:** PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output. PORTD can be configured as an 8-bit wide microprocessor port (Parallel Slave Port) by setting control bit, PSPMODE (TRISE<4>). In this mode, the input buffers are TTL.

PORTE(Pin8 to 10)**and TRISE register:** PORTE has three pins (RE0/RD/AN5, RE1/WR/AN6 and RE2/CS/AN7) which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers.The PORTE pins become the I/O control inputs for the microprocessor port when bit PSPMODE (TRISE<4>) is set. In this mode, the user must make certain that the TRISE<2:0> bits are set and that the pins are configured as digital inputs. Also, ensure that ADCON1 is configured for digital I/O. In this mode, the input buffers are TTL. PORTE pins are multiplexed with analog inputs. When selected for analog input, these pins will read as '0's.

MPLAB IDE:

MPLAB IDE is a free integrated toolset for the development of embedded application on microchip IC and dsPIC microcontroller.

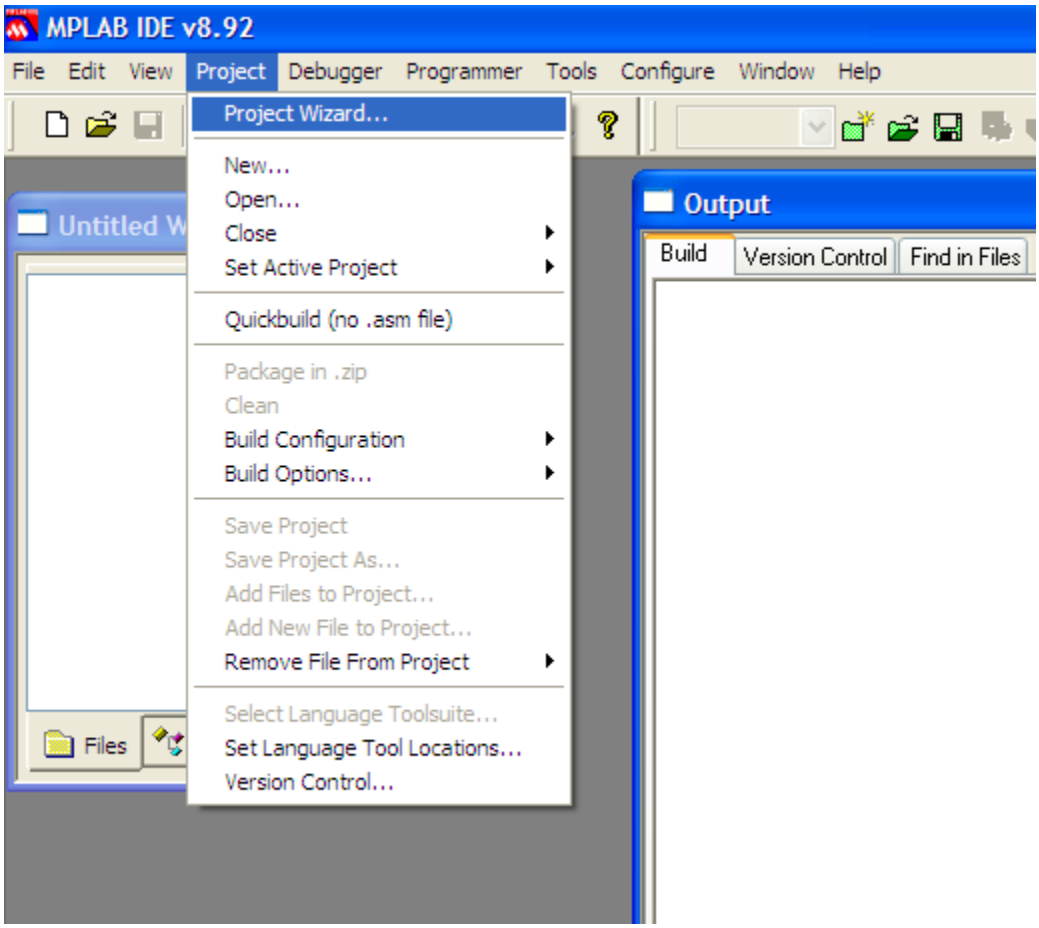
Install MPLAB by following the instructions sets provided in your software.

Creating a new project:

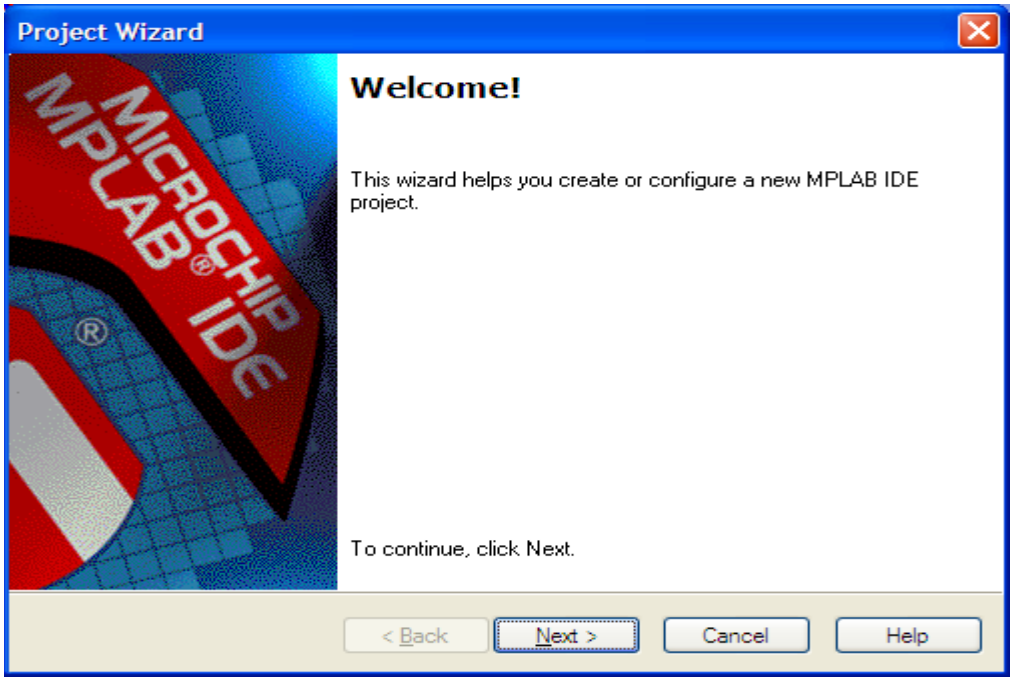
1) Open MPLAB

2) Create a folder in any drive.

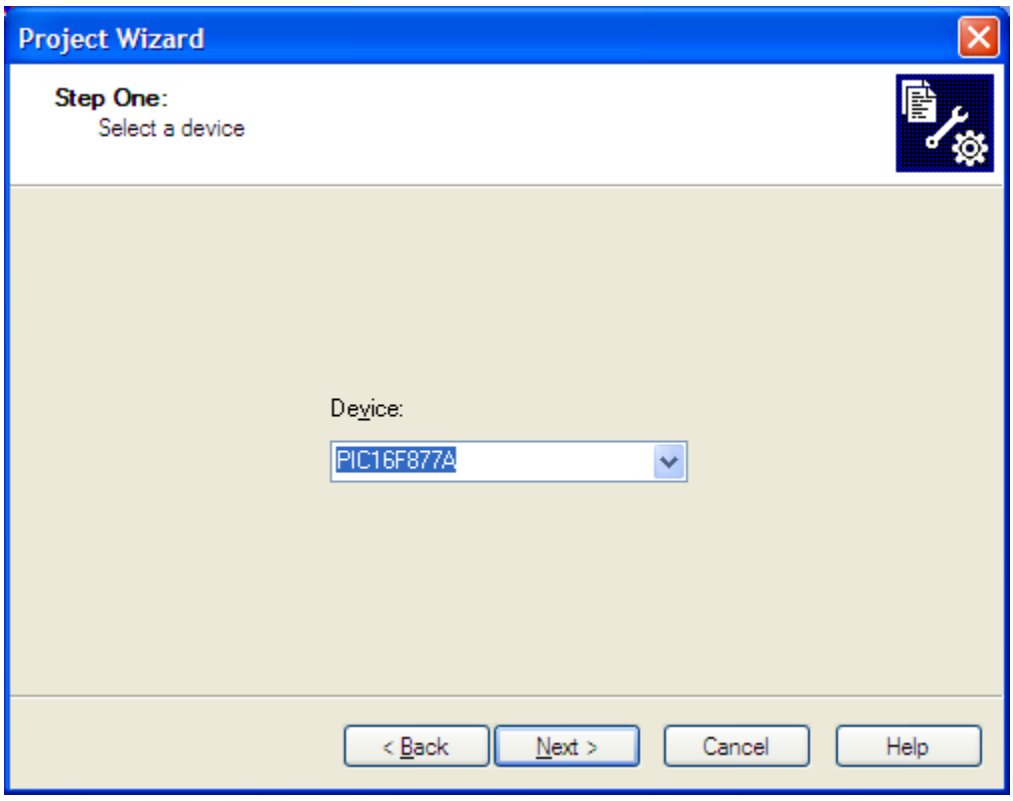
3) Select project->project wizard



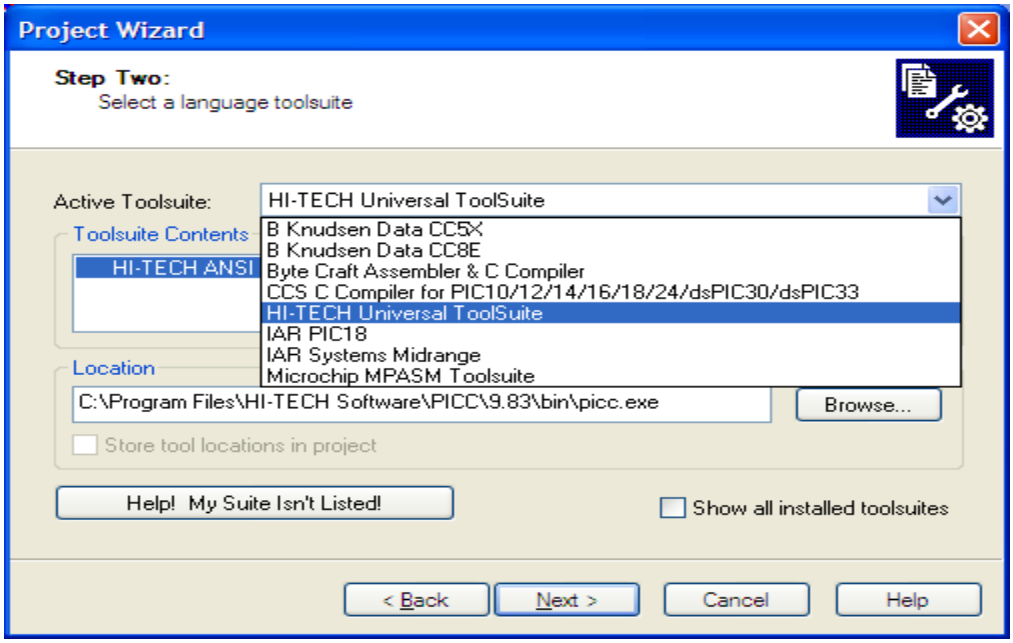
4) Click on next

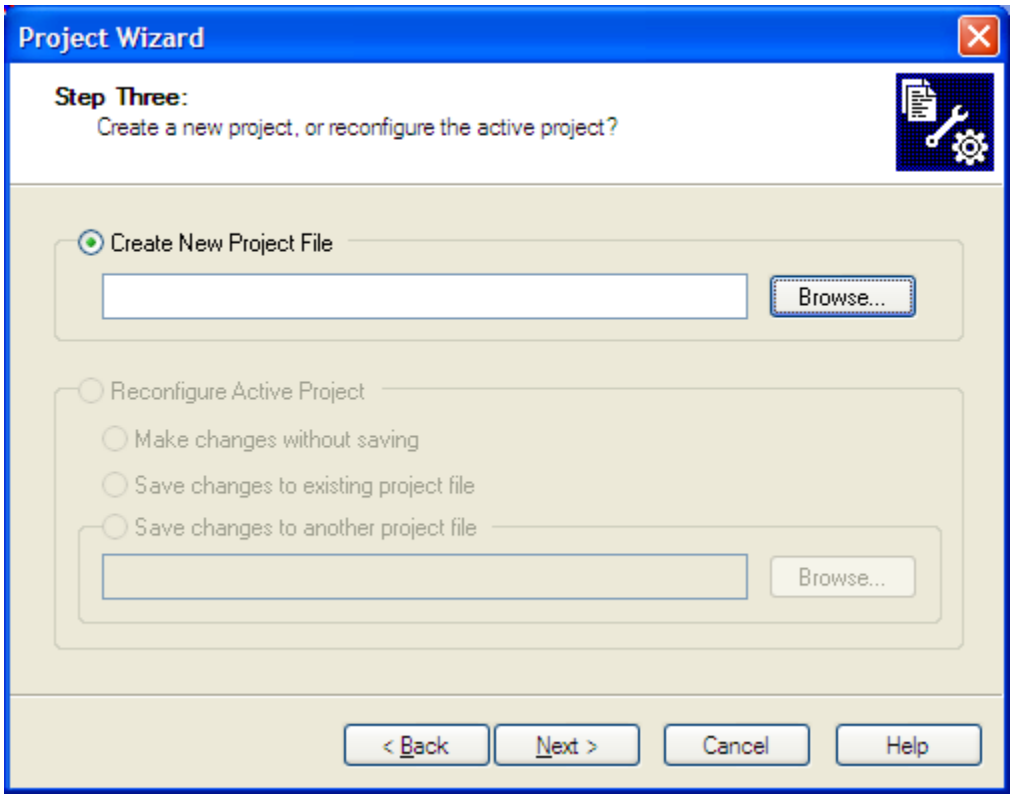


5) Select PIC16F7877A then click on next.

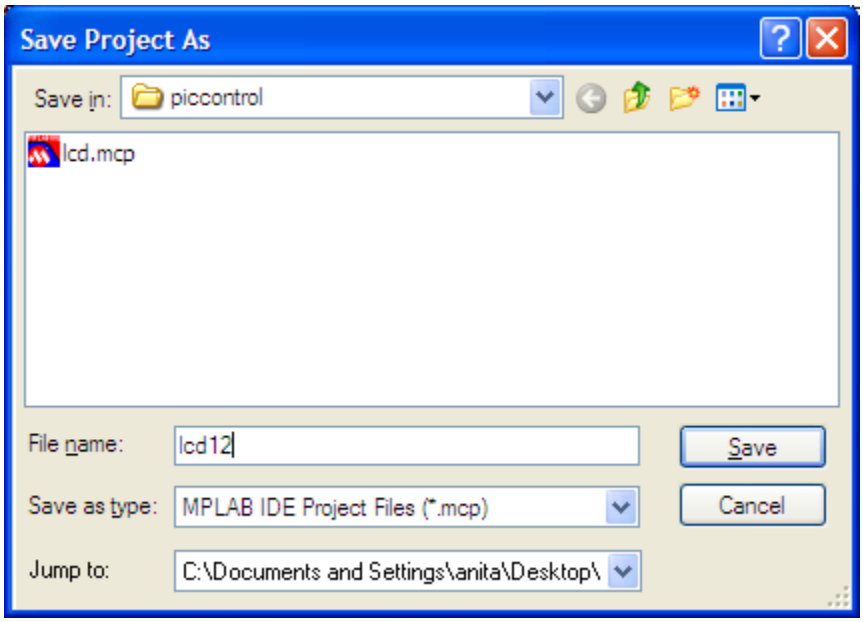


6) Select HI-TECH Universal Tool Suite and click next

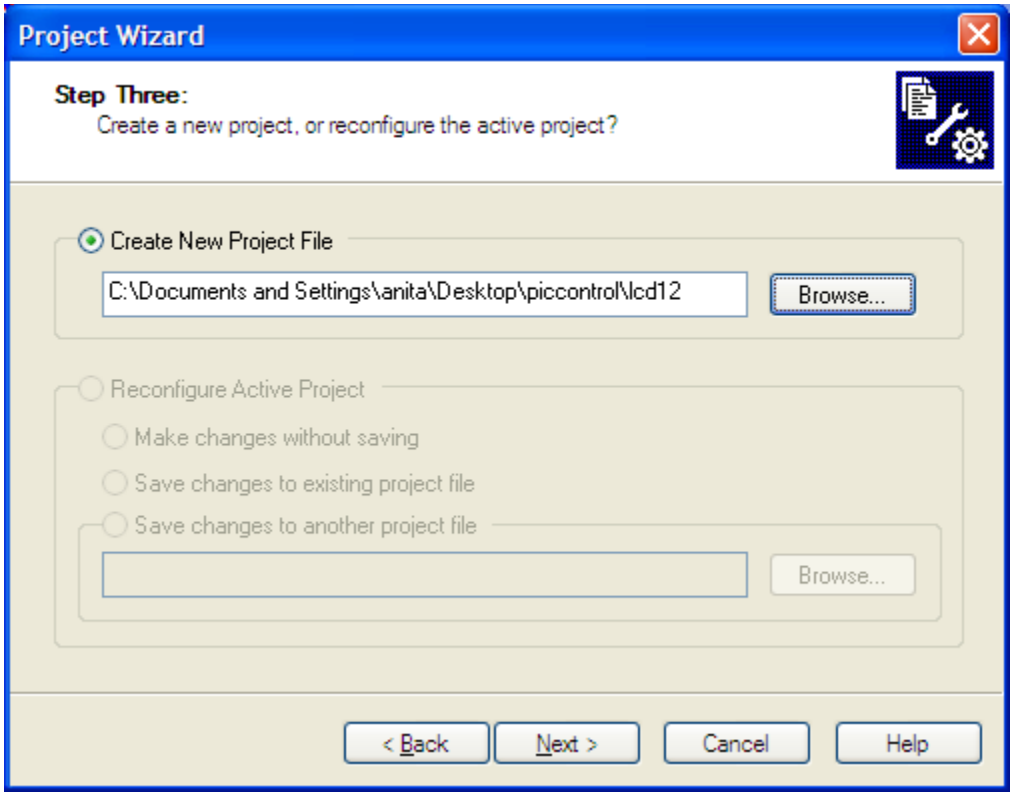




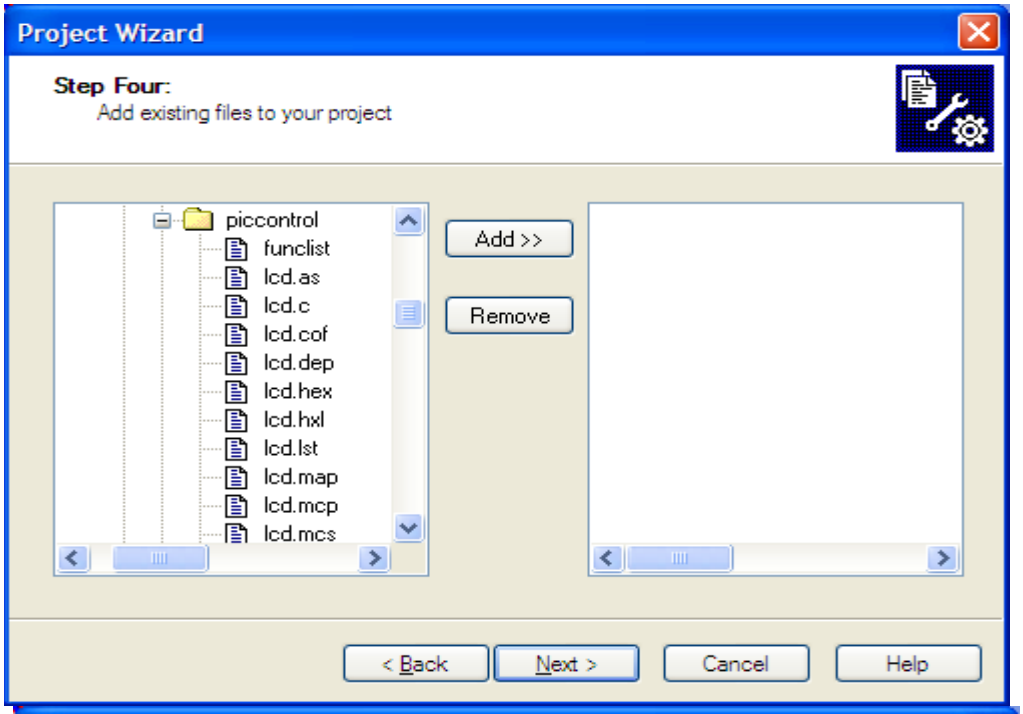
7) Click on browse and select the folder you saved on the drive and write a filename ex: lcd12.

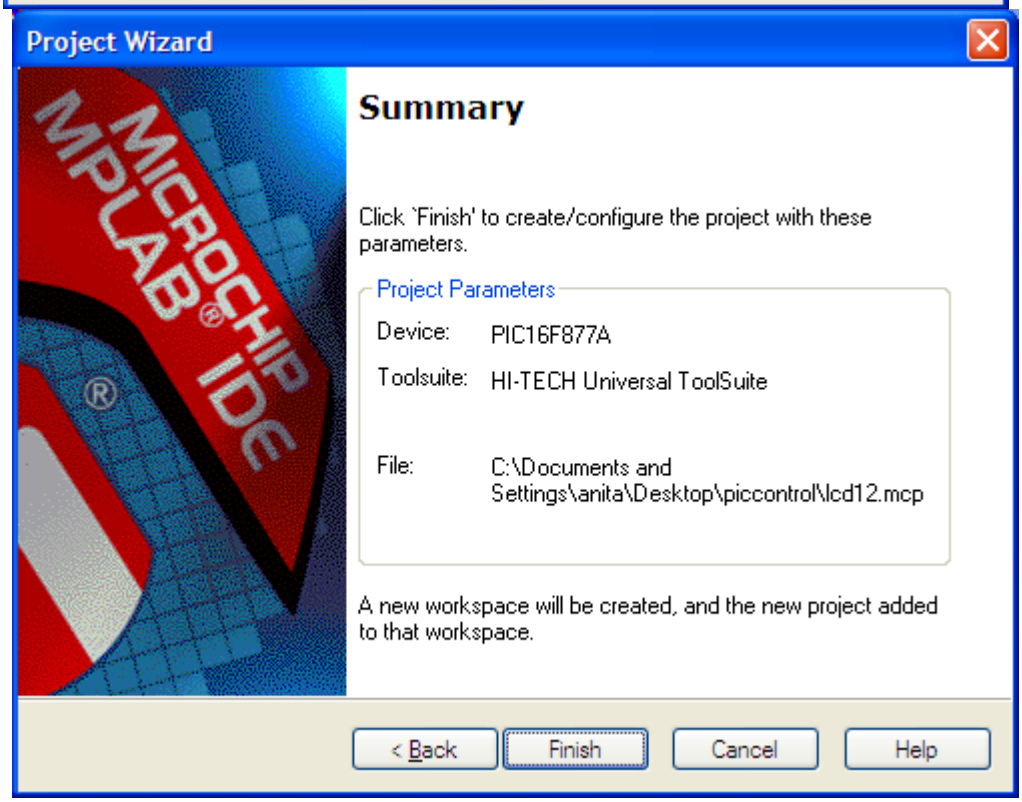


8) Click on save

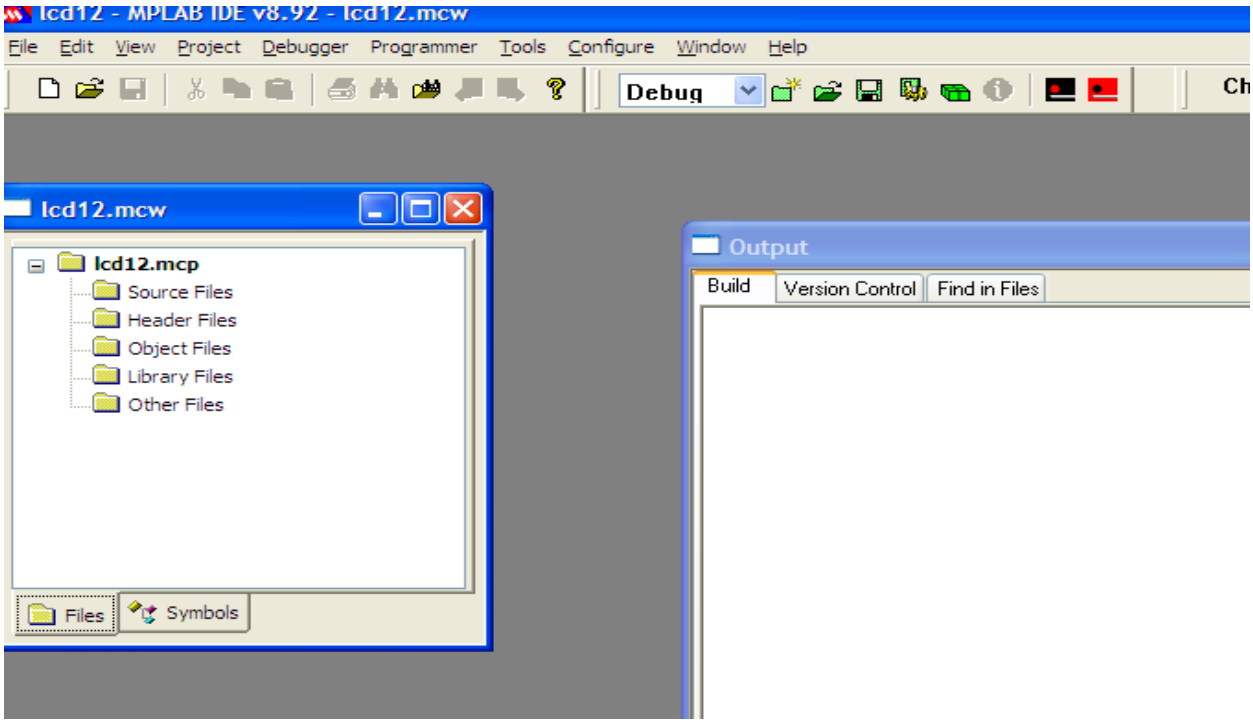


9)Click on next->next->next->finish

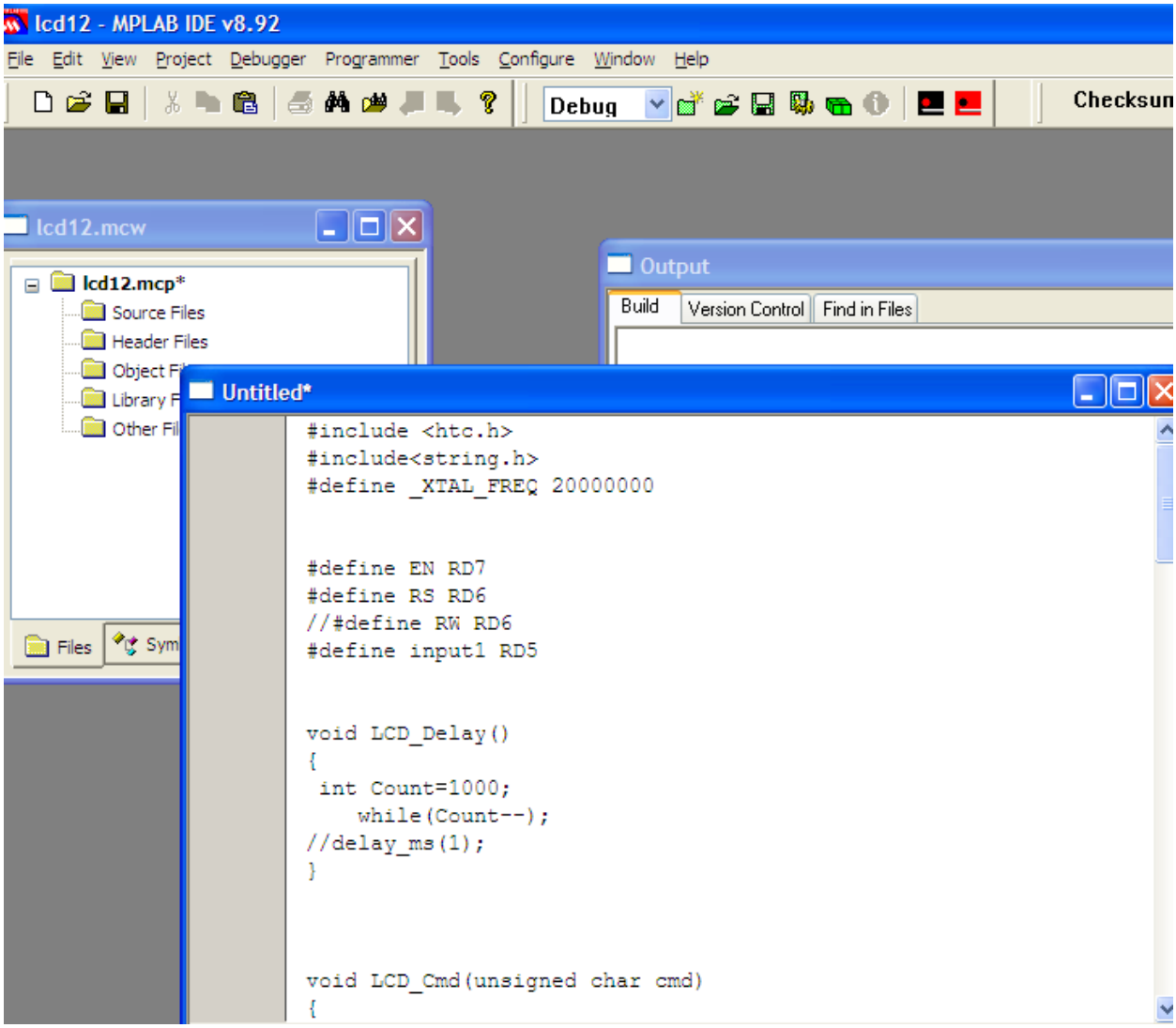




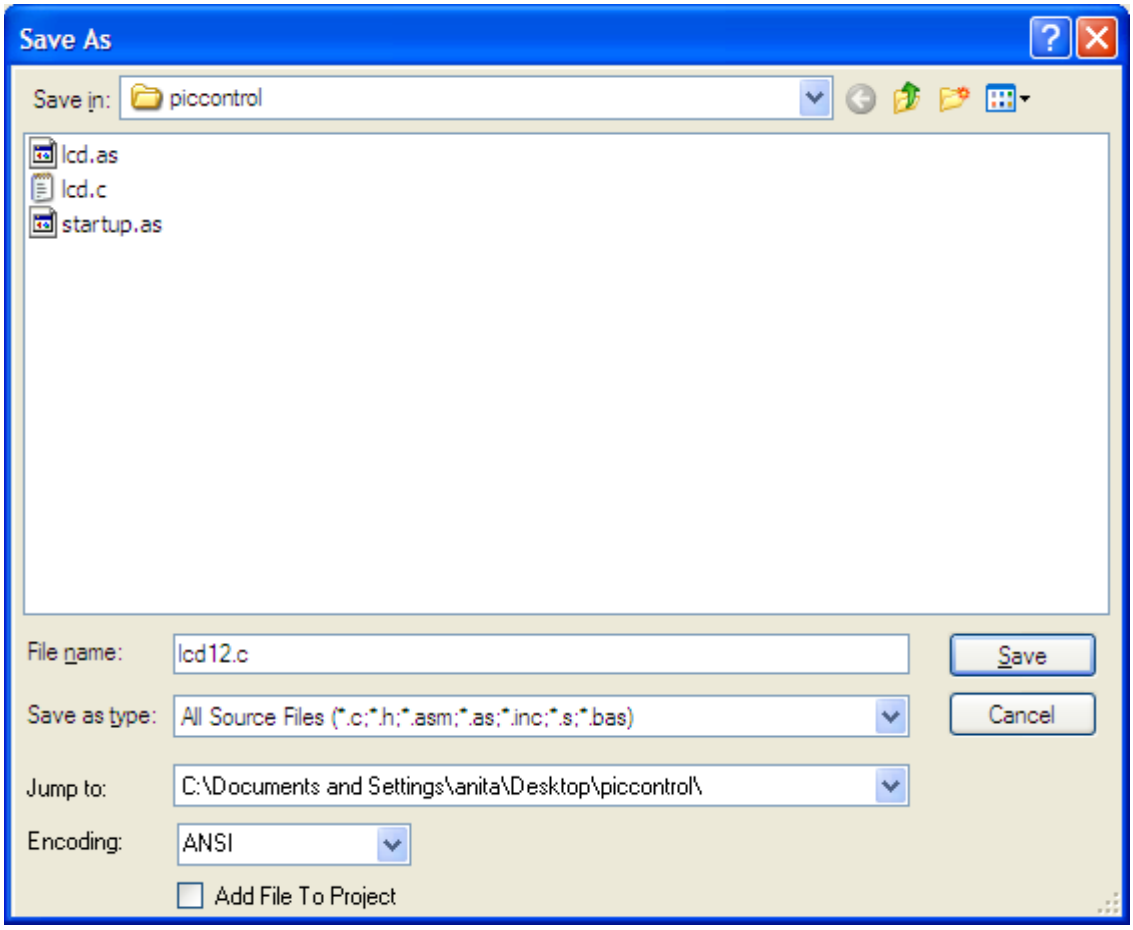
10) You will get the following window.



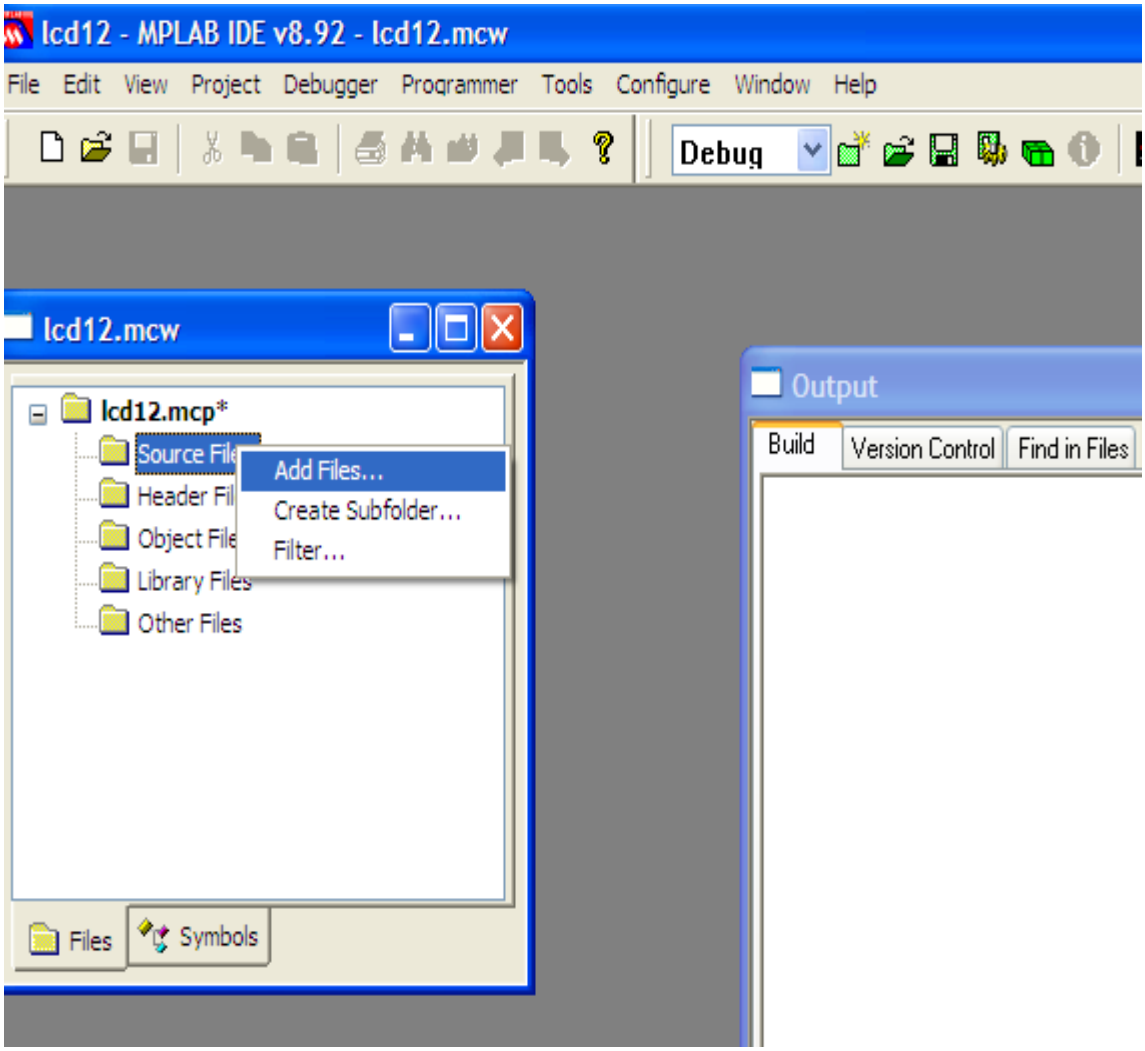
11) Click on file->new->type a program

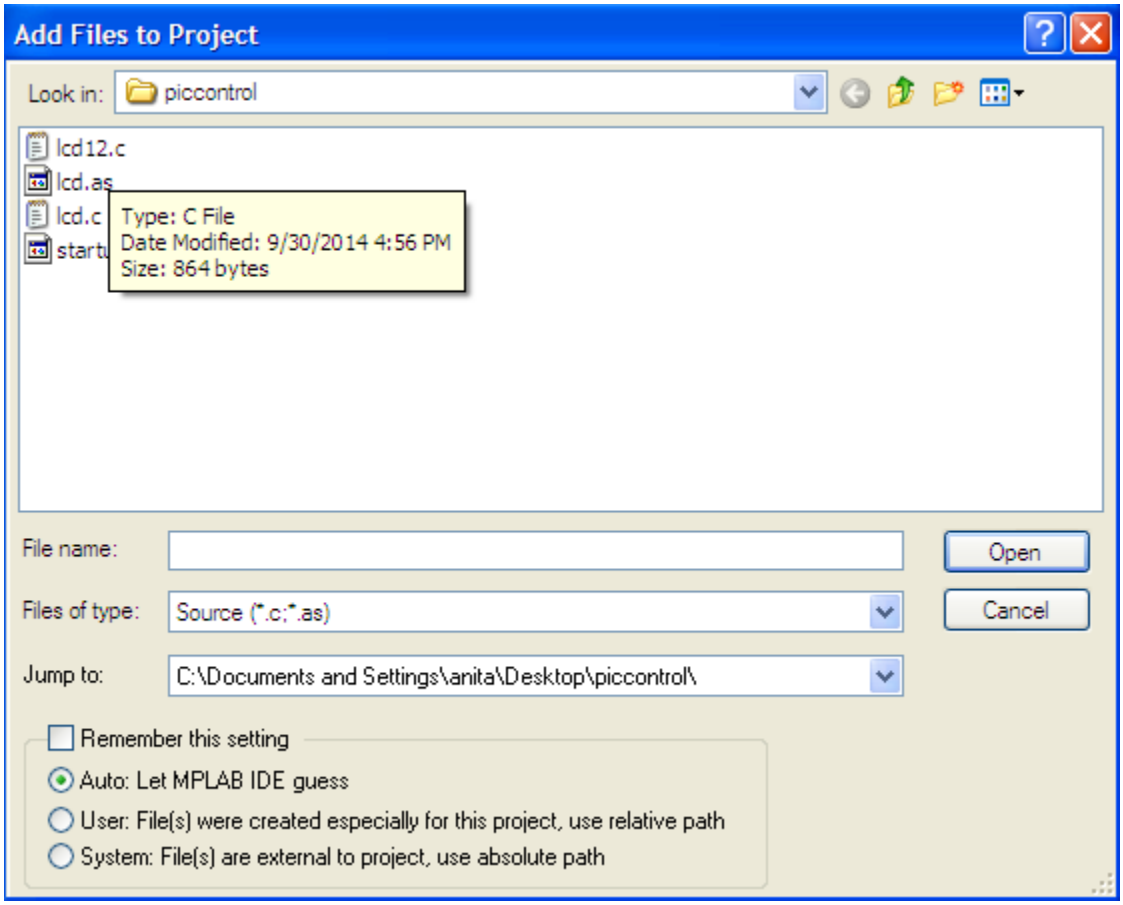


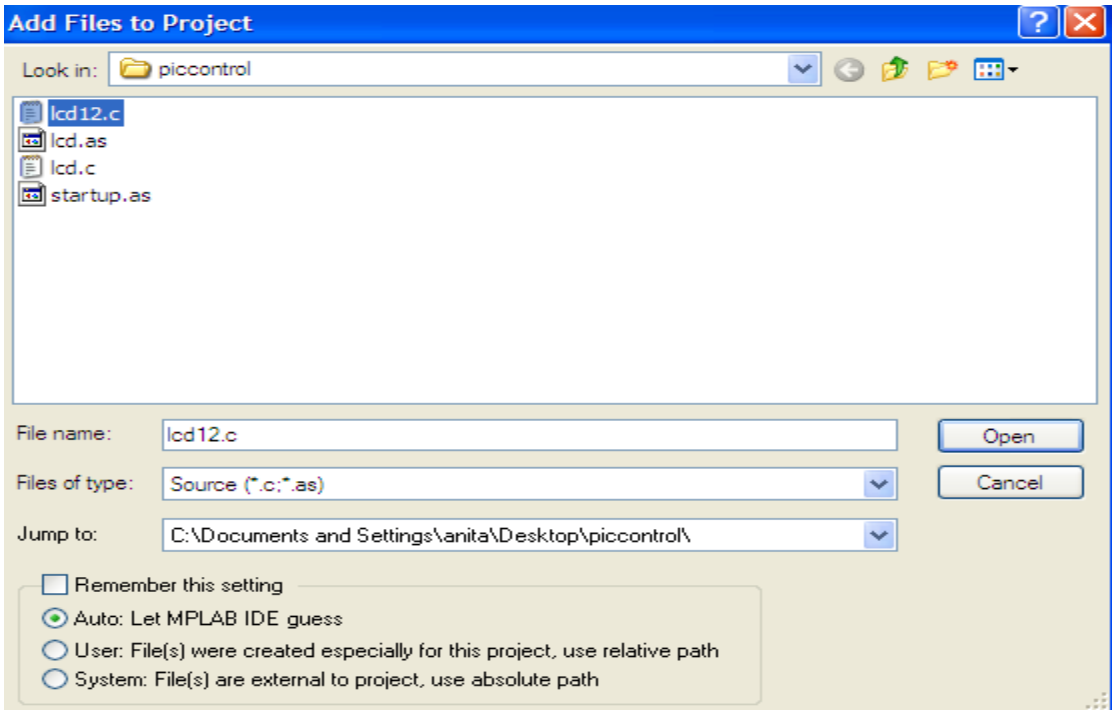
12) Click on save->save it in the same folder with .c extension and click on save.

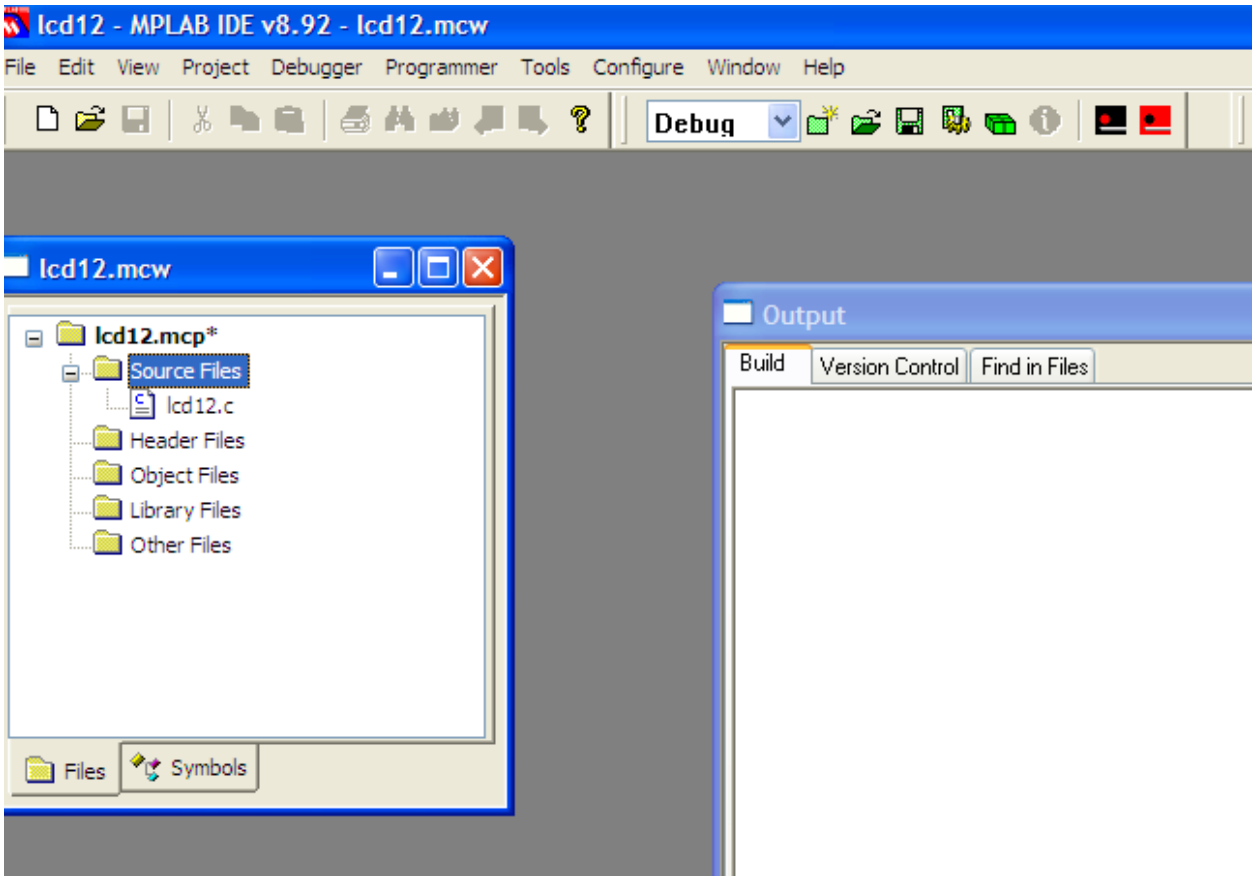


13) Right click on source file ->add files->select your .c file->click on open.

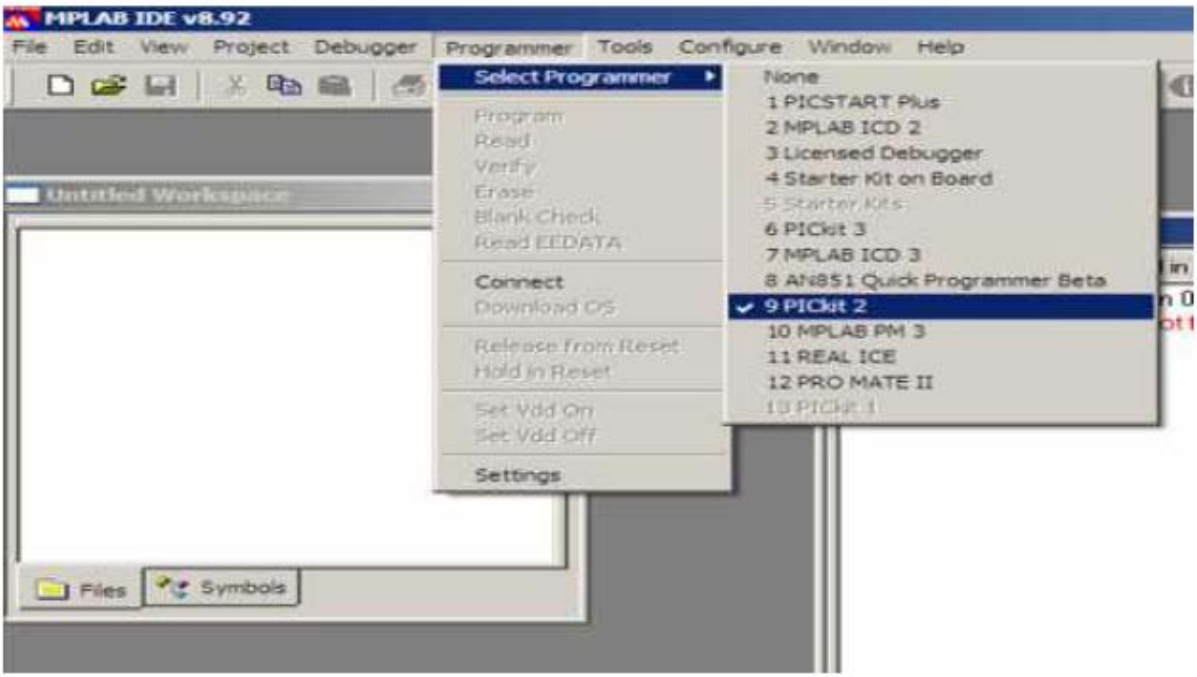

lcd12 - MPLAB IDE v8.92 - lcd12.mcw
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
lcd12.mcw
lcd12.mcp*
Source File
Header File
Object File
Library Files
Other Files
Add Files...
Create Subfolder...
Filter...
Files
Symbols
Output
Build
Version Control
Find in Files


Add Files to Project
Look in: piccontrol
lcd12.c
lcd.as
lcd.c
startu
Type: C File
Date Modified: 9/30/2014 4:56 PM
Size: 864 bytes
File name:
Open
Files of type: Source (*.c;*.as)
Cancel
Jump to: C:\Documents and Settings\anita\Desktop\piccontrol\
Remember this setting
Auto: Let MPLAB IDE guess
User: File(s) were created especially for this project, use relative path
System: File(s) are external to project, use absolute path

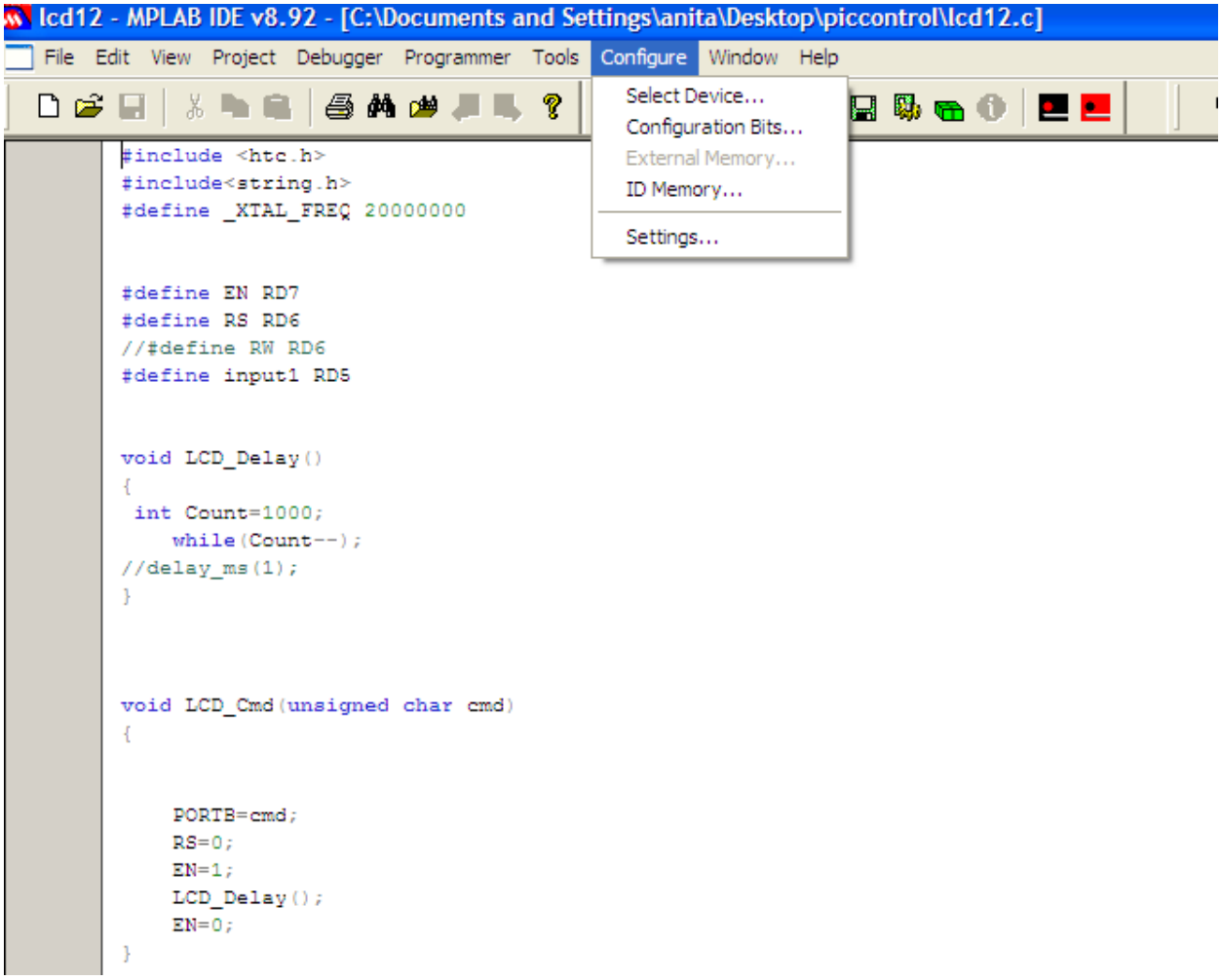

Add Files to Project
Look in: piccontrol
lcd12.c
lcd.as
lcd.c
startup.as
File name: lcd12.c
Open
Files of type: Source (*.c;*.as)
Cancel
Jump to: C:\Documents and Settings\anita\Desktop\piccontrol\
Remember this setting
Auto: Let MPLAB IDE guess
User: File(s) were created especially for this project, use relative path
System: File(s) are external to project, use absolute path

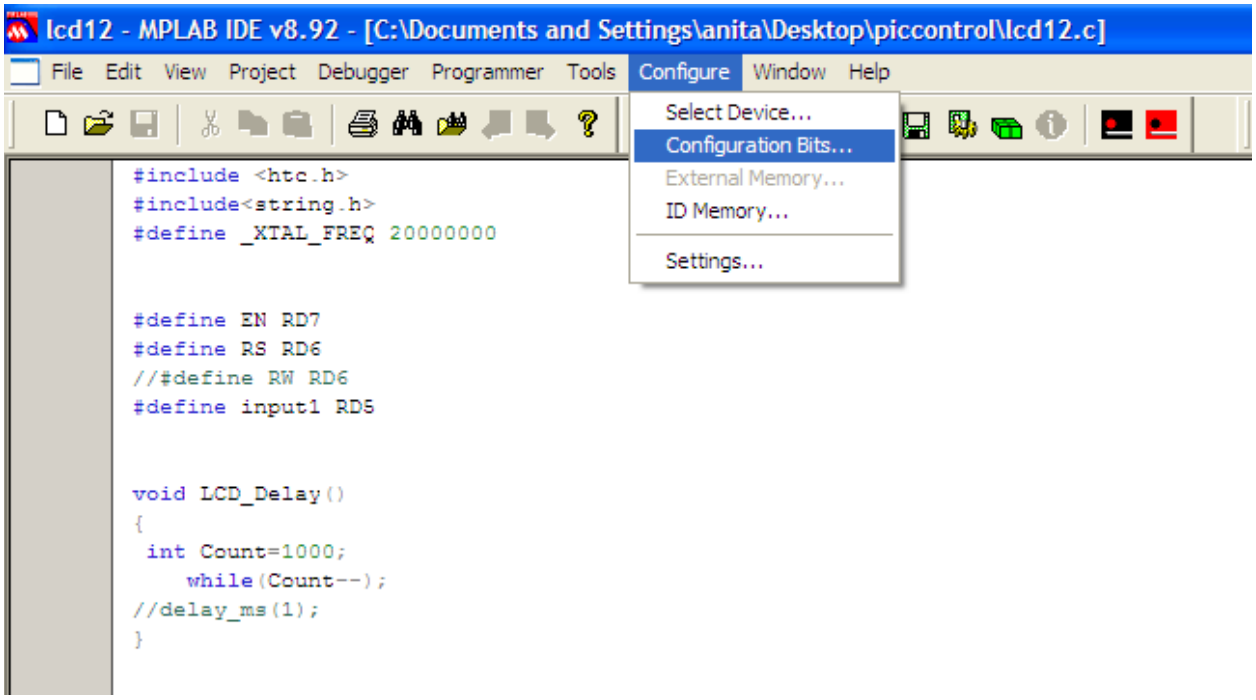


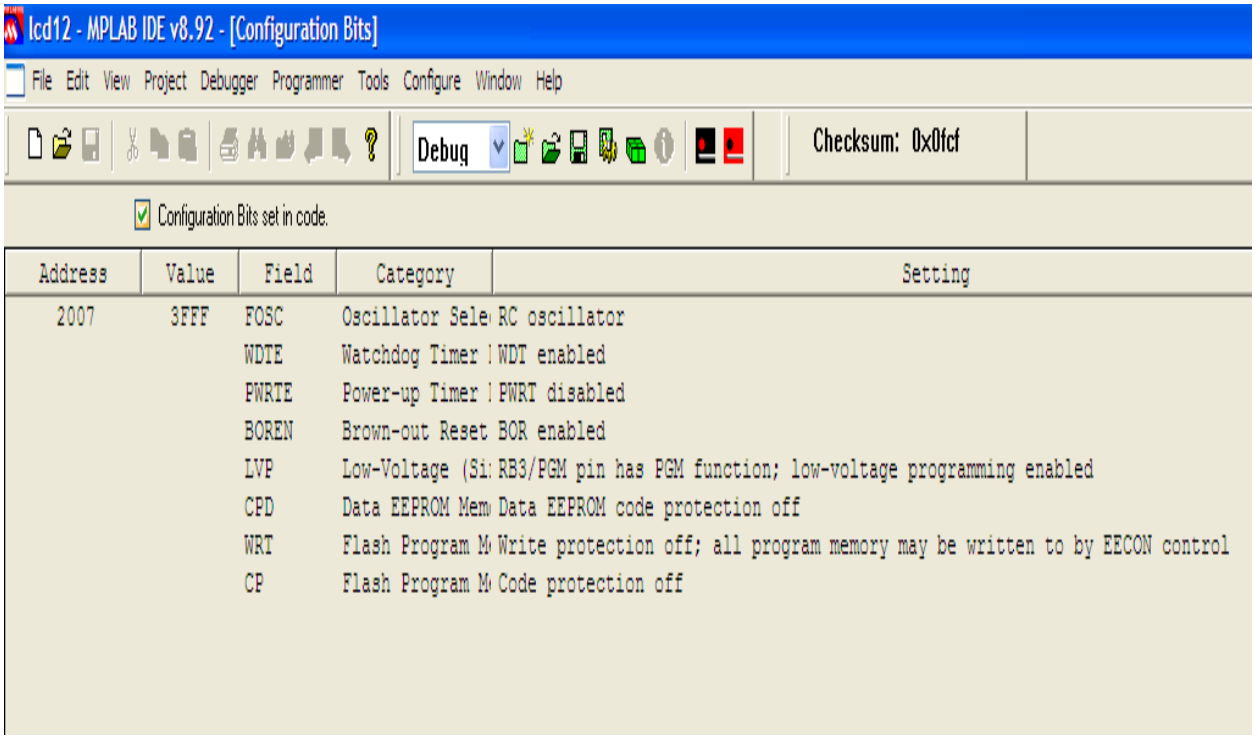
14)click on programmer->select programmer->9PICkit 2

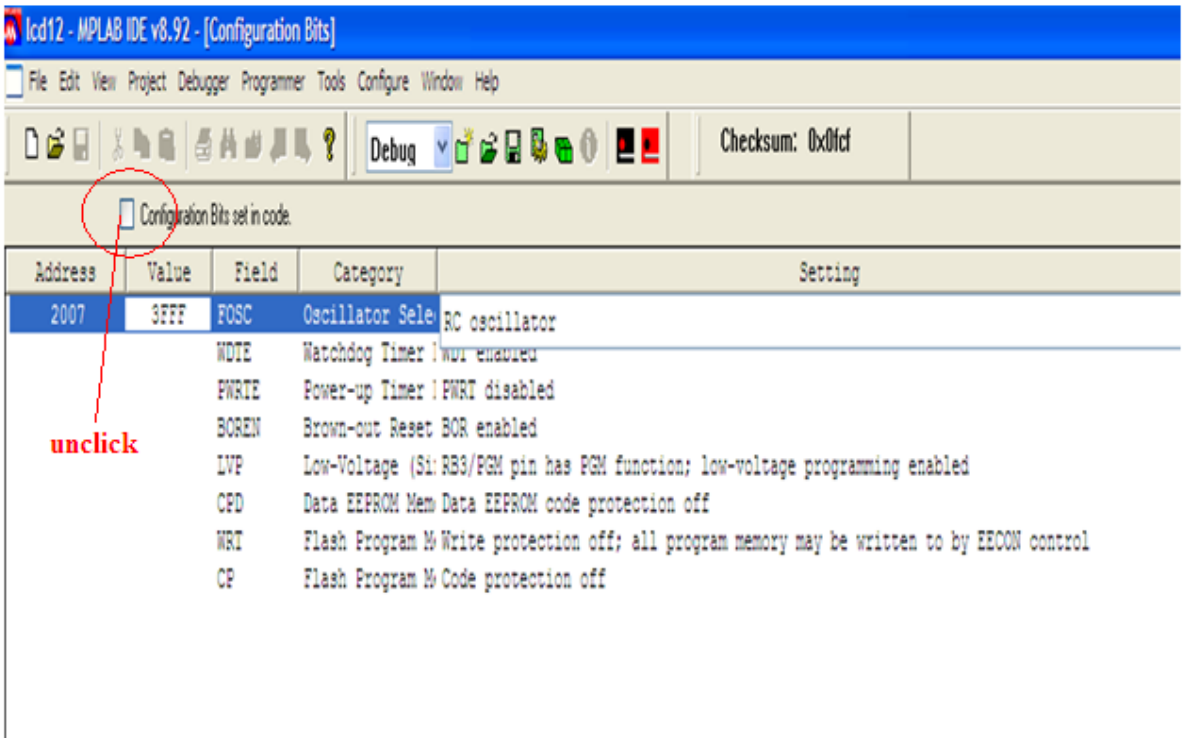


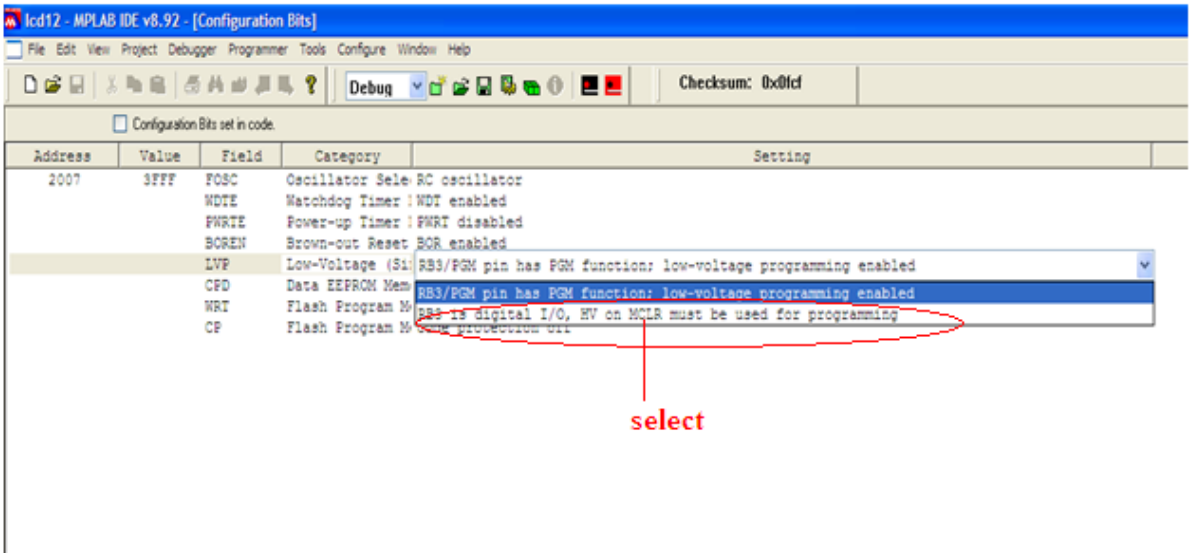
15) Click on configure ->configuration bits->unclick the configuration bits set in code->click ok-select low voltage programming->then click the configuration set in code

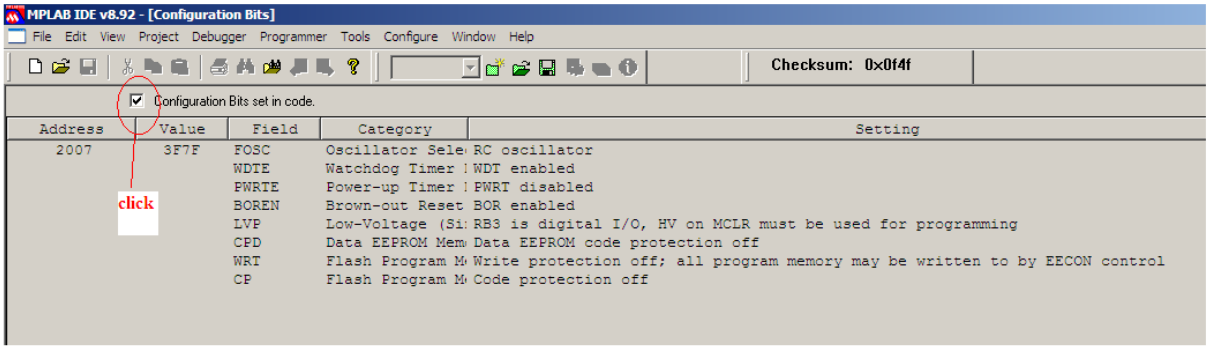

lcd12 - MPLAB IDE v8.92 - [C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.c]
File Edit View Project Debugger Programmer Tools Configure Window Help
Select Device...
Configuration Bits...
External Memory...
ID Memory...
Settings...
#include <htc.h>
#include<string.h>
#define _XTAL_FREQ 20000000
#define EN RD7
#define RS RD6
//#define RW RD6
#define input1 RD5
void LCD_Delay()
{
int Count=1000;
while(Count--);
//delay_ms(1);
}
void LCD_Cmd(unsigned char cmd)
{
PORTB=cmd;
RS=0;
EN=1;
LCD_Delay();
EN=0;
}


lcd12 - MPLAB IDE v8.92 - [C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.c]
File Edit View Project Debugger Programmer Tools Configure Window Help
Select Device...
Configuration Bits...
External Memory...
ID Memory...
Settings...
#include <htc.h>
#include<string.h>
#define _XTAL_FREQ 20000000
#define EN RD7
#define RS RD6
//#define RW RD6
#define input1 RD5
void LCD_Delay()
{
int Count=1000;
while(Count--);
//delay_ms(1);
}

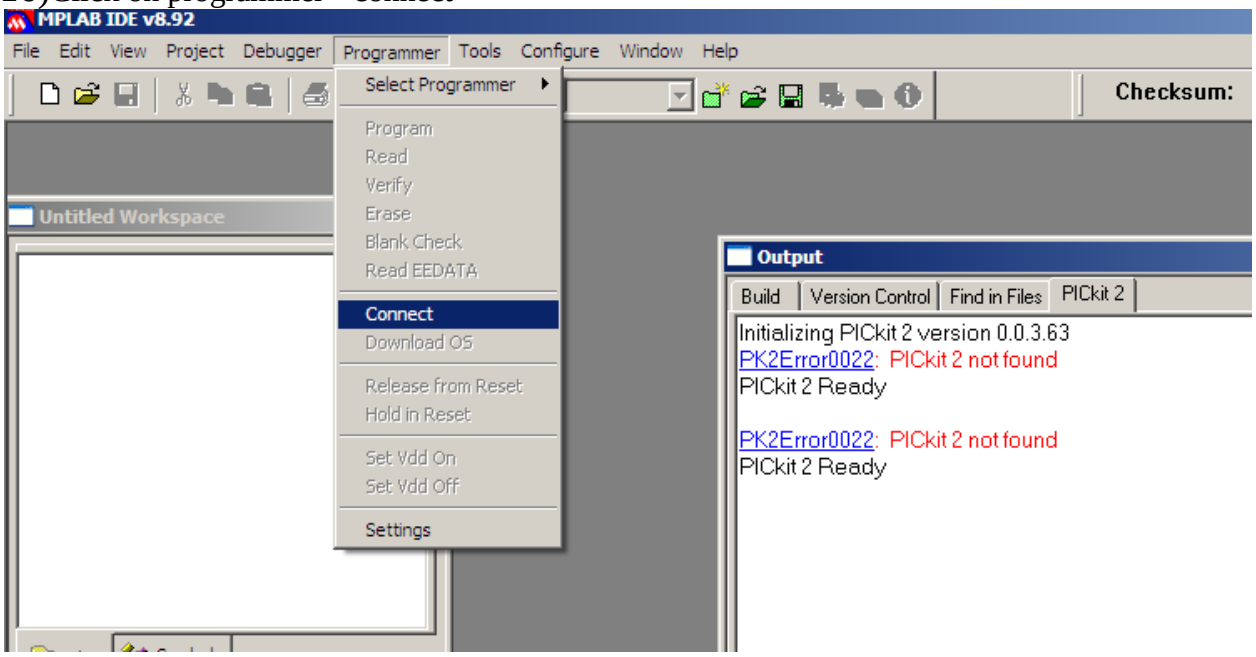

lcd12 - MPLAB IDE v8.92 - [Configuration Bits]
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x0fcf
Configuration Bits set in code.
Address Value Field Category Setting
2007 3FFF FOSC Oscillator Sele RC oscillator
WDTE Watchdog Timer I WDT enabled
PWRTE Power-up Timer I PWRT disabled
BOREN Brown-out Reset BOR enabled
LVP Low-Voltage (Si RB3/PGM pin has PGM function; low-voltage programming enabled
CPD Data EEPROM Mem Data EEPROM code protection off
WRT Flash Program M Write protection off; all program memory may be written to by EECON control
CP Flash Program M Code protection off


lcd12 - MPLAB IDE v8.92 - [Configuration Bits]
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x0fcf
Configuration Bits set in code.
Address
Value
Field
Category
Setting
2007
3FFF
FOSC
Oscillator Sele
RC oscillator
WDTE
Watchdog Timer
PWRTE
Power-up Timer
PWRT disabled
BOREN
Brown-out Reset
BOR enabled
LVP
Low-Voltage (Si
RB3/PGM pin has PGM function; low-voltage programming enabled
CPD
Data EEPROM Mem
Data EEPROM code protection off
WRT
Flash Program M
Write protection off; all program memory may be written to by EECON control
CP
Flash Program M
Code protection off
unclick


lcd12 - MPLAB IDE v8.92 - [Configuration Bits]
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x0fcf
Configuration Bits set in code.
Address
Value
Field
Category
Setting
2007
3FFF
FOSC
Oscillator Sele
RC oscillator
WDTE
Watchdog Timer
WDT enabled
PWRTE
Power-up Timer
PWRT disabled
BOREN
Brown-out Reset
BOR enabled
LVP
Low-Voltage (Si
RB3/PGM pin has PGM function; low-voltage programming enabled
CPD
Data EEPROM Mem
WRT
Flash Program M
CP
Flash Program M
RB3/PGM pin has PGM function; low-voltage programming enabled
RB3 is digital I/O, HV on MCLR must be used for programming
select

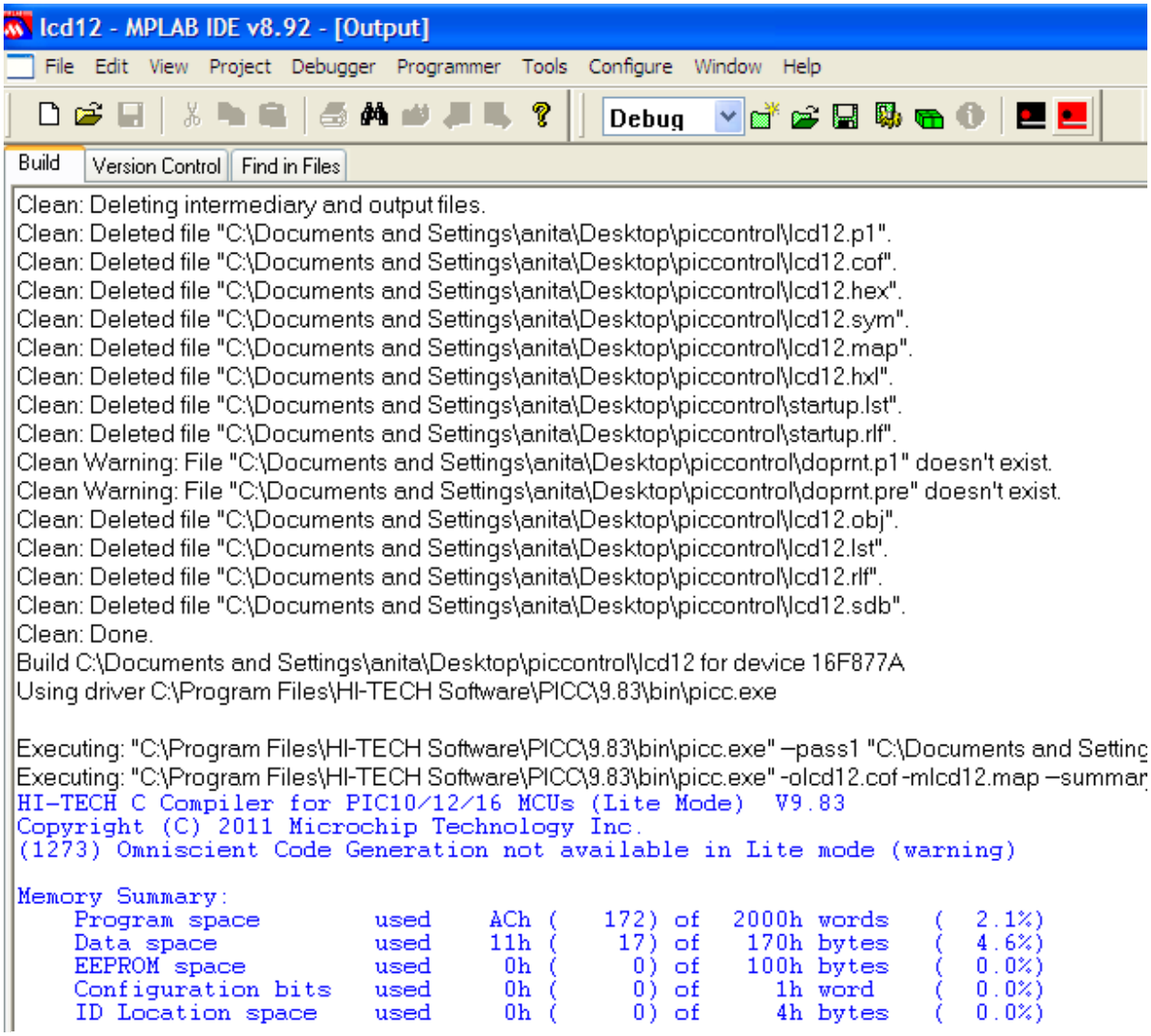


16)Click on programmer->connect



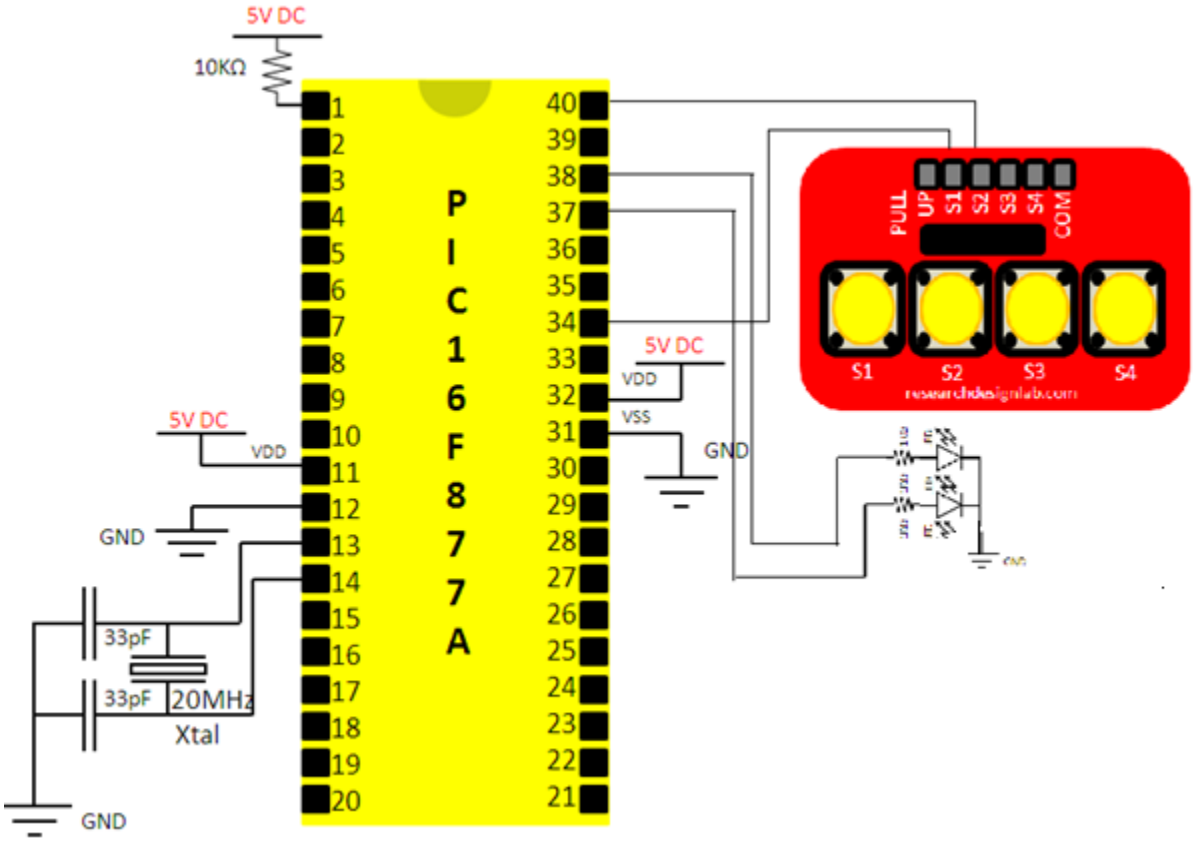
17) Click on compile


lcd12 - MPLAB IDE v8.92 - [C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.c]
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Checksum: 0x5af1
Rebuild with Compiler for PIC10/12/16 MCUs (Lite Mode) V9.83
click
#include <htc.h>
#include<string.h>
#define _XTAL_FREQ 20000000
#define EN RD7
#define RS RD6
//#define RW RD6
#define input1 RD5
void LCD_Delay()
{
int Count=1000;
while(Count--);
//delay_ms(1);
}
void LCD_Cmd(unsigned char cmd)
{
PORTB=cmd;
RS=0;
EN=1;
LCD_Delay();
EN=0;


lcd12 - MPLAB IDE v8.92 - [Output]
File Edit View Project Debugger Programmer Tools Configure Window Help
Debug
Build
Version Control
Find in Files
Clean: Deleting intermediary and output files.
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.p1".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.cof".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.hex".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.sym".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.map".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.hxl".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\startup.lst".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\startup.rlf".
Clean Warning: File "C:\Documents and Settings\anita\Desktop\piccontrol\doprnt.p1" doesn't exist.
Clean Warning: File "C:\Documents and Settings\anita\Desktop\piccontrol\doprnt.pre" doesn't exist.
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.obj".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.lst".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.rlf".
Clean: Deleted file "C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.sdb".
Clean: Done.
Build C:\Documents and Settings\anita\Desktop\piccontrol\lcd12 for device 16F877A
Using driver C:\Program Files\HI-TECH Software\PICC\9.83\bin\picc.exe
Executing: "C:\Program Files\HI-TECH Software\PICC\9.83\bin\picc.exe" –pass1 "C:\Documents and Setting
Executing: "C:\Program Files\HI-TECH Software\PICC\9.83\bin\picc.exe" -olcd12.cof -mlcd12.map –summar
HI-TECH C Compiler for PIC10/12/16 MCUs (Lite Mode) V9.83
Copyright (C) 2011 Microchip Technology Inc.
(1273) Omniscient Code Generation not available in Lite mode (warning)
Memory Summary:
Program space used ACh (172) of 2000h words (2.1%)
Data space used 11h (17) of 170h bytes (4.6%)
EEPROM space used 0h (0) of 100h bytes (0.0%)
Configuration bits used 0h (0) of 1h word (0.0%)
ID Location space used 0h (0) of 4h bytes (0.0%)
Running this compiler in PRO mode, with Omniscient Code Generation enabled,
produces code which is typically 40% smaller than in Lite mode.
See http://microchip.htsoft.com/portal/pic_pro for more information.
Loaded C:\Documents and Settings\anita\Desktop\piccontrol\lcd12.cof.
********** Build successful! **********

GETTING STARTED WITH EMBED C PROGRAMMING:

Lab 1 . LED Blinking using PIC controller (16F877A) with MPLAB:



I/O Connections :

PORT B4→LED1

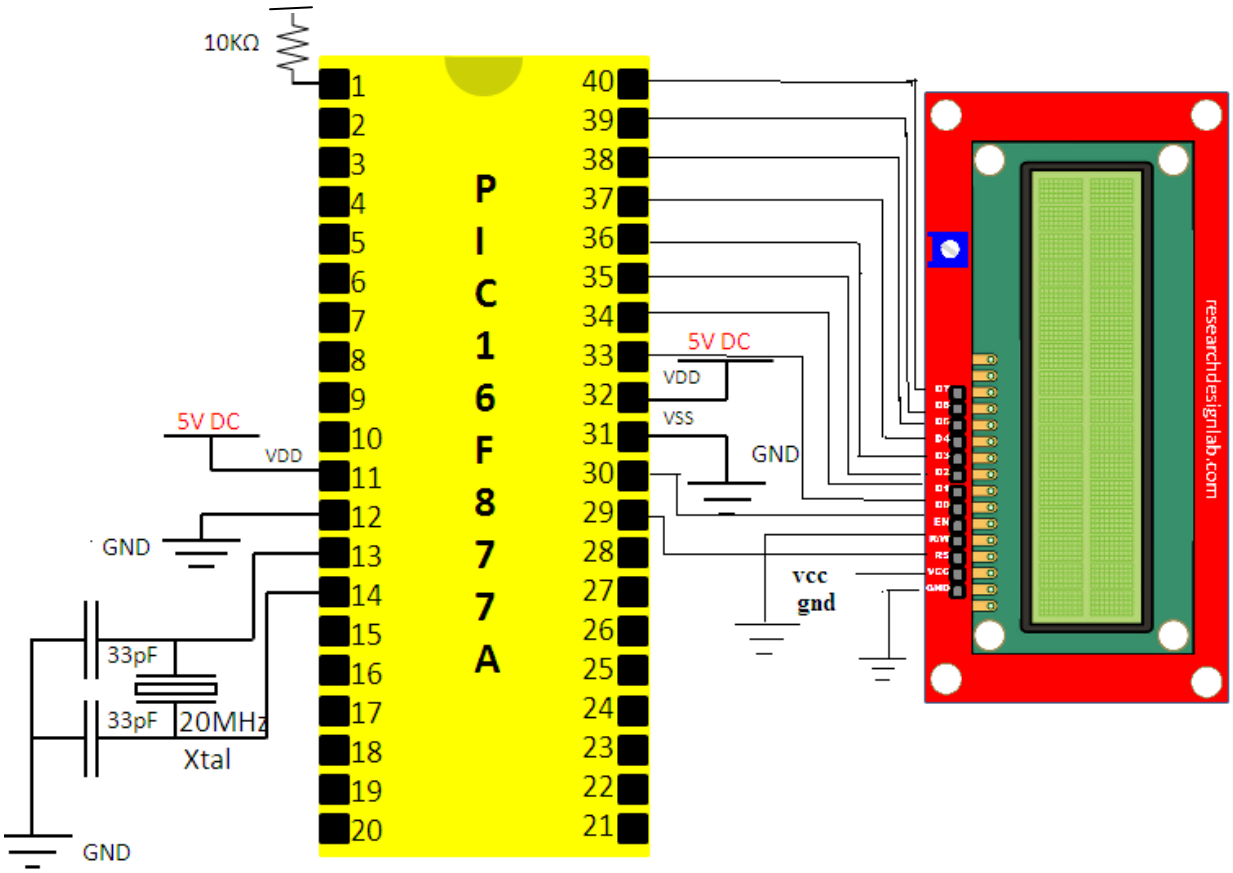
PORTB5→LED2

PORTB6→Switch1

PORTB7→Switch2

```
#include <htc.h>
#define _XTAL_FREQ 20000000                  //crystal frequency of 20MHZ
#define Input1 RB7                           //set port RB7 as input port
#define Input2 RB1                           //set port RB1 as input port
#define Output1 RB4                          //set port RB4 as output port
#define Output2 RB5                          //set port RB5 as output port

void main()
{
        TRISB=0X82;                          //use portB register as input as well as output  port
   while(1)                                  //infinite loop
     {
          if(Input1==0)                      //if switch1  is pressed ie connect port RB7 to sw1
          {

                Output1=1;                   //blink both the LED'S
                Output2=1;

          }
        else if(Input2==0)                   //If switch2 is pressed ie connect port RB1 to sw2

          {
                Output1=0;                   //both the LED'S are turned off
                Output2=0;

          }
     }
}
```

Lab2.To display a message on LCD using pic controller



I/O connection:

PORT B0 tO B7→DO to D7 of LCD

ENABLE→D7

R/W→GROUND

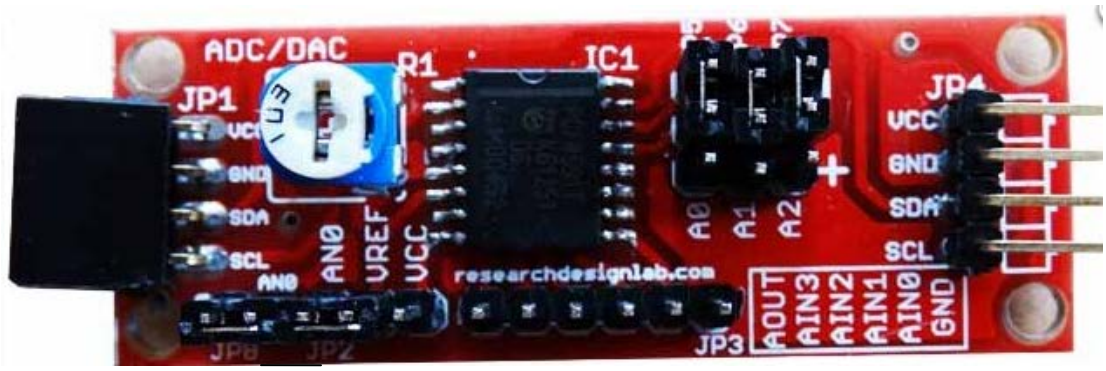
R/S→D6

```
#include <htc.h>
#include<string.h>
#define _XTAL_FREQ 20000000                    //crystal frequency of 20MHZ
#define EN RD7                                 //connect enable pin of LCD to port D7
#define RS RD6                                 //connect  Register select pin of LCD
                                                 to port D6

void LCD_Delay()                               //delay routine
{
 __delay_ms(1);
}

void LCD_Cmd(unsigned char cmd)                //this function is to write command to
                                                 the LCD
{
        PORTB=cmd;
              RS=0;                            //Set RS pin to low in order to send a
                                                 command to the LCD
            EN=1;                              //set EN pin to high in order to send
                                                 high pulse
        LCD_Delay();                           //give a small delay
              EN=0;                            //set EN pin to low in order to make
                                                 pulse low
      LCD_Delay();                             //give a small delay
```

```
}

void LCD_Init()                                    //Initializing LCD
{
unsigned char cmd[5]={0X38,0X06,0X0F,0X01,0X80},Count;
        //0x38 represents 5x7 matrix ,0x06 represent entry mode,0x0f represent display on cursor
                                    blinking,0x01 represents clearing the LCD,0x80 represents 1st row
        for(Count=0;Count<5;Count++)
        LCD_Cmd(cmd[Count]);
}

void LCD_SendDataByte(unsigned char data)          //this function is to write a byte on LCD
{
        PORTB=data;
           RS=1;                                   //make RS pin high inorder to send a
                                                                                    data
                EN=1;                              //set enable pin to high in order to
                                                   sendhigh to low pulse
        LCD_Delay();                               //provide a small delay
        EN=0;
       LCD_Delay();
}
     void LCD_Display( char *addr)                 //this function is to display a string on
                                                                                    LCD
{
        while(*addr)
        {
                LCD_SendDataByte(*addr);
                addr++;
        }
```

```
}


void main()
{
        TRISB=0x00;                                  //make the registerB as ouput
        TRISD=0x00;                                 //make the registerD as ouput
        LCD_Init();                                  //Initialize the LCD
        __delay_ms(1000);
        while(1)                                     //infinite loop

        {

              LCD_Cmd(0X01);                         //clearing the LCD
             LCD_Cmd(0X84);                          //1st row 4th position
              LCD_Display("123");                    //display 123 on LCD

           LCD_Cmd(0xc0);                            //2nd row
              LCD_Display("    RDL");                //display RDL on LCD
              __delay_ms(1000);                      //delay by 1s

             LCD_Cmd(0x01);                          //clear the LCD
           LCD_Cmd(0x80);                            //1st row
             LCD_Display("   LCD");                  //display LCD


           LCD_Cmd(0xc0);                            //2nd row
           LCD_Display("  Display");                 //display on LCD
           __delay_ms(1000);                         //delay by 1s
        }
```

}



Lab3.Interfacing ADC to display analog to digital conversion values on LCD.

I/O connection:

PORT B0 to B7→DO to D7 of LCD

ENABLE→D7

R/W→GROUND

R/S→D6

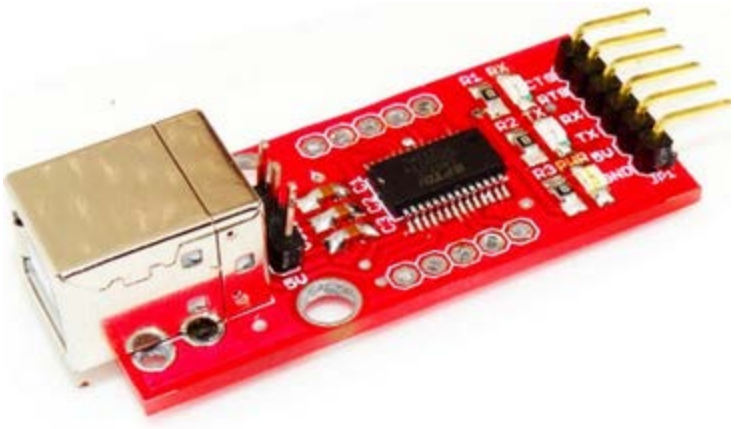
A0→PORTC0

```
#include <htc.h>
#include<string.h>
#define _XTAL_FREQ 20000000                    //crystal frequency of 20MHZ
#define EN RD7                                 //connect enable pin of LCD to port D7
#define RS RD6                                 //connect  Register select pin of LCD to port
                                                D6

/*LCD code */

void LCD_Delay()                               //delay routine
{
 __delay_ms(1);
}



  void LCD_Cmd(unsigned char cmd)              //this function is to write command to the LCD
{
```

```
        PORTB=cmd;
            RS=0;                                   //Set RS pin to low in order to send a
                                                      command to the LCD
            EN=1;                                   //set EN pin to high in order to send
                                                      high pulse
        LCD_Delay();                                //give a small delay
            EN=0;                                   //set EN pin to low in order to make
                                                      pulse low
      LCD_Delay();                                  //give a small delay
}

void LCD_Init()                                     //Initializing LCD
{
                                unsigned char cmd[5]={0X38,0X06,0X0F,0X01,0X80},Count;
        //0x38 represents 5x7 matrix ,0x06 represent entry mode,0x0f represent display on cursor
                                blinking,0x01 represents clearing the LCD,0x80 represents 1st row
        for(Count=0;Count<5;Count++)
        LCD_Cmd(cmd[Count]);
}

          void LCD_SendDataByte(unsigned char data) //this function is to write a byte on LCD
{
        PORTB=data;
            RS=1;                                   //make RS pin high inorder to send a data
             EN=1;                                  //set enable pin to high in order to send
                                                      high to low pulse
        LCD_Delay();                                //provide a small delay
        EN=0;
       LCD_Delay();
}
void LCD_Display( char *addr)                       //this function is to display a string on LCD
{
        while(*addr)
        {
                LCD_SendDataByte(*addr);
                addr++;
        }
}



void ADC_Init()
{
  ADCON0 = 0x41;                                    //set A/D control register0 to 0x41
  ADCON1 = 0xC0;                                    //set A/D control register1 0xc0
```

```
}

unsigned int ADC_Read(unsigned char channel)
{
   if(channel > 7)
     return 0;

   ADCON0 &= 0xC5;
   ADCON0 |= channel<<3;
   __delay_ms(2);
   GO_nDONE = 1;
   while(GO_nDONE);
     return ((ADRESH<<8)+ADRESL);  //left shift the higherorder bits and add   the  lower order
                                                                                              bits
}

            void display(unsigned int number)          //this function is for (0-1024)A/D conversion
{
          unsigned char digit1,digit2,digit3,digit4,digit[4];
          unsigned char x;
          unsigned char temp;

          digit1 = number / 1000u ;                        // extract thousands digit

          digit2 = (number / 100u) % 10u;                 // extract hundreds digit

          digit3 = (number / 10u) % 10u;                  // extract tens digit

          digit4 = number % 10u;                          // extract ones digit
          digit[3]=digit4;
          digit[2]=digit3;
          digit[1]=digit2;
          digit[0]=digit1;

          for(x=0;x<4;x++)                               //loop for upto 4 digits
          {
                    temp=digit[x]|0x30;                  //convert to ACII

                    LCD_SendDataByte(temp);          //display the value on LCD
          }
}


void main()
{
```

```
  unsigned int value;
  unsigned int a;
  TRISB = 0x00;                          //Set registerB as output
  TRISC = 0x00;                          //Set registerC as output
  TRISD=0x00;                            //set registerD as output
  LCD_Init();                            //initialize the LCD
  __delay_ms(1000);                      //provide delay for 1s
  ADC_Init();                            //ADC initialisation

   do
   {

     a = ADC_Read(0);                    //read port (A0)
     __delay_ms(2000);                   //provide delay for 2s
     LCD_Cmd(0x80);                      //1st row
     LCD_ display(a);                    //display the value on LCD
     __delay_ms(1000);                   //provide delay
    } while(1);

}
```



Lab 4.Interfacing UART toTransmit and Receive the message

I/O connection:

TX and RX→UART RX and TX

Ground of Ic →UART ground

```
#include<htc.h>
#define _XTAL_FREQ 20000000                    //crystal frequency of 20MHZ
#include "uart.h"                              //header file
#include "string.h"                            //header file

char val;

void main()
{
   __delay_ms(1000);                           //provide delay for 1s
    UART_Init(9600);                           //calling initialization function with 9600 baud
                                                                                    rate
   __delay_ms(1000);                           //provide delay for 1s
UART_Write_Text("RDL");                        //Display RDL on hyper terminal

    do
  {

    if(UART_Data_Ready())                      //check whether it is ready to receive a data
{
     recieve = UART_Read();                    //read a data and store in variable
           UART_Write(recieve);                //display on terminal
         UART_Write(10);                       //enter
      UART_Write(13);                          //carriage return
```

```
    __delay_ms(1000);                          //provide delay of 1s
}
  }while(1);
}
char UART_Init(const long int baudrate)
{
        unsigned int x;
        x = (_XTAL_FREQ - baudrate*64)/(baudrate*64);
        if(x>255)
        {
                x = (_XTAL_FREQ - baudrate*16)/(baudrate*16);
                 BRGH = 1;                                  //High Baud Rate Select bit set to high
        }
        if(x<256)
        {
            SPBRG = x;                                      //Writing SPBRG register
            SYNC = 0;                                       //Selecting Asynchronous Mode
            SPEN = 1;                                       //enables serial port
            TRISC7 = 1;
             TRISC6 = 1;
              CREN = 1;                                     //enables continuous reception
              TXEN = 1;                                     //enables continuous transmission
          return 1;
        }
          return 0;
}

char UART_TX_Empty()
{
  return TRMT;                                              //Returns Transmit Shift Status bit
```

```
}

char UART_Data_Ready()
{
   return RCIF;                                    //Flag bit
}

char UART_Read()                                   //this function is used to read a byte
{
  while(!RCIF);                                    //Waits for Reception to complete
  return RCREG;                                    //Returns the 8 bit data
}
    void UART_Read_Text(char *Output, unsigned int length)//this function is used to read a text
{
          int i;
          for(int i=0;i<length;i++)
          Output[i] = UART_Read();
}

void UART_Write(char data)                         //this function is used to write a byte
{
  while(!TRMT);
  TXREG = data;                                    //transmit register
}
void UART_Write_Text(char *text)                   //this function is used to write a string
{
  int i;
  for(i=0;text[i]!='\0';i++)
  UART_Write(text[i]);
}
```

Lab 5.Interfacing PWM to vary the brightness of LED

I/O connection:

PORT C1→LED1.

PORTC2→LED2

```
#include<htc.h>
#define XTAL      20000                          //20Mhz=20000Khz
#define PWM_Freq  1                              //1Khz PWM frequency
#define TMR2_PRE  16                             //Timer2 Prescale
#define PR2_Val     ((char)((XTAL/(4*TMR2_PRE*PWM_Freq))-1))
                                                 //Calculation for Period register PR2 (2Khz)
#define Duty_Cyc  PR2_Val*2
 unsigned int i;
void PWM_init(void);                             // This function is to initialize the PWM
void PWM_change(unsigned int);                   //This function is to change theDuty cycle
                                                 routine
void DelayMs(unsigned int);                      //this function is to provide a delay
void main(void)
{
  PWM_init();
  while(1)
  {
    i=0;
    PWM_change(i);
    DelayMs(10);
    while(i<PR2_Val)
    {
      i=i+1;
      PWM_change(i);
```

```
      DelayMs(200);
    }
  }
}

void PWM_init(void)
{
  TRISC2=0;                          //PWM channel 1 and 2 configured as output
  TRISC1=0;
  PORTC = 0x00;
  CCP1CON=0x0c;                      //CCP1 and CCP2 are configured for PWM
  CCP2CON=0x0c;
  PR2=PR2_Val;                       //Move the PR2 value

  T2CON=0x03;                        //Timer2 Prescale is 16
  TMR2=0x00;
  TMR2ON=1;                          //Turn ON timer2
}

void PWM_change(unsigned int DTY)   //Duty cycle change routine
{
  CCPR1L=DTY;                        //Value is between 0 to 255
  CCPR2L=DTY;
}

void DelayMs(unsigned int Ms)        //Delay Routine
{
  int delay_cnst;

  while(Ms>0)
```

```
        {
          Ms--;
 for(delay_cnst = 0;delay_cnst <220;delay_cnst++);      //delay constant for 1Ms @20Mhz
          }
}
```

Lab 6. Interfacing KEYPAD to display value on LCD when a key is pressed.

I/O connection:

PORT D0 tO D7→DO to D7 of LCD

ENABLE→C0

R/W→GROUND

R/S→C1

R1,R2,R3,R4→PORT B0 to B3

C1,C2,C3,C4→PORT4 to B7

```
#include <htc.h>
#include <stdio.h>                                  // Define I/O functions
#define XTAL      20000000
#define BAUD_RATE  9.6                              //9600 Baudrate
```

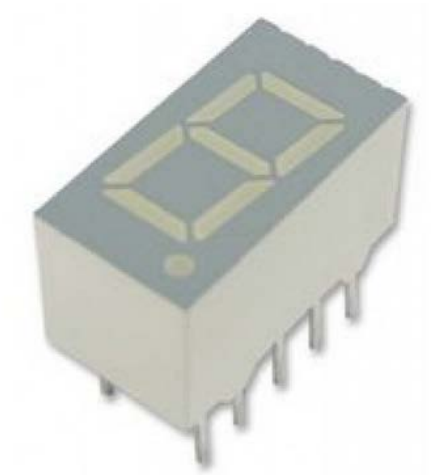
```
#define BAUD_VAL   (char)(XTAL/ (16 * BAUD_RATE )) - 1;
                                                    //Calculation For9600 Baudrate @20Mhz
 #define EN RC0
#define RS RC1
void ScanCol(void);                                 //Column Scan Function
void ScanRow(void);                                 //Row Scan Function
void DelayMs(unsigned int);
void LCD_Cmd(unsigned char);
void LCD_Init(void);
void LCD_Display( char *addr);
void LCD_SendDataByte(unsigned char);


unsigned char KeyArray[4][4]= {   '1','2','3','4',
                                  '5','6','7','8',
                                  '9','A','B','C',
                                 'D','E','F','0'};
                                                    //Keypad value Initialization Function

unsigned char Count[4][4]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int Col=0,Row=0,count=0,i,j;

void main()
{

   TRISD=0x00;                                      //set registerD as output
        TRISC=0x00;                                 //set register C as output
        LCD_Init();                                 //initialize LCD

      DelayMs(1000);
```

```
        nRBPU=0;                                    //Enable PORTB Pullup values

  while(1)
  {
    TRISB=0X0f;                                   // Enable the 4 LSB as I/P & 4 MSB as
                                                     O/P
     PORTB=0X00;
    while(PORTB==0x0f);                           // Get the ROW value
    ScanRow();
     TRISB=0Xf0;                                 // Enable the 4 LSB as O/P & 4 MSB as
                                                     I/P
    PORTB=0X00;
    while(PORTB==0xf0);                           // Get the Column value
    ScanCol();

    DelayMs(1000);                                //provide a delay of 1s
    Count[Row][Col]++;                            // Count the Pressed key
   LCD_Cmd(0X01);                                 //clear the LCD
    LCD_Cmd(0X80);                                //1st row of the LCD
   LCD_SendDataByte(KeyArray[Row][Col]);          //send keypad value and display on LCD
 DelayMs(1000);                                   //provide delay of 1s
  }

}

void ScanRow()                                    // Row Scan Function
{

  switch(PORTB)
  {
```

```
    case 0x07:
    Row=3;                                  // 4th Row
    break;
    case 0x0b:
    Row=2;                                  // 3rd Row
    break;
    case 0x0d:
    Row=1;                                  // 2nd Row
    break;
    case 0x0e:
    Row=0;                                  // 1st Row
    break;
  }
}

void ScanCol()                              // Column Scan Function
{

  switch(PORTB)
  {
    case 0x70:
      Col=3;                                // 4th Column
    break;
    case 0xb0:
      Col=2;                                // 3rd Column
    break;
    case 0xd0:
      Col=1;                                // 2nd Column
    break;
    case 0xe0:
```

```
        Col=0;                                  // 1st Column
      break;
    }
}
/*LCD CODE*/

void LCD_Delay()                                //delay routine
{
 __delay_ms(1);
}



void LCD_Cmd(unsigned char cmd)                 //this function is to write command to the LCD
{


          PORTB=cmd;
          RS=0;                                 //Set RS pin to low in order to send a
                                                command to the LCD
          EN=1;                                 //set EN pin to high in order to send high pulse
          LCD_Delay();                          //give a small delay
          EN=0;                                 //set EN pin to low in order to make pulse low
        LCD_Delay();                            //give a small delay
}

void LCD_Init()                                 //Initializing LCD
{
                                    unsigned char cmd[5]={0X38,0X06,0X0F,0X01,0X80},Count;
```

```
        //0x38 represents 5x7 matrix ,0x06 represent entry mode,0x0f represent display on cursor
                                blinking,0x01 represents clearing the LCD,0x80 represents 1st row
        for(Count=0;Count<5;Count++)
        LCD_Cmd(cmd[Count]);
}

          void LCD_SendDataByte(unsigned char data) //this function is to write a byte on LCD
{
        PORTB=data;
        RS=1;                                   //make RS pin high inorder to send a data
        EN=1;                                   //set enable pin to high in order to send high
                                                  to low pulse
        LCD_Delay();                            //provide a small delay
        EN=0;
       LCD_Delay();
}
  void LCD_Display( char *addr)                 //this function is to display a string on LCD
{
        while(*addr)
        {
                LCD_SendDataByte(*addr);
                addr++;
        }
}
```



Lab7. Interfacing 7segment

I/O connection:

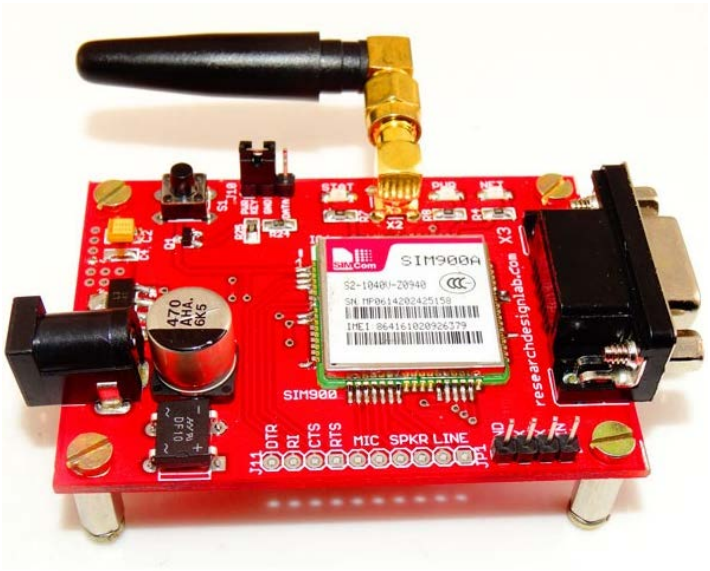
A,B,C,D,E,F,G,DP→B0 to B7

DIG1,DIG2,DIG3,DIG4 →A0 to A3

```
#include<htc.h>
 #define CNTRL_PORT PORTA
#define DATA_PORT  PORTB

void hex2dec(unsigned char);                          //function to convert hex value to decimal
  void send_seg(unsigned char,unsigned char,unsigned char,unsigned char); //Function to display
                                                                          count on 7seg
void DelayMs(unsigned int);                           //function to provide delay
unsigned char x;
unsigned char thou=0,hun=0,ten=0,single=0;
unsignedcharCA[10]  = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
unsignedchar CC[10]  =  {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
unsigned char CA_CNTRL[4]   =   {0x07,0x0b,0x0d,0x0e};
unsigned char CC_CNTRL[4]   =   {0x08,0x04,0x02,0x01};
unsigned char n=1;
void main()
{
  unsigned char number;
  nRBPU =0;
  TRISB=0x00;                                          //PORTB configured as O/P
  ADCON1=0x07;                                        //Configure PORTA & PORTE as Digital
                                                       port
```

```
  TRISA=0x00;                                   //PORTA Configured as O/P
  while(1)
  {
    if(x == 200)
    {
            x=0;
      single++;                                  //Increment up to 9 in unit place
      if(single>9)
            {
        single=0;
              ten++;                             //Increment up to 9 in Tenth place
        if(ten>9)
              {
          ten=0;
                hun++;                           //Increment up to 9 in Hundredth place
                 if(hun>9)
          {
                  hun=0;
                  thou++;                        //Increment up to 9 in Thousandth place
            if(thou>9)
                    thou=0;
          }
        }
      }
    }
    x++;
    send_seg(thou,hun,ten,single);
  }
}
void send_seg(unsigned char thou,unsigned char hun,unsigned char ten,unsigned char single)
{
  if(n==1)
  {
 CNTRL_PORT=CA_CNTRL[0];                 //Eanble Unit place 7-Segment
    DATA_PORT=CA[single];                 //Display Unit Place Number
    n=2;
    DelayMs(5);
  }

  else if(n==2)
     {
CNTRL_PORT=CA_CNTRL[1];                 //Eanble Tenth place 7-Segment
DATA_PORT=CA[ten];                      //Display Tenth Place Number
       n=3;
          DelayMs(5);
```

```
      }
      else if(n==3)
         {
           CNTRL_PORT=CA_CNTRL[2];       //Enable Hundredth place 7-Segment
           DATA_PORT=CA[hun];            //Display Hundredth Place Number
           n=4;
           DelayMs(5);
         }
         else if(n==4)
         {
          CNTRL_PORT=CA_CNTRL[3];      //Eanble Thousandth place 7-Segment
          DATA_PORT=CA[thou];          //Display Thousandth Place Number
          n=1;
          DelayMs(5);
         }
}
void DelayMs(unsigned int Ms)
{
  int delay_cnst;
  while(Ms>0)
  {
 Ms--;
    for(delay_cnst = 0;delay_cnst <220;delay_cnst++);
  }
}
```



Lab 8. Interfacing GSM modem to send and receive the message

I/Oconnection:

Vin of GSM→12v

Ground of GSM→Ground

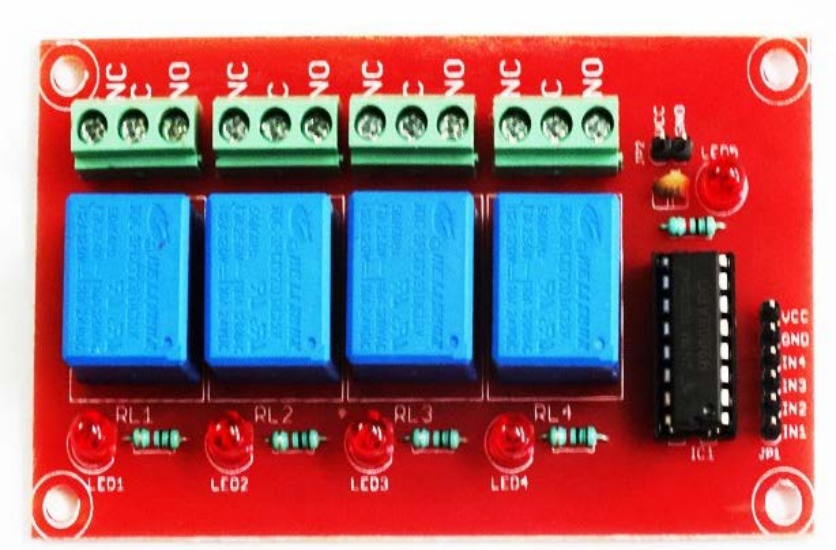
D0,D1 of GSM→TX,RX

```
#define <htc.h>
#define _XTAL_FREQ 20000000                    //crystal frequency of 20MHZ
#include "uart.h"                              //header file
#include "string.h"                            //header file

char UART_Init(const long int baudrate)
{
        unsigned int x;
        x = (_XTAL_FREQ - baudrate*64)/(baudrate*64);
        if(x>255)
        {
            x = (_XTAL_FREQ - baudrate*16)/(baudrate*16);
```

```
            BRGH = 1;                    //High Baud Rate Select bit set to high
        }
        if(x<256)
        {
            SPBRG = x;                   //Writing SPBRG register
            SYNC = 0;                    //Selecting Asynchronous Mode
            SPEN = 1;                    //enables serial port
            TRISC7 = 1;
            TRISC6 = 1;
            CREN = 1;                    //enables continuous reception
            TXEN = 1;                    //enables continuous transmission
             return 1;
        }
             return 0;
}

char UART_TX_Empty()
{
  return TRMT;                           //Returns Transmit Shift Status bit
}

char UART_Data_Ready()
{
   return RCIF;                          //Flag bit
}

char UART_Read()                         //this function is used to read a byte
{
  while(!RCIF);                          //Waits for Reception to complete
  return RCREG;                          //Returns the 8 bit data
```

```
}
void UART_Read_Text(char *Output, unsigned int length)
                                        //this function is used to read a text
{
        int i;
        for(int i=0;i<length;i++)
        Output[i] = UART_Read();
}

void UART_Write(char data)              //this function is used to write a byte
{
 while(!TRMT);
 TXREG = data;                          //transmit register
}

void UART_Write_Text(char *text)        //this function is used to write a string
{
 int i;
 for(i=0;text[i]!='\0';i++)
 UART_Write(text[i]);
}
void main()
{
UART_Init(9600);                        //initialize the UART function
__delay_ms(1000);                       //provide the delay of 1s

while(1)                                //infinite loop
{
__delay_ms(1000);                               //provide a delay of 1s
UART_Write_Text("AT");                          //attention command
```

```
UART_Write(13);                                  //enter
UART_Write(10);                                  //carriage return
__delay_ms(1000);                                //provide delay of 1s
UART_Write_Text("AT+CMGF=1");                    //initialize the modem

UART_Write(13);                                  //enter
UART_Write(10);                                  //carriage return
__delay_ms(1000);                                //provide delay of 1s
UART_Write_Text("AT+CMGS=\"1234567890\"");       //send a message

UART_Write(13);                                  //enter
UART_Write(10);                                  //carriage return
__delay_ms(1000);                                //provide delay of 1s
UART_Write_Text("GSM");                          //display on hyper terminal
UART_Write(13);                                  //enter
UART_Write(10);                                  //carriage return
__delay_ms(1000);                                //provide delay of 1s
UART_Write(26);                                  //Ctr +Z
}
}
```



Lab 9. Interfacing RELAY to turn the relays ON and OFF.

I/O connection:

B0,B1,B2,B3→ to relay shield.

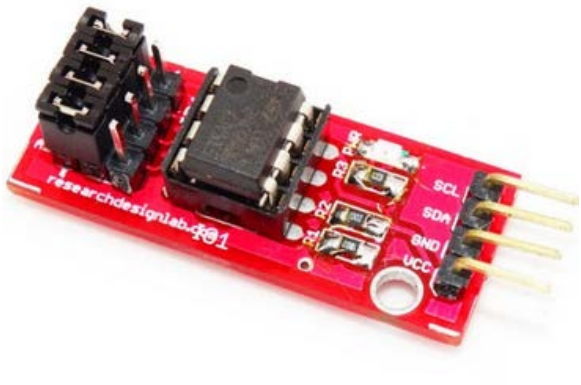
```
#define _XTAL_FREQ 20000000          //crystal frequency of 20MHZ
#include "uart.h"                    //header file
#include "string.h"                  //header file
#define relay1 RB1
#define relay2 RB2
#define relay3 RB3
#define relay4 RB4

char UART_Init(const long int baudrate)
{
        unsigned int x;
        x = (_XTAL_FREQ - baudrate*64)/(baudrate*64);
```

```
        if(x>255)
        {
                x = (_XTAL_FREQ - baudrate*16)/(baudrate*16);
                BRGH = 1;                       //High Baud Rate Select bit set to high
        }
        if(x<256)
        {
            SPBRG = x;                          //Writing SPBRG register
            SYNC = 0;                           //Selecting Asynchronous Mode
            SPEN = 1;                           //enables serial port
            TRISC7 = 1;
            TRISC6 = 1;
            CREN = 1;                           //enables continuous reception
           TXEN = 1;                            //enables continuous transmission
            return 1;
        }
         return 0;
}

char UART_TX_Empty()
{
 return TRMT;                                   //Returns Transmit Shift Status bit
}

char UART_Data_Ready()
{
  return RCIF;                                  //Flag bit
}

char UART_Read()                                //this function is used to read a byte
```

```
{
  while(!RCIF);                                    //Waits for Reception to complete
  return RCREG;                                    //Returns the 8 bit data
}
void UART_Read_Text(char *Output, unsigned int length)//this function is used to read a text
{
        int i;
        for(int i=0;i<length;i++)
        Output[i] = UART_Read();
}

void UART_Write(char data)                         //this function is used to write a byte
{
  while(!TRMT);
  TXREG = data;                                    //transmit register
}

void UART_Write_Text(char *text)          //this function is used to write a string
{
  int i;
  for(i=0;text[i]!='\0';i++)
  UART_Write(text[i]);
}


void main()
{
unsigned char ReceivChar;
TRISB=0X00;                                 //make register as the output
PORTB=0X00;                                //make the PORTB as the output port
```

```
UART_Init(9600);                    //inititalise the UART
DelayMs(1000);                      //provide delay of 1s

while(1)
{
if(UART_Data_Ready())               //check if the data is ready
 {
     ReceivChar = UART_Read();      //store the data in a variable
          UART_Write(ReceivChar);   //display on hyperterminal
    __delay_ms(1000);               //provide delay of 1s

        if(ReceivChar=='1')         //check if the  received char is 1if 1
        {
         ReceivChar = UART_Read();  //store the data in a variable
          UART_Write(ReceivChar);   //display on hyperterminal


         if(ReceivChar=='N')        //if received character is N
        relay1=1;                   //turn ON the 1st relay
        else if(ReceivChar=='F')    //if received character is F
        relay1=0;                   //turn OFF the 1st relay
        }

        else if(ReceivChar=='2')    //check if the  received char is 2if 2

        {
ReceivChar = UART_Read();           //store the data in a variable
UART_Write(ReceivChar);             //display on hyperterminal
        if(ReceivChar=='N')         //if received character is N
        relay2=1;                   //turn ON the 2nd relay
```

```
        else if(ReceivChar=='F')          //if received character is F
        relay2=0;                         //turn OFF the 2nd relay
        }

        else if(ReceivChar=='3')          //check if the  received char is 3if 3
        {
        ReceivChar = UART_Read();         //store the data in a variable
          UART_Write(ReceivChar);         //display on hyperterminal
        if(ReceivChar=='N')               //if received character is N
        relay3=1;                         //turn ON the 3rd relay
        else if(ReceivChar=='F')          //if received character is N
        relay3=0;                         //turn OFF the 3rd relay
        }
else if(ReceivChar=='4')                  //check if the  received char is 4if 4
        {
        ReceivChar = UART_Read();         //store the data in a variable
          UART_Write(ReceivChar);         //display on hyperterminal
        if(ReceivChar=='N')                 //if received character is N
        relay4=1;                           //turn ON the 4th relay
        else if(ReceivChar=='F')            //if received character is N
        relay4=0;                           //turn OFF the 4th relay
        }
```



Lab 10. Display a message using I2c Protocol

I/O connection:

SCL of EEPROM→C3

SDA of EEPROM→C4

```
#include<htc.h>
#include"string.h"
#include<stdio.h>
#define _XTAL_FREQ 20000000
#define I2C_FREQ    100                 // 100khz at 4Mhz
#define FOSC        20000               // 20Mhz==>20000Khz


void WaitMSSP(void);                    //function to wait for a operation to complete

void i2c_init(void);                    //Initialize the UART

void I2C_Start(void);                   //function to send a start bit
```

```
void I2C_Stop(void);                    //function to send a stop bit

char I2C_Read_Data(void);               //function to read a data

char I2C_Write_Data(unsigned char);     //function to write the data

void I2C_Reset(void);                   //function to reset the bit

void DelayMs(unsigned int);             //function to provide a delay

char UART_Init(const long int);         // function to initialize UART

void UART_Write_Text(char *);           //function to write the string

void UART_Write(char );                 //function to write the byte

char UART_Data_Ready(void);             //function to check if data ready

char UART_Read(void);                   //function to read the data


void main()
{
char a;
UART_Init(9600);                        //initialize the UART
 DelayMs(1000);                         //Provide a delay of 1s
i2c_init();                             //initialize the I2C
 DelayMs(1000);                         //Provide a delay of 1s

while(1)
```

```
{
I2C_Start();                    //start bit is set in this function
DelayMs(100);                   //Provide a delay
I2C_Write_Data(0xa0);           //write the data on to the location 0xa0(device address)
DelayMs(100);                   //Provide a delay
I2C_Write_Data(0x20);           //write the data on to location 0x20
DelayMs(100);                   //Provide a delay
I2C_Write_Data('a');            //send character ‘a’
DelayMs(100);                   //Provide a delay
I2C_Stop();                     //stop bit is set in this function
DelayMs(100);                   //Provide a delay
I2C_Start();                    //start bit is set in this function
DelayMs(100);                   //Provide a delay
I2C_Write_Data(0xa0);           //write the data on to the location 0xa0(device address)
DelayMs(100);                   //Provide a delay
I2C_Write_Data(0x20);           //write the data on to location 0x20
DelayMs(100);                   //Provide a delay
I2C_Reset();                    //this function is used to reset
DelayMs(100);                   //Provide a delay
I2C_Write_Data(0xa1);           //write the data on to the location 0xa0(device address)
DelayMs(100);                   //Provide a delay
a=I2C_Read_Data();              //this function reads the data stored in EEPROM
UART_Write(a);                  //display the character on hyper terminal
DelayMs(100);                   //Provide a delay
I2C_Stop();                     //stop bit is set in this function
DelayMs(100);                   //Provide a delay
}
}

char I2C_Write_Data(unsigned char data)
```

```
                                    //This function is used to write the data onto EEPROM

{

  //WaitMSSP();                     // wait for the operation to be finished

  SSPBUF=data;                      //Send Slave address write command

  WaitMSSP();                       //wait for operation to complete
}

void I2C_Start()                    //this function is used to set start bit
{
 SEN=1;                             //start bit is set
 WaitMSSP();                        //wait for operation to complete

}


void I2C_Stop()                     //this function is used to set start bit
{
 PEN=1;                             //stop bit is set
WaitMSSP();                         //wait for operation to complete
}


void I2C_Reset()                    //this function is used to reset start bit
{
 RSEN=1;                           // Send re-start bit
```

```
  WaitMSSP();                    //wait for operation to complete

}
char I2C_Read_Data()             //this function is used to read data from EEPROM

{

   RCEN=1;                       // Enable receive

     WaitMSSP();                 //wait for operation to complete

     ACKDT=1;                    // Acknowledge data 1: NACK, 0: ACK

     ACKEN=1;                    // Enable ACK to send

      WaitMSSP();                //wait for operation to complete


     return SSPBUF;              // Send the received data to PC

   DelayMs(30);

}

void WaitMSSP()                  // function for wait for operation to complete

{

  while(!SSPIF);                 // while SSPIF=0 stay here else exit the loop
```

```
  SSPIF=0;                    // operation completed clear the flag

}

void i2c_init()               //function to initialize I2C
{

  TRISC3=1;                   // Set up I2C lines by setting as input

  TRISC4=1;

  SSPCON=0x28;                // SSP port, Master mode, clock = FOSC / (4 * (SSPADD+1))

 SSPADD=(FOSC / (4 * I2C_FREQ)) - 1; //clock 100khz


  SSPSTAT=80;                 // Slew rate control disabled

}



void DelayMs(unsigned int Ms)   //function to provide a delay

{

  int delay_cnst;


  while(Ms>0)
```

```
  {

    Ms--;

    for(delay_cnst = 0;delay_cnst <220;delay_cnst++);

  }

}


char UART_Init(const long int baudrate)
{
        unsigned int x;
        x = (_XTAL_FREQ - baudrate*64)/(baudrate*64);
        if(x>255)
        {
                x = (_XTAL_FREQ - baudrate*16)/(baudrate*16);
                BRGH = 1;                       //High Baud Rate Select bit set to high
        }
        if(x<256)
        {
            SPBRG = x;                          //Writing SPBRG register
            SYNC = 0;                           //Selecting Asynchronous Mode
            SPEN = 1;                           //enables serial port
            TRISC7 = 1;
            TRISC6 = 1;
            CREN = 1;                           //enables continuous reception
            TXEN = 1;                           //enables continuous transmission
```

```
            return 1;
        }
         return 0;
}

char UART_TX_Empty()
{
 return TRMT;                                    //Returns Transmit Shift Status bit
}

char UART_Data_Ready()
{
  return RCIF;                                    //Flag bit
}

char UART_Read()                                  //this function is used to read a byte
{
 while(!RCIF);                                   //Waits for Reception to complete
 return RCREG;                                   //Returns the 8 bit data
}
void UART_Read_Text(char *Output, unsigned int length)//this function is used to read a text
{
        int i;
        for(int i=0;i<length;i++)
        Output[i] = UART_Read();
}

void UART_Write(char data)                        //this function is used to write a byte
{
 while(!TRMT);
```

```
  TXREG = data;                                  //transmit register
}

void UART_Write_Text(char *text)                 //this function is used to write a string
{
  int i;
  for(i=0;text[i]!='\0';i++)
  UART_Write(text[i]);
}
```

Lab 11. Working with RTC and controller

Pinconnection:

SCL of RTC→C3

SDA of RTC→C4

```
#include<htc.h>

#define _XTAL_FREQ 20000000

#include "string.h"

#define LC01CTRLIN    0xd0

#define LC01CTRLOUT 0xd1

#define I2C_FREG   100

#define FOSC       10000

unsigned char sec,min,hour,day,date,month,year;

unsigned char data[7]={0x45,0x59,0x71,0x01,0x13,0x10,0x13};

int i;

void DS1307Write(unsigned char,unsigned char);

void WaitMSSP();

unsigned char DS1307Read(unsigned char);

void i2c_init(void);

char UART_Init(const long int);
```

```
void ds1307_init(void);

void DelayMs(unsigned int);

void main()

{

  int count=0;

  DelayMs(20);                          //provide a delay

   ds1307_init();                       //initialize ds1307

UART_Init(9600);                        //initialize the UART

  for(i=0;i<7;i++)

  DS1307Write(i,data[i]);

  DelayMs(20);                                  //provide a delay

while(1)

  {

    sec=DS1307Read(0);                          // Read second

    min=DS1307Read(1);                          // Read minute
```

```
    hour=DS1307Read(2);                                    // Read hour

    day=DS1307Read(3);                                     // Read day

    date=DS1307Read(4);                                    // Read date

     month=DS1307Read(5);                                  // Read month

    year=DS1307Read(6);                                    // Read year

    printf("Time: %x : %x : %x  ",(hour&0x1f),min,sec);    //Display the Hours, Minutes,
                                                           Seconds(hours is taken from 5 LSB bits  )

     printf("Date: %x / %x / %x    \r",date,month,year);   //Display the Date, Month, Year

    DelayMs(150);                                          //provide a delay

  }

}

 void DS1307Write(unsigned char addr, unsigned char data)

{

  SEN=1;                              //Initiate Start condition on SDA & SCL pins

  WaitMSSP();

  SSPBUF=LC01CTRLIN;            // Slave address + Write command
```

```
  WaitMSSP();
  SSPBUF=addr;                    // Write the location
  WaitMSSP();
  SSPBUF=data;                    // Write the Data
  WaitMSSP();
  PEN=1;                          // Enable the Stop bit
  WaitMSSP();
}


unsigned char DS1307Read(unsigned char addr)
{
  unsigned char x;
  RSEN=1;                         // Enable the repeated Start Condition
  WaitMSSP ();
```

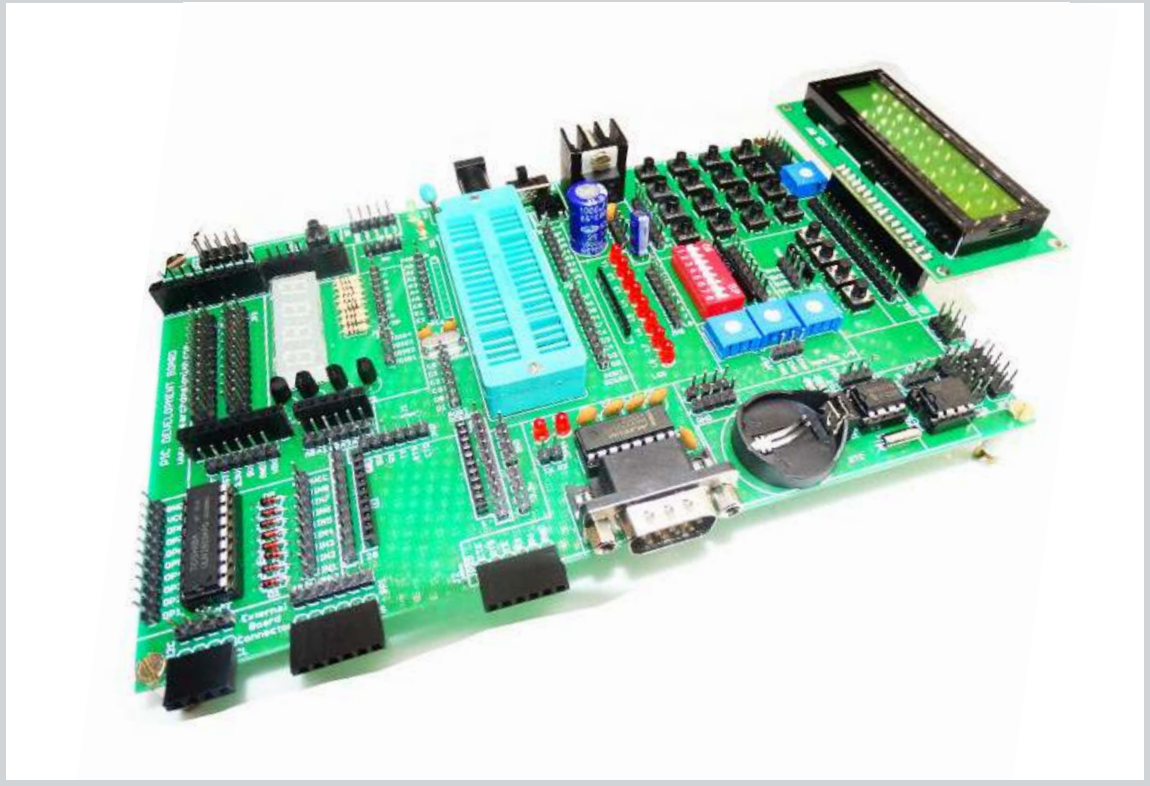
```
SSPBUF=LC01CTRLIN;          // Slave address + Write command

WaitMSSP ();

SSPBUF=addr;                //Write the location (memory address of Hour, minute, etc...)

WaitMSSP ();

RSEN=1;                     // Enable the repeated Start Condition

WaitMSSP ();

SSPBUF=LC01CTRLOUT;         // Slave address + Read command

WaitMSSP ();

RCEN=1;                     // Enable to receive data

WaitMSSP ();

ACKDT=1;                    // Acknowledge the operation (Send NACK)

ACKEN=1;                    // Acknowledge sequence on SDA & SCL pins

PEN=1;                      // Enable the Stop bit

WaitMSSP ();

x=SSPBUF;                   // Store the Receive value in a variable
```

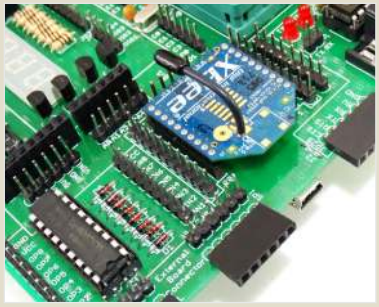
```
  return (x);

}

 void WaitMSSP()

{

  while(!SSPIF);                  // SSPIF is zero while TXion is progress

  SSPIF=0;

}

void ds1307_init()

{

  TRISC3=1;                       // RC3,RC4 set to I2C Mode(Input)

  TRISC4=1;

  SSPCON=0x28;                              // Enable the SDA,SCL & I2C Master Mode

  SSPADD=(FOSC / (4 * I2C_FREG)) – 1;// SSP baud rate 100Khz

  SSPSTAT=0x80;                             // Disable slew Rate control

  PORTC=0x18;
```

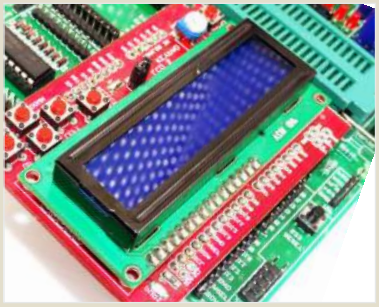
```
  DS1307Write(0,0x00);
}

void putch(unsigned char byte)                    //Required for printf statement
{

  while(!TXIF);                                   // Wait for the Transmit Buffer to be empty
  TXREG = byte;                                   // Transmit the Data
}

void DelayMs(unsigned int Ms)                     //Function to provide a delay
{
  int delay_cnst;
  while(Ms>0)
  {
```

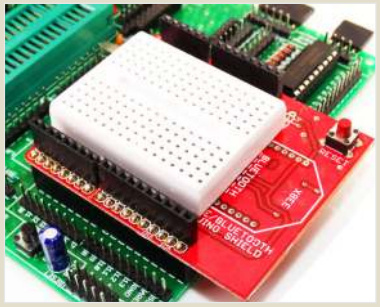
```
    Ms--;

    for(delay_cnst = 0;delay_cnst <220;delay_cnst++);

  }

}

char UART_Init(const long int baudrate)        //function to initialize the UART
{
        unsigned int x;
        x = (_XTAL_FREQ - baudrate*64)/(baudrate*64);
        if(x>255)
        {
                x = (_XTAL_FREQ - baudrate*16)/(baudrate*16);
                BRGH = 1;                       //High Baud Rate Select bit set to high
        }
        if(x<256)
        {
            SPBRG = x;                          //Writing SPBRG register
             SYNC = 0;                                  //Selecting Asynchronous Mode
             SPEN = 1;                                  //enables serial port
             TRISC7 = 1;
              TRISC6 = 1;
              CREN = 1;                                 //enables continuous reception
              TXEN = 1;                                 //enables continuous transmission
              return 1;
        }
               return 0;
}
```

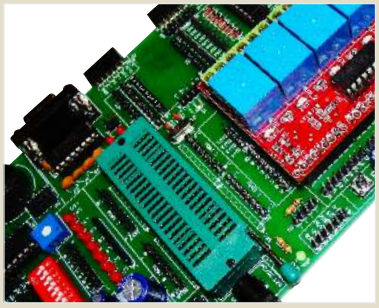
Pic Development Board

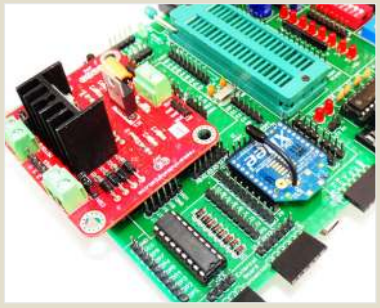












Research Design Lab

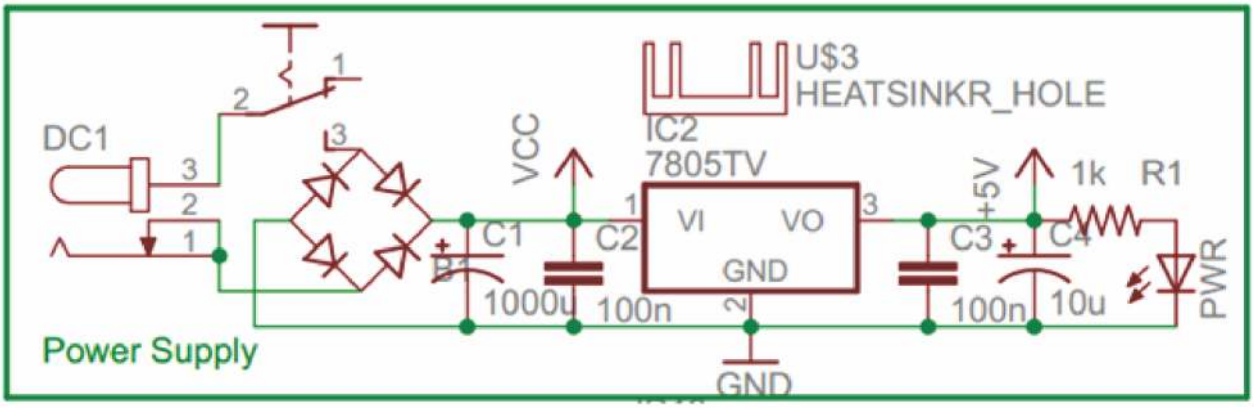
www.researchdesignlab.com

Email: sales@researchdesignlab.com I www.researchdesignlab.com

An ISO 9001- 2008 Certified Company

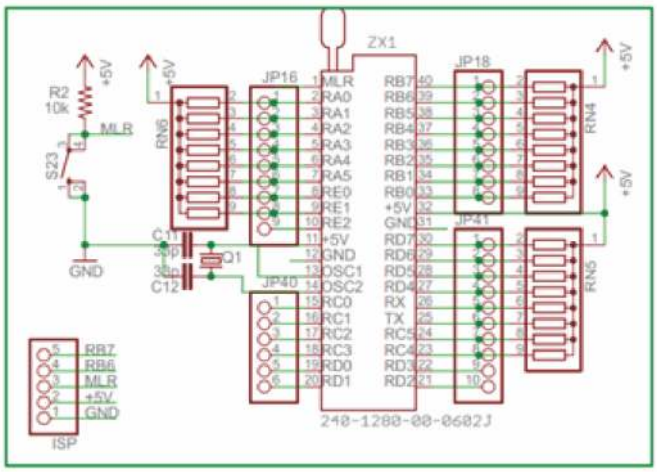
Power supply, 5V-12V

All digital circuits require regulated power supply. Here is a simple power supply circuit diagram used on this board.
You can use AC or DC source (12V) which converts into regulated 5V which is required for driving the development board circuit.



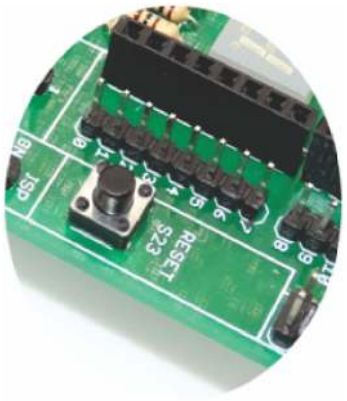
1. 40 pin ZIF socket for IC mount & ISP connector*

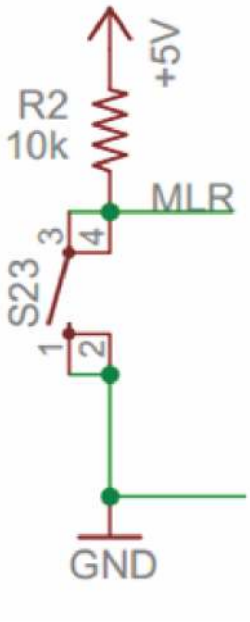
Select the IC's from the given list and mount on the ZIF socket. ZIF socket pin maps out PORT1 PORT2 PORT3 PORT4 for easy making connections for the rest of the circuit. Port 1 is enabled with pull up circuit and also connected ISP for easy on board Programming.



2. Reset

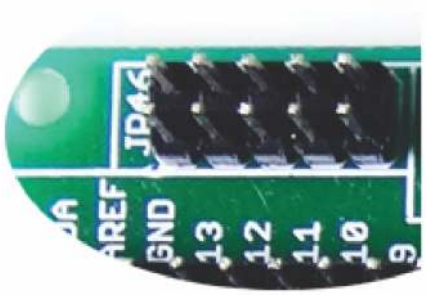
Resets your microcontroller by pressing s23

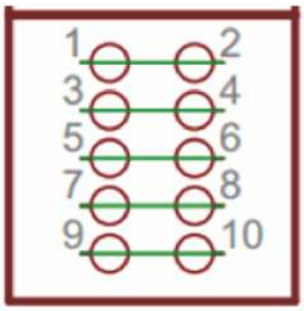




3. Node connector

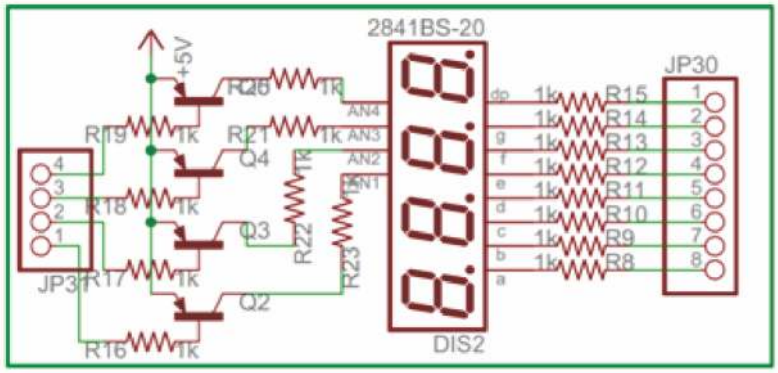
Node connector is an additional on board connection extender or 1 connection IN and 1 connection out





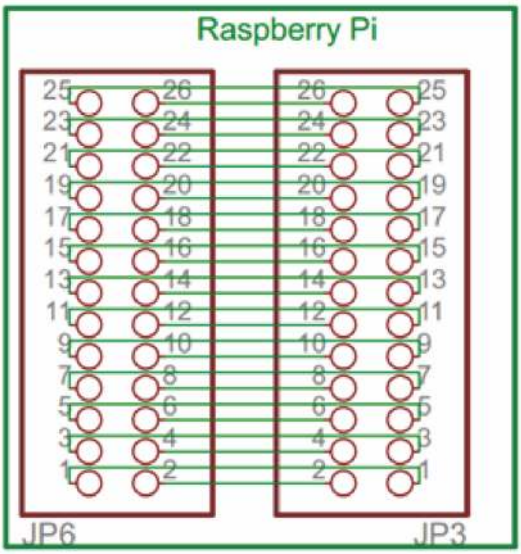
4. 4 digit 7 segment display

One seven segment digit consist of 7+1 LEDs which are arranged in a specific formation which can be used to represent digits from 0 to 9 and even some letters. One additional LED is used for marking the decimal dot, in case you want to write a decimal point in the desired segment.



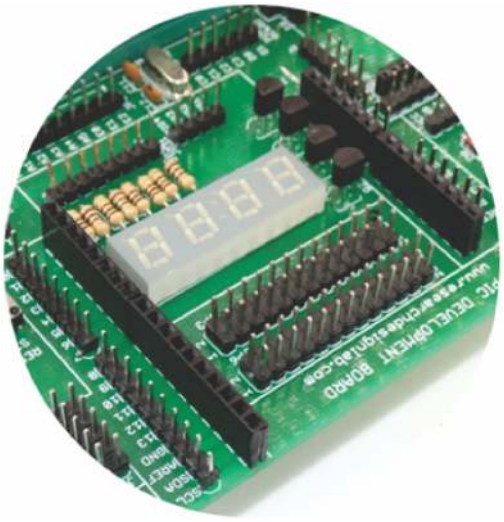
5. 26 pin raspberry connector

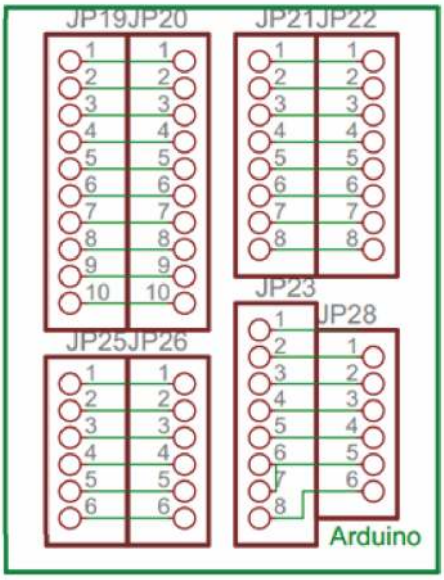
26 Pin raspberry connector is an easy way for making connections with raspberry pi with this development board.



6. Arduino Shield footprint

Arduino Shield footprint is provided in the board to mount different types of Arduino compatible shields on this development board.





7. ULN 2803 driver

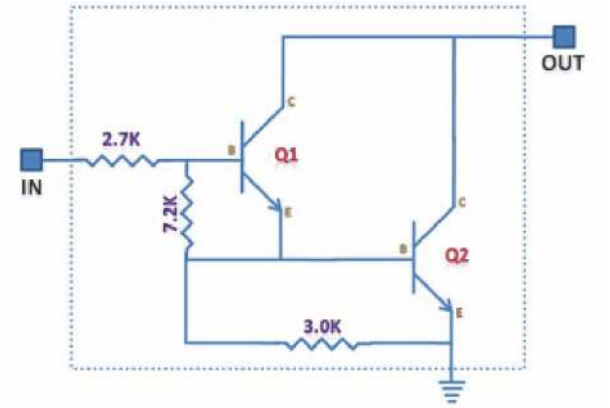
IC ULN2803 consists of octal high voltage, high current darlington transistor arrays. The eight NPN Darlington connected transistors in this family of arrays are ideally suited for interfacing between low logic level digital circuitry (such as TTL, CMOS or PMOS/NMOS) and the higher current/voltage requirements of lamps, relays, printer hammers or other similar loads for a broad range of computer, industrial, and consumer applications.

Features

- Eight Darlingtons with Common Emitter.
- Open-collector outputs.
- Free wheeling clamp diodes for transient suppression.
- Output Current to 500 mA.
- Output Voltage to 50 V.
- Inputs pinned opposite outputs to simplify board layout.

Working

The ULN 2803 IC consists of eight NPN Darlington connected transistors (often called a Darlington pair). Darlington pair consists of two bipolar transistors such that the current amplified by the first is amplified further by the second to get a high current gain ß or hFE. The figure shown below is one of the eight Darlington pairs of ULN 2803 IC.



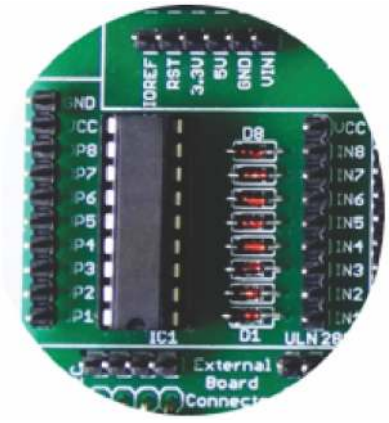
Now 2 cases arise:-

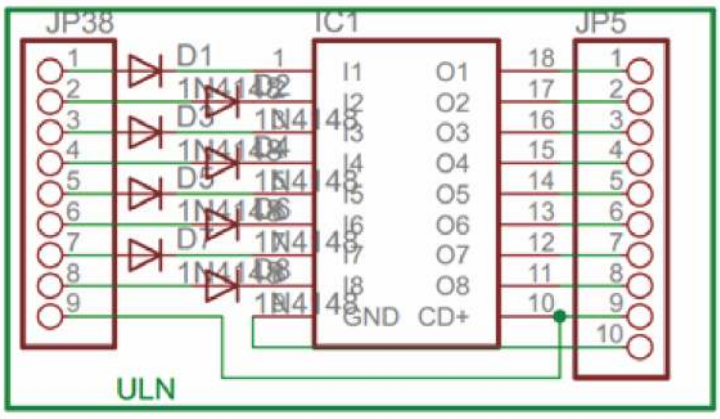
Case 1: When IN is 0 volts.

Q1 and Q2 both will not conduct as there is no base current provided to them. Thus, nothing will appear at the output (OUT).

Case 2: When IN is 5 volts.

Input current will increase and both transistors Q1 and Q2 will begin to conduct. Now, input current of Q2 is combination of input current and emitter current of Q1, so Q2 will conduct more than Q1 resulting in higher current gain which is very much required to meet the higher current requirements of devices like motors, relays etc. Output current flows through Q2 providing a path (sink) to ground for the external circuit that the output is applied to. Thus, when a 5V input is applied to any of the input pins (1 to 8), output voltage at corresponding output pin (11 to 18) drops down to zero providing GND for the external circuit. Thus, the external circuit gets grounded at one end while it is provided +Vcc at its other end. So, the circuit gets completed and starts operating.





8. I2C bus

One IC that wants to talk to another must: (Protocol)

1) Wait until it sees no activity on the I2C bus. SDA and SCL are both high. The bus is 'free'.

2) Put a message on the bus that says 'its mine' - I have STARTED to use the bus. All other ICs then LISTEN to the bus data to see whether they might be the one who will be called up (addressed).

3) Provide on the CLOCK (SCL) wire a clock signal. It will be used by all the ICs as the reference time at which each bit of DATA on the data (SDA) wire will be correct (valid) and can be used. The data on the data wire (SDA) must be valid at the time the clock wire (SCL) switches from 'low' to 'high' voltage.

4) Put out in serial form the unique binary 'address'(name) of the IC that it wants to communicate with.

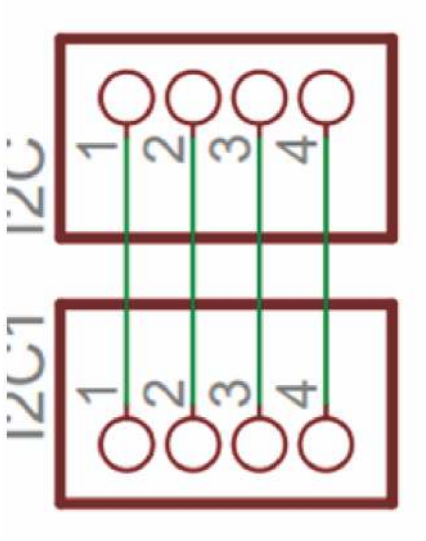
5) Put a message (one bit) on the bus telling whether it wants to SEND or RECEIVE data from the other chip. (The read/write wire is gone!)

6) Ask the other IC to ACKNOWLEDGE (using one bit) that it recognized its address and is ready to communicate.

7) After the other IC acknowledges all is OK, data can be transferred.

8) The first IC sends or receives as many 8-bit words of data as it wants. After every 8-bit data word the sending IC expects the receiving IC to acknowledge the transfer is going OK.

9) When all the data is finished the first chip must free up the bus and it does that by a special message called 'STOP'. It is just one bit of information transferred by a special 'wiggling' of the SDA/SCL wires of the bus.



9. SPI bus

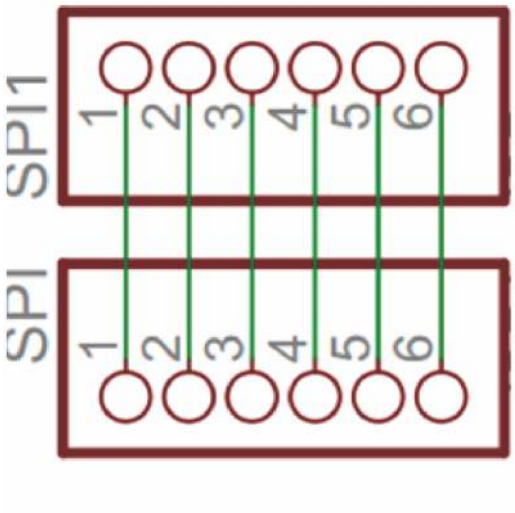
Serial to Peripheral Interface (SPI) is a hardware/firmware communications protocol developed by Motorola and later adopted by others in the industry. Microwire of National Semiconductor is same as SPI. Sometimes SPI is also called a "four wire" serial bus.

The Serial Peripheral Interface or SPI-bus is a simple 4-wire serial communications interface used by many microprocessor/microcontroller peripheral chips that enables the controllers and peripheral devices to communicate each other. Even though it is developed primarily for the communication between host processor and peripherals, a connection of two processors via SPI is just as well possible.

The SPI bus, which operates at full duplex (means, signals carrying data can go in both directions simultaneously), is a synchronous type data link setup with a Master / Slave interface and can support up to 1 megabaud or 10Mbps of speed. Both single-master and multi-master protocols are possible in SPI. But the multi-master bus is rarely used and look awkward, and are usually limited to a single slave.

The SPI Bus is usually used only on the PCB. There are many facts, which prevent us from using it outside the PCB area. The SPI Bus was designed to transfer data between various IC chips, at very high speeds. Due to this high-speed aspect, the bus lines cannot be too long, because their reactance increases too much, and the Bus becomes unusable. However, its possible to use the SPI Bus outside the PCB at low speeds, but this is not quite practical.

The peripherals can be a Real Time Clocks, converters like ADC and DAC, memory modules like EEPROM and FLASH, sensors like temperature sensors and pressure sensors, or some other devices like signal-mixer, potentiometer, LCD controller, UART, CAN controller, USB controller and amplifier.

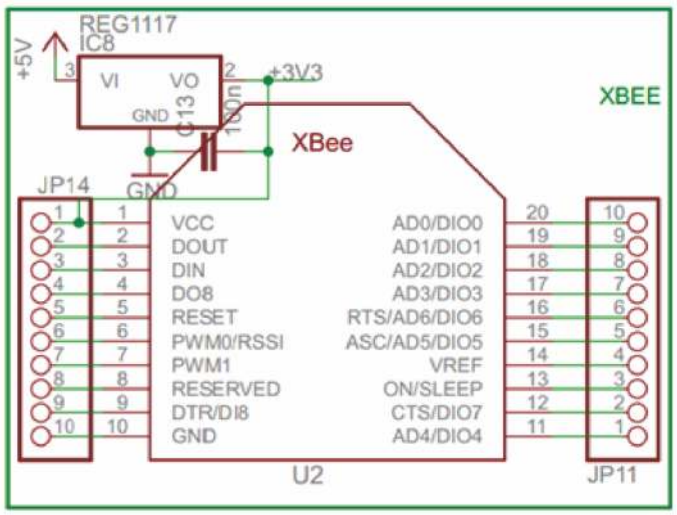


10. XBEE footprint/ XBEE Adaptor module

All XBeeZNet 2.5 modules can be identified by their unique 64-bit addresses or a user-configurable ASCII string identifier The 64-bit address of a module can be read using the SH and SL commands. The ASCII string identifier is configured using the NI command.

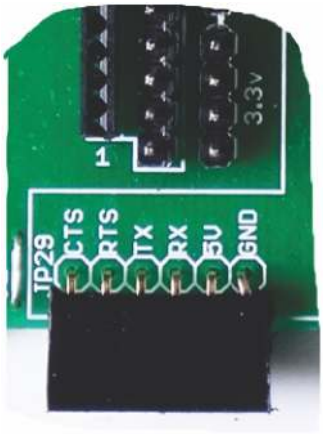
To transmit using device addressing, only the destination address must be configured. The destination address can be specified using either the destination device's 64-bit address or its NI-string. The XBee modules also support coordinator and broadcast addressing modes. Device addressing in the AT firmware is configured using the DL, DH, or DN commands. In the API firmware, the ZigBee Transmit Request API frame (0x10) can be used to specify destination addresses.

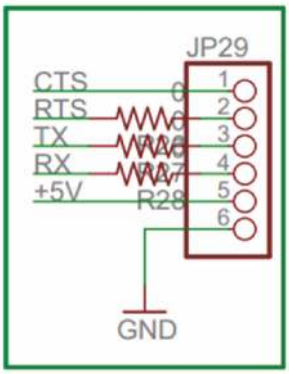
To address a node by its 64-bit address, the destination address must be set to match the 64-bit address of the remote. In the AT firmware, the DH and DL commands set the destination 64-bit address. In the API firmware, the destination 64-bit address is set in the ZigBee Transmit Request frame. ZigBee end devices rely on a parent (router or coordinator) to remain awake and receive any data packets destined for the end device. When the end device wakes from sleep, it sends a transmission (poll request) to its parent asking if the parent has received any RF data destined for the end device. The parent, upon receipt of the poll request, will send an RF response and the buffered data (if present). If the parent has no data for the end device, the end device may return to sleep, depending on its sleep mode configuration settings. The following figure demonstrates how the end device uses polling to receive RF data through its parent.

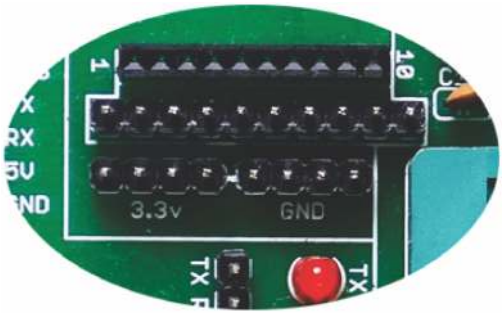


11. FT232 breakout board connector

A standard FT232 breakout board from researchdesignlab.com could be used to interface on these connectors, whose other end is connected to a USB.





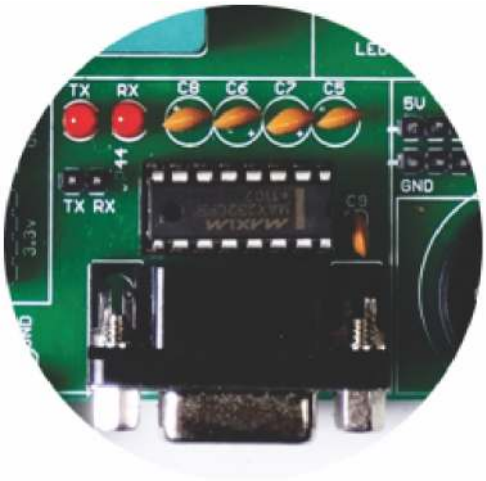


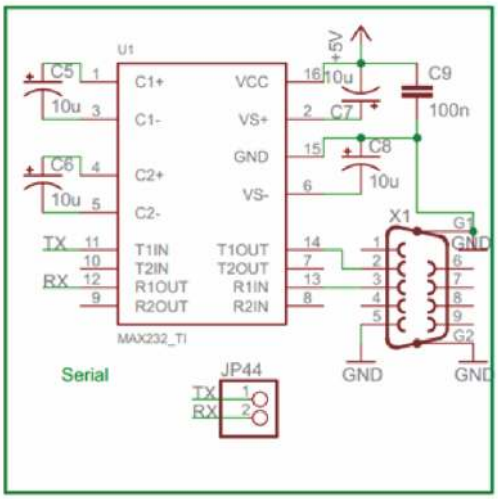
12. DC 3.3V connectors

These connectors provide on board 3.3V DC connections.

13. DB-9 female connector

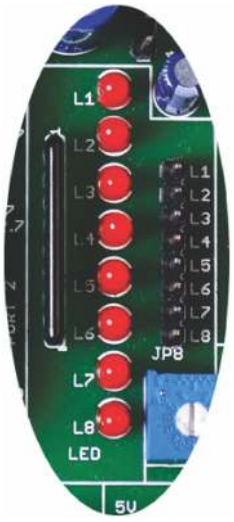
RS-232 is a standard communication protocol for linking computer and its peripheral devices to allow serial data exchange. In simple terms RS232 defines the voltage for the path used for data exchange between the devices. It specifies common voltage and signal level, common pin wire configuration and minimum, amount of control signals.

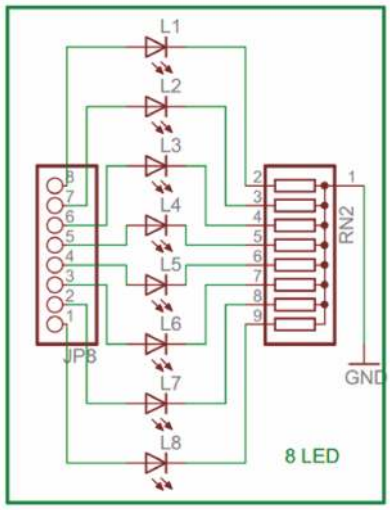




14. 8x1 LED's

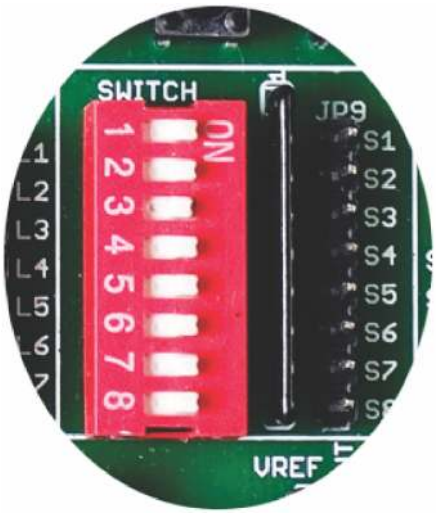
LED's are used to indicate something, whether any pin is high or indicating the output for many purposes like indicating I/O status or program debugging running state. We have four led outputs on board which can be used by the programmer as per the requirement for testing and development.

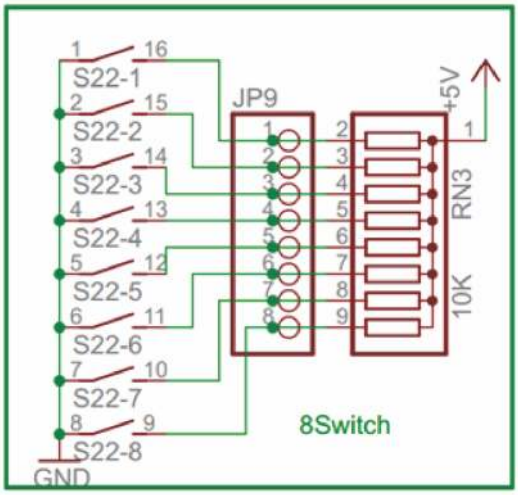




15. 8 way DIP switch

DIP switches are an alternative to jumper blocks. Their main advantages are that they are quicker to change and there are no parts to lose.





16. RTC Module

The DS1307 Serial Real Time Clock is a low power, full BCD clock/calendar plus 56 bytes of nonvolatile SRAM. Address and data are transferred serially via a 2-wire bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit which detects power failures and automatically switches to the battery supply.

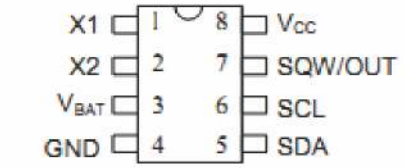
Operation

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When VCC falls below 1.25 x VBAT the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out of tolerance system. When VCC falls below VBAT the device switches into a low current battery backup mode. Upon power up, the device switches from battery to VCC when VCC is greater than VBAT +0.2V and recognizes inputs.

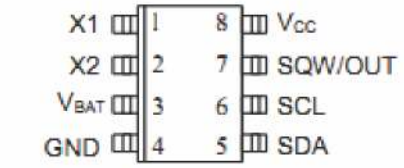
Features:
1. 56 byte nonvolatile RAM for data storage
2. 2-wire serial interface
3. Programmable square wave output signal
4. Automatic power-fail detect and switch circuitry
5. Consumes less than 500 nA in battery backup mode with oscillator running
6. Optional industrial temperature range -40°C to +85°C
7. Available in 8-pin DIP or SOIC
8. Recognized by Underwriters Laboratory

PIN DESCRIPTION
1. VCC - Primary Power Supply
2. X1, X2 - 32.768 kHz Crystal Connection
3. VBAT - +3V Battery Input
4. GND - Ground
5. SDA - Serial Data
6. SCL - Serial Clock
7. SQW/OUT - Square wave/Output Driver

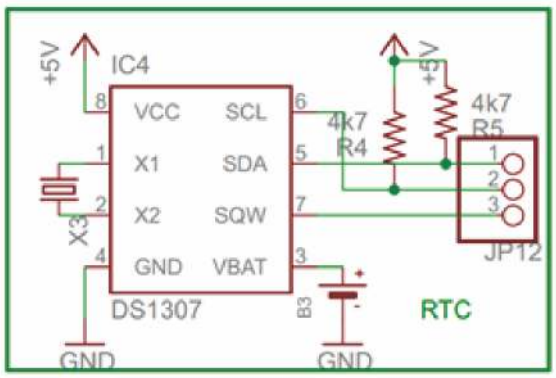
PIN ASSIGNMENT

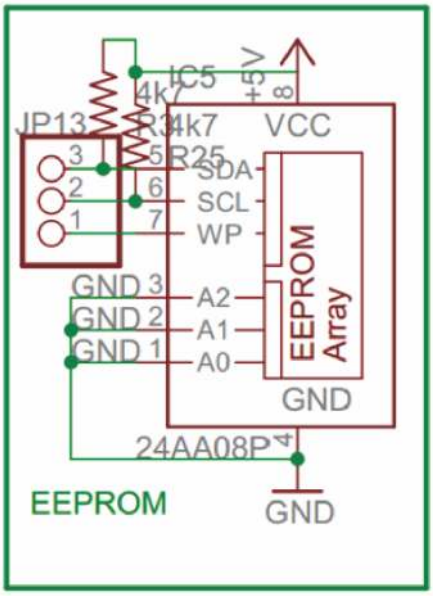


DS1307 8-Pin DIP (300 mil)



DS1307Z 8-Pin SOIC (150 mil)



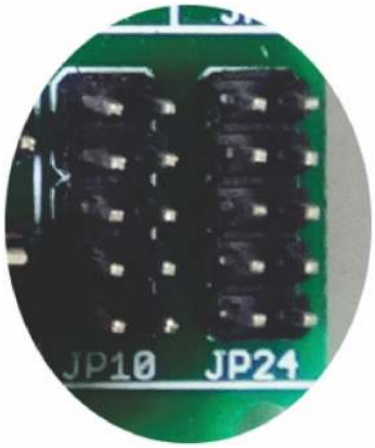


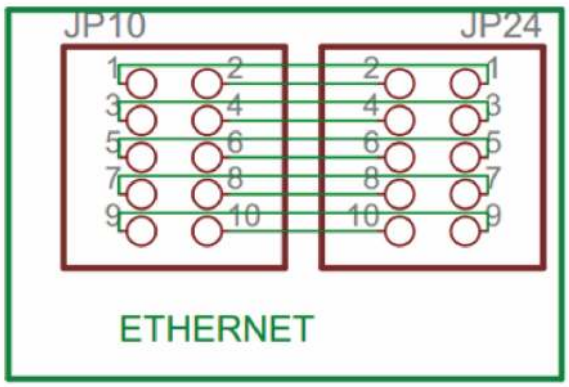
17. EEPROM

IC, EEPROM I2C 4K, 24C04, DIP8
Memory Size: 4Kbit
Memory Configuration: 512 x 8
Interface Type: I2C, Serial
Clock Frequency: 400kHz
Supply Voltage Range: 2.5V to 5.5V
Memory Case Style: DIP
No. of Pins: 8
Operating Temperature Range: -40°C to +85°C
SVHC: No SVHC (19-Dec-2011)
Base Number: 24
Device Marking: M24C04
IC Generic Number: 24C04
Interface: I2C
Interface Type: Serial, I2C
Logic Function Number: 24C04
Memory Configuration: 512 x 8
Memory Size: 4Kbit
Memory Type: EEPROM
Memory Voltage Vcc: 2.5V
Operating Temperature Max: +85°C
Operating Temperature Min: -40°C
Package / Case: DIP
Supply Voltage Max: 5.5V
Supply Voltage Min: 2.5V
Termination Type: Through Hole
Voltage Vcc: 2.5V

18. 2x5x2 jumper node

Node connector is an additional on board connection extender
or 1 connection IN and 1 connection OUT





19. DC 5V connectors

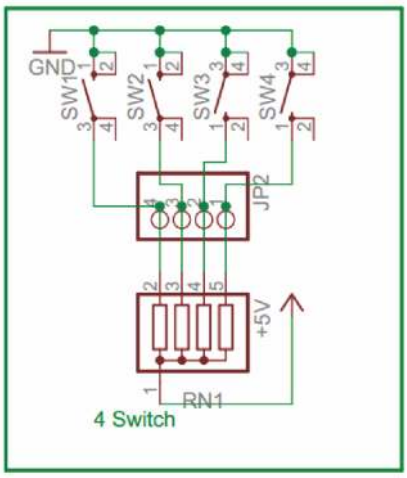
These connectors provide
on board 5V DC connections.

20. Potentiometer

The Potentiometer Option allows the user to adjust the frequency reference by rotating a potentiometers dial. Turning the potentiometer changes the frequency reference making it easier to adjust the motor speed and also to set the duty cycle for PWM values.

21. 4x1 keypad

Switches are mainly used to switch the controls of a module. We have four switches on board which can be used by the programmer as per the requirement for testing and development



22. 16x2 LCD connectors

LCD screen consists of two lines with 16 characters each. Each character consists of 5x7 dot matrix. Contrast on display depends on the power supply voltage and whether messages are displayed in one or two lines. For that reason, variable voltage 0-Vdd is applied on pin marked as Vee. Trimmer potentiometer is usually used for that purpose. Some versions of displays have built in backlight (blue or green diodes). When used during operating, a resistor for current limitation should be used (like with any LE diode). LCD Connection Depending on how many lines are used for connection to the microcontroller, there are 8-bit and 4-bit LCD modes. The appropriate mode is determined at the beginning of the process in a phase called “initialization”. In the first case, the data are transferred through outputs D0-D7 as it has been already explained. In case of 4-bit LED mode, for the sake of saving valuable I/O pins of the microcontroller, there are only 4 higher bits (D4-D7) used for communication, while other may be left unconnected.

Consequently, each data is sent to LCD in two steps: four higher bits are sent first (that normally would be sent through lines D4-D7), four lower bits are sent afterwards. With the help of initialization, LCD will correctly connect and interpret each data received. Besides, with regards to the fact that data are rarely read from LCD (data mainly are transferred from microcontroller to LCD) one more I/O pin may be saved by simple connecting R/W pin to the Ground. Such saving has its price. Even though message displaying will be normally performed, it will not be possible to read from busy flag since it is not possible to read from display.

Features:

1. Can display 224 different symbols.
2. Low power consumption.
3. 5x7 dot matrix format.
4. Powerful command set and user produced characters.

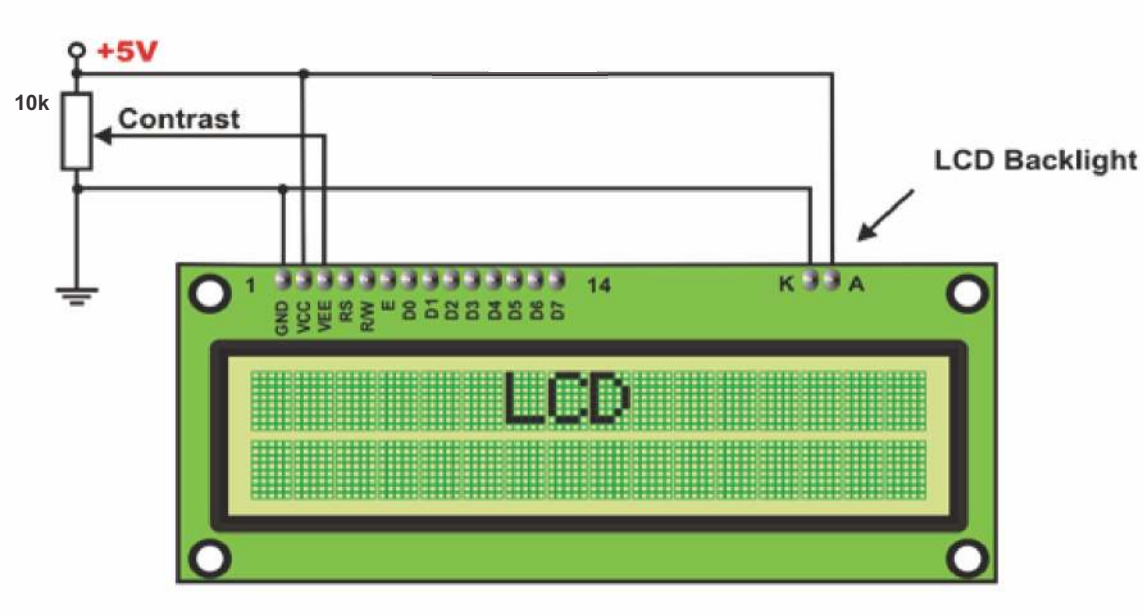
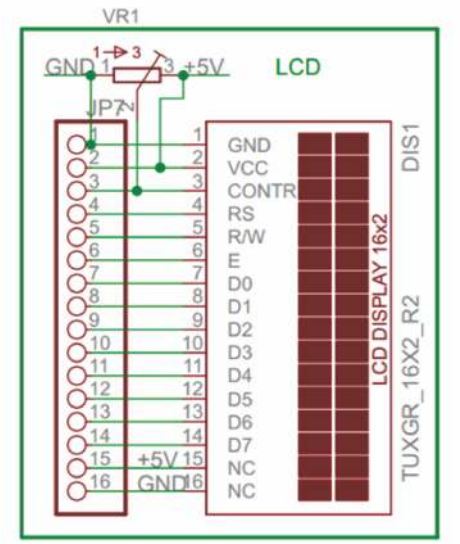


Fig: Circuit connections of LCD

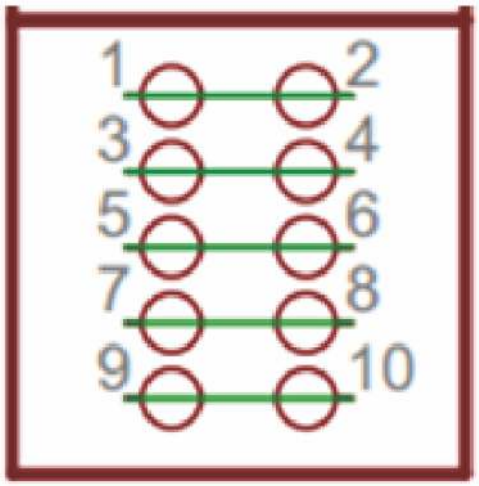
Pin Description

1. Gnd:- Power supply ground
2. VCC:-+5v Power supply input
3. RS:- Reset pin
4. R/W:- Read/Write pin
5. En:-Enable pin
6. D0-D7:- Data lines



23. Node connector

Node connector is an additional on board connection extender or 1 connection IN and 1 connection out



24. 4x4 Matrix Keypad

In a 4x4 matrix keypad eight Input/Output ports are used for interfacing with any microcontrollers. Rows are connected to Peripheral Input/Output (PIO) pins configured as output. Columns are connected to PIO pins configured as input with interrupts. In this configuration, four pull-up resistors must be added in order to apply a high level on the corresponding input pins as shown in below Figure. The corresponding hexadecimal value of the pressed key is sent on four LEDs.

Working

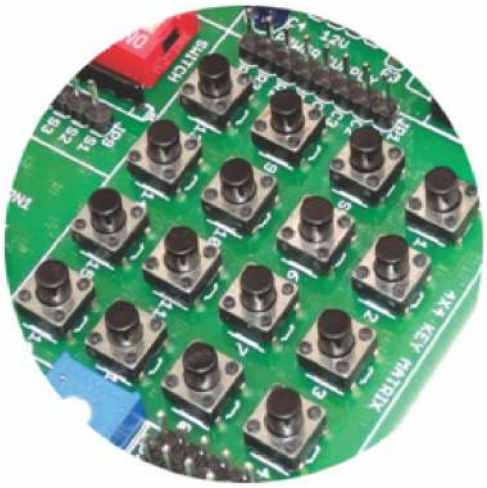
This Application Note describes programming techniques implemented on the AT91 ARM-based microcontroller for scanning a 4x4 Keyboard matrix usually found in both consumer and industrial applications for numeric data entry.AT91 Keyboard interface In this application, a 4x4 matrix keypad requiring eight Input/Output ports for interfacing is used as an example. Rows are connected to Peripheral Input/Output (PIO) pins configured as output. Columns are connected to PIO pins configured as input with interrupts. In this configuration, four pull-up resistors must be added in order to apply a high level on the corresponding input pins as shown in Figure 1. The corresponding hexadecimal value of the pressed key is sent on four LEDs.

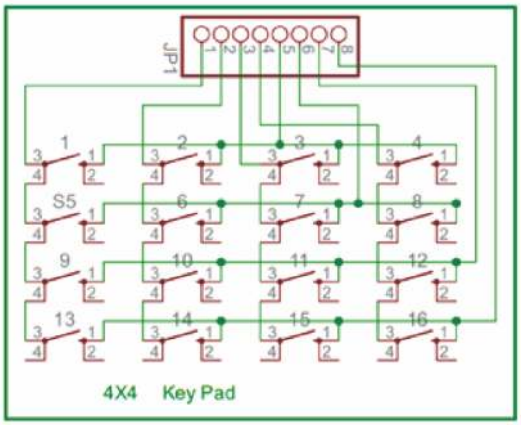
FEATURES

1. Contact debouncing.
2. Easy to interface.
3. Interfaces to any microcontroller or microprocessor.
4. Data valid output signal for interrupt activation.

PIN DETAILS

pin 1-4: R0-R3:- Rows
pin 5-8: C0-C3:- Columns





25. DC 12V connectors

These connectors provide on board 12V DC connections.