



SSG EMBEDDED SOLUTIONS

SSG

info@ssges.co.in

Embedded solutions

SSG EMBEDDED SOLUTIONS | 7123559635

Microcontroller Development Kit

8051

(NUVOTON W78E052DDG)

Documentation



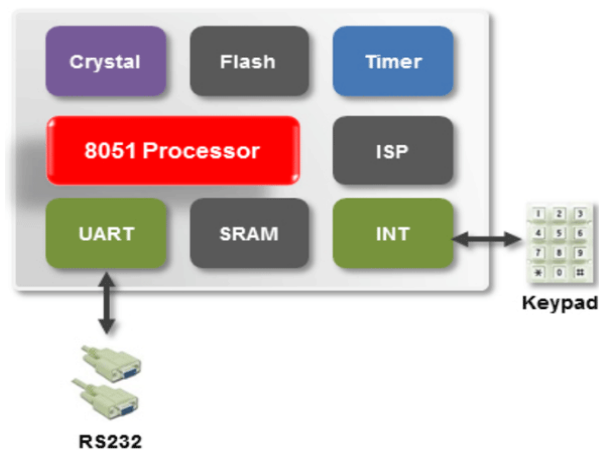


List of Contents

	Page No.
1. Description of 8051	1-2
<ul style="list-style-type: none">• Introduction• Pin Diagram• Features and Specifications• Application	
2. Installing Software	3-8
<ul style="list-style-type: none">• Install Keil software• Installing programmer ESPISP• Installing X-CTU	
3. User guide	9-16
<ul style="list-style-type: none">• How to use Keil software• How to use Nuvoton ISP-ICP programmer	
4. Examples	17-85
<ul style="list-style-type: none">• LED Blinking• LCD Display• RS232 Serial Communication37-44• Seven Segment• ADC• DAC• Stepper Motor• Multiplex Keyboard• Relay• RTC	

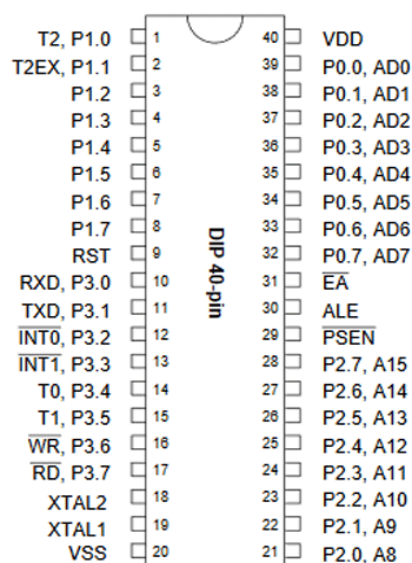
Introduction

The W78E052DDG 8-bit microcontroller is an **8051 based CMOS microcontroller** unit that has limited peripheral support having a vast range of GPIOs and a single full-duplex UART. This controller has a 3pcs 16bits Timer/Counter. A **full-duplex single UART** is very good to make communication with PC-based applications. Here is the diagram that is offered by the **nuvoton W78E052DDG**.



W78E052DDG is a successive low-cost, high pin count microcontroller mainly designed for I/O based operations where peripheral support is less important but I/O counts are required.

Pin Diagram





Features and Specifications

- Core: 8051-based CMOS
- 8052 compatible instruction sets
- Data Bus width: 8bit
- Maximum Clock Frequency: 40 MHz (12T mode, 12 clocks per machine cycle operation), 20 MHz (6T Mode, 6 clocks per machine cycle operation)
- Program Memory size: 8kB
- Data RAM size: 256B
- MCS-51 compatible pin and instruction set.
- Operating supply: 2.4V to 5.5 V
- Minimum operating Temperature: - 40C
- Maximum Operating Temperature : + 85C
- Program Memory Type: Flash
- Interface Type: UART
- Number of 8-bit I/O ports: 4
- Number of Times/counters: 3
- Timer / Counter Resolution - 16 bits
- Product Type: 8-bit 8051 based Microcontrollers MCU
- EMI Reduction mode
- Full-duplex serial port - 1

Application

- I/O operations
- Multi-segment display driver
- Low power Embedded devices.
- Small data retention-related purposes.
- Keypad interfacing

Install Keil software

STEP 1: Go to <https://www.keil.com/download/product/> → **Download** → **Product Downloads** → Hit on **C51 Setup**. Enter your contact information with valid address, phone number and email. Download is free for evaluation version.



STEP 2: Then click on **C51V954A.EXE** and **Download** it on your computer.



STEP 3: Next step is to run setup file **C51V954A.EXE** and then we'll get pop-up box, hit on **Next** and Proceed Installation.



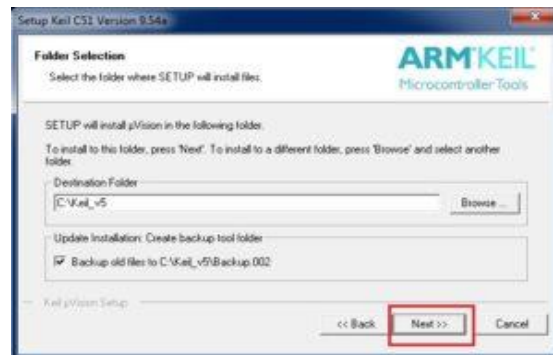
Proceed Installation Keil 8051

STEP 4: Read license agreement, check **I agree to all the terms....**, and click **Next**.



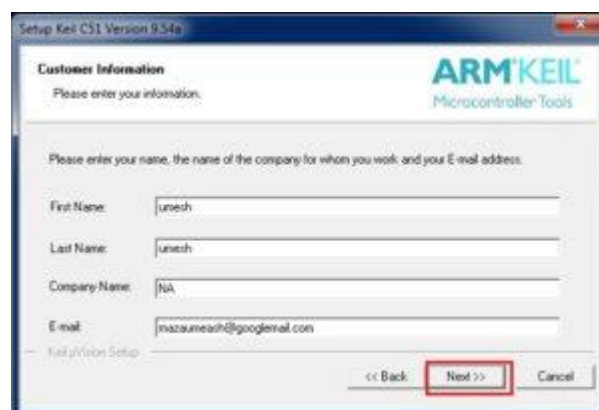
Accept Terms and Conditions

STEP 5: Select Destination folder where you want to install Keil or default destination is already there. And hit on **Next**.



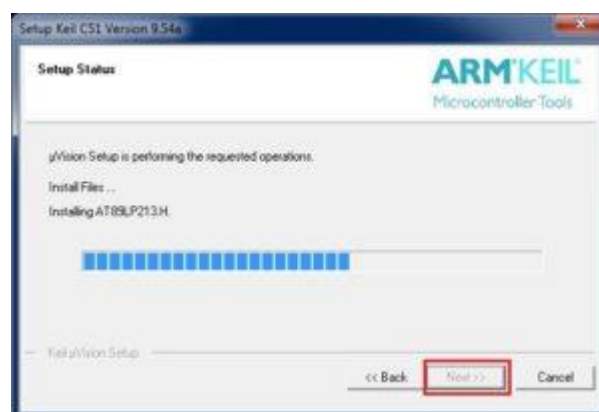
Choose Installation Directory

STEP 6: Fill up required fields with all relevant information and click on **Next**.



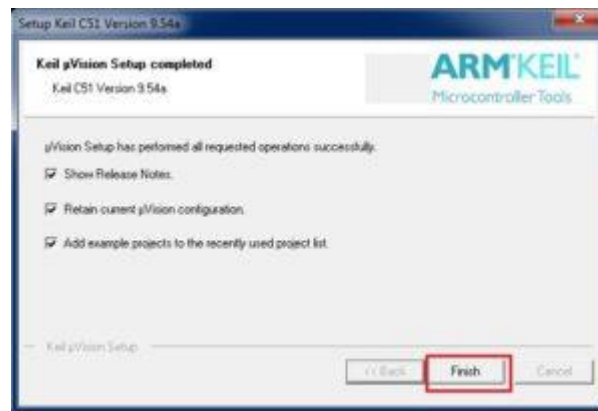
Proceed with Registered Credentials

STEP 7: Wait for installation completes and hit on **Next**.



Wait until Installation finishes

STEP 8: Tick on show release notes, deselect remaining (as per your choice) and click on **Finish**.

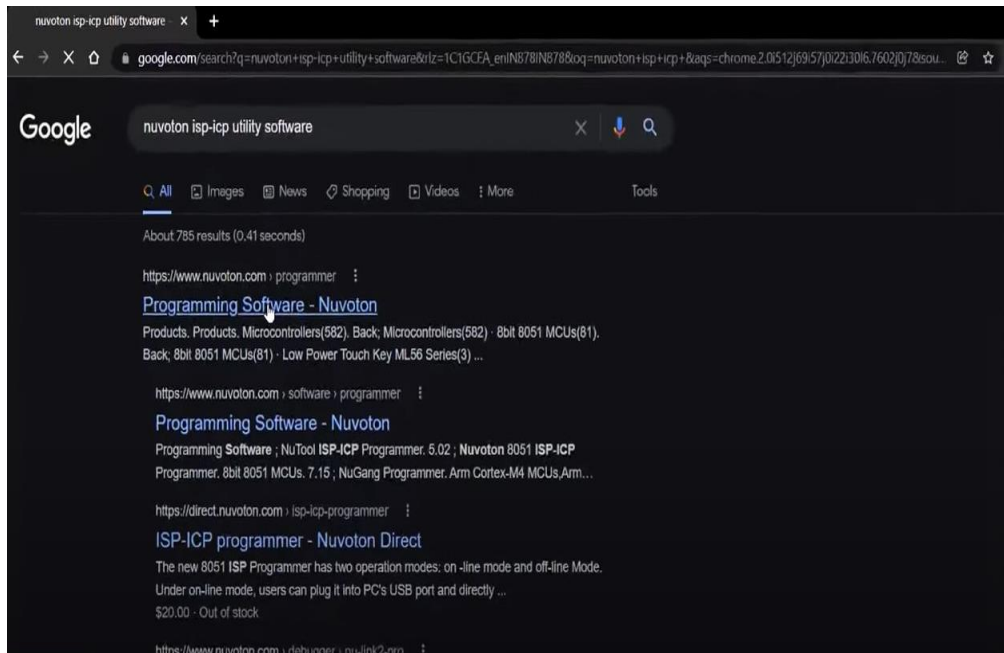


Click Finish And Installation Done

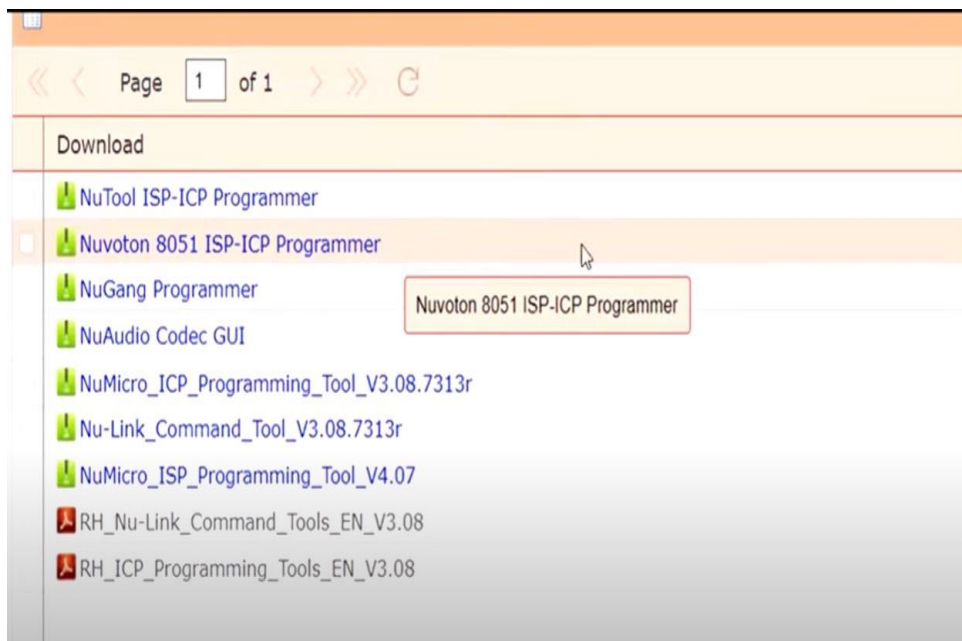
Now we're done, install Keil to Program 8051 Microcontroller under windows. We hope you will find this tutorial educational. If you have any question then please feel free to leave a comment. Thanks and see you in next tutorial where we'll create fresh new Keil uVision project for 8051 Microcontroller.

programmer ISP-ICP

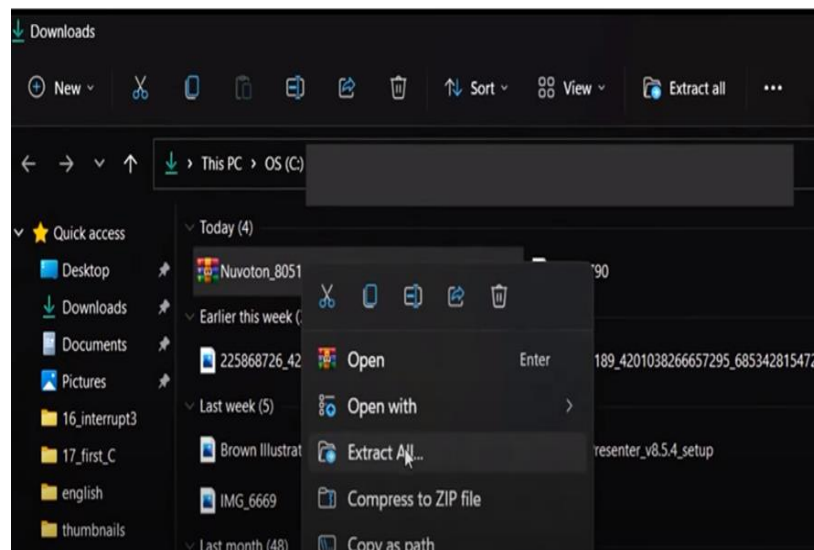
STEP 1:



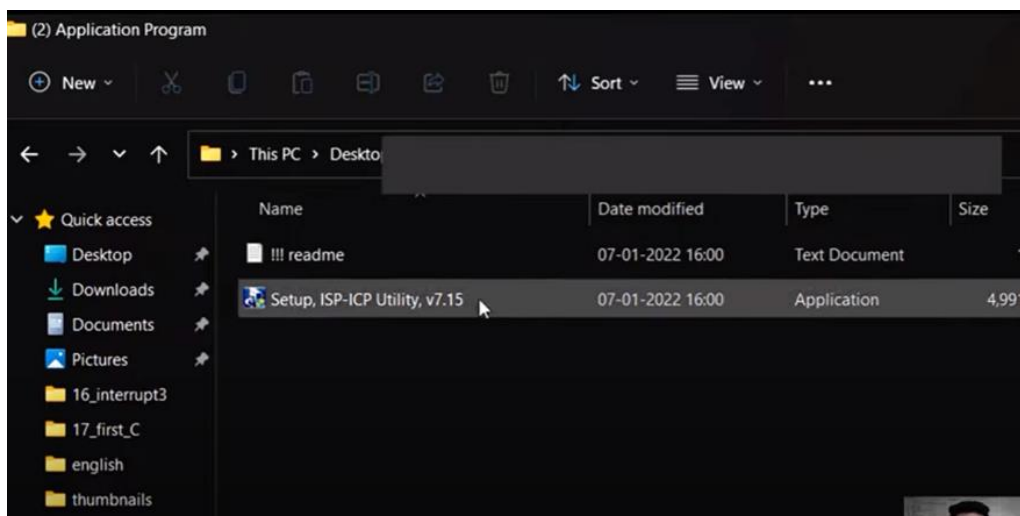
STEP 2:



STEP 3:



STEP 4:



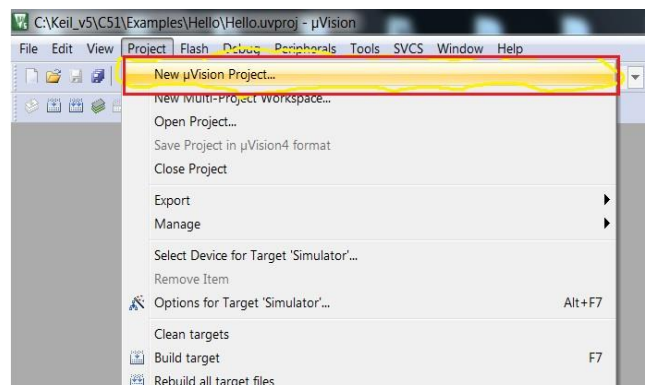
NOTE:

If any issue to download the programmer go through tool folder open nuvoton ic programmer and install it as well as X-CTU.

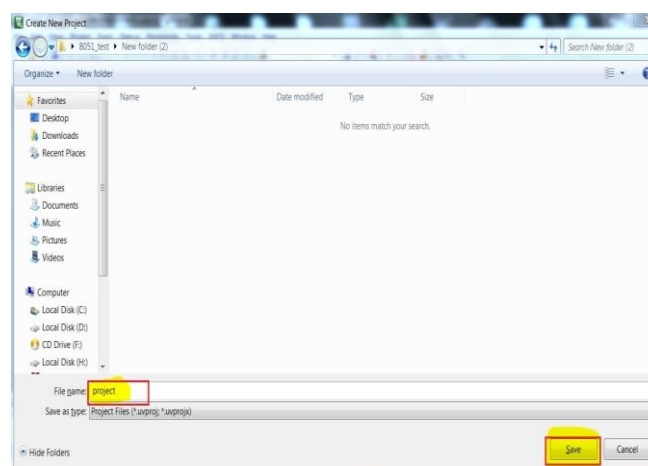
How to use Keil software

STEP 1: Before we proceed any further, first launch Keil uVision5 application from computers program menu.

STEP 2: Now click on **Project** tab and select **New uVision Project...**

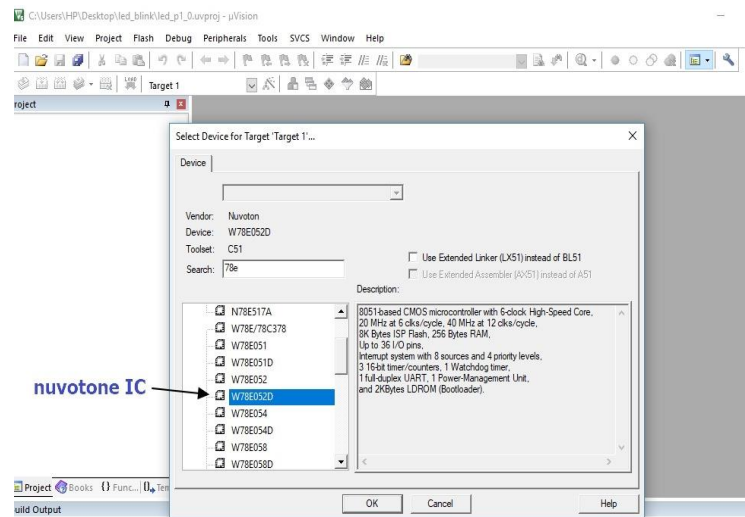


STEP 3: Give any name to project (Here we have given a name as **project**). And **Save** project in new folder.

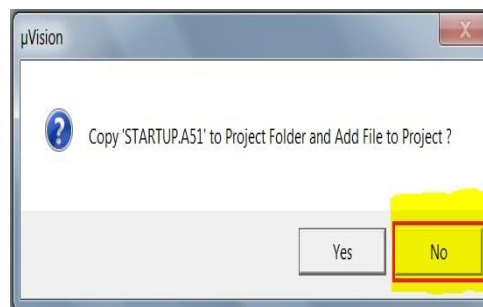


STEP 4: Now in select **Device for Target 'Target 1'...** Search for appropriate derivative from family of 8051 microcontroller (for example: P89V51RD2, AT89C51, AT89S51 etc). Here in this case we have chosen **Nuvoton W78E052D**. As we'll be experimenting in future with W78E052D

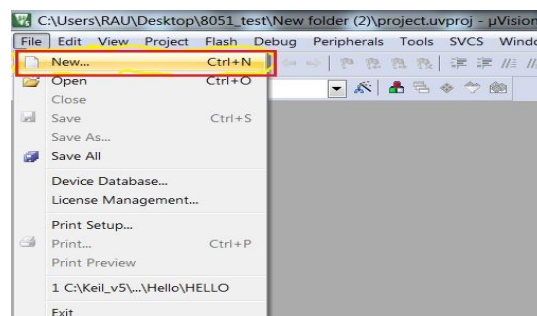
Then select your device and hit on **OK**.



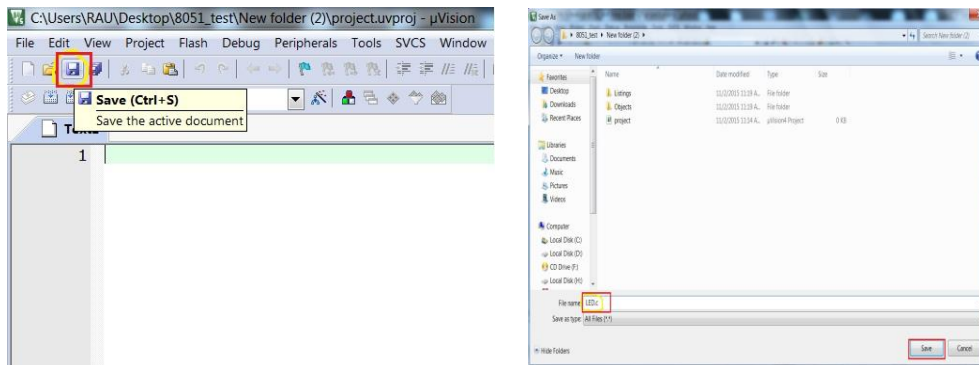
STEP 5: When ask for **Copy 'STARTUP.A51' to Project Folder and Add File to Project?**
Click **NO**



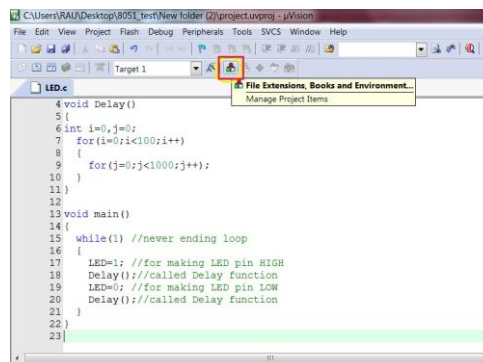
STEP 6: Now to create source file go to **File** tab and click on **New...**



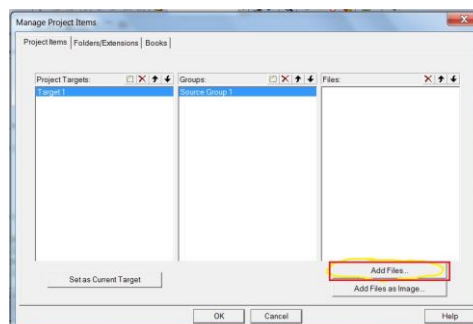
STEP 7: Here **save** this file with **.c** (e.g. LED.c) extension in **project** folder. which we have created earlier.



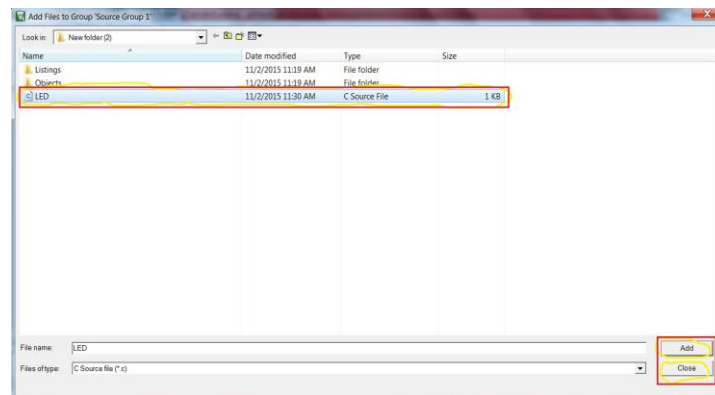
STEP 8: At this point, we're ready to write code/program. Now click on **File Extension, Books and Environment...** tab



STEP 9: Then click on **Add Files...**

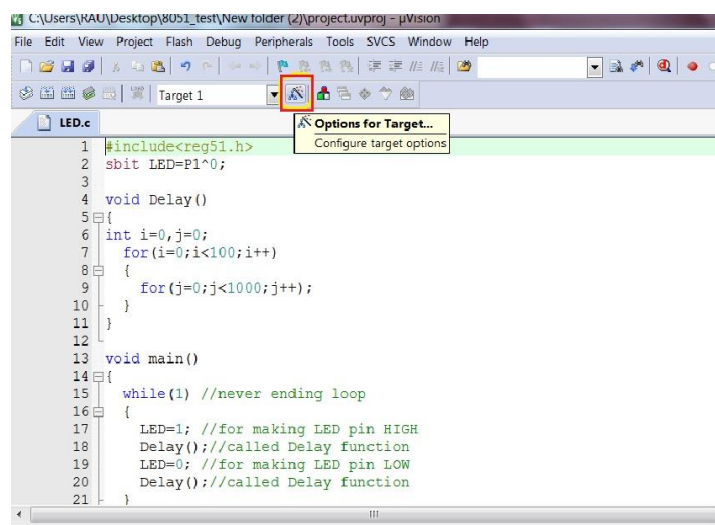


Now **select file** which we have saved with .c extension. Then click on **Add** button. And then hit on **Close**.

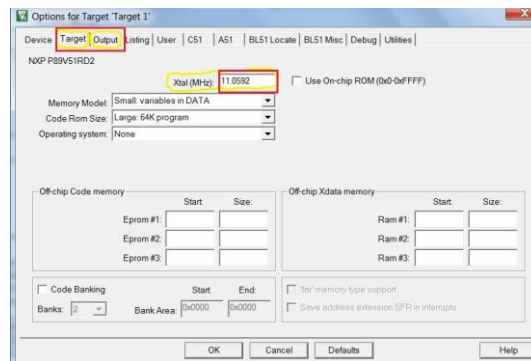


Here you can see your file gets added in Files: window. Now click **OK**

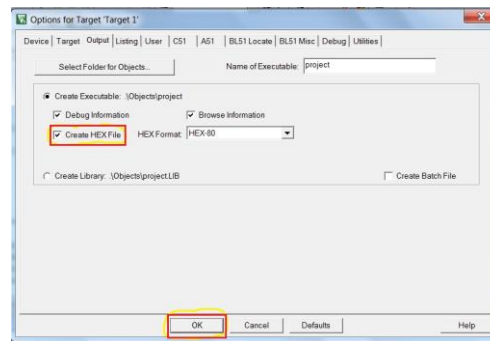
STEP 10: Now click on **Options for Target...** tab



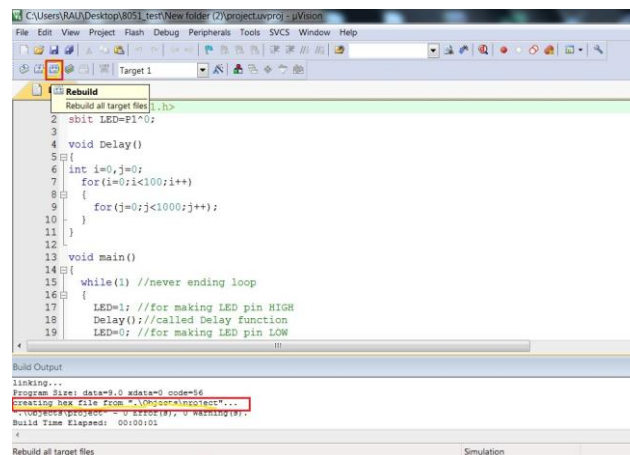
In Target window type clock frequency which we're using for 8051 microcontroller. Throughout this tutorial series we are going to use **11.0592MHz crystal** clock.



Now in **Output** tab click on check box of **Create HEX File**. Then hit on **OK**



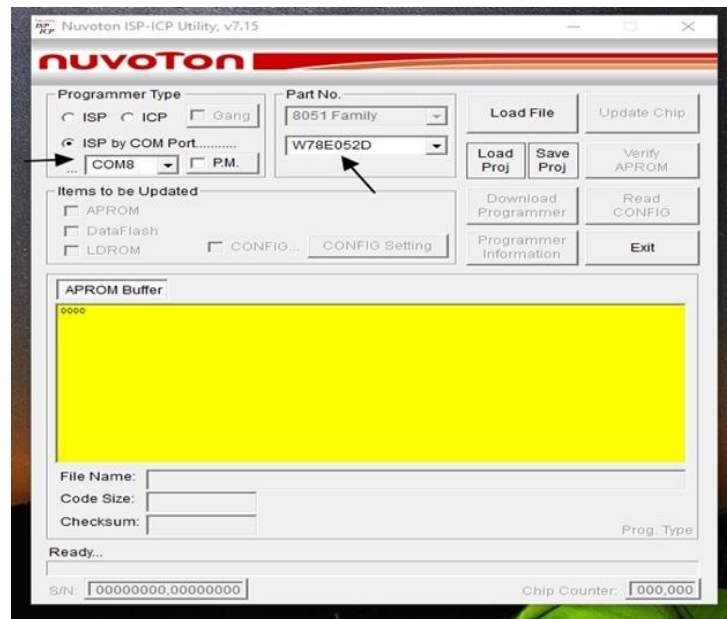
STEP 11: Now click on **“Build”** tab for building project file. Then click on **Rebuild**



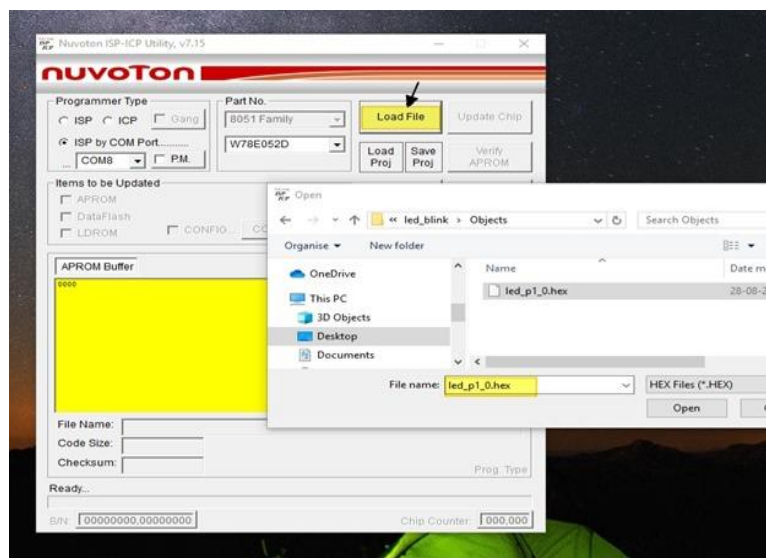
At this point, if there isn't be any error then we'll get HEX file into **“Objects folder”** with .HEX extension. This is how we'll Create Keil project for 8051 Microcontroller using uVision5.

How to use Nuvoton ISP-ICP programmer

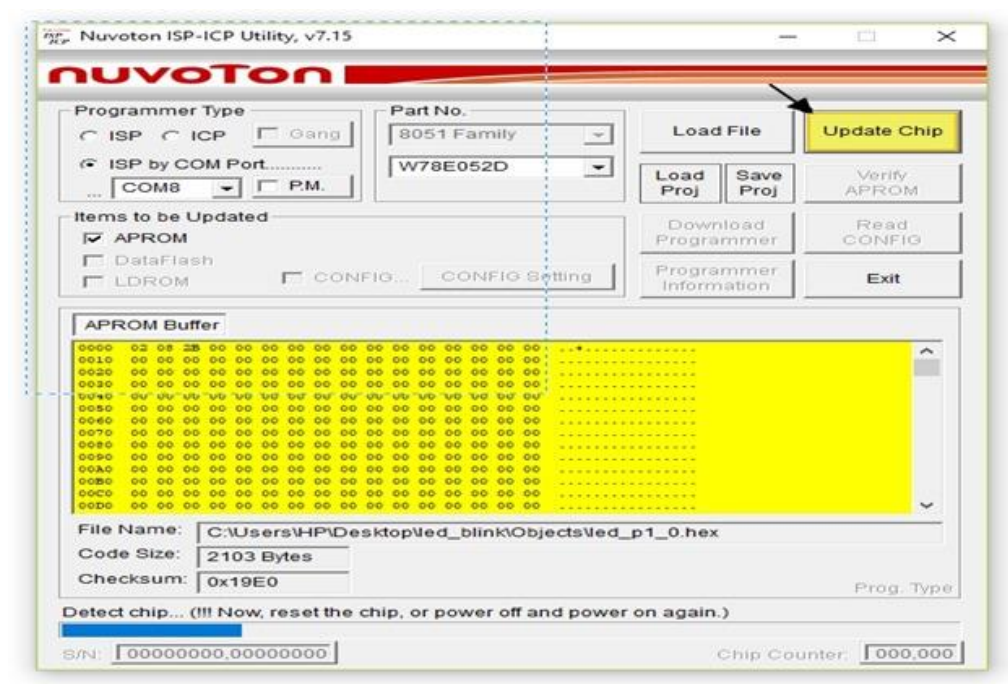
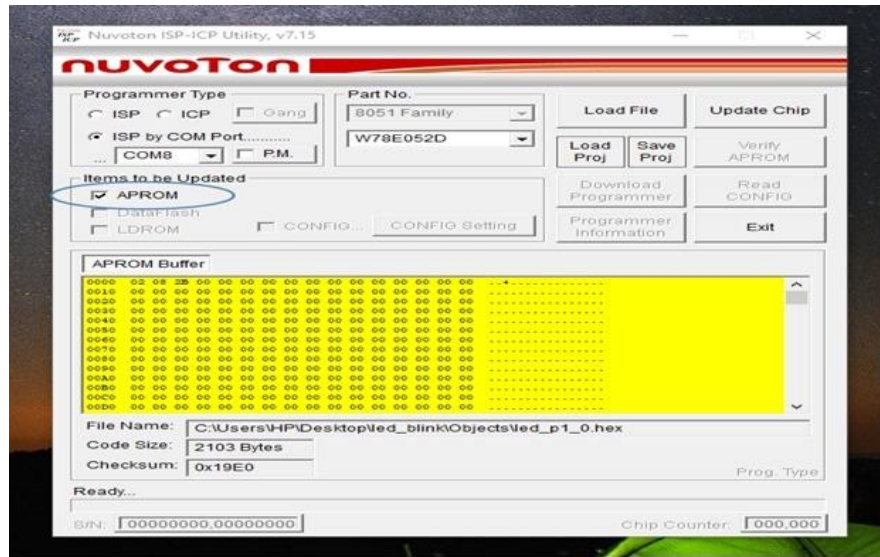
STEP 1: Select IC and Port



STEP 2: Load File



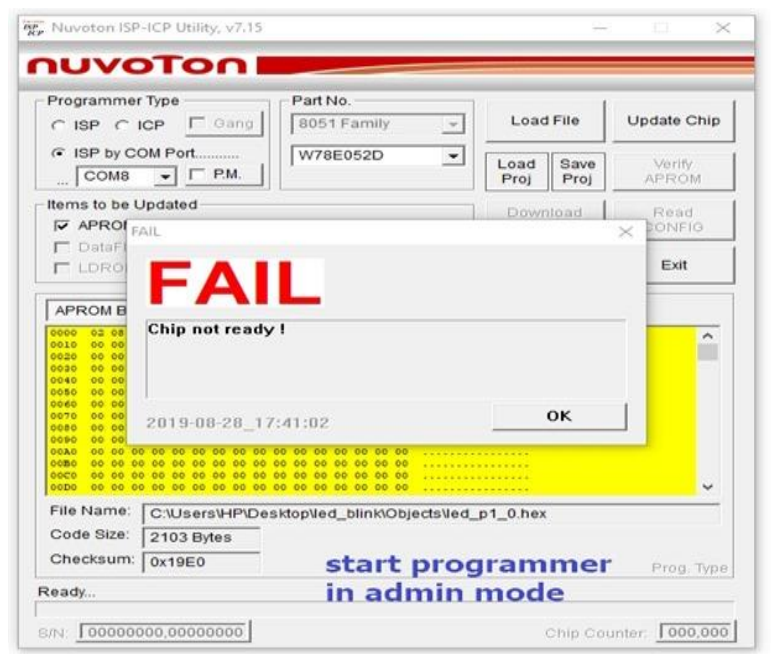
STEP 3: Select APROM



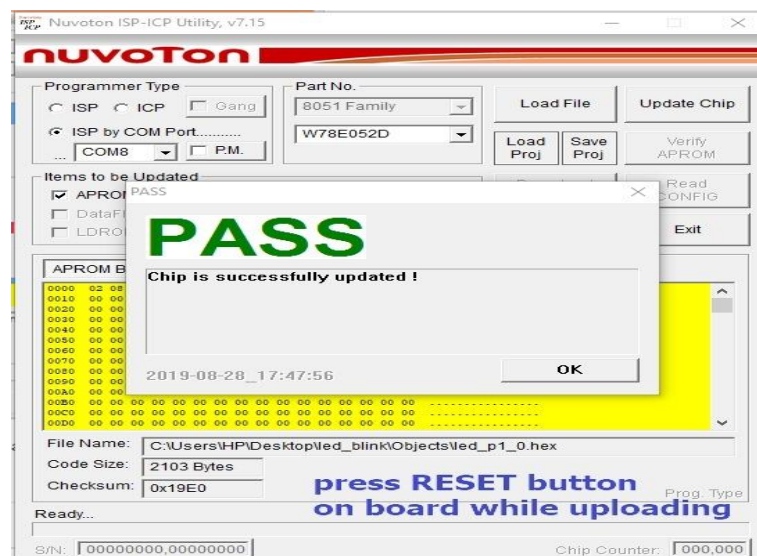
NOTE:

If RESET button is not pressed , you get FAIL report. NUVOTONE Utility must be started in ADMIN mode.

It shows below snap

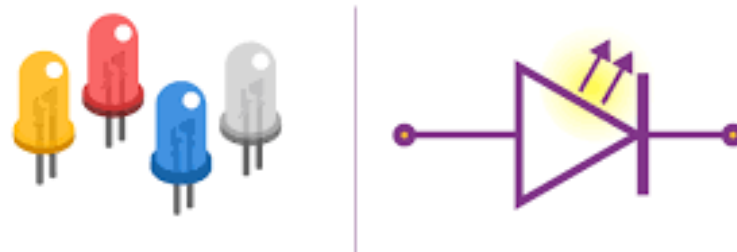


After press reset shows below snap



LED

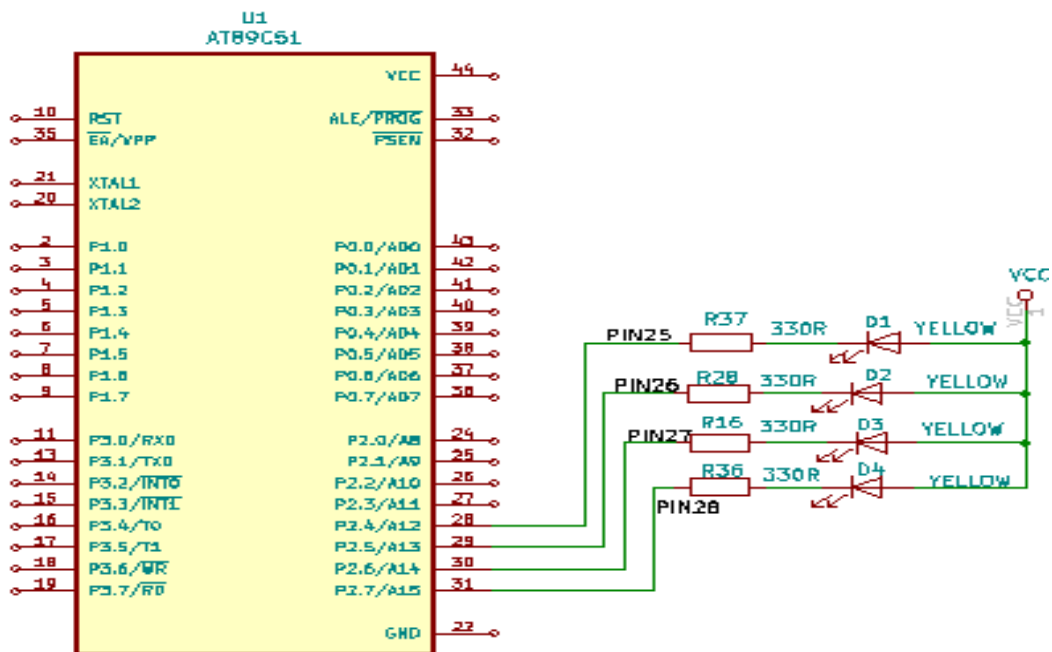
It is most widely used semiconductor which emit either visible light or invisible infrared light when forward biased. Remote controls generate invisible light. A Light emitting diodes (LED) is an optical electrical energy into light energy when voltage is applied.



These are the applications of LEDs:

- Digital computers and calculators.
- Traffic signals and Burglar alarms systems.
- Camera flashes and automotive heat lamps
- Picture phones and digital watches.

Blinking LED Schematic and Code



Code

```
#include<REG51.H>

//Fuction Prototypes
void delay(void);

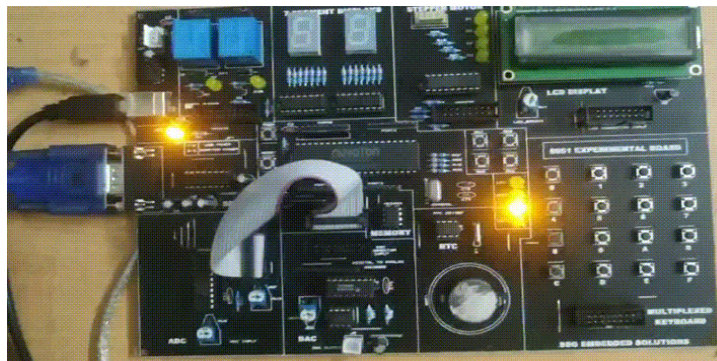
// Program Starts Here
void main()
{
    P2 = 0x00; // Leds are ON
    delay();
    P2 = 0xff; // Leds are OFF
    delay();
}

void delay(void)
{
    unsigned int a,b;
```

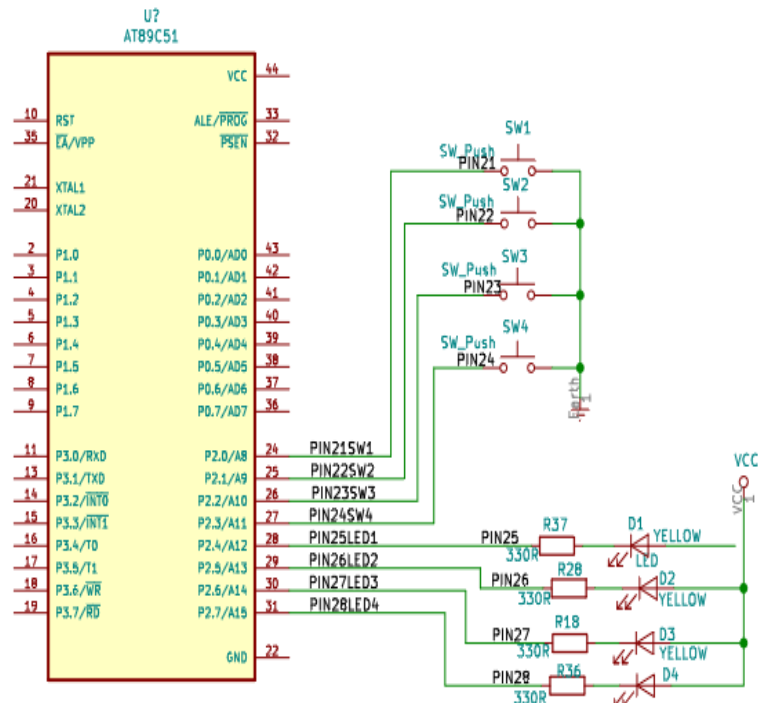
```

for(a=0;a<1000;a++)
{
    for(b=0;b<120;b++)
        ;
}
}


```



Blinking LED blink (two at a time) Schematic and Code



Code



```

#include<REG51.H>

//Fuction Prototypes
void delay(void);
sbit sw1=P2^0;

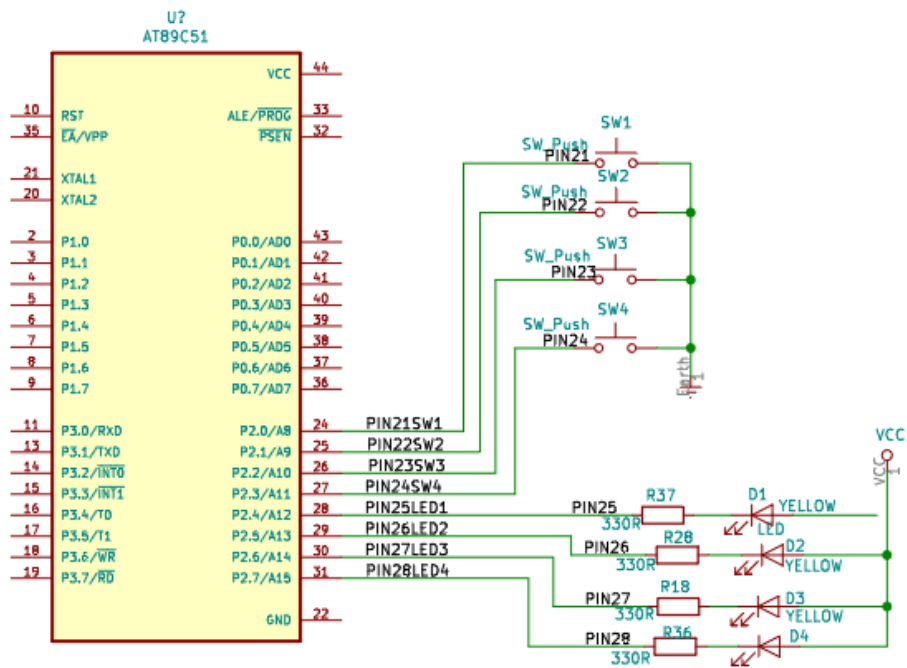
// Program Starts Here
void main()
{
    P2 = 0xFF; // makes the Port 1 as input port
    while(1)
    {
        if(sw1 == 0) //checked the button is pressed or not 0=pressed, 1= not pressed
        {
            P2=(P2 & 0x0f)|0x50;
            delay();
        }
        P2=(P2 & 0x0f)|0xaf;
    }
}

void delay(void)
{
    unsigned int a,b;
    for(a=0;a<100;a++)
    {
        for(b=0;b<120;b++)
        ;
    }
}

```




Blinking LED with Switch Schematic and Code



Code

```
#include<REG51.H>


//Fuction Prototypes

void delay(void);

// Program Starts Here

void main()
{
    P2 = 0xFF; // makes the Port 1 as input port

    while(1)
    {
        if((P2 & 0x0f) == 0x0e)
        {
            P2= (P2 & 0x0f)| 0xe0;
        }
    }
}
```



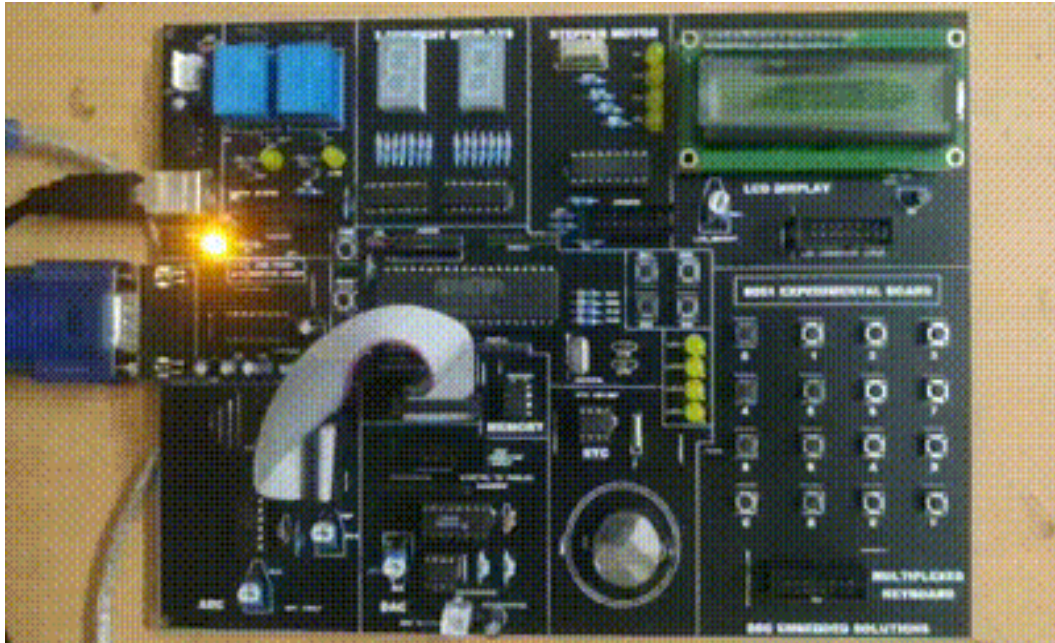
```
        delay();
    }
    else if ((P2 & 0x0f) == 0x0d)
    {
        P2=(P2 & 0x0f) | 0xd0;
        delay();
    }
    else if ((P2 & 0x0f) == 0x0b)
    {
        P2=(P2 & 0x0f) | 0xb0;
        delay();
    }
    else if ((P2 & 0x0f) == 0x07)
    {
        P2=(P2 & 0x0f) | 0x70;
        delay();
    }

    P2=0xff;
}

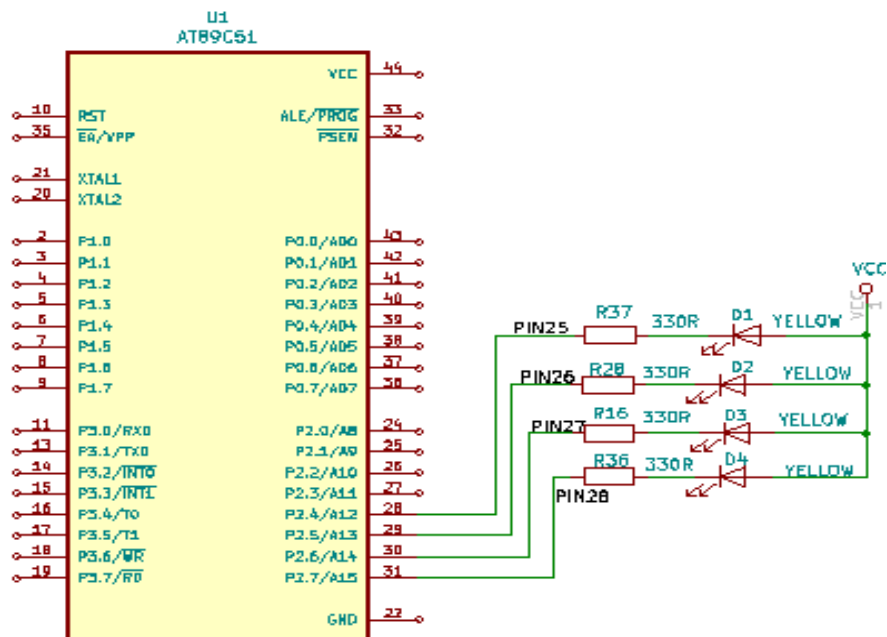
}

void delay(void)
{
    unsigned int a,b;
    for(a=0;a<10;a++)
    {
        for(b=0;b<120;b++);
    }
}
```

}



Blinking LED with Chaser and Code



Code

```
#include<REG51.H>
```

```
//Fuction Prototypes
```

```
void delay(void);
```

```
// Program Starts Here
```

```
void main()
```

```
{
```

```
    unsigned char a;
```

```
    a=0xef;
```

```
    while(1)
```

```
    {
```

```
        P2= a;
```

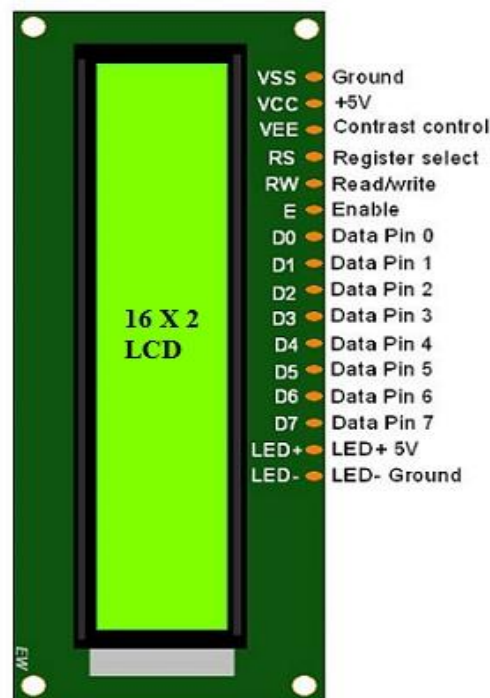


```
a = a<<1;
if (a==0xf0)
a=0xef;
delay();
}
}
void delay(void)
{
    unsigned int a,b;
    for(a=0;a<1000;a++)
    {
        for(b=0;b<50;b++)
        ;
    }
}
```



LCD Display


The term LCD stands for liquid crystal display. It is one kind of electronic display module used in an extensive range of applications like various circuits & devices like mobile phones, calculators, computers, TV sets, etc. These displays are mainly preferred for multi-segment light-emitting diodes and seven segments. The main benefits of using this module are inexpensive; simply programmable, animations, and there are no limitations for displaying custom characters, special and even animations, etc.



The features of this LCD mainly include the following.

- The operating voltage of this LCD is 4.7V-5.3V
- It includes two rows where each row can produce 16-characters.
- The utilization of current is 1mA with no backlight
- Every character can be built with a 5×8 pixel box
- The alphanumeric LCDs alphabets & numbers
- Is display can work on two modes like 4-bit & 8-bit

Display “Welcome have a nice day” Schematic and Code



```
#define lcddata P1;


// Program Starts Here

void main()
{

    light=0; //ON the lcd light
    lcd_cmd(0x38);
    delay(1);
    lcd_cmd(0x06);    // display move cursor to right
    delay(1);
    lcd_cmd(0x0E);    // LCD On, & cursor on
    delay(1);
    lcd_cmd(0x01);    // LCD Clear
    delay(1);
    lcd_cmd(0x80);    // LCD start 1st line
    delay(1);
    lcd_string("  WELCOME  ");
    delay(25);
    lcd_cmd(0xc0);    // LCD start 2nd line
    delay(1);
    lcd_string("HAVE A NICE DAY ");
    while(1);

}

void lcd_string(unsigned char *p)
```



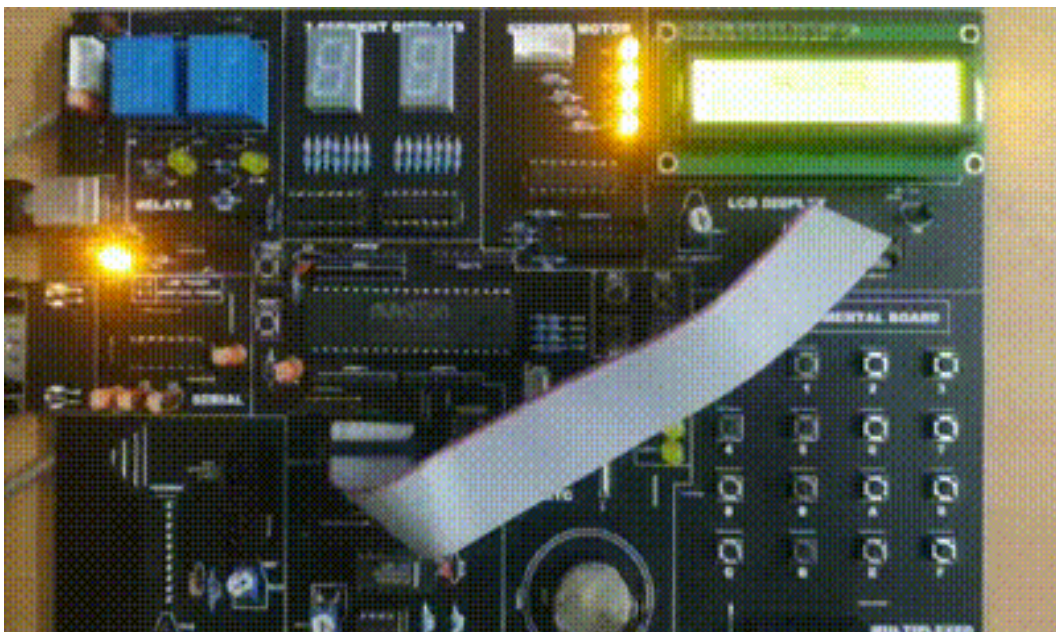
```
{  
    while(*p != '\0')  
    {  
        lcd_data(*p);  
        p++;  
        delay(10);  
    }  
}
```

```
void lcd_data(unsigned char x)  
{  
    RW=0;  
    RS=1;  
    P1 = x;  
    EN=1;  
    delay(1);  
    EN=0;  
}
```

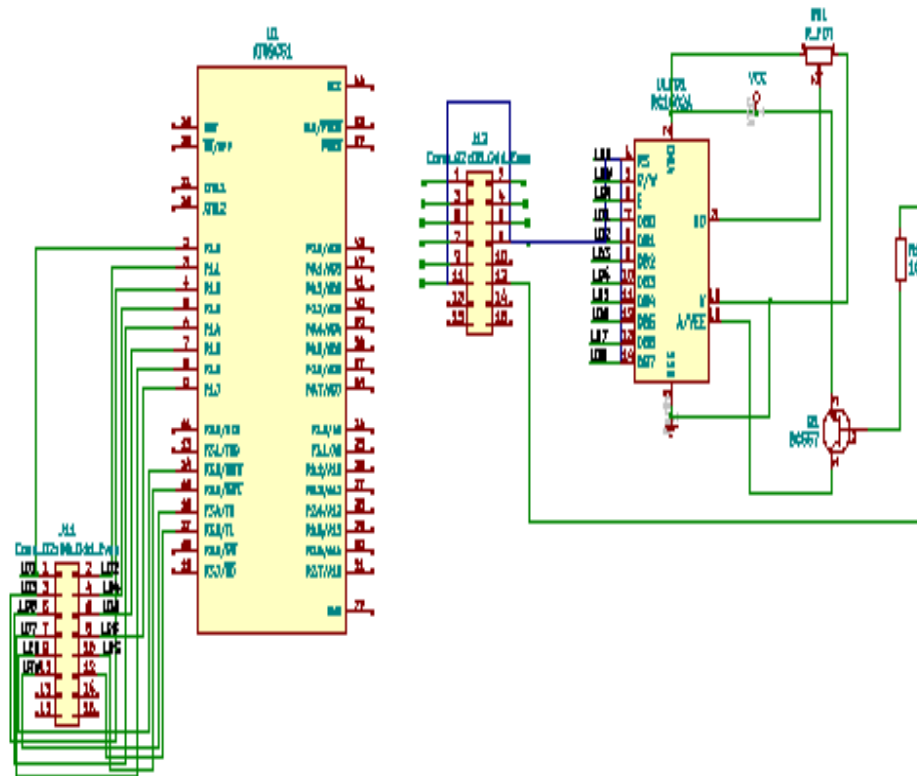
```
void lcd_cmd(unsigned char z)  
{  
    RW=0;  
    RS=0;  
    P1 = z;  
    EN =1;  
    delay(1);  
    EN=0;  
}
```



```
void delay(unsigned int tt)
{
    unsigned int a,b;
    for(a=0;a<tt;a++)
    {
        for(b=0;b<1275;b++)
        ;
    }
}
```



Display LCD Functions Schematic and Code




Code

```
#include<REG51.H>


//Fuction Prototypes
void delay(unsigned int);
void lcd_string(unsigned char *p);
void lcd_data(unsigned char );
void lcd_cmd(unsigned char );

sbit EN = P3^2;
sbit RS = P3^3;
sbit RW = P3^4;
sbit light = P3^5;
```




```
#define Lcddata P1;


// Program Starts Here
void main()
{
    light=0;
    lcd_cmd(0x38);    //LCD 4 bit Mode
    delay(1);
    lcd_cmd(0x06);    // display move cursor to right
    delay(1);
    lcd_cmd(0x0C);    // LCD On, & cursor OFF
    delay(1);
    lcd_cmd(0x01);    // LCD Clear
    delay(1);
    lcd_cmd(0x80);    // LCD start 1st line
    delay(1);
    lcd_string("  HELLO  ");
    delay(100);
    lcd_cmd(0xc0);    // LCD start 2nd line
    delay(1);
    lcd_string(" AND WELCOME ");
    delay(400);
    while(1)
    {
        lcd_cmd(0x01);    // LCD Clear
        delay(1);
```




```
lcd_string("LCD FUCTIONS ");
delay(400);
lcd_cmd(0x01);    // LCD Clear
delay(1);
lcd_string("FIRST LINE ");
delay(1);
delay(400);
lcd_cmd(0x01);    // LCD Clear
delay(1);
lcd_cmd(0xc0);    // LCD start 2nd line
delay(1);
lcd_string("SECOND LINE ");
delay(400);
lcd_cmd(0x01);    // LCD Clear
delay(1);
lcd_string("CURSOR ON ");
delay(1);
lcd_cmd(0xc0);    // LCD start 2nd line
lcd_cmd(0x0E);    // CURSOR ON
delay(500);
lcd_cmd(0x01);    // LCD Clear
delay(1);
lcd_string("CURSOR OFF ");
delay(1);
lcd_cmd(0xc0);    // LCD start 2nd line
lcd_cmd(0x0C);    // CURSOR OFF
delay(500);
```



```
lcd_cmd(0x01); // LCD Clear
delay(1);
lcd_string("CURSOR BLINK");
delay(1);
lcd_cmd(0xc0); // LCD start 2nd line
lcd_cmd(0x0F); // CURSOR BLINK
delay(500);
lcd_cmd(0x01); // LCD Clear
delay(1);
lcd_string("CURSOR RIGHT");
delay(1);
lcd_cmd(0xc0); // LCD start 2nd line
lcd_cmd(0x14); // CURSOR RIGHT
delay(100);
lcd_cmd(0x14); // CURSOR RIGHT
delay(100);
lcd_cmd(0x14); // CURSOR RIGHT
delay(100);
lcd_cmd(0x14); // CURSOR RIGHT
delay(100);
lcd_cmd(0x14); // CURSOR RIGHT
delay(100);
delay(500);
lcd_cmd(0x01); // LCD Clear
delay(1);
lcd_string("CURSOR LEFT");
delay(1);
```

```
lcd_cmd(0xc5);    // LCD start 2nd line
lcd_cmd(0x10);    // CURSOR LEFT
delay(100);
lcd_cmd(0x10);    // CURSOR LEFT
delay(100);
lcd_cmd(0x10);    // CURSOR LEFT
delay(100);
lcd_cmd(0x10);    // CURSOR LEFT
delay(100);
lcd_cmd(0x10);    // CURSOR LEFT
delay(100);
delay(500);
lcd_cmd(0x01);    // LCD Clear
lcd_cmd(0x0C);    // CURSOR OFF
delay(1);
lcd_string("LCD OFF");
delay(200);
light=1;
lcd_cmd(0x0A);    // LCD OFF
delay(500);
lcd_cmd(0x01);    // LCD Clear
delay(1);
lcd_string("LCD ON");
delay(1);
light=0;
lcd_cmd(0x0C);    // LCD ON
delay(500);
```



```
    }  
  
}  
  
void lcd_string(unsigned char *p)  
{  
    while(*p != '\0')  
    {  
        lcd_data(*p);  
        p++;  
    }  
}
```

```
void lcd_data(unsigned char x)  
{  
    RW=0;  
    RS=1;  
    P1 = x;  
    EN=1;  
    delay(1);  
    EN=0;  
}
```

```
void lcd_cmd(unsigned char z)  
{  
    RW=0;  
    RS=0;
```

```
P1 = z;  
EN =1;  
delay(1);  
EN=0;  
}
```

```
void delay(unsigned int tt)  
{  
    unsigned int a,b;  
    for(a=0;a<tt;a++)  
    {  
        for(b=0;b<1000;b++)  
        ;  
    }  
}
```



RS232 Serial Communication

UART (RS232)

UART (Universal Asynchronous Receiver Transmitter) are one of the basic interfaces which provide a cost effective simple and reliable communication between one controller to another controller or between a controller and PC.

Usually all the digital ICs work on TTL or CMOS voltage levels which cannot be used to communicate over RS-232 protocol. So a voltage or level converter is needed which can convert TTL to RS232 and RS232 to TTL voltage levels. The most commonly used RS-232 level converter is MAX232.

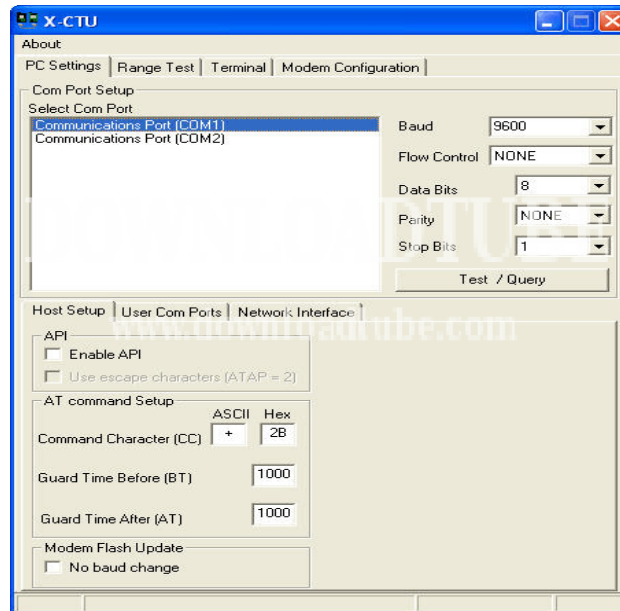


This IC includes charge pump which can generate RS232 voltage levels (-10V and +10V) from 5V power supply. It also includes two receiver and two transmitters and is capable of full-duplex UART/USART communication.

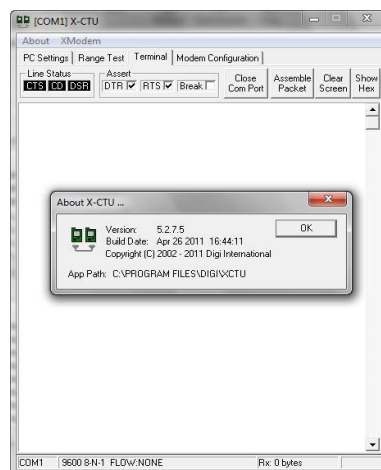
RS-232 communication enables point-to-point data transfer. It is commonly used in data acquisition applications, for the transfer of data between the microcontroller and a PC. The voltage levels of a microcontroller and PC are not directly compatible with those of RS-232, a level transition buffer such as MAX232 be used.

X-CTU

To see above output open XCTU, you will get following window.

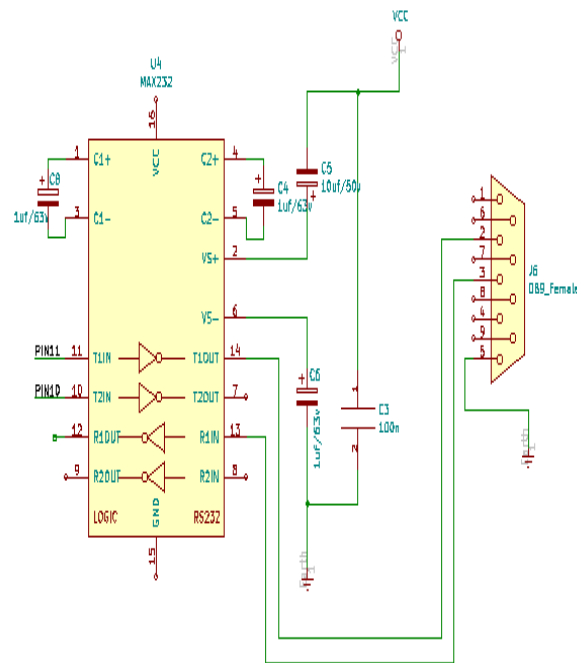


- In XCTU first select Communication Port (COM5).
- Then click on Terminal, you will get following window.



- Then click on Open Com Port.
- See the output as "This is Serial Test".
- Then Close Com Port.
- Click on Clear Screen.

Display on X-CTU “This is serial tet” Schematic and Code



Code


```
#include<REG51.H>

#include<stdio.h>

//Fuction Prototypes
void delay(unsigned int);
void serial_init(void);
```

```
// Program Starts Here
```

```
void main()
{
    unsigned char a=0x0a;
    serial_init();
    while(1)
```



```

{
    printf("This is Serial Test \n");
    delay(100);
}
}

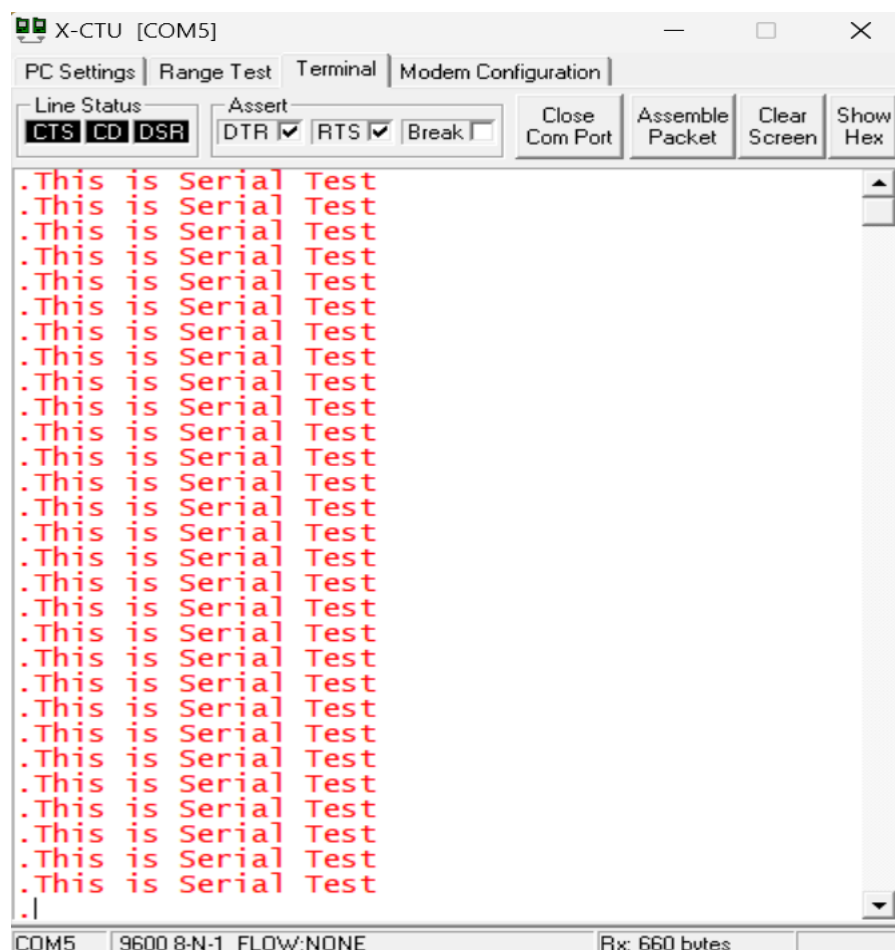
void serial_init()
{
    SCON = 0x50; // Setup serial port control register Mode 1:
                // 8-bit uart var baud rate REN: enable receiver

    TMOD |= 0x20; // Set M1 for 8-bit autoreload timer
    TH1 = 0xFD;  // Set autoreload value for timer1 9600 baud
                // with 11.0592 MHz xtal

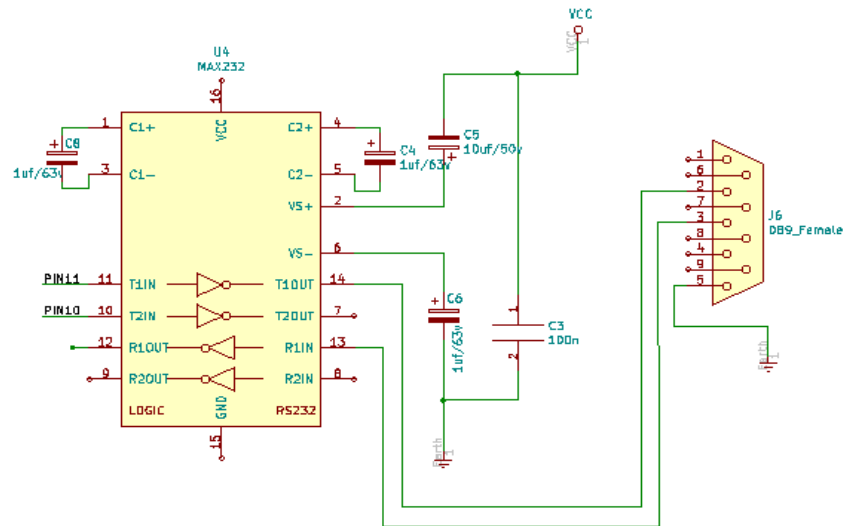
    TR1 = 1;    // Start timer 1
    TI = 1;     // Set TI to indicate ready to xmit
}

void delay(unsigned int tt)
{
    unsigned int a,b;
    for(a=0;a<tt;a++)
    {
        for(b=0;b<1000;b++)
        ;
    }
}

```



Display on X-CTU “Sum of two number” Schematic and Code



Code

```
#include<REG51.H>

#include<stdio.h>

//Fuction Prototypes
void serial_init(void);

// Program Starts Here
void main()
{
    unsigned char a,b,c,tmp;

    serial_init();

    while(1)
    {
        printf("***** \n");
```

```

printf("FIND THE SUM OF TWO NUMBERS \n");
printf("***** \n");
printf("\n\n");
printf("Enter the First No :- ");
while(!(a=_getkey()));
putchar(a);
printf("\n");
printf("Enter the Second No :- ");
while(!(b=_getkey()));
putchar(b);
printf("\n\n");
printf(" The Answer is   :- ");
c=(a & 0x0f)+(b & 0x0f);
tmp = c/10;
tmp |= 0x30;
printf("%c",tmp);
tmp = c%10;
tmp |= 0x30;
printf("%c",tmp);
printf("\n\n\n");
}

}

void serial_init()
{
    SCON = 0x50; // Setup serial port control register Mode 1:
                // 8-bit uart var baud rate REN: enable receiver
    TMOD |= 0x20; // Set M1 for 8-bit autoreload timer

```

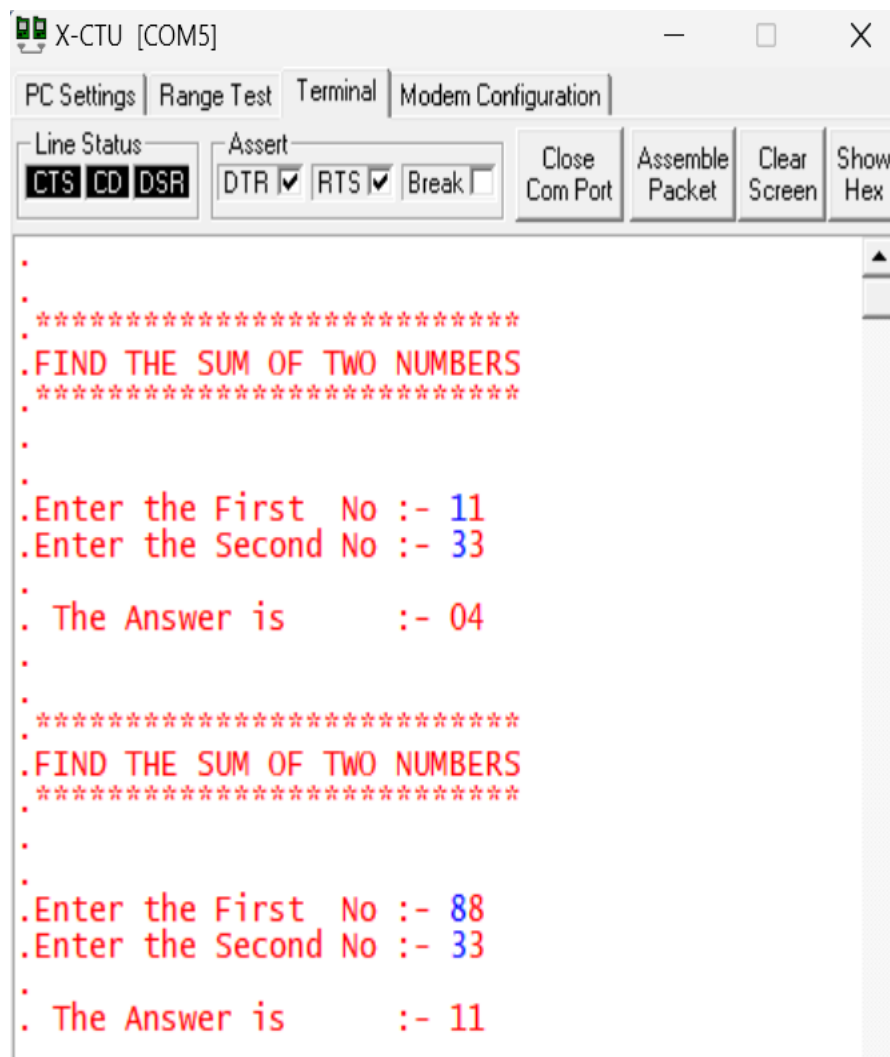


```
TH1 = 0xFD;    // Set autoreload value for timer1 9600 baud
               // with 11.0592 MHz xtal

TR1 = 1;       // Start timer 1

TI = 1;        // Set TI to indicate ready to xmit

}
```

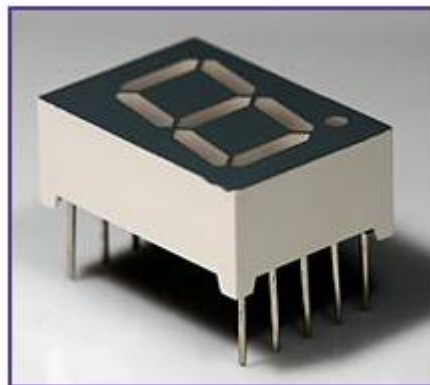
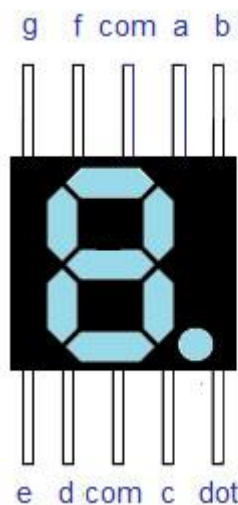


The screenshot shows the X-CTU [COM5] terminal window. The window has a menu bar with 'PC Settings', 'Range Test', 'Terminal', and 'Modem Configuration'. Below the menu bar, there are two sections: 'Line Status' and 'Assert'. The 'Line Status' section shows 'CTS', 'CD', and 'DSR' as active. The 'Assert' section shows 'DTR' and 'RTS' as checked, and 'Break' as unchecked. To the right of these sections are four buttons: 'Close Com Port', 'Assemble Packet', 'Clear Screen', and 'Show Hex'. The main terminal area displays the following text in red:

```
.
.
. *****
. FIND THE SUM OF TWO NUMBERS
. *****
.
.
. Enter the First No :- 11
. Enter the Second No :- 33
.
. The Answer is      :- 04
.
.
. *****
. FIND THE SUM OF TWO NUMBERS
. *****
.
.
. Enter the First No :- 88
. Enter the Second No :- 33
.
. The Answer is      :- 11
```

Seven Segment

Seven segment displays are important display units in Electronics and widely used to display numbers from 0 to 9. It can also display some character alphabets like A,B,C,H,F,E etc. It's the simplest unit to display numbers and characters. It just consists 8 LEDs, each LED used to illuminate one segment of unit and the 8th LED used to illuminate DOT in 7 segment display. We can refer each segment as a LINE, as we can see there are 7 lines in the unit, which are used to display a number/character. We can refer each line/segment "a,b,c,d,e,f,g" and for dot character we will use "h". There are 10 pins, in which 8 pins are used to refer a,b,c,d,e,f,g and h/dp, the two middle pins are common anode/cathode of all the LEDs. These common anode/cathode are internally shorted so we need to connect only one COM pin.

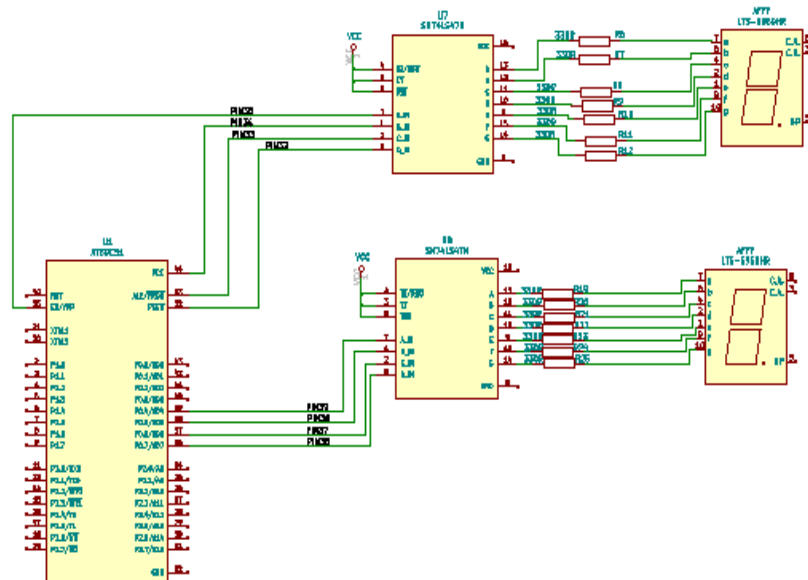


There are two types of 7 segment displays: Common Anode and Common Cathode:

Common Anode: In this all the Negative terminals (cathode) of all the 8 LEDs are connected together (see diagram below), named as COM. And all the positive terminals are left alone.

Common Cathode: In this all the positive terminals (Anodes) of all the 8 LEDs are connected together, named as COM. And all the negative terminals are left alone.

Display Rolling Schematic and Code



Code

/* A seven segment display is connected through a 7447 ic which is bcd to 7 segment driver IC. The BCD inputs are connected to port3.4 to port3.7 of the 8051. the table for 1-9 is:

BCD	7 SEGMENT OUTPUT
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8

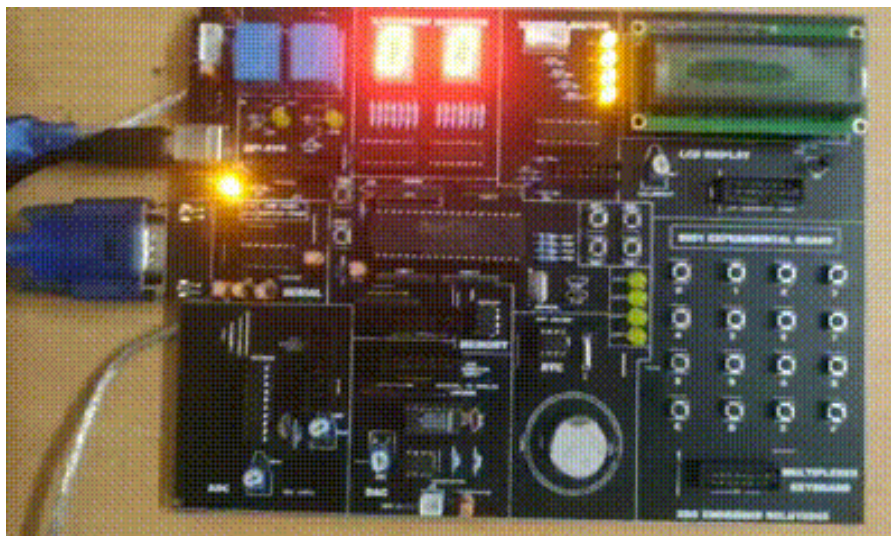
```
#include<REG51.H>

//Fuction Prototypes
void delay(void);

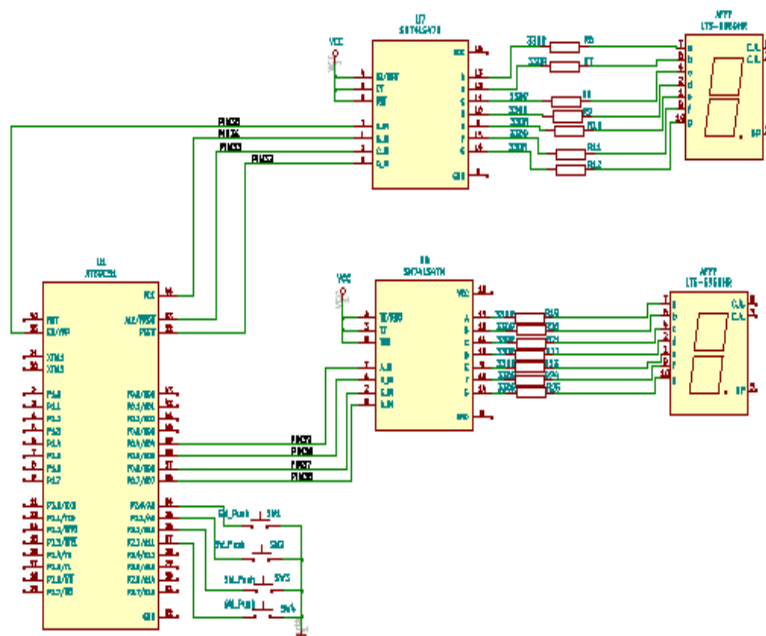
// Program Starts Here
void main()
{
    unsigned char a,b;
    while(1)
    {
        for(a=0;a<30;a++)
        {
            b=a>>4;
            b |= a<<4;
            P0 =b ;
            delay();
        }
    }
}
```



```
void delay(void)
{
    unsigned int a,b;
    for(a=0;a<1000;a++)
    {
        for(b=0;b<120;b++);
    }
}
```



Display with Switch Schematic and Code



Code

/* A seven segment display is connected through a 7447 ic which is bcd to 7 segment driver IC. The BCD inputs are connected to port3.4 to port3.7 of the 8051. the table for 1-9 is:

BCD	7 SEGMENT OUTPUT
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9




When each button is pressed the display
shows the no of pressed button*/

```
#include<REG51.H>

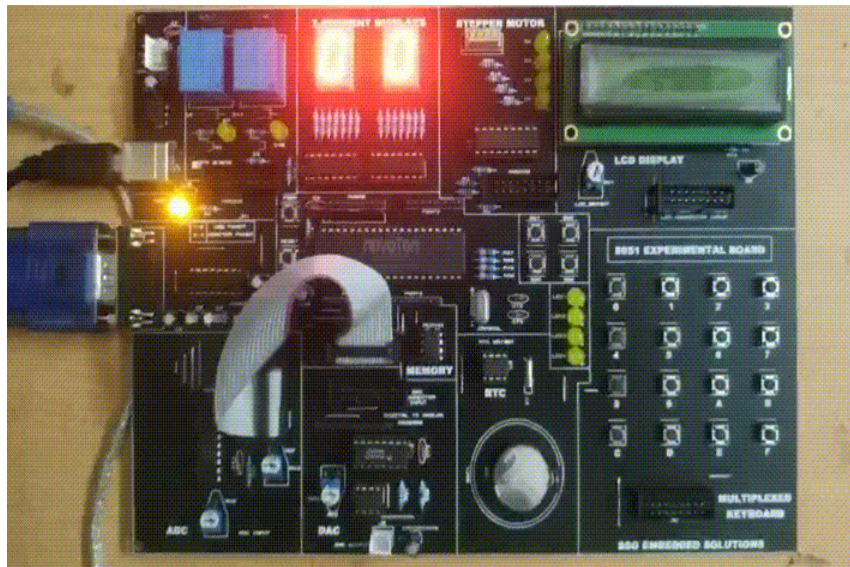
//Fuction Prototypes
void delay(void);

// Program Starts Here
void main()
{
    P2=0xff;
    P0=0x00;
    while(1)
    {
        if((P2 & 0x0f) == 0x0e)
        {
            P0= 0x10;
            delay();
        }
        else if ((P2 & 0x0f) == 0x0d)
        {
            P0=0x20;
            delay();
        }
        else if ((P2 & 0x0f) == 0x0b)
        {
            P0=0x30;
            delay();
        }
    }
}
```

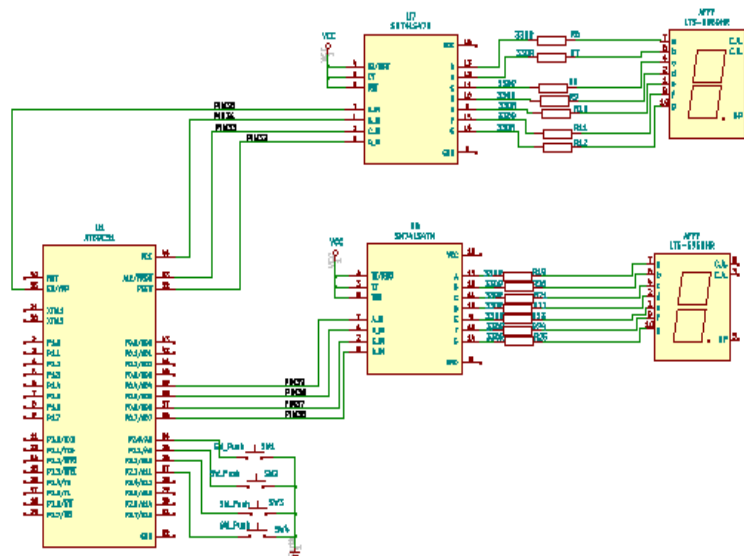


```
        else if ((P2 & 0x0f) == 0x07)
        {
            P0=0x40;
            delay();
        }
        else
            P0=0x00;
    }
}
```

```
void delay(void)
{
    unsigned int a,b;
    for(a=0;a<10;a++)
    {
        for(b=0;b<120;b++)
            ;
    }
}
```



Display Up & Down Schematic and Code



Code

/* A seven segment display is connected through a 7447 ic which is bcd to 7 segment driver IC. The BCD inputs are connected to port3.4 to port3.7 of the 8051. the table for 1-9 is

BCD	7 SEGMENT OUTPUT
0000	0
0001	1



0010		2
0011		3
0100		4
0101		5
0110		6
0111		7
1000		8
1001		9

WHEN SW1 IS PRESSED THE COUNTER IN INCREMENTED

WHEN SW2 IS PRESED THE COUNTER IS DECREMENTED */

```
#include<REG51.H>
```

```
//Fuction Prototypes
```

```
void delay(void);
```

```
sbit sw1=P2^0;
```

```
sbit sw2=P2^1;
```

```
void delayms(unsigned int tt);
```

```
// Program Starts Here
```

```
void main()
```

```
{
```

```
    unsigned char a,b;
```

```
    P2=0xff;
```

```
    a=0;
```

```
    while(1)
```

```
    {
```

```
        if(sw1 == 0)
```

```
        {
```

```
            delayms(20);
```



```

        while(sw1==0);

        a++;

        b=a>>4;

        b |= a<<4;

    }

    else if (sw2 == 0)
    {

        delayms(20);

        while(sw2==0);

        if(a>0)

            a--;

        b=a>>4;

        b |= a<<4;

    }

    P0 = b ;

}

void delayms(unsigned int tt)
{

    unsigned int a,b;

    for(a=0;a<tt;a++)

        {

            for(b=0;b<1275;b++)

                ;

        }

}

```



ADC

ADC is the Analog to Digital converter, which converts analog data into digital format; usually it is used to convert **analog voltage** into digital format. Analog signal has infinite no of values like a sine wave or our speech, ADC converts them into particular levels or states, which can be measured in numbers as a physical quantity.

When we select **8051 microcontroller** family for making any project, in which we need of an ADC conversion, then we use **external ADC**. Some external ADC chips are 0803,0804,0808,0809 and there are many more.

ADC0808 is a commonly used External 8 bit ADC and it has 28 pins. It can measure up to eight ADC values from 0 to 5 volt since it has eight channels. when voltage reference is +5V, its Step size will be 19.53mV. That is, for every increase of 19.53mV on the input side there will be an increase of 1 bit at the output side.

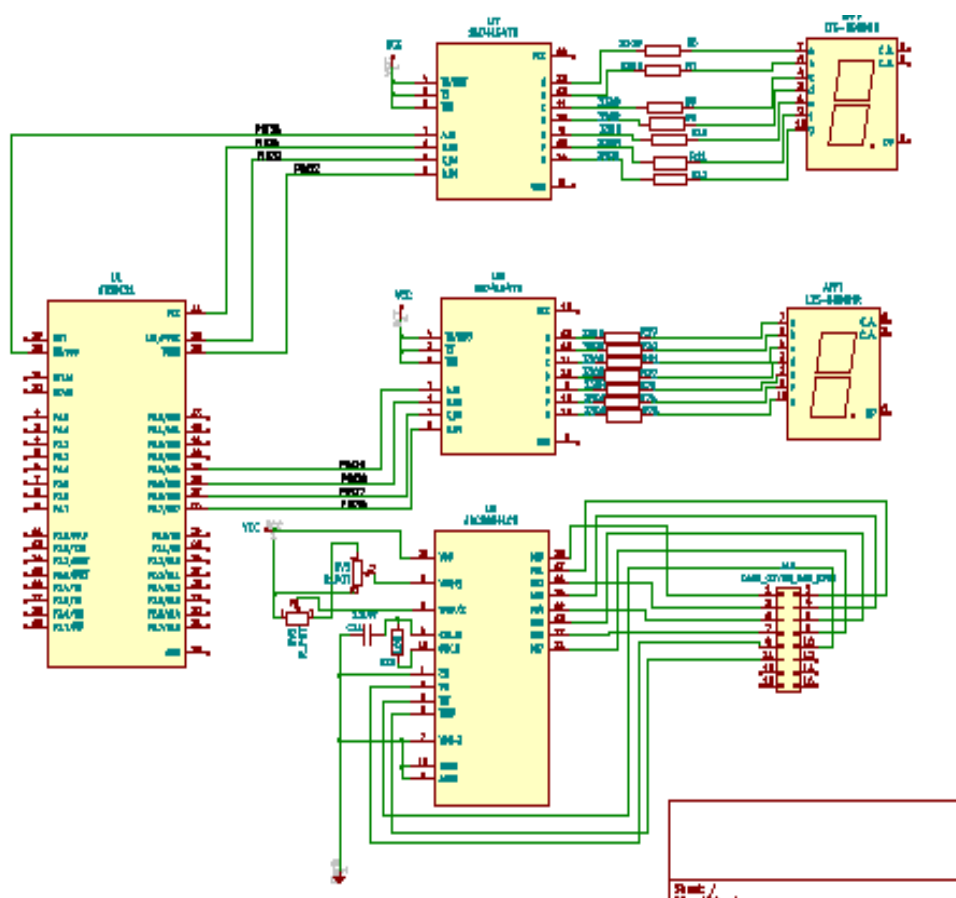
ADC0808 needs an external clock to operate. The ADC needs some specific control signals for its operations like start conversion and bring data to output pins. When the conversion is complete the EOC pins go low to indicate the end of a conversion and that the data is ready to be picked up.

Features

- Easy interface to all microprocessors
- Operates ratio metrically or with 5 V DC or analog span adjusted voltage reference
- No zero or full-scale adjust required

-

ADC Schematic and Code



Code

```
#include<reg51.h>

#define input P1 //Input port (read values of ADC)
#define output P2 // Output port (connected to LED's)

void delay(unsigned int msec);

void adc();


sbit wr = P3 ^ 6; // Write pin.
sbit rd = P3 ^ 7; // Read pin.
sbit intr = P3 ^ 4; // Interrupt pin.

void main()
{
    input = 0xff; // Declare port 1 as input port.
    output = 0x00; // Declare port 0 as output port.

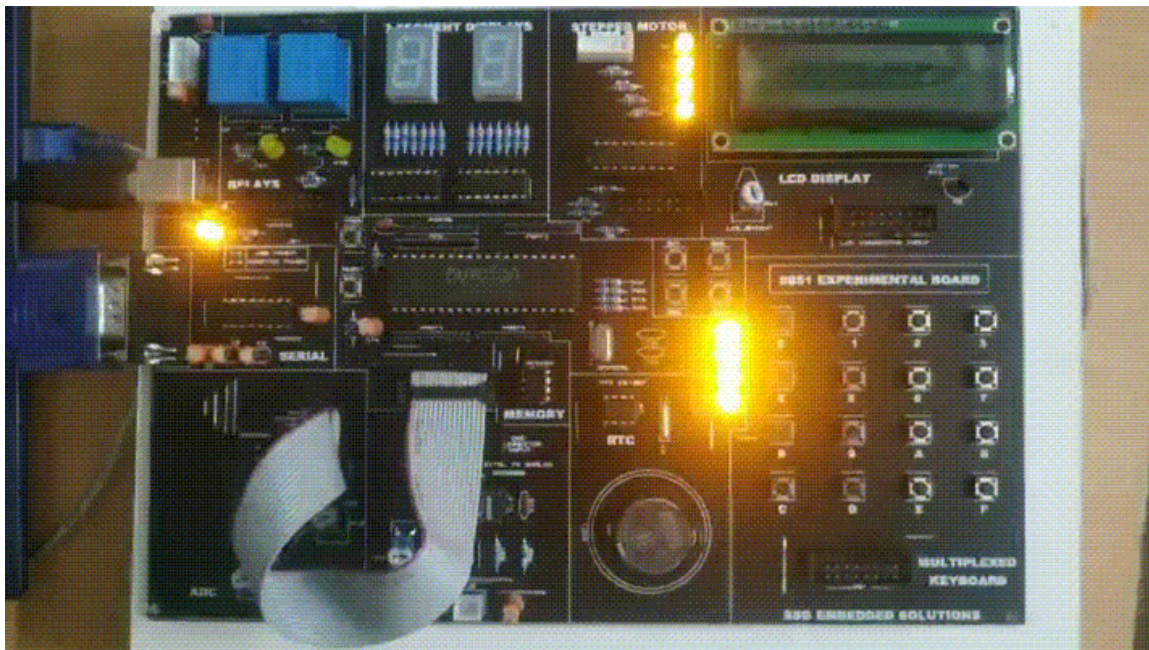
    while (1)
    {
        adc();
    }
}

void delay(unsigned int msec) // Delay function
{
    int i, j;

    for (i = 0; i < msec; i++){
```




```
for (j = 0; j < 1275; j++){  
  
    }  
}  
}  
  
voidadc() // Reading values from ADC and display on the LED's  
{  
    rd = 1;  
    wr = 0;  
  
    delay(10);  
  
    wr = 1;  
  
    while (intr == 1);  
  
    rd = 0;  
  
    output = input;  
  
    delay(10);  
  
    intr = 1;  
}
```

DAC

Digital-to-analog conversion is a process in which signals having a few (usually two) defined levels or states are converted into signals having a theoretically infinite number of states. A common example is the processing of computer data into audio-frequency (AF) tones that can be transmitted over a twisted pair by a modem of computer data telephone line. The circuit that performs this function is a digital-to-analog converter (DAC). Basically, digital-to-analog conversion is the opposite of analog-to-digital conversion. In most cases, if an analog-to-digital converter (ADC) is placed in a communications circuit after a DAC, the digital signal output is identical to the digital signal input. Also, in most instances when a DAC is placed after an ADC, the analog signal output is identical to the analog signal input.

DAC Schematic and Code



```

P3 = 0x00;

InitDAC();                                // Initialize DAC0808 data bus


while(1)
{
    output    Generate_DAC_Voltage(1000);    // Generate 1000mV = 1v at
    delay_sec(2);                            // Two second delay
    output    Generate_DAC_Voltage(2000);    // Generate 2000mV = 2v at
    delay_sec(2);                            // Two second delay
    output    Generate_DAC_Voltage(3000);    // Generate 3000mV = 3v at
    delay_sec(2);                            // Two second delay
}
}

// Function Purpose: Produce approximate delay in Secs.
void delay_sec(unsigned int d)
{
    unsigned int i;

    for(i=0;i<(d*20);i++)
        __delay_us(50000);
}

// Function Purpose: Produce approximate delay in given uSecs.
void __delay_us(unsigned int d)
{
    unsigned int i, limit;

```



```
limit = d/15;
```

```
for(i=0;i<limit;i++);  
}
```

“Includes.h”

```
`#include "Includes.h"
```

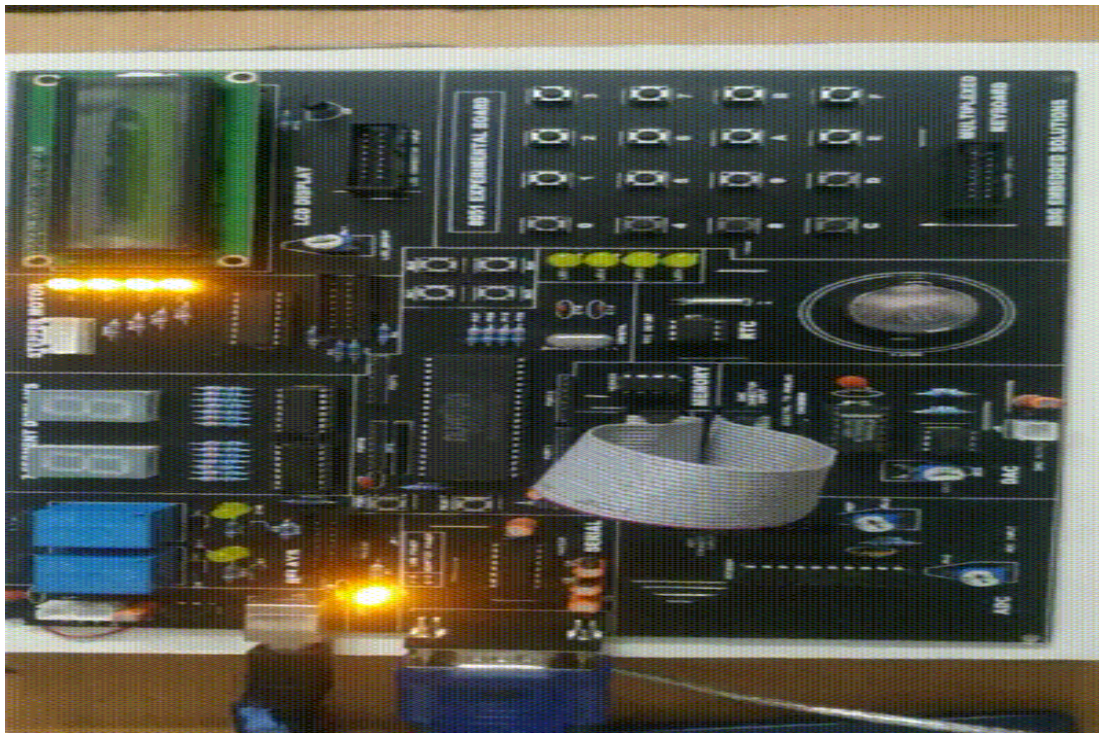
```
void InitDAC(void)
```

```
{  
    DAC_Data_Bus = 0x00; // Make Outputs  
}
```

```
void Generate_DAC_Voltage(unsigned int mV) // Input should be in mV
```

```
{  
    unsigned long V = ((unsigned long)mV * 25)/VREF; // Scale the input value  
    V = V/98; //  
    Conversion factor
```

```
    DAC_Data_Bus = (unsigned char)V; // Assign proper value to  
    DAC inputs  
}
```

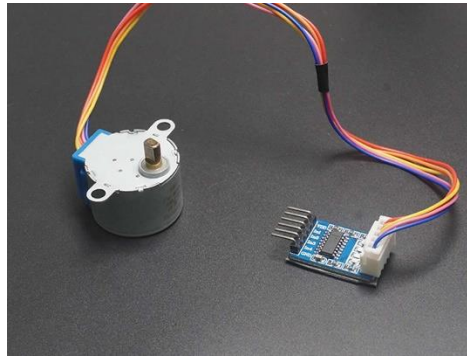



Stepper Motor

Stepper Motor is a brushless DC Motor. Control signals are applied to stepper motor to rotate it in steps.

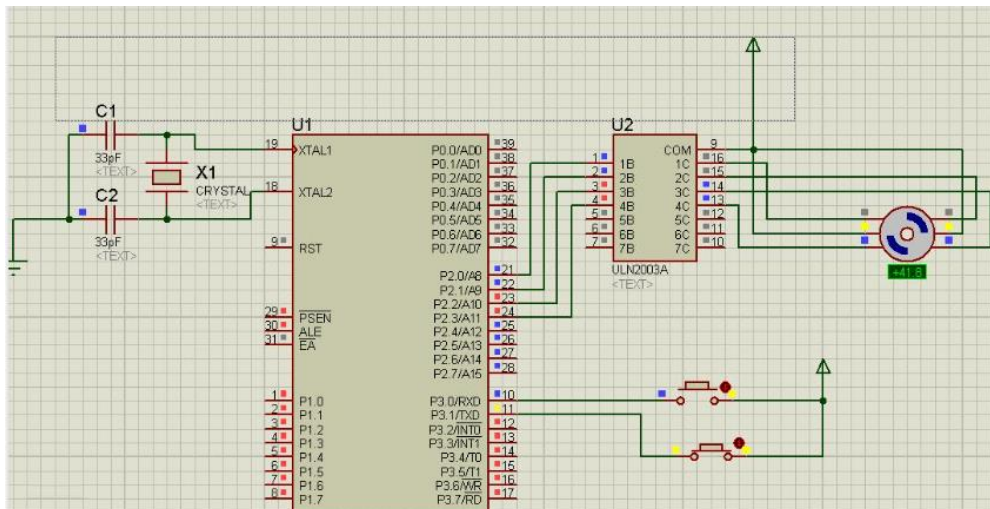
Its speed of rotation depends upon rate at which control signals are applied. There are various stepper motors available with minimum required step angle.

Stepper motor is made up of mainly two parts, a stator and rotor. Stator is of coil winding and rotor is mostly permanent magnet or ferromagnetic material.



Step angle is the minimum angle that stepper motor will cover within one move/step. Number of steps required to complete one rotation depends upon step angle. Depending upon stepper motor configuration, step angle varies e.g. 0.72° , 1.8° , 3.75° , 7.5° , 15° etc.

Stepper Motor Schematic and Code



Code

```
#include<reg51.h>


sbit stepper1=P1^0;

sbit stepper2=P1^1;

sbit stepper3=P1^2;

sbit stepper4=P1^3;

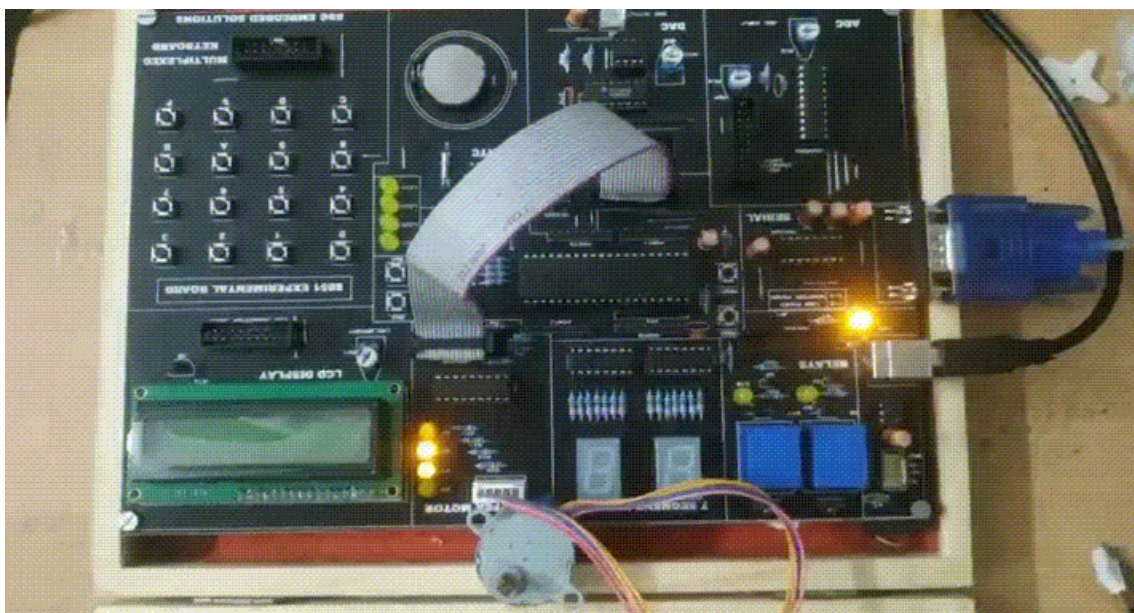
void delay(unsigned int count) {
```



```
    unsigned int i, j;  
    for (i = 0; i < count; i++)  
        for (j = 0; j < 500; j++);  
}
```

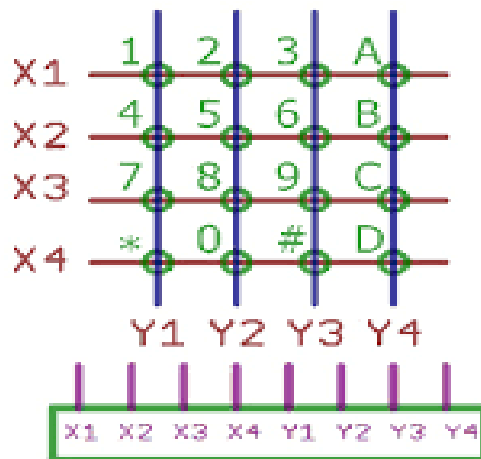
```
void main()  
{  
    delay(50);  
    while(1)  
    {  
  
        stepper1 = 1;  
        stepper2 = 0;  
        stepper3 = 0;  
        stepper4 = 0;  
        delay(5);  
        stepper1 = 0;  
        stepper2 = 1;  
        stepper3 = 0;  
        stepper4 = 0;  
        delay(5);  
        stepper1 = 0;  
        stepper2 = 0;  
        stepper3 = 1;  
        stepper4 = 0;  
        delay(5);  
        stepper1 = 0;  
        stepper2 = 0;
```

```
stepper3 = 0;
stepper4 = 1;
    delay(5);
}
}
```



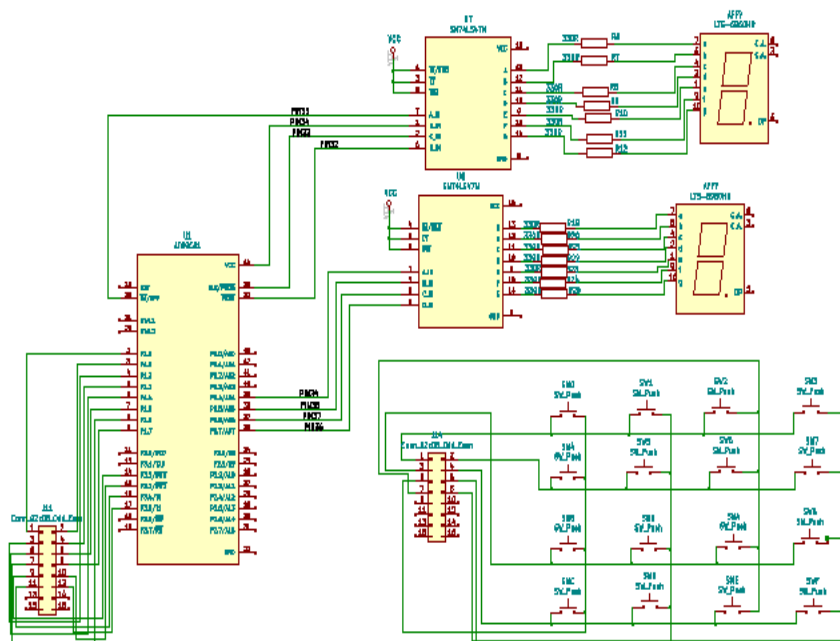
Multiplex Keyboard

Matrix keypads are very common input devices in embedded systems. They have simple architecture and are easy to interface. One good thing about them is that they allow you to interface a large number of input keys to a microcontroller with minimum usage of I/O resources.



The big advantage of using a matrix keypad is that it allows to interface a large number of keys with a relatively small number of microcontroller pins.

Multiplex Keyboard Schematic and Code




Code

```
#include <at89c51xd2.h>

void show(unsigned char);


void delay(void);
```


```
sbit col1= P1^7;
sbit col2= P1^6;
sbit col3= P1^5;
sbit col4= P1^4;
sbit row1= P1^0;
sbit row2= P1^1;
sbit row3= P1^2;
sbit row4= P1^3;
unsigned char code keytable[4][4]={0 ,1 ,2 ,3 ,
                                     4 ,5 ,6 ,7 ,
                                     8 ,9 ,0x10,0x11,
                                     0x12,0x13,0x14,0x15
                                     };
```

```
void main(void)
{
    unsigned char col,rowno;
    P1=0xf0;    //make cols high and rows low

    while(1)
    {
        do
        {
            P1=0xf0;
            col=P1;
            col &=0xf0;
        }while(col != 0xf0);
        do
        {
```

```
do
{
col=P1;
col &=0xf0;
}while(col == 0xf0);
delay();
col=P1;
col &=0xf0;
}while(col == 0xf0);
P1 |= 0xf0;
while(1)
{
row1=0;row2=1;row3=1;row4=1;
col=P1;
col &=0xf0;
if(col != 0xf0)
{
rowno=0;
break;
}
row1=1;row2=0;row3=1;row4=1;
col=P1;
col &=0xf0;
if(col != 0xf0)
{
rowno=1;
break;
}
```



```
    row1=1;row2=1;row3=0;row4=1;
    col=P1;
    col &=0xf0;
    if(col != 0xf0)
    {
        rowno=2;
        break;
    }
    row1=1;row2=1;row3=1;row4=0;
    col=P1;
    col &=0xf0;
    if(col != 0xf0)
    {
        rowno=3;
        break;
    }
}
if(col == 0xe0)
show(keytable[rowno][0]);
else if(col == 0xd0)
show(keytable[rowno][1]);
else if(col == 0xb0)
show(keytable[rowno][2]);
else if(col == 0x70)
show(keytable[rowno][3]);
}

}
```

```
void show(unsigned char key)
```

```
{
```

```
    unsigned char b;
```

```
    b= key >> 4;
```

```
    b |= key << 4;
```

```
    P0=b;
```

```
}
```

```
void delay(void)
```

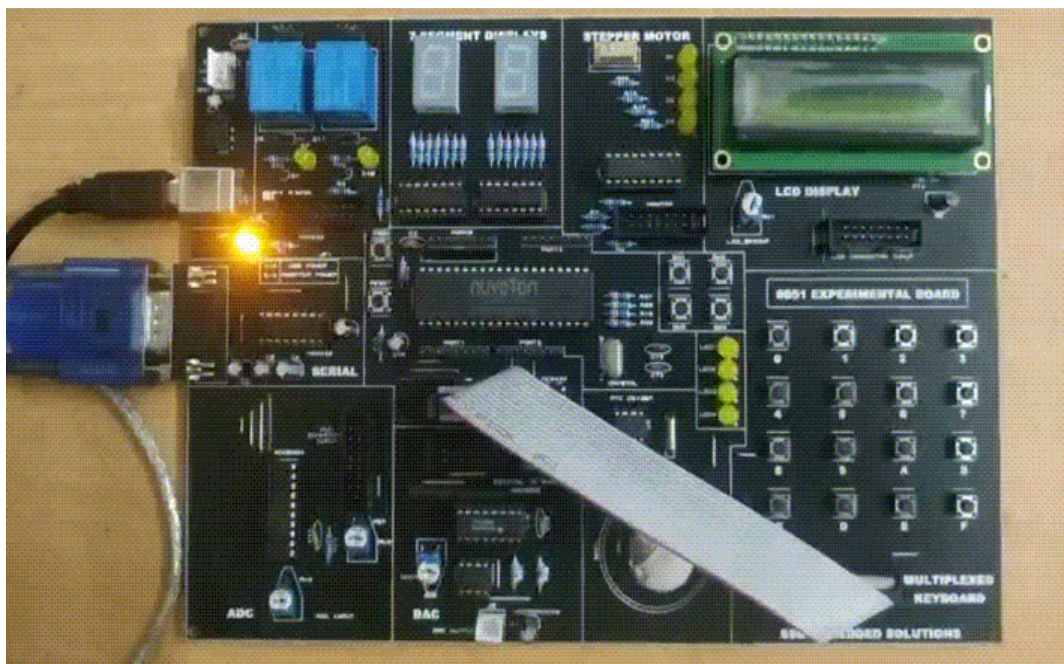
```
{
```

```
    unsigned char a,b;
```

```
    for(a=0;a<20;a++)
```

```
        for(b=0;b<120;b++);
```

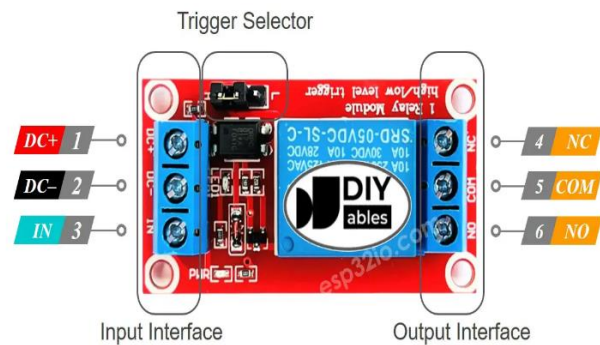
```
}
```



Relay

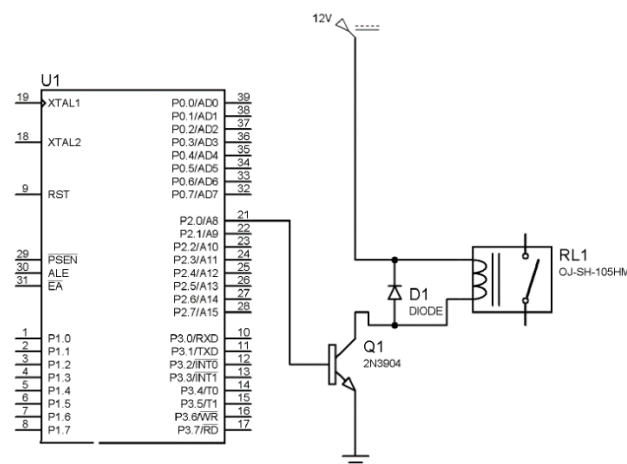
Relays are the switches that aim at closing and opening the circuits electronically as well as electromechanically. It controls the opening and closing of the circuit contacts

of an electronic circuit. When the relay contact is open (NO), the relay isn't energized with the open contact. However, if it is closed (NC), the relay isn't energized given the closed contact. However, when energy (electricity or charge) is supplied, the states are prone to change.



Relays are normally used in the control panels, manufacturing, and building automation to control the power along with switching the smaller current values in a control circuit. However, the supply of amplifying effect can help control the large amperes and voltages because if low voltage is applied to the relay coil, a large voltage can be switched by the contacts.

Relay Schematic and Code



Code



```
#include<REG51.H>
```

```
//Fuction Prototypes
```

```
void delay(void);
```

```
// Program Starts Here
```

```
void main()
```

```
{
```

```
    P1 = 0x00; // Leds are ON
```

```
    delay();
```

```
    P1 = 0xff; // Leds are OFF
```

```
    delay();
```

```
}
```

```
void delay(void)
```

```
{
```

```
    unsigned int a,b;
```

```
    for(a=0;a<1000;a++)
```

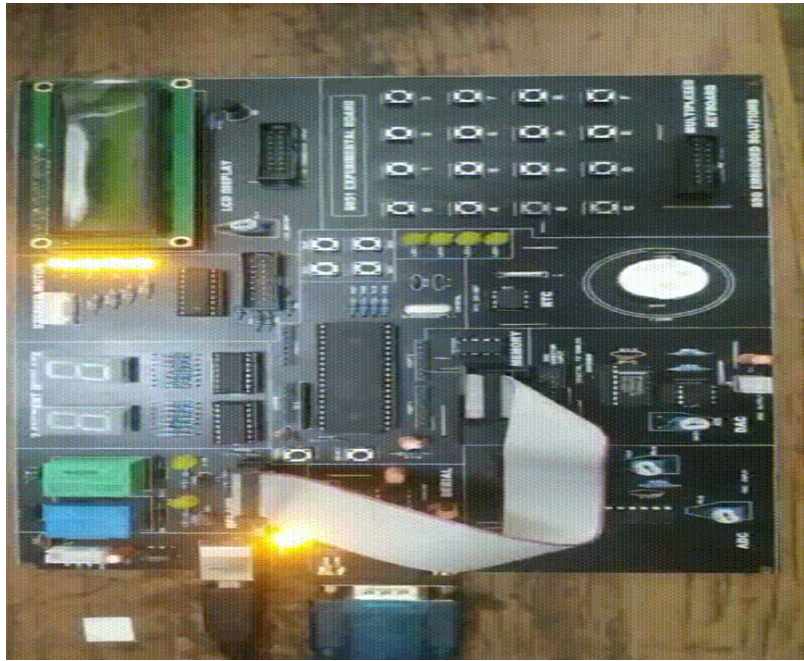
```
    {
```

```
        for(b=0;b<120;b++)
```

```
        ;
```

```
    }
```

```
}
```

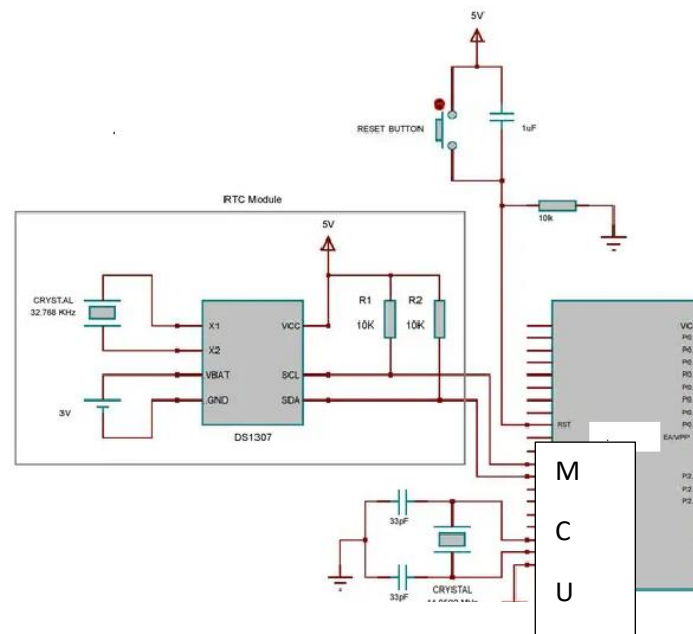
RTC

The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. This chip works on I²C protocol. 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 fewer 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 that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply. DS1307 chip can continuously run till 10 year.

RTC Schematic and Code



Code

/*The ds1307 is a Real time clock it has a small 3v battery so the clock setting remains while power off it works on I2C protocol*/

```
#include<reg52.h>
```

```
#include<stdio.h>
```

```
#include<intrins.h>
```

```
void InitSerial(void);
```

```
void DelayMs(unsigned int count);
```

```
void ReadRTC(unsigned char * buff);
```

```
void WriteRTC(unsigned char * buff);
```

```
char * Int2Day(unsigned char day);
```

```
char * Int2Month(unsigned char month);
```

```
#define ACK          1
```

```
#define NO_ACK       0
```

```
#define SLAVE 0xD0
```



```
#define WRITE 0x00
```

```
#define READ 0x01
```

```
#define ERR_ACK 0x01
```

```
unsigned char RTC_ARR[7]; // Buffer for second,minute,.....,year
```

```
unsigned char p;
```

```
unsigned char i;
```

```
const unsigned char * DayStr[7] = {"Sun",
```

```
    {"Mon"},
```

```
    {"Tue"},
```

```
    {"Wed"},
```

```
    {"Thu"},
```

```
    {"Fri"},
```

```
    {"Sat"}};
```

```
const unsigned char * MonthStr[12] = {"Jan",
```

```
    {"Feb"},
```

```
    {"Mar"},
```

```
    {"Apr"},
```

```
    {"May"},
```

```
    {"Jun"},
```

```
    {"Jul"},
```

```
    {"Aug"},
```

```
    {"Sep"},
```

```
    {"Oct"},
```

```
    {"Nov"},
```



```
 {"Dec"}};
```

```
sbit SDA = P3^6;    // connect to SDA pin (Data)
sbit SCL = P3^7;    // connect to SCL pin (Clock)

//-----
// Main program
//-----

void main(void)
{


    InitSerial();        // Initialize serial port
    ReadRTC(&RTC_ARR[0]);

    //  RTC_ARR[0] = RTC_ARR[0] & 0x7F;  // enable oscillator (bit 7=0)
    //RTC_ARR[1] = 0x44; // minute = 59
    //RTC_ARR[2] = 0x11; // hour = 05 ,24-hour mode(bit 6=0)
    //RTC_ARR[3] = 0x02; // Day = 1 or sunday
    //RTC_ARR[4] = 0x18; // Date = 30
    //RTC_ARR[5] = 0x09; // month = August
    //RTC_ARR[6] = 0x23;    // year = 05 or 2005
    //  WriteRTC(&RTC_ARR[0]);    // Set RTC

    while(1)
    {

        ReadRTC(&RTC_ARR[0]);

        putchar(0x0C); // clear Hyper terminal
```

```

        printf("Day : %s\r\n",Int2Day(RTC_ARR[3]));

        printf("Time : %02bX:%02bX:%02bX\r\n",RTC_ARR[2],RTC_ARR[1],RTC_ARR[0]);

        printf("Data : %02bX-%s-
20%02bX",RTC_ARR[4],Int2Month(RTC_ARR[5]),RTC_ARR[6]);

        DelayMs(1000);    // delay about 1 second
    }
}

void InitSerial(void)
{
    SCON = 0x50; // Setup serial port control register Mode 1:
                // 8-bit uart var baud rate REN: enable receiver

    TMOD |= 0x20; // Set M1 for 8-bit autoreload timer

    TH1 = 0xFD;  // Set autoreload value for timer1 9600 baud
                // with 11.0592 MHz xtal

    TR1 = 1;    // Start timer 1


    TI = 1;     // Set TI to indicate ready to xmit
}

void DelayMs(unsigned int count)
{
    unsigned int i;
    while(count)
    {
        i = 115;


        while(i>0) i--;

        count--;
    }
}


```



```
//-----  
// Convert BCD 1 byte to HEX 1 byte  
//-----  
unsigned char BCD2HEX(unsigned int BCD)  
{  
    unsigned char temp;  
    temp=((BCD>>8)*100)|((BCD>>4)*10)|(BCD&0x0f);  
    return temp;  
}  
  
//-----  
// start I2C  
//-----  
void Start(void)  
{  
    SDA = 1;  
    SCL = 1;  
    _nop();_nop();  
    SDA = 0;  
    _nop();_nop();  
    SCL = 0;  
    _nop();_nop();  
}
```



```
//-----  
// stop I2C  
//-----  
void Stop(void)  
{  
    SDA = 0;  
    _nop();_nop();  
    SCL = 1;  
    _nop();_nop();  
    SDA = 1;  
}  
  
//-----  
// Write I2C  
//-----  
void WriteI2C(unsigned char Data)  
{  
  
    for (i=0;i<8;i++)  
    {  
        SDA = (Data & 0x80) ? 1:0;  
        SCL=1;SCL=0;  
        Data<<=1;  
    }  
  
    SCL = 1;  
    _nop();_nop();  
    SCL = 0;
```



```
}

//-----
// Read I2C
//-----
unsigned char ReadI2C(bit ACK_Bit)
{

    unsigned char Data=0;

    SDA = 1;
    for (i=0;i<8;i++)
    {
        SCL = 1;
        Data<<= 1;
        Data = (Data | SDA);
        SCL = 0;
        _nop_();
    }

    if (ACK_Bit == 1)
        SDA = 0; // Send ACK
    else
        SDA = 1; // Send NO ACK

    _nop_();_nop_();
    SCL = 1;
```

```

        _nop();_nop();

        SCL = 0;

        return Data;
    }

//-----
// Read RTC (all real time)
//-----
void ReadRTC(unsigned char * buff)
{

    Start();

    Writel2C(0xD0);

    Writel2C(0x00);


    Start();

    Writel2C(0xD1);

    *(buff+0)=ReadI2C(ACK);    // Second
    *(buff+1)=ReadI2C(ACK);    // Minute
    *(buff+2)=ReadI2C(ACK);    // hour
    *(buff+3)=ReadI2C(ACK);    // Day
    *(buff+4)=ReadI2C(ACK);    // date
    *(buff+5)=ReadI2C(ACK);    // month
    *(buff+6)=ReadI2C(NO_ACK); // year

    Stop();
}

```

```
//-----  
// Write RTC  
//-----  
void WriteRTC(unsigned char *buff)  
{  
  
    Start();  
    Writel2C(0xD0);  
    Writel2C(0x00);  
    Writel2C(*(buff+0));  
    Writel2C(*(buff+1));  
    Writel2C(*(buff+2));  
    Writel2C(*(buff+3));  
    Writel2C(*(buff+4));  
    Writel2C(*(buff+5));  
    Writel2C(*(buff+6));  
    Stop();  
}
```

```
char * Int2Day(unsigned char day)  
{  
    return DayStr[day-1];  
}
```

```
char * Int2Month(unsigned char month)
```

```
{  
    return MonthStr[BCD2HEX(month)-1];  
}
```

