Exp name: Blinking a LED with the help of 8051

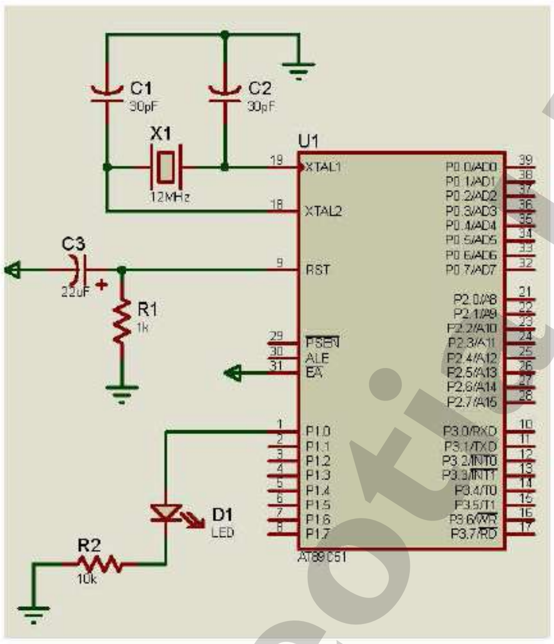
Objective:i) To use GPIO of 8051 as output port

ii)To produce some arbitrary amount of delay

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: As the LED is connected with port 1- line 0, this particular pin of 8051 is made logic '1' and logic '0' alternately with some amount of delay in between. For this particular program the delay produced is arbitrary

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include<AT89X52.h>
voiddelay_ms(unsigned intms);
sbit LED = P1^0;
void main()
{
    // Infinity loop to continue LED Blink
     while(1)
    {
          LED = 0;
delay_ms(2000);
          LED = 1;
delay_ms(2000);
    }
}
voiddelay_ms(unsigned intms)
```

```
{
unsignedinti,j;
for(i=0;i<ms;i++)
for(j=0;j<127;j++);

}
```

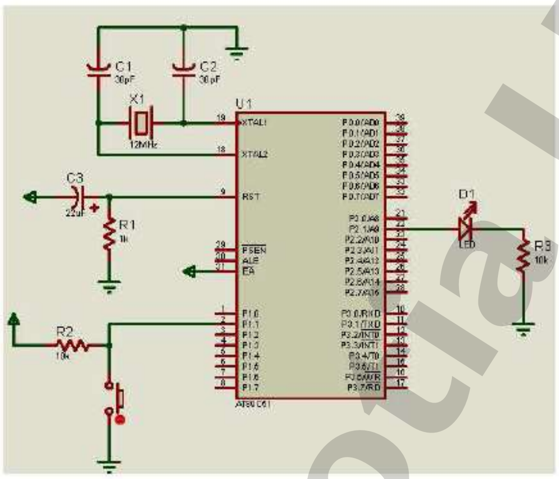
Exp name: Control the LED by a switch

Objective: i)To use the GPIO as input port

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: Here the switch (button type) is connected with port 1-line1 which is configured to work as input port and the LED to be controlled is connected with port 2-line1 which is configured as output port. With the pattern the switch is connected as shown in the circuit(note one can have opposite connection also), when switch is pressed the input line p1.1 gets logic '0' otherwise it remains at logic '1'. This line status is read by the controller and the LED line (the output line) is made logic '0' or '1' accordingly (here what is made for what is not important, what is important is whether we can control the LED as desired according to the input line status or not).

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include<reg51.h>

sbit Led    = P2^1;     //pin connected to toggle Led
sbit Switch =P1^1;   //Pin connected to toggle led


int main()
{
Led   = 0; //configuring as output pin
    Switch = 1; //Configuring as input pin
while(1) //Continuous monitor the status of the switch.
    {

if(Switch == 0)
        {
            Led =1; //Led On
        }
else
```

```
          {
               Led =0; //Led Off
          }
     }
return 0;
}
```

Exp name: Trying to execute multiple tasks without interrupt

Objective: i)To understand the need of interrupt.

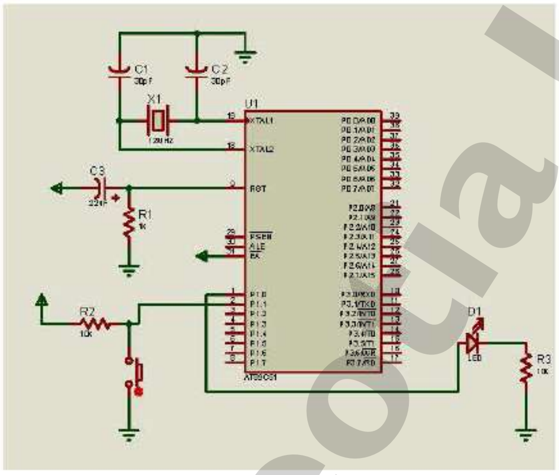
Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle:Here we implement two tasks: one is to read switch status and to reflect it to the LED(both are connected to port 1 as shown in the circuit). Another task is to make port 2 on and off alternately with some delay. As can be understood from the program that the controller executes these two tasks in a sequential manner i.e first complete one then next.

Let us consider that it is currently executing the second task. With some considerable amount of delay, controller will be busy with this task. If in the mean time one presses the switch, controller will miss it as it is busy with second task.

This is the problem in executing multiple tasks in a sequential manner.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include<AT89X52.h>
voiddelay_ms(unsigned intms);
sbit Led = P1^0;
sbit Switch =P1^1;
void main()
{
unsignedinti,j;
Led  = 0; //configuring as output pin
    Switch = 1; //Configuring as input pin

while(1) //Continuous monitor the status of the switch.
    {
   //task1
      if(Switch == 0)
{ Led =~Led;}
```

```
        //task 2
        P2=0xff;
        for(i=0;i<1000;i++)
for(j=0;j<100;j++);
        P2=0x00;
                for(i=0;i<1000;i++)
for(j=0;j<100;j++);

       }
         }
```

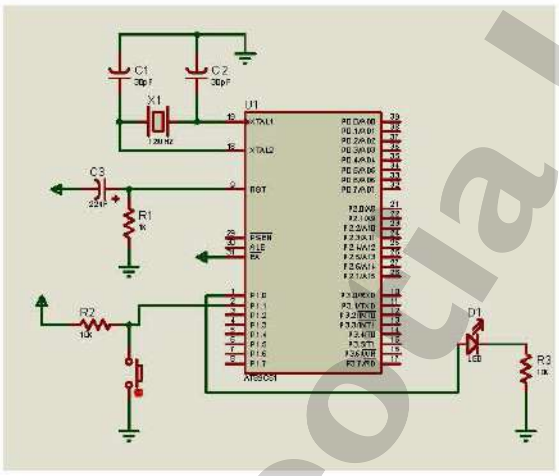
Exp name: To execute multiple tasks with interrupt

Objective: i)To understand the need of interrupt.

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: As with the previous experiment, here we again implement two tasks: one is to read switch status and to reflect it to the LED(both are connected to port 1 as shown in the circuit). Another task is to make port 2 on and off alternately with some delay. But now, the switch is connected in such a manner that if it is pressed, the controller will be interrupted and so the task 1 is xecuted once it is interrupted irrespective of current execution status of the controller. Thus even if it was busy in executing the second task, whenever the switch is pressed then and then the controller will be interrupted and will come out for a moment from second ask and executes the first task and go back again to execute the second task. Thus there is no chance of missing any task thereby implementing multi tasking.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include<AT89X52.h>
voiddelay_ms(unsigned intms);
sbit Led = P1^0;
sbit Switch =P1^1;
void main()
{
unsignedinti,j;
Led  = 0; //configuring as output pin
    Switch = 1; //Configuring as input pin
      IE=0x81;              // Enable INT0
     IT0=1;

while(1) //Continuous monitor the status of the switch.
    {
     //task 2
```

```
      P2=0xff;
      for(i=0;i<1000;i++)
for(j=0;j<100;j++);
      P2=0x00;
            for(i=0;i<1000;i++)
for(j=0;j<100;j++);

      }
       }
      void ISR_ex0(void) interrupt 0      // ISR for external
interrupt INT0
       {
         Led =~Led;
        }
```

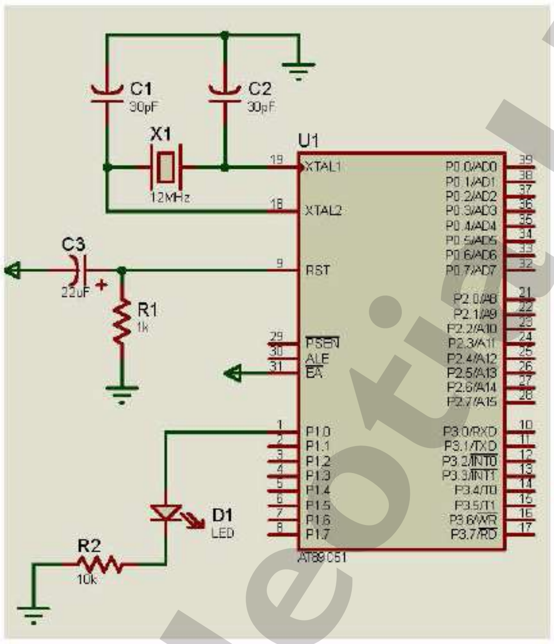
Exp name: Blinking a LED with 8051 timer to produce delay

Objective: ii)How to produce a predefind amount of delay using timers

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: As the LED is connected with port 1- line 0, this particular pin of 8051 is made logic '1' and logic '0' alternately with some amount of delay in between. For this particular program the delay produced is not arbitrary, rather it is well defined. As given in the program, the timer 0 is used in mode 1 to produce 100 us delay, which is repeated for 1000 times to produce an overall 1 second delay.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
// Use of Timer mode 1 for blinking LED using polling method
// XTAL frequency 11.0592MHz
#include<reg51.h>
sbit led = P1^0;                          // LED connected to 1st pin of
port P1
void delay();

main()
{
      unsignedinti;
      while(1)
      {
      led=~led;                    // Toggle LED
      for(i=0;i<1000;i++)
      delay();                        // Call delay
      }
}

void delay()                  // Delay generation using Timer 0 mode 1
{
      TMOD = 0x01;                          // Mode1 of Timer0
      TH0= 0xFC;                            // FC66 evaluated hex value
for 1millisecond delay
      TL0 = 0x66;
      TR0 = 1;                              // Start Timer
      while(TF0 == 0);                      // Using polling method
      TR0 = 0;                        // Stop Timer
      TF0 = 0;                        // Clear flag
}
```

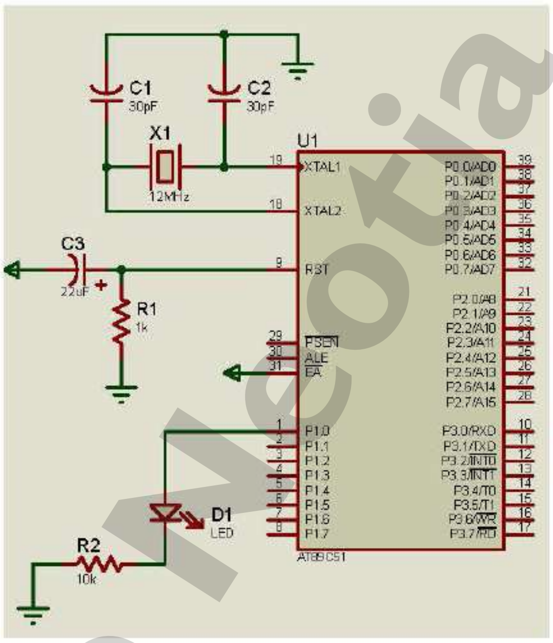
Exp name: Blinking a LED with 8051 timer to produce delay in interrupt mode

Objective: ii)How to produce a predefind amount of delay using timers in interrput mode thereby multitasking.

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: As the LED is connected with port 1- line 0, this particular pin of 8051 is made logic '1' and logic '0' alternately with some amount of delay in between. For this particular program the delay produced is not arbitrary, rather it is well defined. As given in the program, the timer 0 is used in mode 1 to produce 1ms delay, which is repeated for 1000 times to produce an overall 1 second delay. Please note that whether the timer has finished its counting or not, is not being observed here by the controller as opposed to the previous project. Here, once the counting is complete by the timer, it is made known to the controller by interrupting it thereby controller can do other jobs as it does not need to seat and onserve.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include <reg51.h>

sbit LED = P1^0;
longinti;

void main()
```

```
{
    TMOD = 0x01;                  // Mode1 of Timer0
     TH0= 0xFC;  // FC66 evaluated hex value for 1millisecond delay
     TL0 = 0x66;
     IE=0x82;
     TR0=1;

while(1)
    {

    }

}

void timer0() interrupt 1  //
{
TR0=0;
i++;
if (i==1000) {LED=~LED; i=0;}
     TH0= 0xFC;  // FC66 evaluated hex value for 1millisecond delay
     TL0 = 0x66;
     TR0=1;
}
```

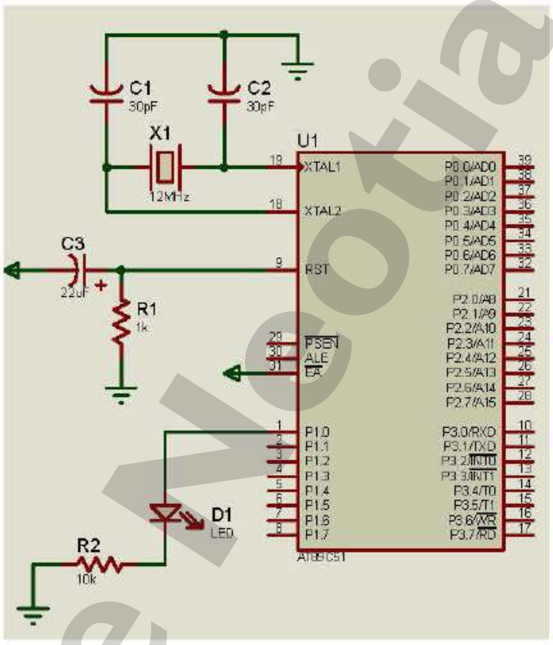
Exp name:Blinking a LED with 8051 timer to produce delay in interrupt mode and mode2

Objective: ii)How to produce a predefind amount of delay using timers in mode 2 and in interrput mode thereby multitasking.

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: As the LED is connected with port 1- line 0, this particular pin of 8051 is made logic '1' and logic '0' alternately with some amount of delay in between. For this particular program the delay produced is not arbitrary, rather it is well defined. As given in the program, the timer 0 is used in mode 1 to produce 1ms delay, which is repeated for 1000 times to produce an overall 1 second delay. Please note that whether the timer has finished its counting or not, is not being observed here by the controller as opposed to the previous project. Here, also like previous project, once the counting is complete by the timer, it is made known to the controller by interrupting it thereby controller can do other jobs as it does not need to seat and observe. The difference is that, here we do not need to initialize the timer with its values everytime it overflows as it is done done automatically. That is why, this mode of timer is normally used to produce a symmetric and repeating amount of delay.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include <reg51.h>

sbit LED = P1^0;
long int i;
```

```
void main()
{
    TMOD = 0x02; //Timer0 mode 2
     TH0=0xA2;
     IE=0x82;
     TR0=1;

    while(1)
    {
    }

}

void timer0() interrupt 1  // Function to generate clock of
frequency 500KHZ using Timer 0 interrupt.
{
i++;
if (i==10000) {LED=~LED;    i=0;}
}

 //  TMOD = 0x01;
// timer_delay()
// {
// TH0=0xBD;
// TL0=0x2F;
// TR0=1;
// while(TF0==0);
// TR0=0;TF0=0;
// }
```

Exp name: Serial (UART) communication between two processors

Objective: i)To understand how serial (using UART) communication is implemented between two processors.

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

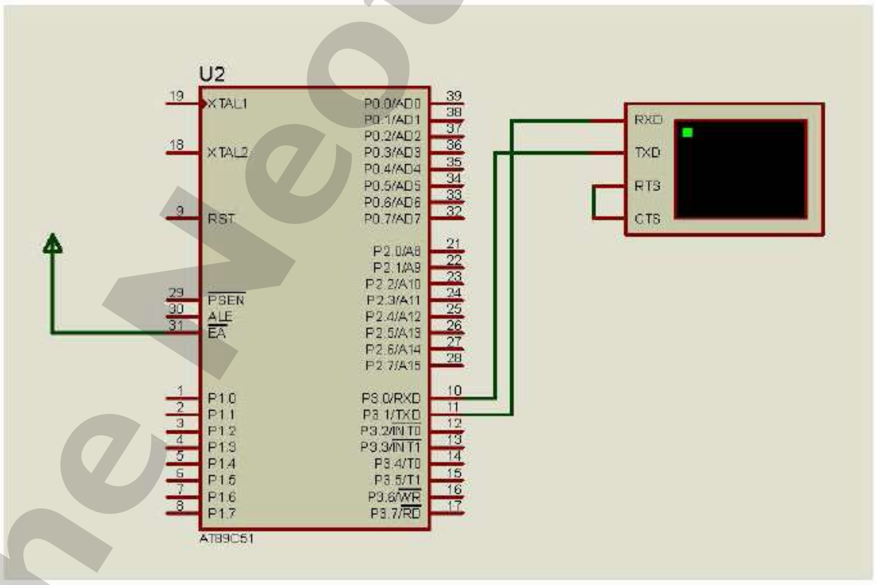
Theory/Principle:Here two processors, one is our controller 8051 and another is the computer itself are used to test whether we can send data serially to and from the controller or not. As shown in the picture, two lines (Tx and Rx) are used for this purpose. In practise, there may be one Rs232 cable in between if we need to connect two processors quite far apart which mainly changes the voltage levels for the purpose of sending data to a long distance.

Here the transmitters and receivers in both sides are used in polling mode to detect whether the previous data byte has been sent or one data byte has been successfully received. This is implemented by polling the TI and RI bits respectively.

The communication is set up by initialising the number of data bits, number of stop bits, parity bit in both the sides (it is to be same in both the sides) by setting the bits of SCON register. The speed is set by initializing the timer (timer 1 in mode 2).

Here the whole process is tested by sending one byte from the computer which is received by the controller (8051) , which again is resent by the controller to the computer. The computer upon receiving the byte, displays it on the screen thereby testing the loop and successful communication.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include<reg51.h>
```

```
voidUART_Init()
{
    SCON = 0x50;  // Asynchrontdatous mode, 8-bia and 1-stop bit
    TMOD = 0x20;  //Timer1 in Mode2.
    TH1 = 0xFd;
    TR1 = 1;      //Turn ON the timer for Baud rate generation
}

voidUART_TxChar(char ch)
{
    SBUF = ch;      // Load the data to be transmitted
while(TI==0);   // Wait till the data is trasmitted
    TI = 0;         //Clear the Tx flag for next cycle.
}


charUART_RxChar(void)
{
char x;
while(RI==0);     // Wait till the data is received
    RI=0;             // Clear Receive Interrupt Flag for next cycle
     x=SBUF;
return(x);     // return the received char
}
putst(char s[])
{
inti;
for(i=0;s[i]!=0;i++)
    {
UART_TxChar(s[i]); // Transmit predefined string
    }
}

int main()
{
   // char i,a[]={"Welcome to 8051 Serial Comm, Type the char to be
echoed: "};
charch;

UART_Init();       //Initialize the UART module with 9600 baud rate

           //  putst("Welcome to 8051 Serial Comm, Type the char to
be echoed :");
while(1)
    {
ch = UART_RxChar(); // Receive a char from serial port
UART_TxChar(ch);    // Transmit the received char
    }
}
```

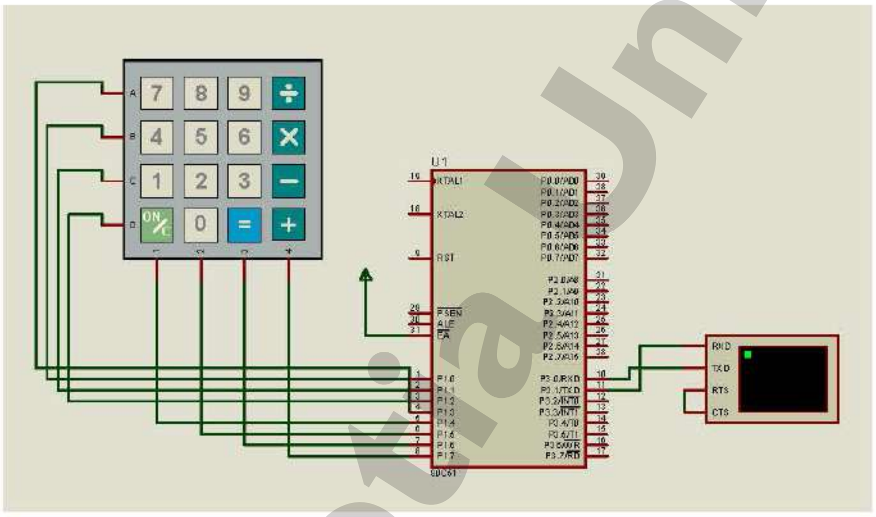
Exp name: Reading a matrix keyboard by 8051

Objective: i) To understand how we can connect and read a matrix keyboard by 8051

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: As shown in the circuit, the rows of the keyboardis driven to logic '0' one by one by the controller and parallelly reading the status of the column lines in search of whether any of them is at logic '0' or not , thereby confirming the switch press and identifying the switch number. Here, in the program we also used the earlier serial communication protocol to send the identified key to the computer such that we can see and confirm the identified switch in the screen.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
  #include<reg51.h> //including sfr registers for ports of the
controller

//Keypad Connections
sbit R1 = P1^0;
sbit R2 = P1^1;
sbit R3 = P1^2;
sbit R4 = P1^3;
sbit C1 = P1^4;
sbit C2 = P1^5;
sbit C3 = P1^6;
sbit C4 = P1^7;
//End Keypad Connections
voidUART_Init()
{
    SCON = 0x50;  // Asynchronous mode, 8-bit data and 1-stop bit
    TMOD = 0x20;  //Timer1 in Mode2.
    TH1 = 0xFD;
    TR1 = 1;      //Turn ON the timer for Baud rate generation
```

```
}

voidUART_TxChar(char ch)
{
    SBUF = ch;       // Load the data to be transmitted
while(TI==0);   // Wait till the data is trasmitted
    TI = 0;          //Clear the Tx flag for next cycle.
}


//char UART_RxChar(void)
//{
//    char x;
//    while(RI==0);     // Wait till the data is received
//    RI=0;             // Clear Receive Interrupt Flag for next
cycle
//   x=SBUF;
//    return(x);     // return the received char
//}
void Delay(int a)
{
int j;
inti;
for(i=0;i<a;i++)
  {
for(j=0;j<100;j++)
    {
    }
  }
}

charRead_Keypad()
{
  C1=1;
  C2=1;
  C3=1;
  C4=1;
  R1=0;
  R2=1;
  R3=1;
  R4=1;
if(C1==0){Delay(50);while(C1==0);return 'A';}
if(C2==0){Delay(50);while(C2==0);return '8';}
if(C3==0){Delay(50);while(C3==0);return '9';}
if(C4==0){Delay(50);while(C4==0);return '/';}
  R1=1;
  R2=0;
  R3=1;
  R4=1;
if(C1==0){Delay(50);while(C1==0);return '4';}
if(C2==0){Delay(50);while(C2==0);return '5';}
if(C3==0){Delay(50);while(C3==0);return '6';}
if(C4==0){Delay(50);while(C4==0);return 'X';}
  R1=1;
  R2=1;
  R3=0;
  R4=1;
if(C1==0){Delay(50);while(C1==0);return '1';}
if(C2==0){Delay(50);while(C2==0);return '2';}
if(C3==0){Delay(50);while(C3==0);return '3';}
if(C4==0){Delay(50);while(C4==0);return '-';}
  R1=1;
  R2=1;
  R3=1;
  R4=0;
if(C1==0){Delay(50);while(C1==0);return 'C';}
if(C2==0){Delay(50);while(C2==0);return '0';}
if(C3==0){Delay(50);while(C3==0);return '=';}
if(C4==0){Delay(50);while(C4==0);return '+';}
return 0;
}

void main()
{
charc,p;
UART_Init();
while(1)
  {
while(!(p = Read_Keypad()));
      UART_TxChar(p);
//    P2=p;
  }
}
```

Exp name: Connecting an external parallel ADC with 8051

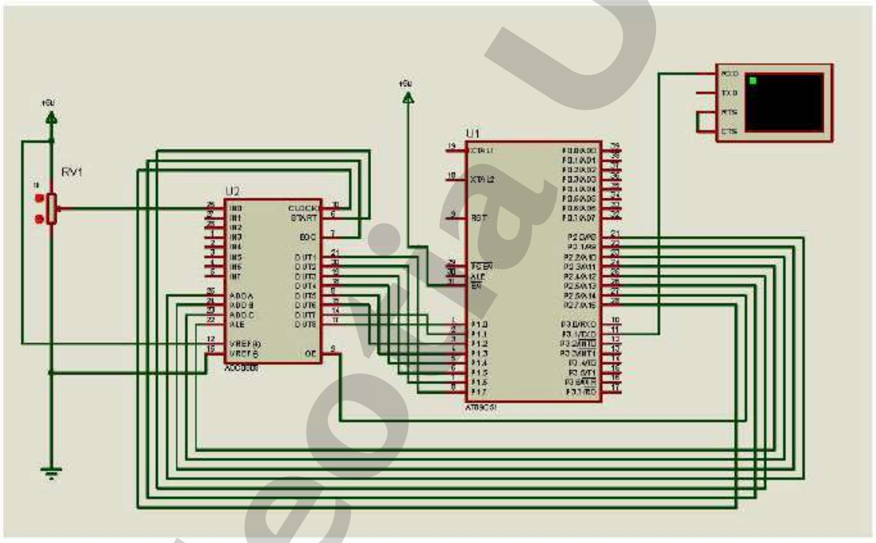
Objective: i) To understand how we can connect and drive an external ADC which is connected parallely with the 8051

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: This is one example of how one can write a driver to drive any external IC, though very small but useful. Here we write to drive IC ADC0808 by 8051. Writing a driver means basically one need to follow the timing diagram and produce/generate the control signals accordingly maintaining the timing diagram. Here also we have done the same thing(please follow the timing diagram of the IC in its datasheet).

In addition to that, here again we have used the serial communication protocol for the purpose of reflecting the read ADC vale into the computer terminal.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include<reg51.h>

sbit ale=P2^3;
sbitoe=P2^6;
sbitsc=P2^4;
sbiteoc=P2^5;
sbitclk=P2^7;
sbit ADDA=P2^0;  //Address pins for selecting input channels.
sbit ADDB=P2^1;
sbit ADDC=P2^2;

#define input_portP1  //ADC
#define output_portP3  //ADC
unsigned char number;
```

```
void timer0() interrupt 1  // Function to generate clock of
frequency 5 KHZ using Timer 0 interrupt.
{
clk=~clk;
}

voidUART_Init()
{
    SCON = 0x50;  // Asynchrontdatous mode, 8-bia and 1-stop bit
    TMOD = 0x20;  //Timer1 in Mode2.
    TH1 = 0xFd;
    TR1 = 1;      //Turn ON the timer for Baud rate generation
}
void delay(unsigned int count)
{
inti,j;
for(i=0;i<count;i++)
for(j=0;j<100;j++);
}

voidread_adc()
{
number=0;
ale=1;
sc=1;
delay(1);
ale=0;
sc=0;
while(eoc==1);
while(eoc==0);
oe=1;
number=input_port;
delay(1);
oe=0;
}
voidadc(inti)  //Function to drive ADC
{
switch(i)
  {
case 0:
    ADDC=0;  // Selecting input channel IN0 using address lines
    ADDB=0;
    ADDA=0;
read_adc();

break;
case 1:
    ADDC=0;  // Selecting input channel IN1 using address lines
    ADDB=0;
    ADDA=1;
read_adc();

break;
case 2:
    ADDC=0;  // Selecting input channel IN2 using address lines
    ADDB=1;
    ADDA=0;
read_adc();

break;
  }
}

voidUART_TxChar(char ch)
{
    SBUF = ch;      // Load the data to be transmitted
while(TI==0);   // Wait till the data is trasmitted
    TI = 0;         //Clear the Tx flag for next cycle.
}

void main()
{
inti=0;
eoc=1;
ale=0;
oe=0;
sc=0;
clk=0;

UART_Init();

 TH0=0xaF;
 IE=0x82;
 TR0=1;
while(1)
{
adc(i);
    // output_port=number;
     UART_TxChar(number);
   }
}
```

Exp name: Connecting an external SPI ADC with 8051

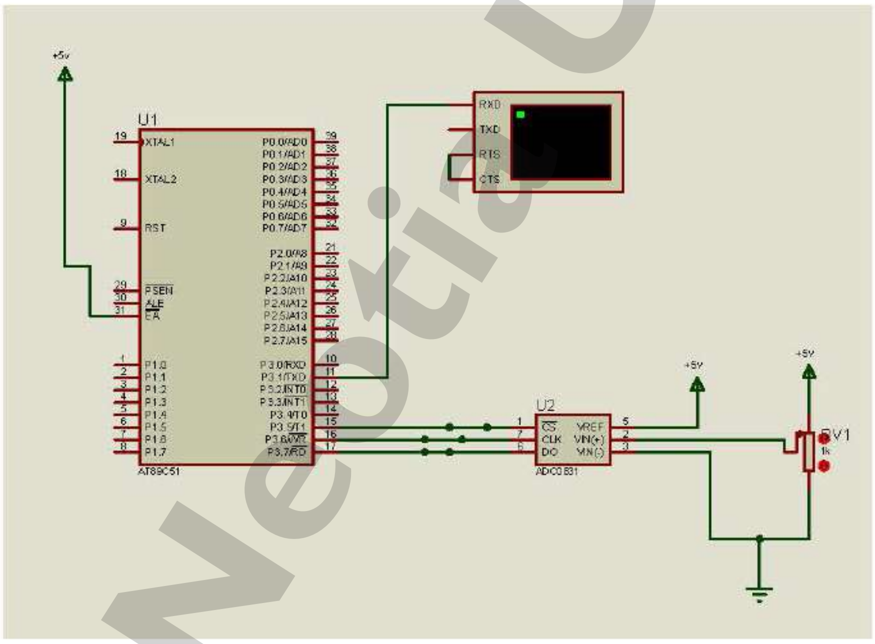
Objective: i) To understand how we can connect and drive an external ADC which is connected by SPI bus with the 8051

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: Here also we have followed the timing diagram (please follow the timing diagram of the IC in its datasheet). Only difference is that here the ADC is following the SPI protocol and so we are generating the same from the controller with the help of software. This process is bit banging. Another option is if the related hardware to generate the control signal is already there within the controller. In that case, we need not have to worry to follow the timing diagram as in that case it is governed by the related hardware module. Here we use the bit banging option.

In addition to that, here again we have used the serial communication protocol for the purpose of reflecting the read ADC vale into the computer terminal.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
#include <AT89x51.h>

sbitadc_CS = P3^5;
sbitadc_CLK = P3^6;
sbitadc_DATA = P3^7;

voidputst(char s[]);
charUART_RxChar(void);
voidUART_TxChar(char ch);
```

```
voidUART_Init();
void delay(unsigned int);
void InitADC0831();
void ReadADC0831();

unsigned char number;
void delay(unsigned int d)
{
unsignedinti;
for(i=0; i<d; i++);
}
void InitADC0831()
{
     adc_CLK = 0;
     adc_CS = 1;
     adc_DATA = 1;
}

void ReadADC0831()
{
     unsigned char i;
     number=0;
 // setup time
     adc_CLK = 0;
     adc_CS = 0;
     delay(500);
 // start conversion
     adc_CLK = 1;
     delay(500);
     adc_CLK = 0;
     delay(500);
 // conversion complete
     adc_CLK = 1;

// data shifted out from adc in -ve edge
     for(i = 0; i< 8; i ++)
     {
          adc_CLK = 0; delay(500);
          number = number<< 1;
          number =number | adc_DATA;
          adc_CLK = 1; delay(500);
     }
     adc_CS = 1;

}

voidUART_Init()
{
    SCON = 0x50;  // Asynchrontdatous mode, 8-bia and 1-stop bit
    TMOD = 0x20;  //Timer1 in Mode2.
    TH1 = 0xFd;
    TR1 = 1;      //Turn ON the timer for Baud rate generation
}

voidUART_TxChar(char ch)
{
    SBUF = ch;      // Load the data to be transmitted
while(TI==0);   // Wait till the data is trasmitted
    TI = 0;         //Clear the Tx flag for next cycle.
}


charUART_RxChar(void)
{
char x;
while(RI==0);     // Wait till the data is received
    RI=0;             // Clear Receive Interrupt Flag for next cycle
     x=SBUF;
return(x);     // return the received char
}
voidputst(char s[])
{
inti;
for(i=0;s[i]!=0;i++)
    {
UART_TxChar(s[i]); // Transmit predefined string
    }
}

voidInitSystem()
{
      InitADC0831();
      UART_Init();
}

void main()
{
unsigned char pnumber;

//    InitSystem();
      UART_Init();

      while(1)
      {
            ReadADC0831();

      if(pnumber !=number)
            UART_TxChar(number);
            pnumber=number;
      }
}
```

Exp name: Connecting an external I2CEEPROM with 8051

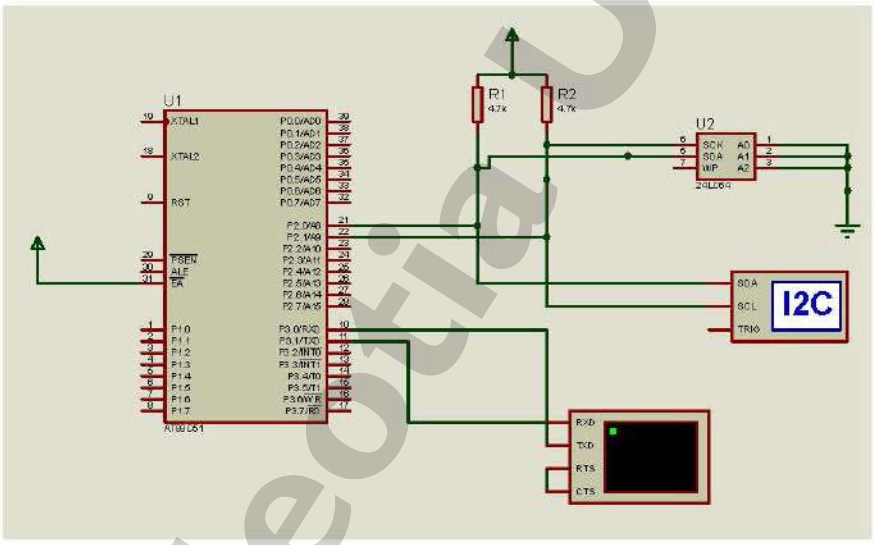
Objective: i) To understand how we can use and connect an external I2C device (here it is one EEPROM) with the 8051

Tools used: Keil IDE and c51 compiler to generate and build the project along with Proteus ISIS for simulation

Theory/Principle: Here also we have followed the timing diagram (please follow the timing diagram of the IC in its datasheet). Only difference is that here the I2C protocol is used (please review the protocol) and so we are generating the same from the controller with the help of software i.ehere we use the bit banging option again.

In addition to that, here again we have used the serial communication protocol for the purpose of interacting with the user in storing (and reading) the value and position of the data to and from the EEPROM.

Circuit: The associated circuit is shown below. Two more things to note: 1) the crystal connection 2) the reset circuit connection (please correlate with the associated theory)



Program:

```
/*=========================================
   Program to write "HELLO" into i2c eeprom memory positions (1:5)
and read by address
 =========================================*/
#include <reg51.h>
//Delay for I2c
#define I2C_DELAY    50
//control address of 241c64
#define device_addr 0xA0
#define ACK_BIT     0
//Define the Pin for the I2c and lec
sbit SDA_BUS = P2^0;
sbit SCL_BUS = P2^1;
```

```
/*===========================================
   Prototypes for I2c functions
 ===========================================*/
void InitI2c(void);
void StartI2c(void);
void RepeatedStartI2c(void);
void StopI2c(void);
voidSendAckBit(void);
voidSendNackBit(void);
void delay(unsigned int);
bit write_i2c(unsigned char);
unsigned char read_i2c(void);
voidwrite_byte_to_eeprom(unsigned int,unsigned char);
unsigned char  read_byte_from_eeprom(unsigned int);
/*===========================================
   Prototypes for UART functions
 ===========================================*/
voidputst(char s[]);
charUART_RxChar(void);
voidUART_TxChar(char ch);
voidUART_Init();
/*===========================================
   Definition of I2c functions
 ===========================================*/
/**
\brief of  delay function.
This function provide the delay which is used in clock generation.
*/
void delay(unsigned int d)
{
unsignedinti;
for(i=0; i<d; i++);
}
/**
\brief of InitI2c function.
This function  use to make the data line and clock line idle to put
the both line high
*/
void InitI2c(void)
{
    SDA_BUS =1;
    SCL_BUS =1;
}
/**
\brief of StartI2c function.
This function performs the start operation to initiate the
communication.
*/
void StartI2c(void)
{
    SDA_BUS  = 1;
    SCL_BUS  = 1;
delay(I2C_DELAY);
    SDA_BUS  = 0;
delay(I2C_DELAY);
}
```

```
/**
\brief of void RepeatedStartI2c function.
When master does not want to relaese the control from the bus then
it assert the repeated
start condition on the i2c bus.
*/
void RepeatedStartI2c()
{
    SCL_BUS  = 0;
delay(I2C_DELAY/2);
    SDA_BUS  = 1;
delay(I2C_DELAY/2);
    SCL_BUS  = 1;
delay(I2C_DELAY/2);
    SDA_BUS  = 0;
delay(I2C_DELAY);
}
/**
\brief of void StopI2c function.
When master want to stop the communication then it will assert the
stop condition to the i2c bus.
*/
void StopI2c(void)
{
    SCL_BUS  = 0;
delay(I2C_DELAY/2);
    SDA_BUS  = 0;
delay(I2C_DELAY/2);
    SCL_BUS  = 1;
delay(I2C_DELAY/2);
    SDA_BUS  = 1;
delay(I2C_DELAY);
}
/**
\brief of  SendAckBit function.
This function use to send the acknoledgement(ACK) bit the i2c bus.
*/
voidSendAckBit()
{
    SCL_BUS  = 0;
delay(I2C_DELAY/2);
    SDA_BUS  = 0;
delay(I2C_DELAY/2);
    SCL_BUS  = 1;
delay(I2C_DELAY);
}
/**
\brief of  SendNackBit function.
This function use to send the Non-acknoledgement(NACK) bit the i2c
bus.
*/
voidSendNackBit(void)
{
    SCL_BUS  = 0;
delay(I2C_DELAY/2);
    SDA_BUS  = 1;
delay(I2C_DELAY/2);
    SCL_BUS  = 1;
delay(I2C_DELAY);
}
/**
\brief of write_i2c function.
This function use to send signle byte to the I2C Data Bus
*/
bit write_i2c(unsigned char byte)
{
unsigned char i;
for(i=0; i<8; i++)
    {
        SCL_BUS  = 0;
delay(I2C_DELAY);
if((byte<<i)&0x80)
            SDA_BUS  = 1;
else
            SDA_BUS  = 0;
delay(I2C_DELAY/2);
        SCL_BUS  = 1;
delay(I2C_DELAY);
    }
//ack from slave //
    SCL_BUS  = 0;
    SDA_BUS  = 0;
delay(I2C_DELAY/2);
    SCL_BUS  = 1;
delay(I2C_DELAY);
return SDA_BUS;
}
/**
\brief of write_i2c function.
This function use to read the data from the I2C data bus
*/
unsigned char read_i2c(void)
{
unsigned char i,d, rxdata=0;
for(i=0; i<8; i++)
    {
        SCL_BUS  = 0;
        SDA_BUS  = 1;
delay(I2C_DELAY);
        SCL_BUS  = 1;
delay(I2C_DELAY/2);
        d=SDA_BUS;
rxdata=rxdata|(d<<7-i);
delay(I2C_DELAY);
    }
returnrxdata;
}
/**
\brief of write_byte_to_eeprom function.
This function use to single byte the eeprom at desire address
*///Write Data to eeprom memory
voidwrite_byte_to_eeprom(unsigned intaddr,unsigned char byte)
```

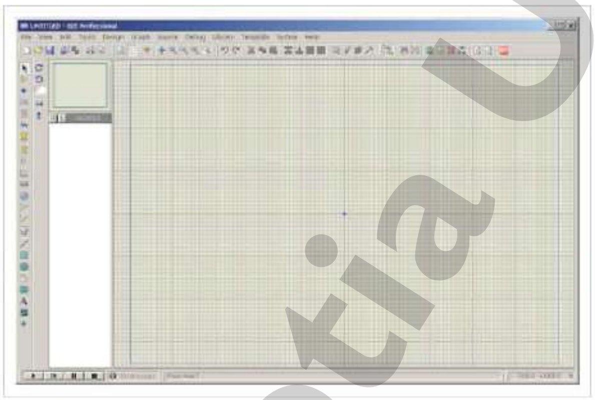
```
{
StartI2c();
while(write_i2c(device_addr|0)==1)
    {
StartI2c();
    }
    write_i2c(addr>>8);
    write_i2c((unsigned char)addr);
    write_i2c(byte);
StopI2c();
}
/**
\brief of read_byte_from_eeprom function.
This function use to read the data byte from eeprom at the desire
the address
*/
unsigned char read_byte_from_eeprom(unsigned intaddr)
{
unsigned char rxdata =0;
StartI2c();
while(write_i2c(device_addr|0)==1)
    {
StartI2c();
    }
    write_i2c(addr>>8);
    write_i2c((unsigned char)addr);
RepeatedStartI2c();
    write_i2c(device_addr|1);
rxdata=read_i2c();
SendNackBit();
StopI2c() ;
returnrxdata;
}
/*==========================================
   Definition of UART functions
 ==========================================*/
voidUART_Init()
{
    SCON = 0x50;  // Asynchrontdatous mode, 8-bia and 1-stop bit
    TMOD = 0x20;  //Timer1 in Mode2.
    TH1 = 0xFd;
    TR1 = 1;      //Turn ON the timer for Baud rate generation
}

voidUART_TxChar(char ch)
{
    SBUF = ch;      // Load the data to be transmitted
while(TI==0);   // Wait till the data is trasmitted
    TI = 0;         //Clear the Tx flag for next cycle.
}


charUART_RxChar(void)
{
char x;
while(RI==0);     // Wait till the data is received
```

```
    RI=0;             // Clear Receive Interrupt Flag for next cycle
     x=SBUF;
return(x);     // return the received char
}
voidputst(char s[])
{
inti;
for(i=0;s[i]!=0;i++)
    {
UART_TxChar(s[i]); // Transmit predefined string
    }
}

// Main function
void main(void)
{   int rd1,i=0;
charrd;
unsigned char rxbyte=0;
unsigned char cSendByte = 'a';

    SDA_BUS = 0;
    SCL_BUS = 0;
InitI2c();
     UART_Init();
write_byte_to_eeprom(0x0001,'H');
     write_byte_to_eeprom(0x0002,'E');
     write_byte_to_eeprom(0x0003,'L');
     write_byte_to_eeprom(0x0004,'L');
     write_byte_to_eeprom(0x0005,'O');

while(1)
  {
putst("type address (1:5) to read data from");
UART_TxChar(0x0d);
rd=UART_RxChar();
  rd1=(int)rd - 0x30;
rxbyte=read_byte_from_eeprom(rd1) ;
delay(5000);
     UART_TxChar(rxbyte);
     UART_TxChar(0x0d);
  }
}
```

HOW TO USE PROTEUS

The simulators I mentioned in my previous post were strictly for beginners. They just show you the output of the microcontroller so you can learn how everything works. They will accompany you as long as you are dealing with manipulating the data on the ports or registers. Sooner or later, you will be going further and attaching external hardware to the 8051 but that's exactly how we deal with it. So if you're talking about simulating a complete circuit then you actually need **PROTEUS** for this.

UPDATE: You can also click this link for an updated version of this post with more insight and explanation.



PROTEUS Environment

WHAT IS PROTEUS?

Basically PROTEUS is also a simulating software but it helps you attach many components with the 8051. Like resistors, capacitors, LEDs, LCDs, keypads, ICs etc. and these are just few that I have named in general. It has a complete library and you will find everything that you will ever need. You can design your complete circuit and then simulate it to view the final output. This means that after perfecting your project on the programming side in KEIL, you'll need to simulate it on PROTEUS to determine the output of the hardware components and change it if need be. This will completely ensure your project's success.

DOWNLOAD PROTEUS

It is a paid application but you can download the free demo version of PROTEUS from their official website here. It is fully functional except that it won't allow you to save your designs.

USING PROTEUS

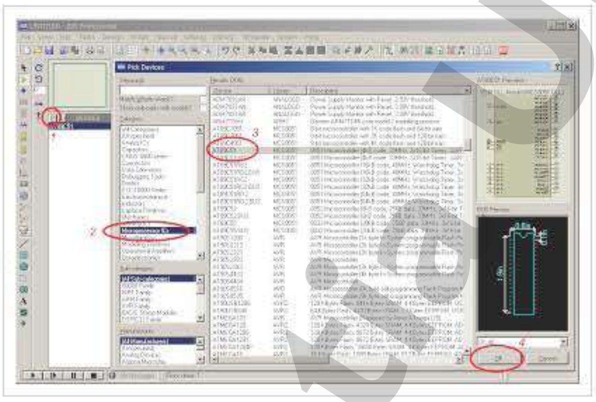
PROTEUS is designed to be user-friendly and you will get the hold of it instantly. There is no need to worry about some complex configuration / settings prior to simulation. Here are the basic steps.

- Place your components from the library
- Connect them accordingly
- Load HEX file (if 8051 is involved)
- Simulate the circuit

Let me explain each step.

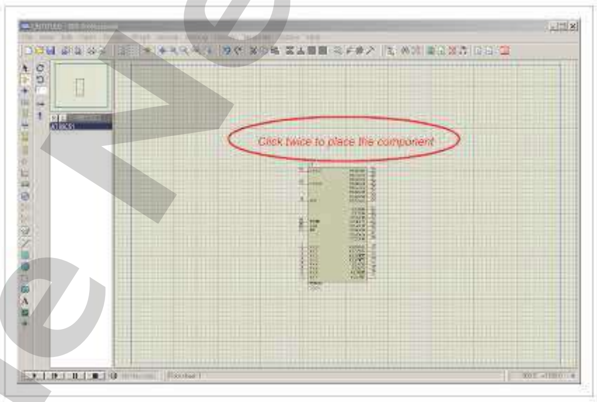
PLACING COMPONENTS

- Click the "Pick from library (P)" button as shown in the figure
- Select any category
- Select item from the list
- Click OK



Click to enlarge

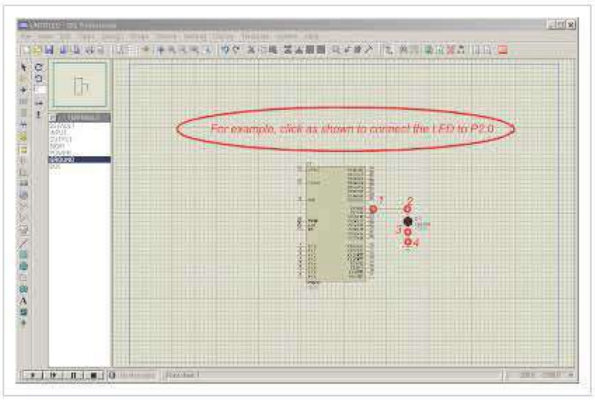
- After selecting component, click anywhere in the design area to select it and then click again to place it



Click to enlarge

CONNECTING COMPONENTS

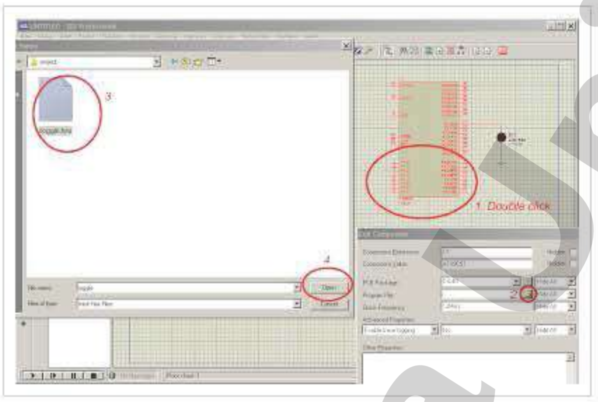
- Place all the required components
- Connect the desired nodes by clicking at starting and ending points



Click to enlarge

LOAD HEX FILE

- Double click the 8051 component to open its properties
- Browse for the HEX file as shown and select it

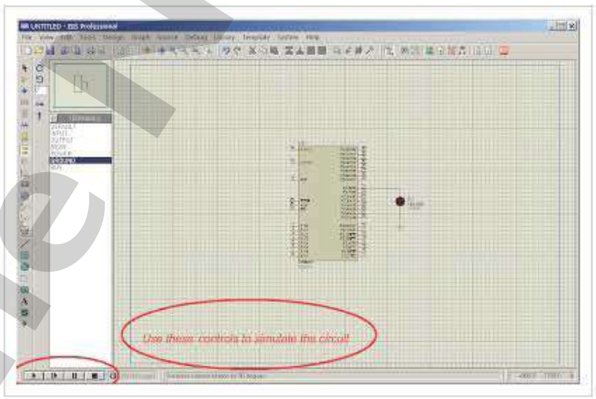


Click to enlarge

And don't worry, in PROTEUS, there is no need to provide the RESET circuit or crystal oscillator to the microcontroller. It will work just fine even without it. The frequency can be adjusted in the properties window as well.

SIMULATING THE CIRCUIT

- The controls at the left-bottom corner will help you simulate the circuit in real time

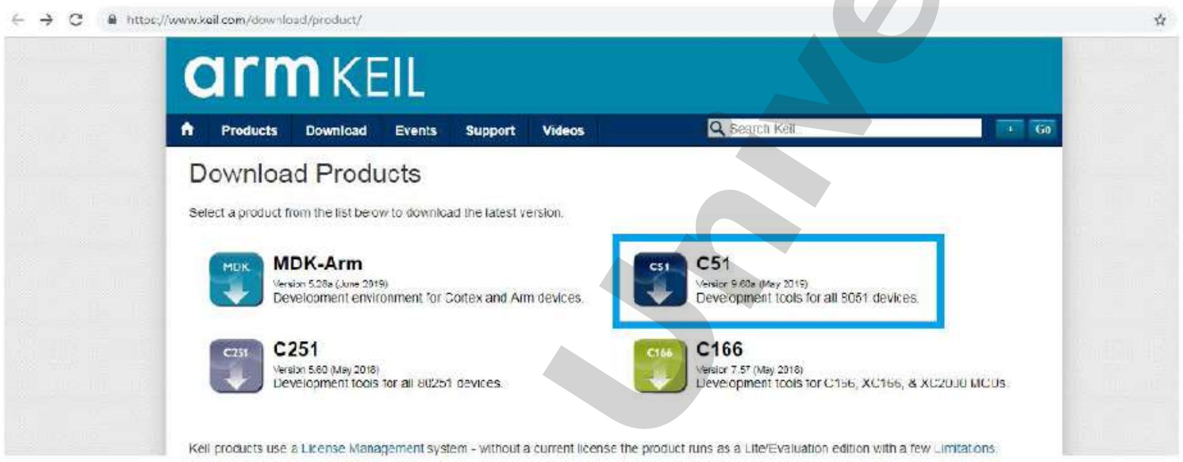


Click to enlarge

The above picture is the complete circuitry for testing an LED on P2.0 like toggling (ON / OFF) through programming but we will get to that part later on. At this point, you will just see the LED glow if you have programmed it to be always ON. Again I am emphasizing that there is no need for other connections to the microcontroller.

8051/52 Programming Using Keil µVision IDE

Step 1: Downloading Keil µVision IDE



Download Products

Select a product from the list below to download the latest version.

MDK-Arm
Version 5.28a (June 2019)
Development environment for Cortex and Arm devices.

C51
Version 9.60a (May 2019)
Development tools for all 8051 devices.

C251
Version 5.60 (May 2018)
Development tools for all 80251 devices.

C166
Version 7.57 (May 2018)
Development tools for C166, XC166, & XC2000 MCUs.

Keil products use a License Management system - without a current license the product runs as a Lite/Evaluation edition with a few Limitations.

Keil provides a code limited (2K bytes) evaluation version for 8051 architecture (C51) which is sufficient for learning purposes.

The main limitations of the evaluation version are the following.

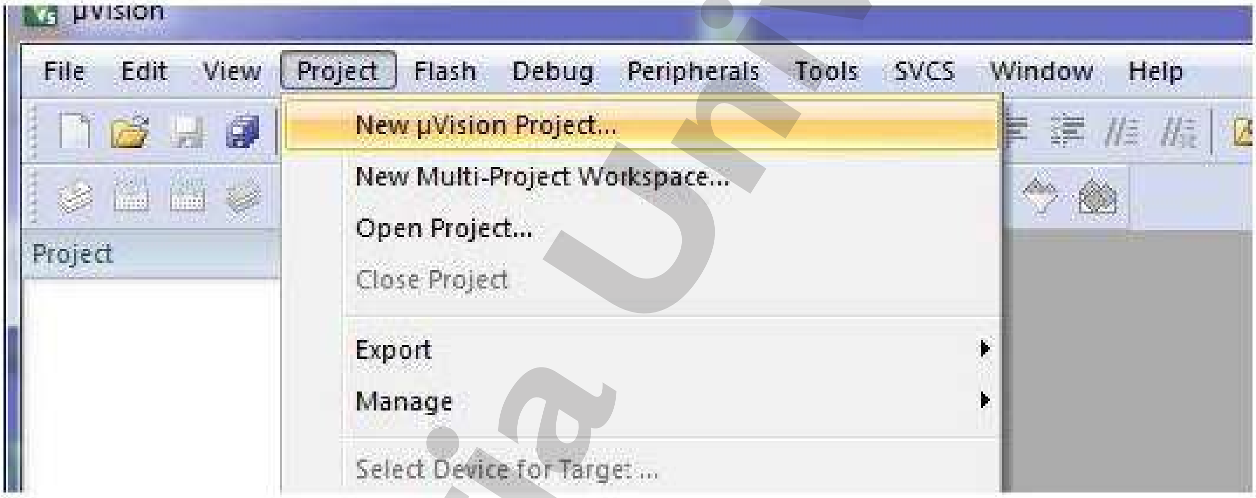
- 8051 compiler, assembler, linker, and debugger are limited to 2 Kbytes of object code
- Programs that generate more than 2 Kbytes of object code will not compile
- The debugger supports programs that are 2 Kbytes or smaller
- No hardware support for multiple DPTR registers is provided

Keil uVision IDE (Evaluation Version) can be downloaded using this link.

On Clicking the above link you will be redirected to Keil Website Download section.

Please click on the **C51** icon to download 8051 development tools (above Figure) .
and download your Windows Executable.

Step 2: Creating a 8051/8052 Project Using Keil Uvision IDE

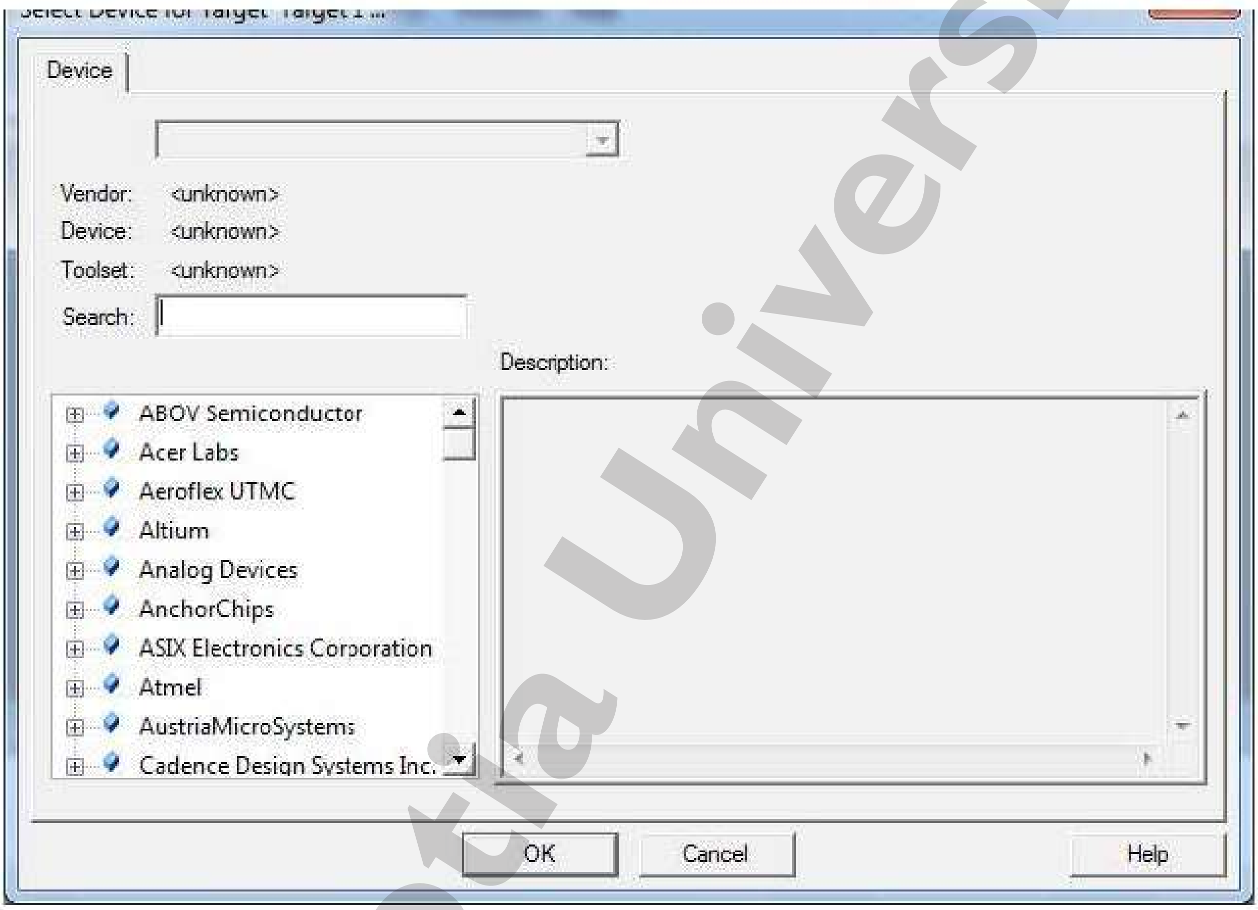


After you have installed the Keil uVision tools for 8051, Double click on the Keil icon on your Windows Desktop to launch the IDE.

To create a new 8051 project using Keil IDE, Click on the ' **Project** ' item on the IDE Menu bar and select ' **New uVision Project...** 'as shown in the above image.

Now create a Folder to store your project and give a name to your Project files (*.uvproj), for eg Test (Test.uvproj).

Step 3: Selecting an 8051 Device in Keil

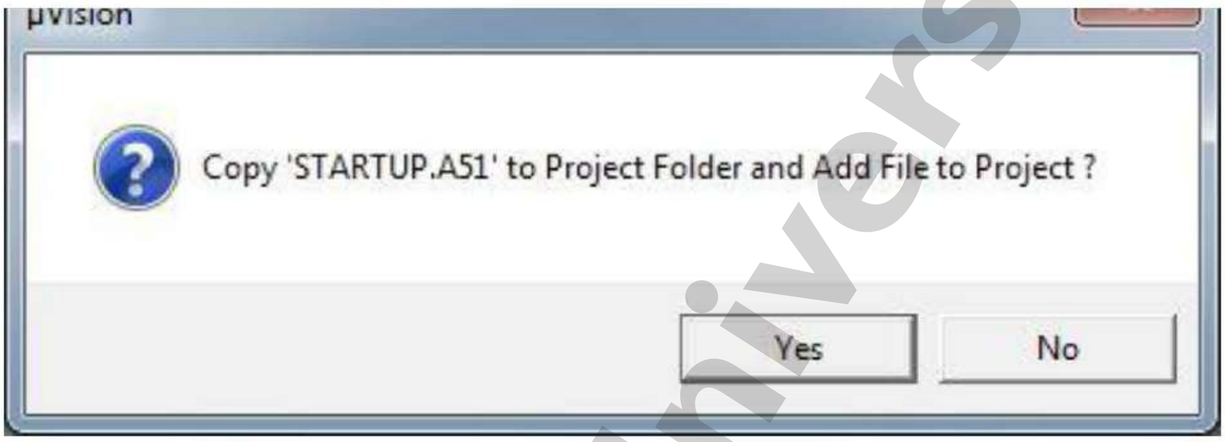


You will then be taken to the device selection dialog, where you can select the 8051 derivative for which you want to develop software.

Keil has support for a wide variety of 8051 derivatives on its IDE.The 8051 derivatives are organised according to their manufacturer's.

On selecting the particular microcontroller the Keil IDE also displays the features of the selected microcontroller on its left pane .You can Click OK to confirm your choice.

Step 4:



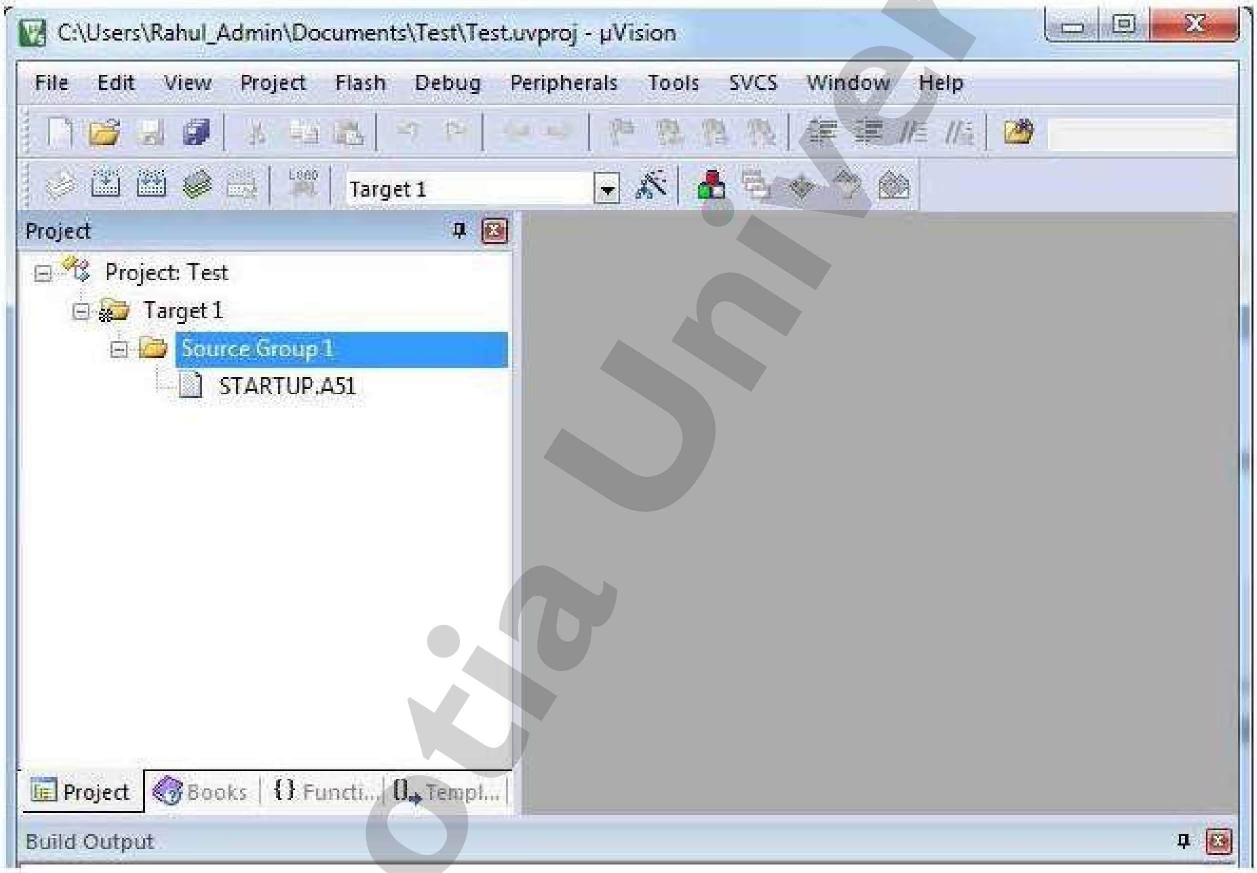
Copy 'STARTUP.A51' to Project Folder and Add File to Project ?

After selecting your 8051 derivative,

You will get another dialog as shown Above.Asking to copy STARTUP.A51

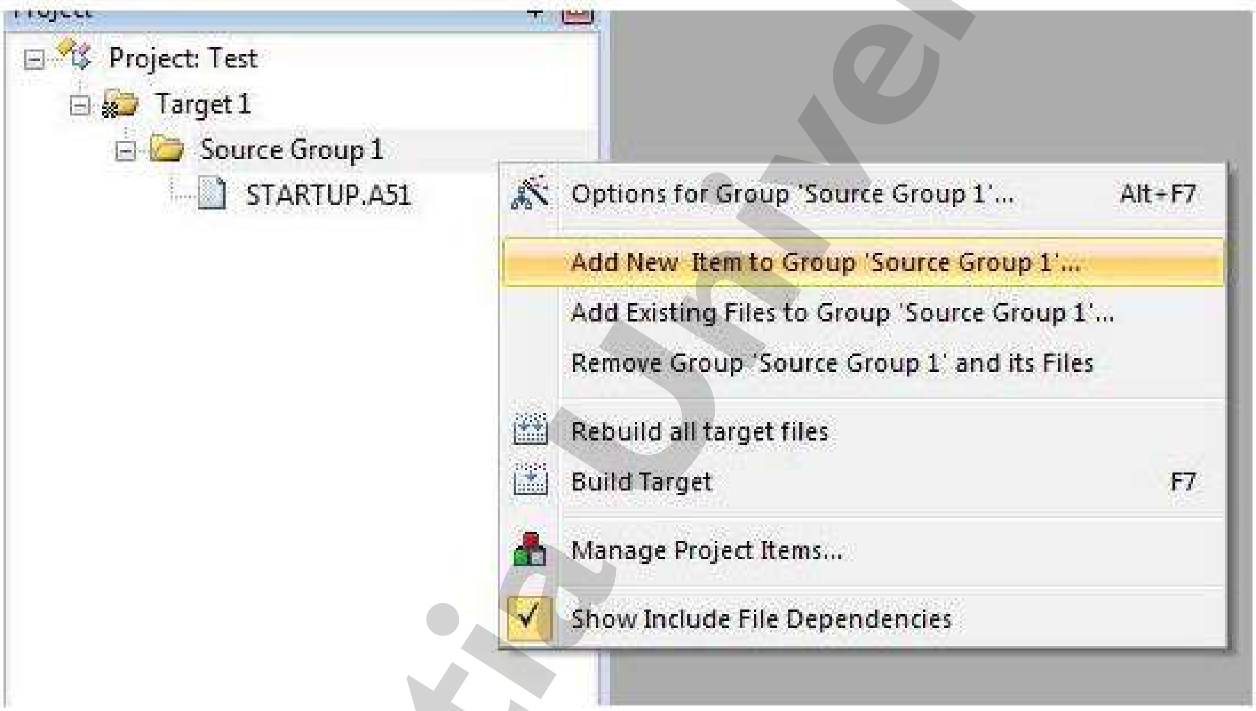
Click ' Yes '

Step 5:


C:\Users\Rahul_Admin\Documents\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Project: Test
Target 1
Source Group 1
STARTUP.A51
Project
Books
{} Functi...
{}. Templ...
Build Output

Now your Project pane on the Kiel IDE would look something like this (above image)

Step 6: Adding C Files to Keil Project


Project: Test
Target 1
Source Group 1
STARTUP.A51
Options for Group 'Source Group 1'... Alt+F7
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files
Rebuild all target files
Build Target F7
Manage Project Items...
Show Include File Dependencies

Now you can add C files to you Project.

Right Click on the **Source Group 1** folder on your Project pane and select **Add New Item to Group 'Source Group1'...**

Step 7:

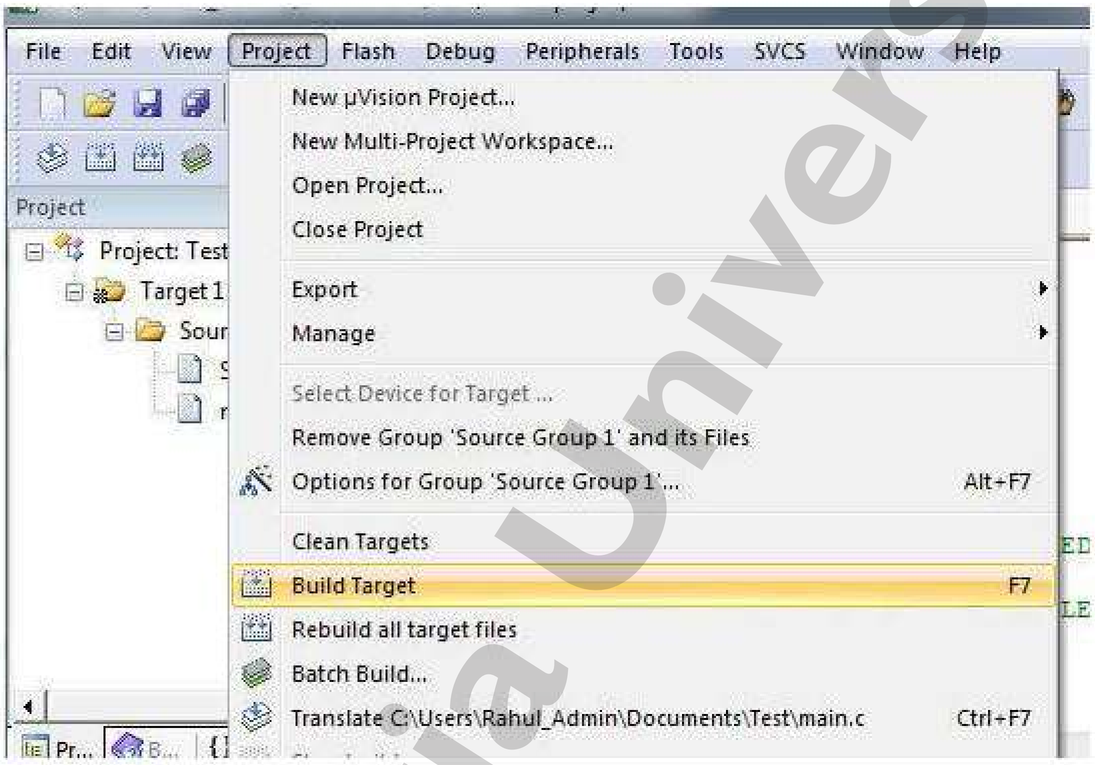
Now you can select the type of file you want to add to your project using Select C File(.c) and give it a name (here main.c) and Click **Add**.Now you can type a small program into the main.c to blink the LED's connected to Port 1 of 8051 .You can find the source code below.

**********************************START**

```
#include<reg52.h>          // special function register declarations
                           // for the intended 8051 derivative
P3 = 0x00;                 //output port used for leds
sbit LED = P3^5;           // Defining LED pin
void Delay(void);          // Function prototype declaration
void main (void)
{
   while(1)             // infinite loop
   {
      LED = 0;            // LED ON
      Delay();
      LED = 1;            // LED OFF
      Delay();
   }
}
void Delay(void)
{
   int j;
   int i;
   for(i=0;i<10;i++)
   {
      for(j=0;j<10000;j++)
      {
      }
   }
}
```

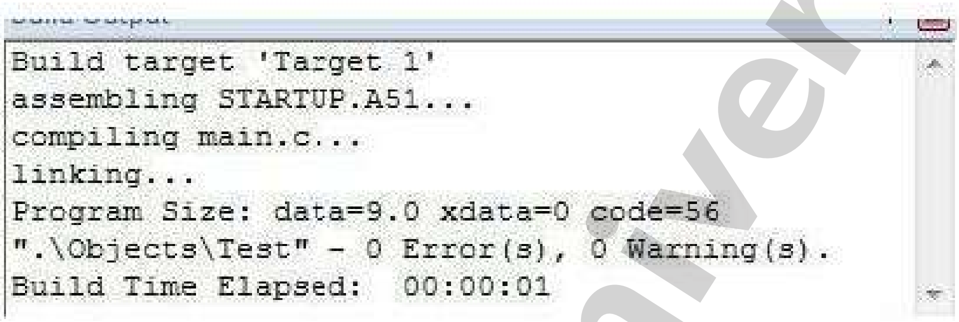
**********************************STOP****************************

Step 8: Building a C Project Using Keil UVision IDE


File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Project
Project: Test
Target 1
New µVision Project...
New Multi-Project Workspace...
Open Project...
Close Project
Export
Manage
Select Device for Target ...
Remove Group 'Source Group 1' and its Files
Options for Group 'Source Group 1'... Alt+F7
Clean Targets
Build Target F7
Rebuild all target files
Batch Build...
Translate C:\Users\Rahul_Admin\Documents\Test\main.c Ctrl+F7

After you have typed out the above c program to your main.c file,You can compile the C file by pressing **F7 key** or by going to '**Project -> Build Target** ' on the IDE menu bar.

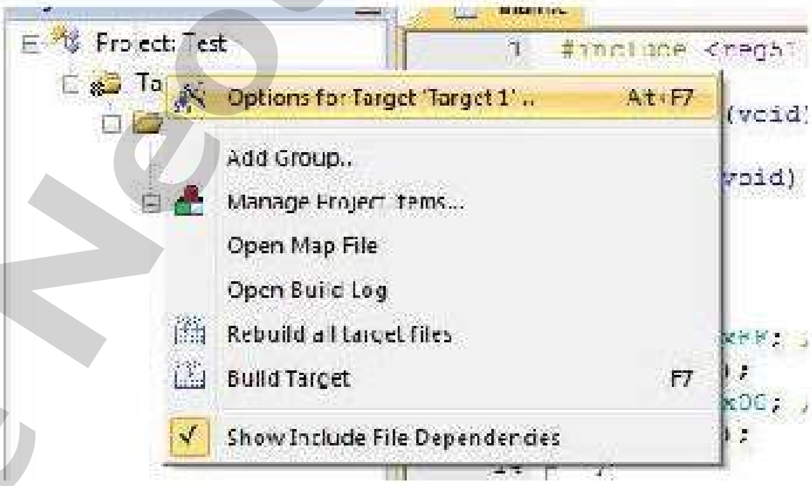
Step 9:



```
Build target 'Target 1'
assembling STARTUP.A51...
compiling main.c...
linking...
Program Size: data=9.0 xdata=0 code=56
".\Objects\Test" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```

If there are no errors the code will compile and you can view the output on the Build Output panel.

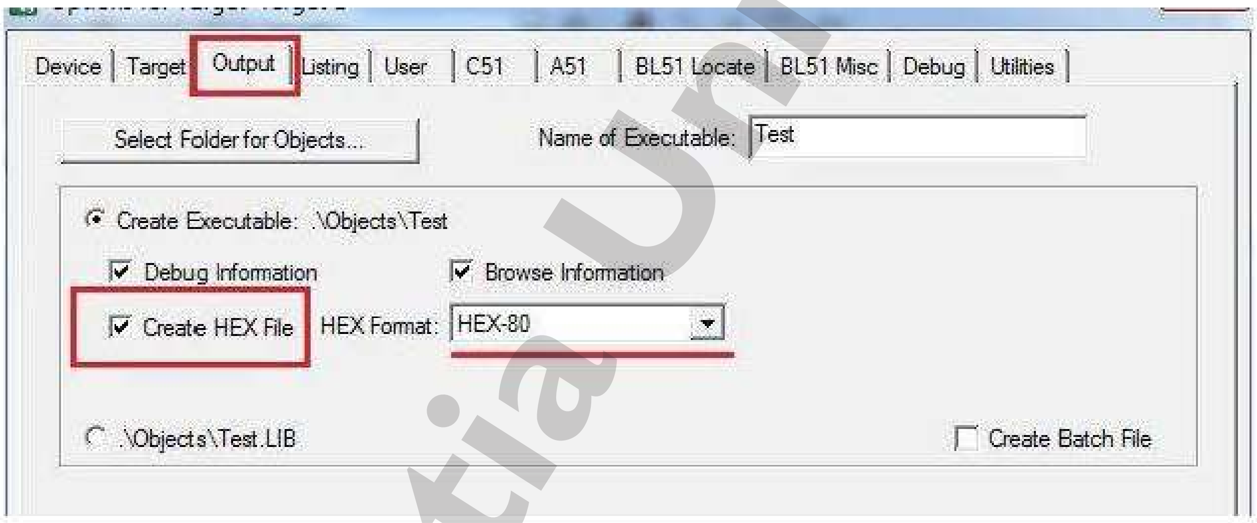
Step 10: Generating 8051 HEX File Using Kiel IDE



Inorder to download the code into the 8051 microcontroller we have to generate the corresponding hex code .

In Keil uVision IDE you can generate hex file for your 8051 derivative by, Right Clicking on the **' Target 1 '** Folder and Selecting **Options for Target 'Target1'....**

Step 11:



Name of Executable: Test

[✓ Create Executable: .\Objects\Test]

☑ Debug Information ☑ Browse Information

☑ Create HEX File HEX Format: HEX-80

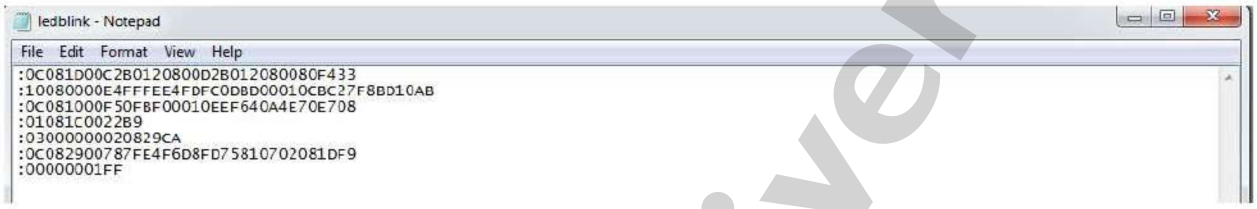
.\Objects\Test.LIB ☐ Create Batch File

Then on the Options for Target **' Target 1'** Dialog ,

Select the **Output tab** and check the **Create Hex File** option and Press **OK.**

Now rebuild your project by pressing F7.
Kiel IDE would generate a hex file with same name (here Test.hex) as your project in the Objects folder .

Step 12: Viewing the Generated Hex File



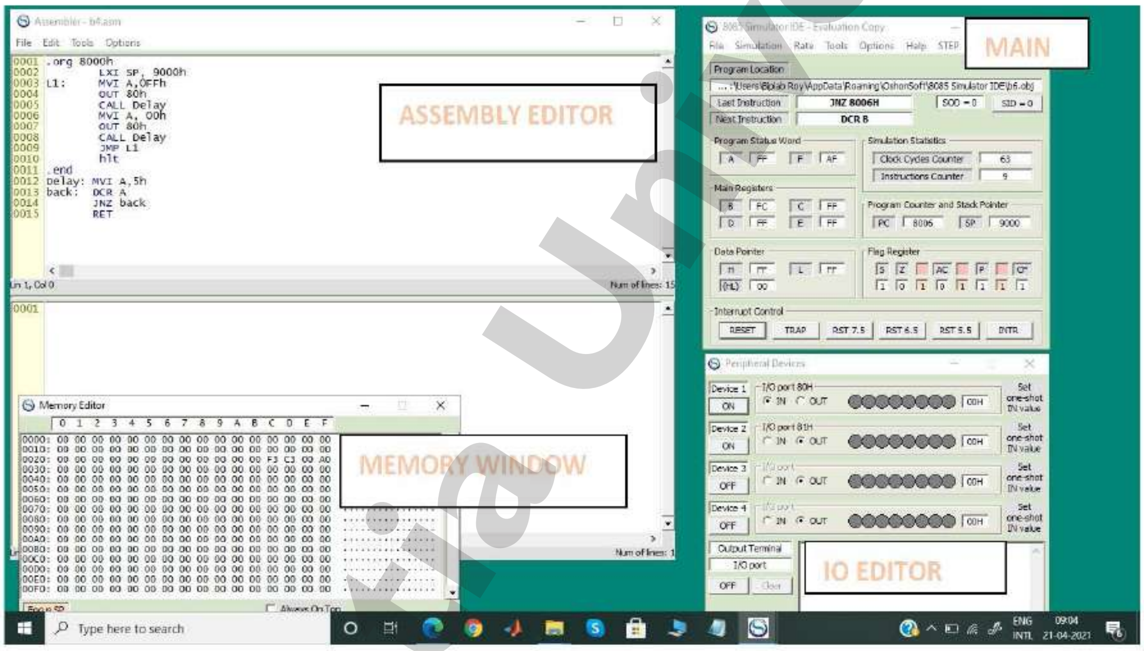
```
:0C081D00C2B0120800D2B012080080F433
:10080000E4FFFEE4FDFC0DBD00010CBC27F8BD10AB
:0C081000F50FBF00010EEF640A4E70E708
:01081C0022B9
:03000000020829CA
:0C082900787FE4F6D8FD7581070208 1DF9
:00000001FF
```

You can open the Test.hex file with notepad to view the contents after creation.

Experiment Name: Familiarization with the 8085 simulator with simple assembly instructions

Objective: To be familiar with the simulator as well as assembly instructions

Procedure: Open the simulator by double clicking the symbol “8085 simulator IDE” from OshonsoftIncon the desktop. From the main window, go to tools->assembler to open the assembly window, which we will be using for all successive programs for the purpose of writing and assembling our assembly programs. Also, one may open IO editor (among many other windows) for input output purposes as shown below the main window. Another very important window is the memory window where from one can see all the memory content of the processor including the code and data.



```
.org 8000h
        LXI SP, 9000h
L1:     MVI A,0FFh
        OUT 80h
        CALL Delay
        MVI A, 00h
        OUT 80h
        CALL Delay
        JMP L1
        hlt
.end
Delay: MVI A,5h
back:  DCR A
       JNZ back
       RET
```

Last Instruction: JNZ 8006H
Next Instruction: DCR B
Clock Cycles Counter: 63
Instructions Counter: 9
PC: 8006
SP: 9000

Write a sample program as given below, in the assembly editor and assemble it. If any error is there, it will be shown in the attached window.

```
.org 8000h
      mvi A, 34h
      mvi B, 53h
      add B
.end
```

Observe that .org and .end statements which are called “assembly directives”, by which we direct the assembler to instruct the starting position of program in memory and where program ends. Please note that, the format of the program written above is specific to an assembler but normally the assembler designers follow almost the same format in most cases.

Once assembled, load the assembled program (the object file) from the main window (file-> load program) into the simulator.

Now one can simulate the program with different speed set from the simulate tab. Here we will use “step by step” such that in every step of execution of instructions, we can observe different register contents and the flag bits. Before starting simulation set the starting address to 8000h from options-> change starting address .

Experiment name: Use of branch instruction to place some finite number of values into memory in a loop

Objective: To be able to implement loops using branch instructions

Procedure: Use the same simulator as used in exp-1 to understand the operation in a loop with the help the program given below-

```
.org 8000h
MVI A,02FH  ;initial value in register A
      LXI B,0FF00H  ;initial value in register pair BC
L1:   STAX B ;load value in A to the memory location addressed by BC
      INX B  ;increment BC
      DCR A  ;decrement A
      JNZ L1  ;loop until value in A is zero
      STAX B  ;load value 00H to memory location FFFFH
      HLT  ;haltcpu
      .END
```

Here in the program, values of register A starting from 2Fh is copied to memory location FF00h upwords and decremented every time in a loop.

Experiment Name: Blinking a LED with 8085

Objective: To be able to use output instruction of 8085 as well as to implement time delay using assembly instructions

Procedure: open the simulator and load the program given below after assembling-

```
.org 8000h
     LXI SP, 9000h
L1:        MVI A,0FFh
        OUT 80h
     CALL Delay
     MVI A, 00h
        OUT 80h
     CALL Delay
     JMP L1
     hlt
.end
Delay: MVI A,5h
back:  DCR A
       JNZ back
       RET
```

Note that, the LED connected to the port number 80h of 8085 is alternatively made on and off in the loop L1. And every time the LED is either made on/off, it is allowed to stay in that status for some time by putting some delay before changing the LED status.

Observe that, here we have used the stack by initializing the SP. As we are using the CALL instruction, we will have to use the stack for storing the PC value to be popped while returning back.

Experiment Name: controlling a LED by a switch with 8085

Objective: To be able to use input/output instruction of 8085

Procedure: open the simulator and load the program given below after assembling-

```
.org 8000h
L1:     IN 80h
        OUT 81h
      JMP L1
      hlt
.end
```

Open the input/output editor and Execute in “step by step” mode. From the input/output editor, apply some input after setting port number to 80h. Observe that the same is reflected into the output port 81h set in the similar manner in the input/output editor window.

Experiment Name: Interrupt based input/output with 8085

Objective: To be able to use interruptwith 8085

Procedure: open the simulator and load the program given below after assembling-

```
.org 8000h
      LXI SP, 9000h
      EI
l1:      MVI B, 0FFh
back: DCR B
      JNZ back
      JMP L1
      hlt
.end


.org 002ch
      DI
      jmp 0A000h

.org 0A000h
      IN 80h
      OUT 81h
      EI
      RET
```

Here, first the interrupt is enabled by EI instruction. Microprocessor runs the program to decrement the value of register B in a loop normally. When it is interrupted, it comes out of the normal flow and goes to the location (predefined vector address) 2ch which is called ISR address. There from it goes to location A000h where actual routine is written. In our case the routine is to read the input switch vale and reflect it to LED.