

# Digital WorkBook-8051

---

Learn-Practice-Grow

[www.pantechsolutions.net](http://www.pantechsolutions.net)

8/19/2021

## Contents

|  |    |
|--|----|
| Introduction to 8051 Controller .....              | 2  |
| Getting Started with 8051 and Keil IDE .....       | 7  |
| Introduction.....                                  | 7  |
| 8051 UART.....                                     | 16 |
| 8051 serial interrupt .....                        | 22 |
| 8051 Power down and idle mode .....                | 24 |
| 8051 Timers .....                                  | 29 |
| Timer interrupt in 8051 .....                      | 41 |
| 8051 Interrupts .....                              | 44 |
| LCD16x2 interfacing in 4-bit mode with 8051 .....  | 51 |
| 4-bit Mode.....                                    | 52 |
| HC-05 Bluetooth Module Interfacing with 8051 ..... | 59 |

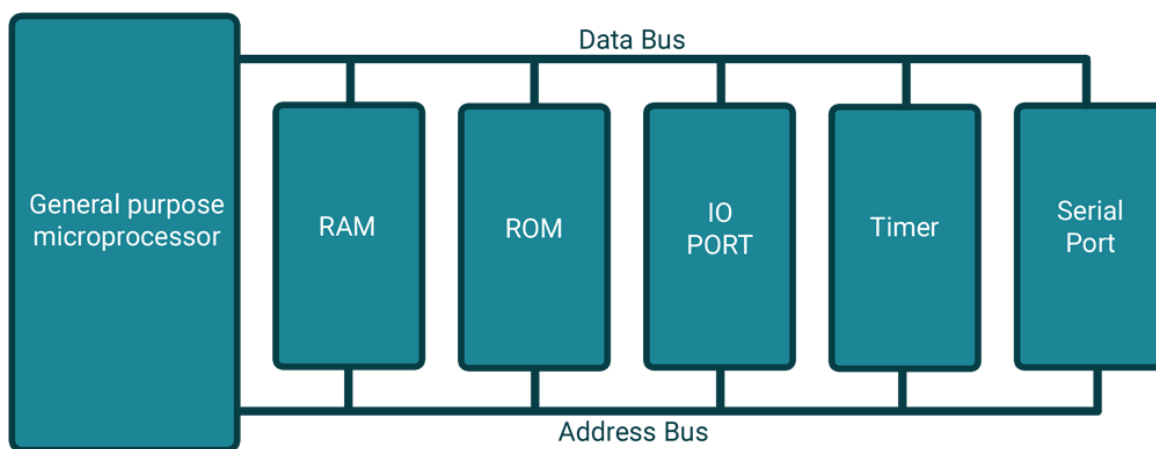
# Introduction to 8051 Controller

## Introduction

Microcontroller consists of all features that are found in microprocessors with additional built-in ROM, RAM, I/O ports, Serial ports, Timers, Interrupts, and Clock circuits. It is an entire computer on a single chip that is embedded within applications. Microcontrollers are widely used in many domestic (washing machines, VCD players, microwave oven, robotics, etc.) as well as industrial and automobile areas.

The 8051 is the first microcontroller of the MCS-51 family developed by Intel Corporation in 1980. It was developed using N-type Metal-Oxide-Semiconductor (NMOS) technology and later it came to be identified by a letter C in their names e.g. 80C51 which was developed with Complementary Metal-Oxide-Semiconductor (CMOS) technology which consumes less power than NMOS and made it better compatible for battery-powered applications.

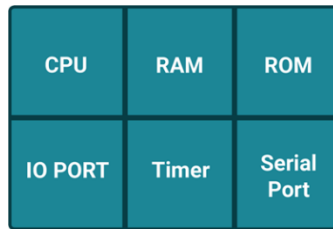
Microcontrollers can be classified on the basis of their bit processing capability e.g. 8-bit microcontroller means it can read, write, and process 8-bit data. Basically, it specifies the size of the data bus. Today microcontrollers are designed with much more compact, cheap, and powerful specifications like AVR and PIC.



### General Purpose Microprocessor System

As shown in the figure, Microprocessors need external devices such as RAM for data storage, ROM for program storage, PPI 8255 for I/O port, 8253 for timer, and USART for serial communication.

All these peripherals are integrated together to form a controlling unit ready to embed within applications.



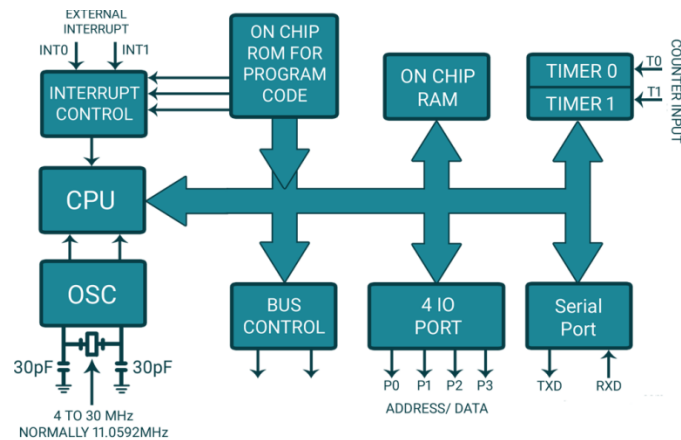
### Microcontroller

Whereas microcontroller has all memories and ports available on-chip as shown in the figure. This makes microcontrollers the most popular. Later many semiconductor companies developed their own microcontrollers with different specifications.

#### Family Member Specifications

Find below, the specifications for various popular 8051 family members developed by mentioned semiconductor companies.

| Developed by | Family members | RAM(bytes) | ROM(Kbytes) | Timers | I/O pins | Serial port | Interrupt sources |
|--------------|----------------|------------|-------------|--------|----------|-------------|-------------------|
| Intel        | 8051           | 128        | 4           | 2      | 32       | 1           | 6                 |
|              | 8052           | 256        | 8           | 3      | 32       | 1           | 8                 |
|              | 8031           | 128        | None        | 2      | 32       | 1           | 6                 |
| Atmel        | AT89C51        | 128        | 4           | 2      | 32       | 1           | 6                 |
|              | AT89C52        | 256        | 8           | 3      | 32       | 1           | 6                 |
| Dallas       | DS5000-8       | 128        | 8           | 2      | 32       | 1           | 6                 |
|              | DS5000-32      | 128        | 32          | 2      | 32       | 1           | 6                 |
| Philips      | P89C51RD2      | 1024       | 64          | 3      | 32       | 1           | 8                 |

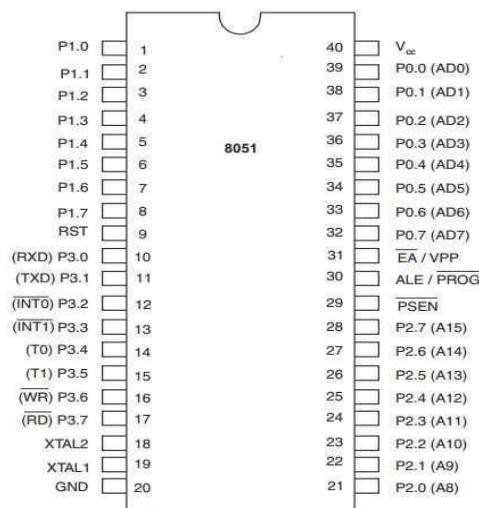


## 8051 Architecture

All 8051 microcontrollers have unique architecture as shown in the figure, which consists of functional blocks to build 8051 powerful controlling machines.

### CPU

Microcontroller 8051 has a central processing unit which is also called ALU (Arithmetic Logic Unit) which performs all arithmetic and logical operation.



## 8051 Pin Diagram

## **RAM (Random-access memory)**

- Microcontroller 8051 has 128-byte RAM for data storage.
- It is a Volatile type of memory. That means the data is lost when power to the device is turned off.
- It is used during execution time to store data temporarily.
- RAM consists of a register bank, stack, and temporary data storage with some special function registers (SFR's).

## **ROM (Read Only Memory)**

- In 8051, 4KB ROM is available for program storage.
- It is a Non-Volatile type of memory. It means that data is not lost even in the event of power failure.
- 8051 has a 16-bit address. It means it can access  $2^{16}$  memory locations and we can interface up to 64 KB of program memory externally in case of large applications.

Sizes specified of RAM and ROM is different by their manufacturer.

## **Timers and Counters**

- Microcontroller 8051 has two timer pins T0 and T1
- By these timers, we can generate a delay of a particular time in timer mode
- We can count external pulses or events in counter mode
- Two 16-bit timer registers are available as T0 (TH0 & TL0) and T1 (TH1 & TL1), e.g. If we want to load T0 then we can load Higher 8-bit in TH0 & Lower 8-bit in TL0
- TMOD and TCON registers are used to select mode and control the timer operation

## **Interrupts**

- Interrupts are requested by internal or external peripherals which are masked while unused.
- Interrupt handler routines are called after each interrupts event occurs.
- These routines are called an Interrupt Service Routine (ISR) and are located in special memory loc.
- INT0 and INT1 pins used to accept external interrupts.

## **Oscillator**

- It is used to provide a clock to the 8051 which decides the speed and baud rate.
- We use crystals of frequency varying from 4MHz to 30 MHz normally we use 11.0592 MHz frequency which is required for 9600 baud rate in serial communication.

## **I/O Ports**

- 8051 has four Input/output port P0, P1, P2, P3
- Each port is 8 bit wide and their SFR (P0, P1, P2, P3) are bit accessible i.e. we can set or reset individual bit.
- Some ports have dual functionality on their pins as,
- P0 I/O pins are multiplexed with an 8-bit data bus and lower order address bus (AD0-AD7) which de-multiplexed by ALE signal and latch used in external memory access operation.
- P2 I/O pins are multiplexed with remaining higher order address bus (A8-A15)
- P3 I/O pins also have dual functions as,
  1. P3.0 – RXD – Serial data receive.
  2. P3.1 – TXD – Serial data transmit.
  3. P3.2 – INT0 – External Interrupt 0.
  4. P3.3 – INT1 – External Interrupt 1.
  5. P3.4 – T0 – Clock input for counter 0.
  6. P3.5 – T1 – Clock input for counter 1.
  7. P3.6 – WR – Signal for writing to external memory.
  8. P3.7 – RD – Signal for reading from external memory.
- P0 and P2 can't be used as I/O pins in the external memory access operation

## **Serial Communication port**

- 8051 has two serial communication pins TXD and RXD used for transmitting and receive data serially via the SBUF register
- SCON SFR used to control serial operation

# Getting Started with 8051 and Keil IDE

## Introduction

8051 microcontroller can be programmed in two languages

- Assembly language
- C language

8051 microcontroller popular development IDE is MCU 8051 IDE and  $\mu$ Vision to develop code.

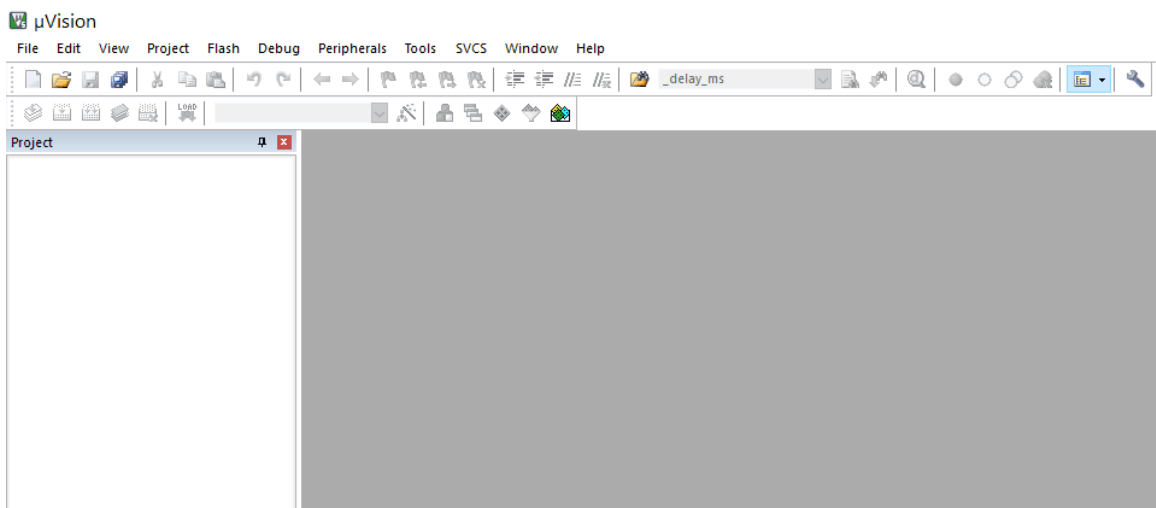
Keil  $\mu$ Vision IDE consists,

- C Compiler - C51.Exe
- Assembler - A51.Exe
- Linker/Locator - BL51.Exe
- Librarian - LIB51.Exe
- Hex Converter - OH51.Exe

Let's develop a simple LED blinking program using Keil  $\mu$ Vision IDE with the C51 compiler. Here we are using the AT89S52 microcontroller from 8051 families.

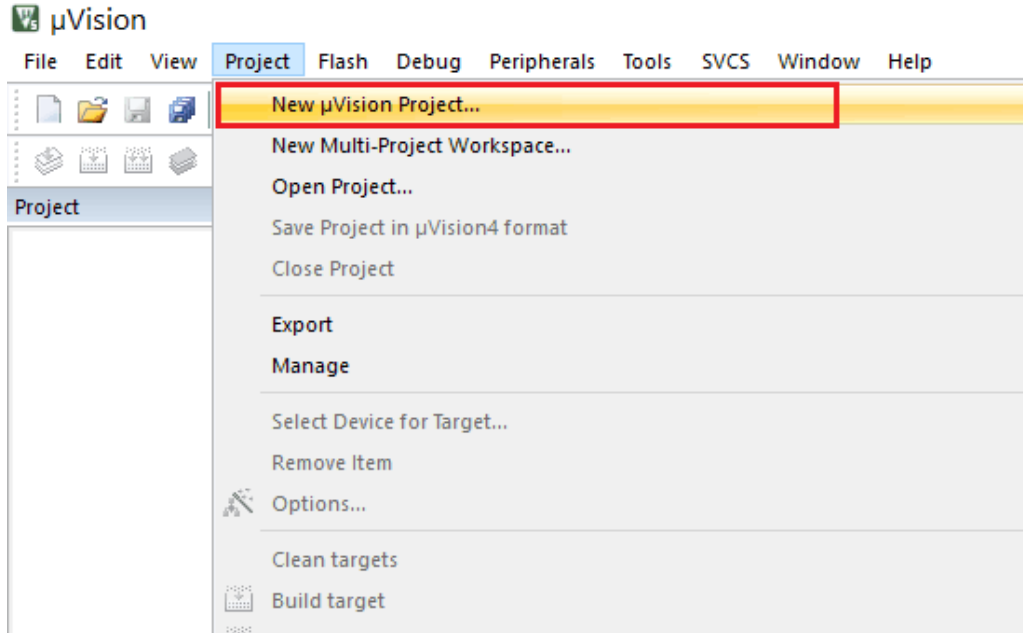
1. Download Keil  $\mu$ Vision and install it.

Now open Keil  $\mu$ Vision

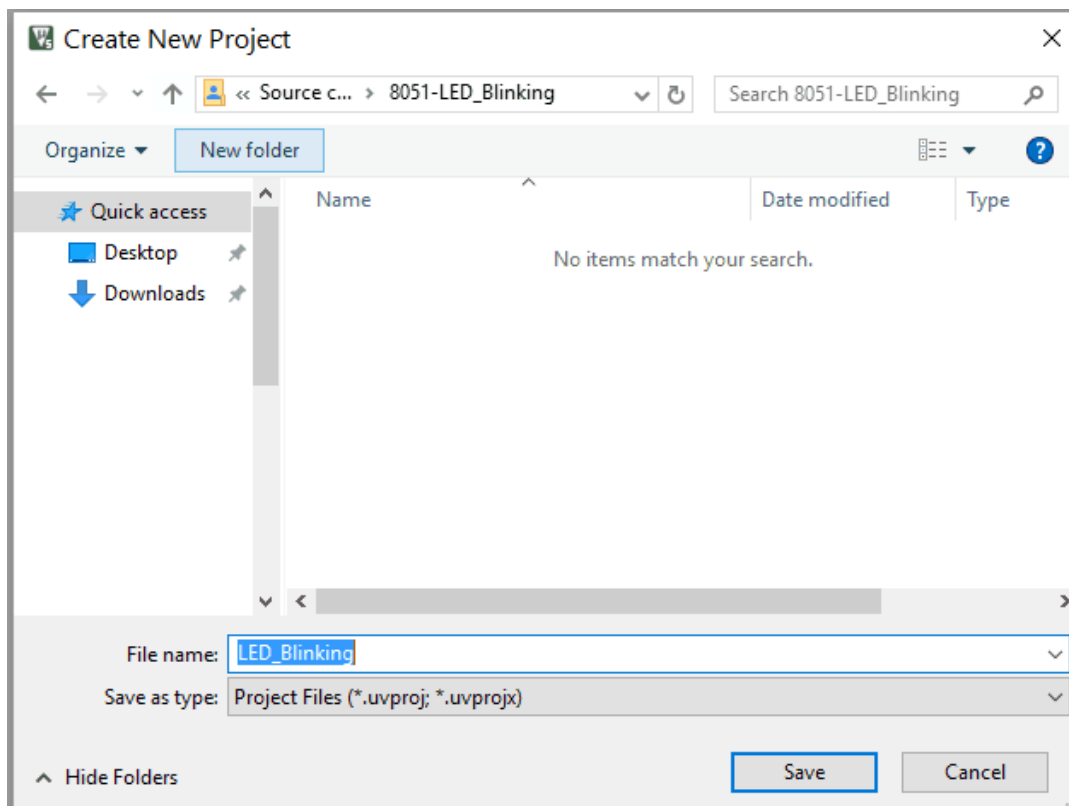




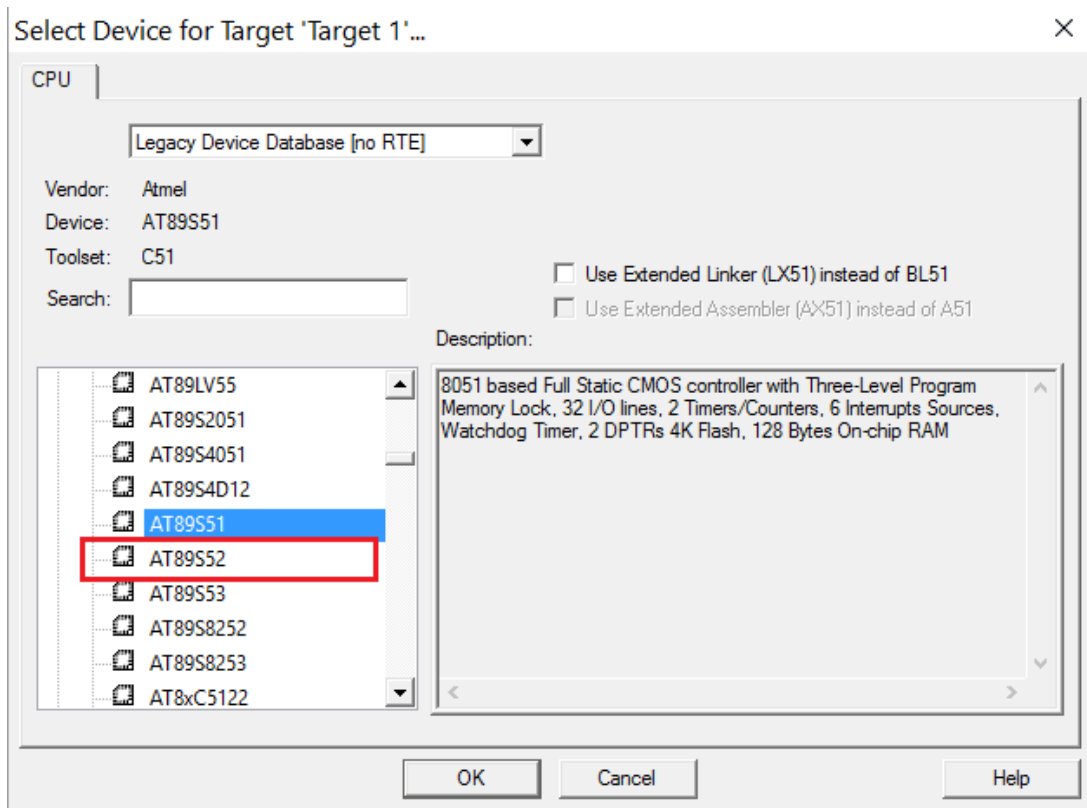
Click on the Project menu and select the new  $\mu$ Vision Project...



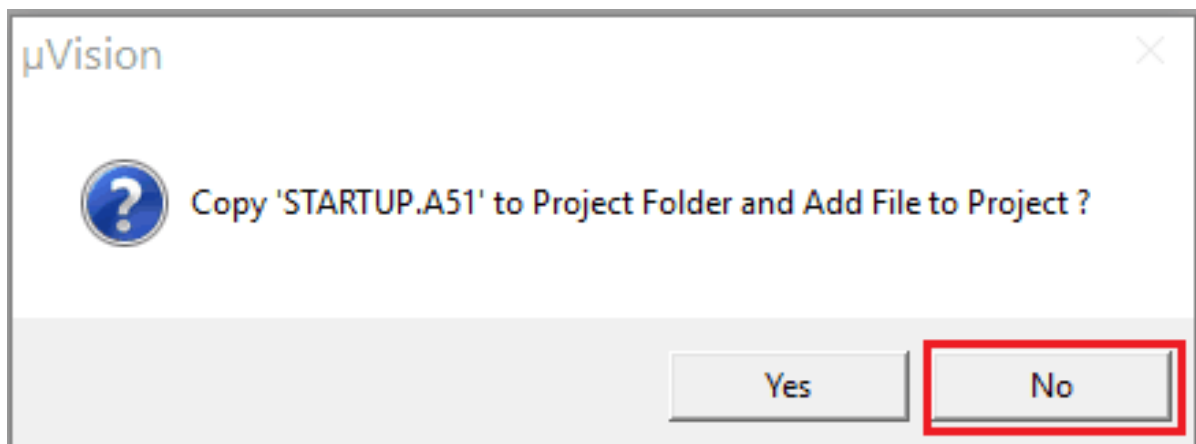
“Create New Project” window will pop up, type a project name and location for the project and save.



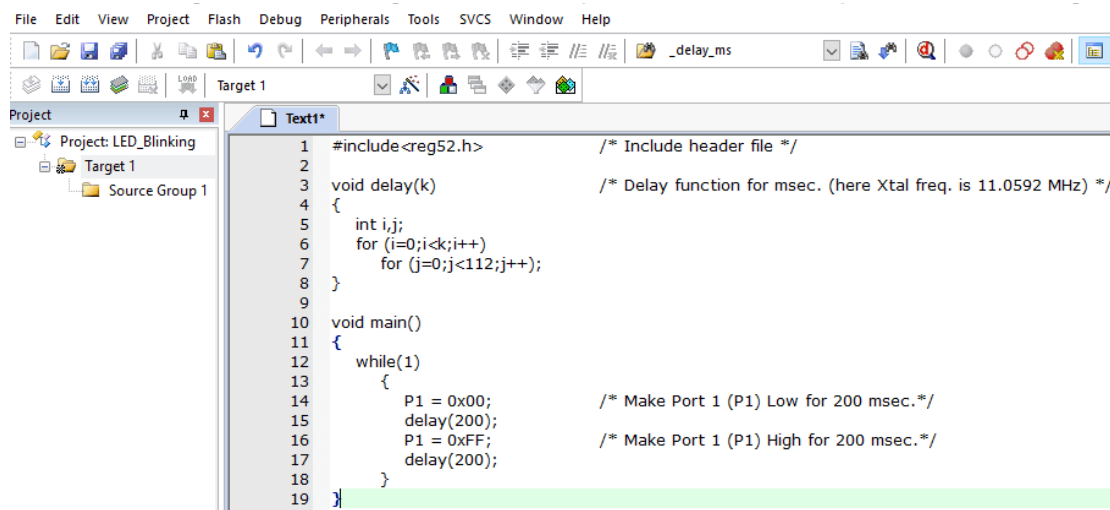
“Select Device for Target” window will pop up, select your device (here we selecting AT89S52)



“μVision” window will ask for copy STARTUP.A51 to the project folder and add a file to the project (here is not necessary so we have selected No)



Now select New file from the File menu and type your program code (here we have typed LED blinking program)



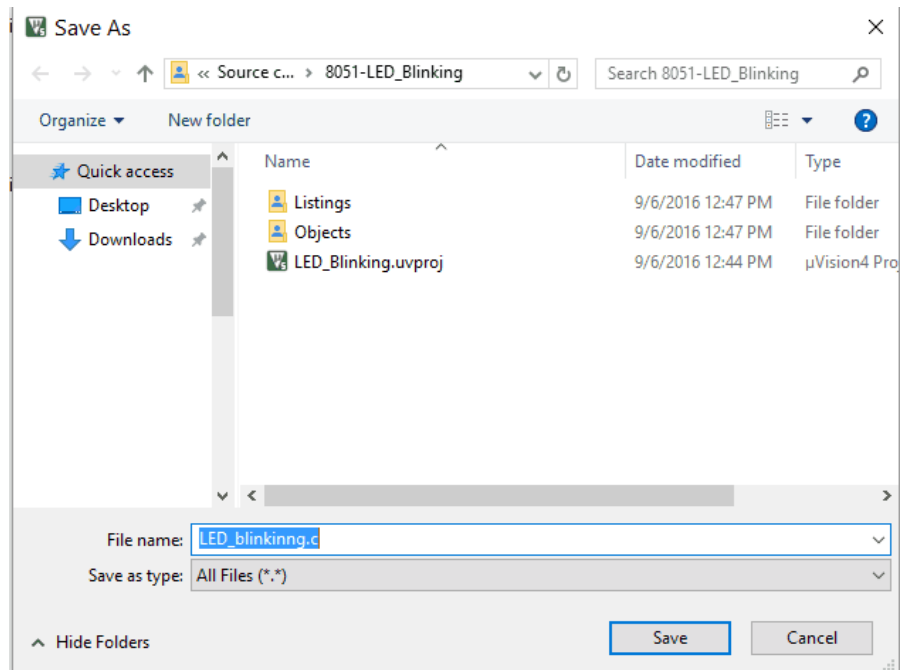
### LED Blinking Program

```

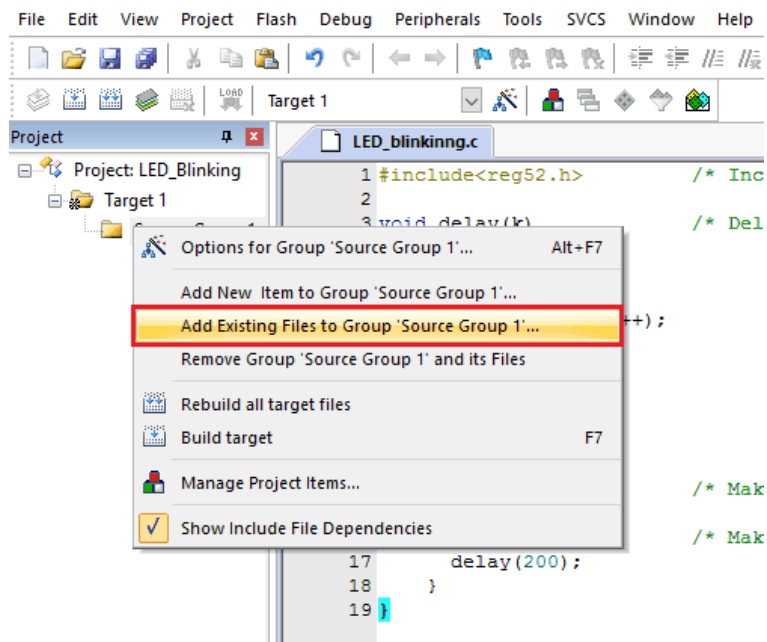
#include<reg52.h>  /* Include header file */
Void delay(k)      /* Delay for msec. (here Xtal freq. is 11.0592 MHz) */
{
    int i,j;
    For (i=0; i<k; i++)
    For (j=0; j<112; j++);
}
Void main () {
    While (1) {
        P1 = 0x00; /* Make Port 1 (P1) Low for 200 msec.*/
        Delay (200);
        P1 = 0xFF; /* Make Port 1 (P1) High for 200 msec.*/
        Delay (200);
    }
}

```

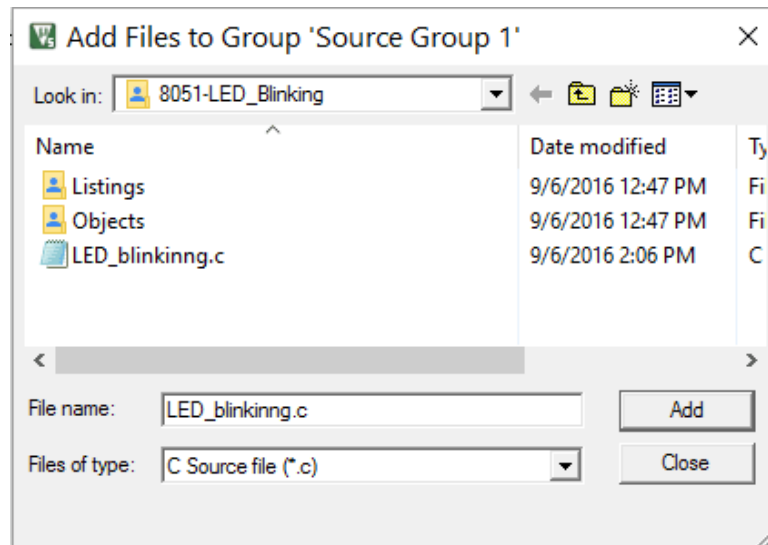
Save program code with “.c “ extension (In case if you are using assembly language then save program code with “.asm “ extension)



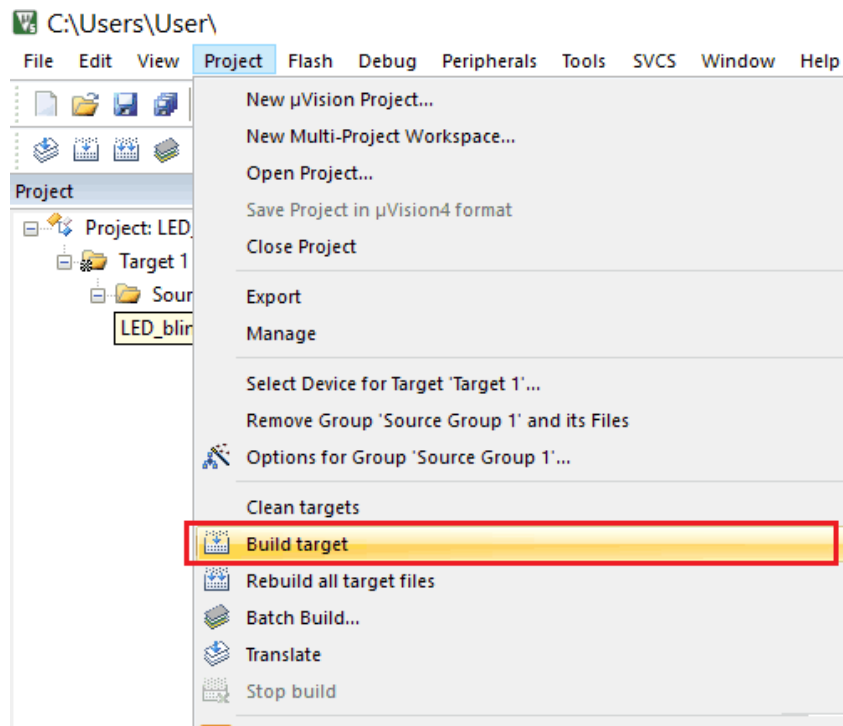
Right-click on Source Group 1 folder from Target 1 and select “Add existing files to Group ‘Source Group 1’”



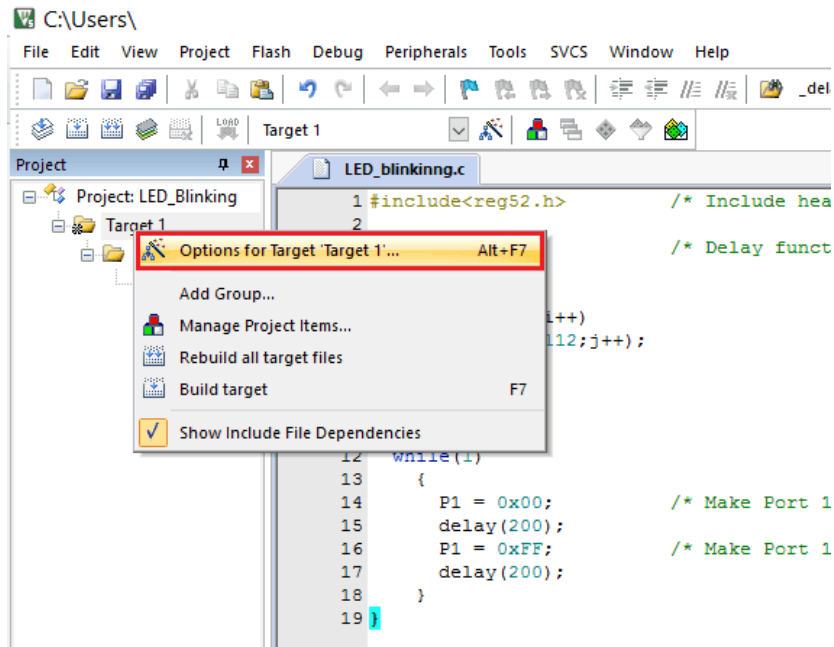
Select the program file saved with “.c “or “.asm” (in case of assembly language) and add it. Then close that window. You can see the added file in the “Source Group 1” folder in the left project window



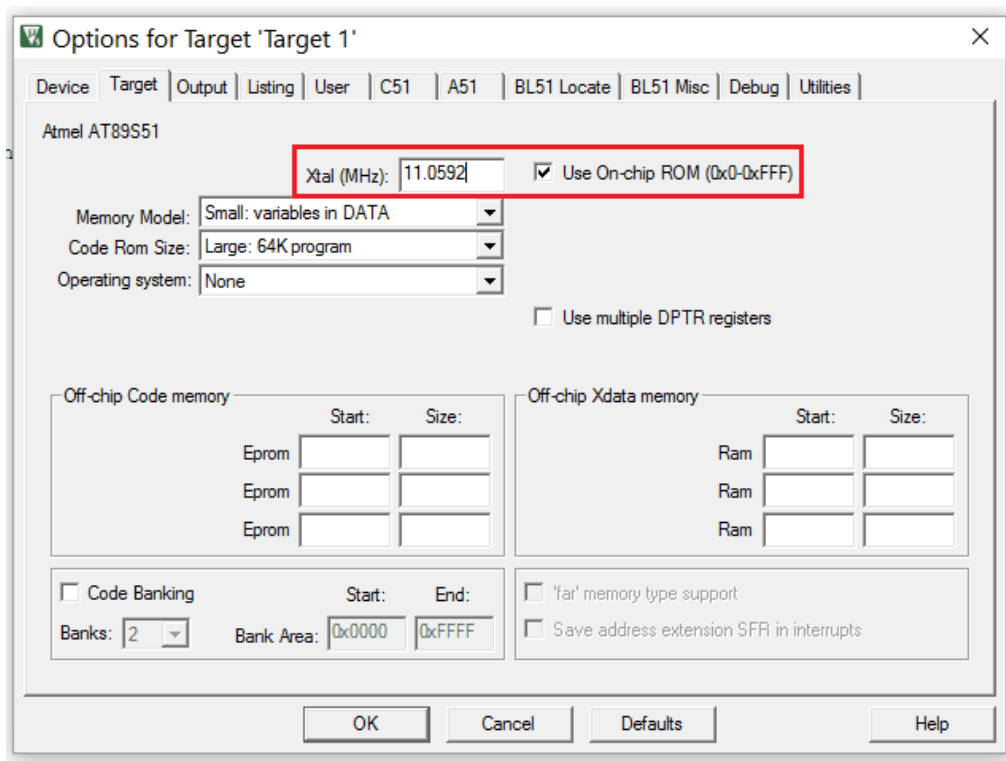
Now select the Project menu and click on “Build target”, it will build a project and give status in the Build output window with Error and Warning count if any.



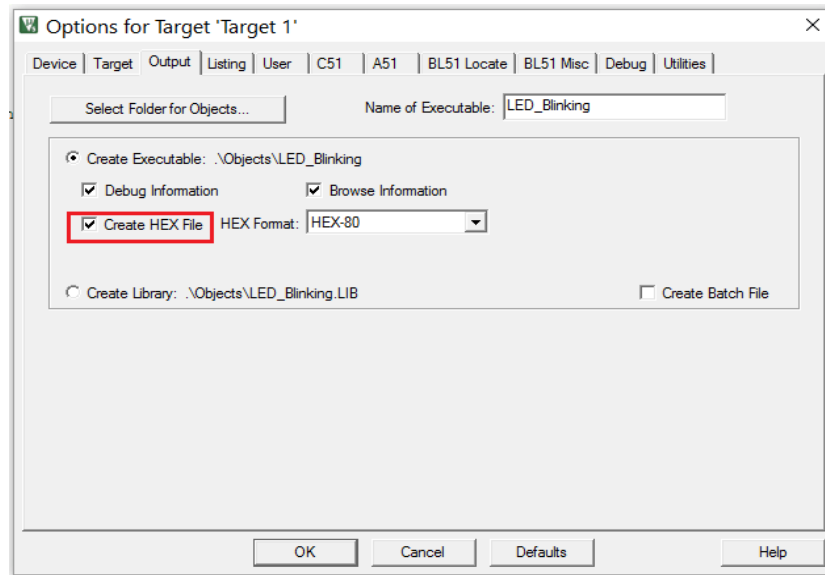
To create a Hex file right click on Target 1 and select Option for Target 'Target 1'



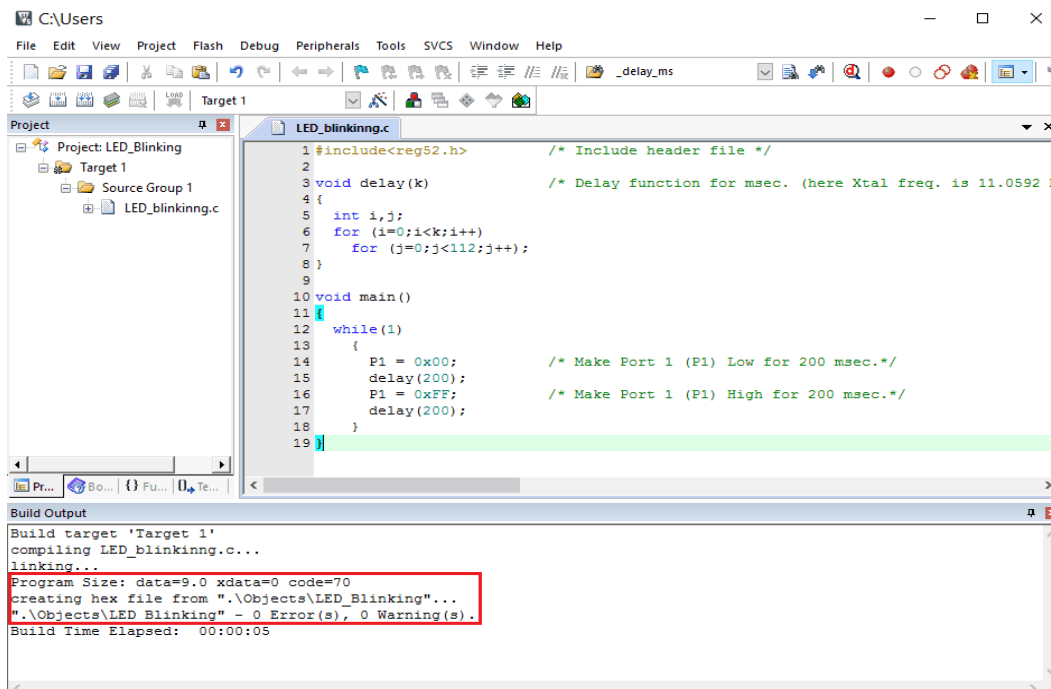
The target window will pop up, enter Xtal (MHz) frequency (here we used 11.0592 MHz), and tick mark in front of the “Use On-chip ROM” tag.



Within the same window select the “Output” option and tick mark in front of the “Create Hex File” tag and click on OK.



Now again select build target from the Project menu or simply hit the F7 shortcut key for the same, it will build target and also create a Hex file. You can see creating a Hex file in the Build output window

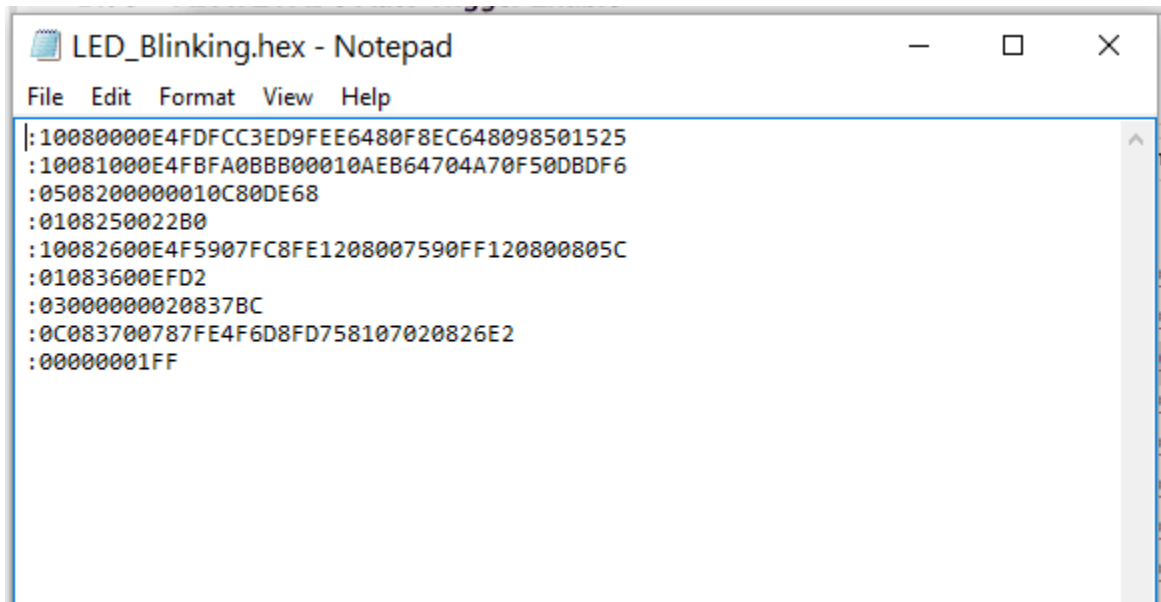


The created Hex file is burned into a microcontroller flash memory using various programming methods based on the programmer or programmer developed by the manufacturer itself.

For example, Flash magic is used for NXP Philips microcontrollers only and is developed by NXP itself. Other manufacturers use serial programmer like an ISP programmer to flash their controllers.

Now load the generated Hex file in one of any programmers available and flash it in your 8051 Microcontroller.

Find below hex file snap.

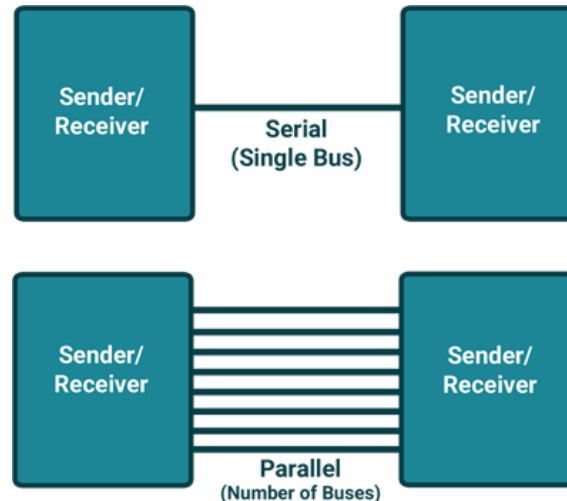


```
LED_Blinking.hex - Notepad
File Edit Format View Help
:10080000E4FDFCC3ED9FEE6480F8EC648098501525
:10081000E4FBFA0B8B00010AEB64704A70F50DBDF6
:0508200000010C80DE68
:0108250022B0
:10082600E4F5907FC8FE1208007590FF120800805C
:01083600EFD2
:03000000020837BC
:0C083700787FE4F6D8FD758107020826E2
:00000001FF
```



# 8051 UART

## Introduction



Serial communication means to transfer data bit by bit serially at a time, whereas in parallel communication, the number of bits that can be transferred at a time depends upon the number of data lines available for communication.

Two methods of serial communication are

- Synchronous Communication: Transfer of bulk data in the framed structure at a time
- Asynchronous Communication: Transfer of a byte data in the framed structure at a time

8051 has built-in UART with RXD (serial data receive pin) and TXD (serial data transmit pin) on PORT3.0 and PORT3.1 respectively.

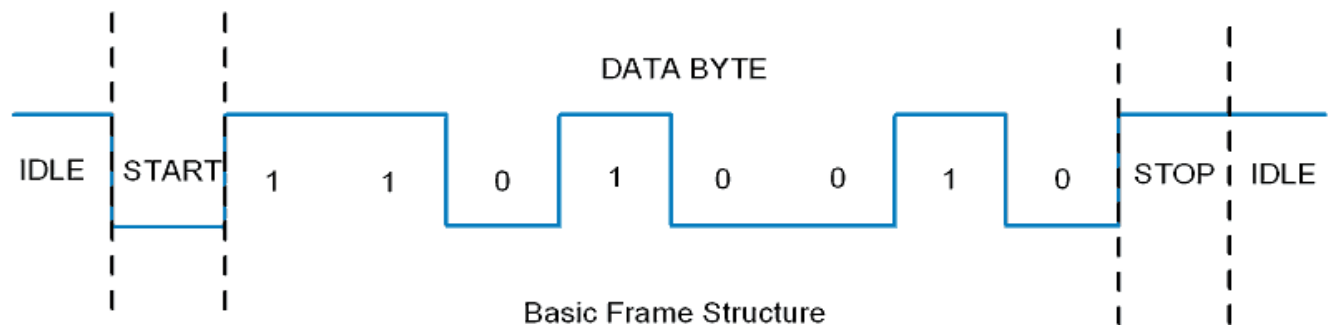
## Asynchronous communication

Asynchronous serial communication is widely used for byte-oriented transmission.

### Frame structure in Asynchronous communication:

- **START bit:** It is a bit with which serial communication starts and it is always low.
- **Data bits packet:** Data bits can be 5 to 9 bits packet. Normally we use 8 data bit packet, which is always sent after the START bit.
- **STOP bit:** This is one or two bits. It is sent after the data bits packet to indicate the end of the frame. The stop bit is always logic high.

In an asynchronous serial communication frame, the first START bit followed by the data byte and at last STOP bit forms a 10-bit frame. Sometimes the last bit is also used as a parity bit.



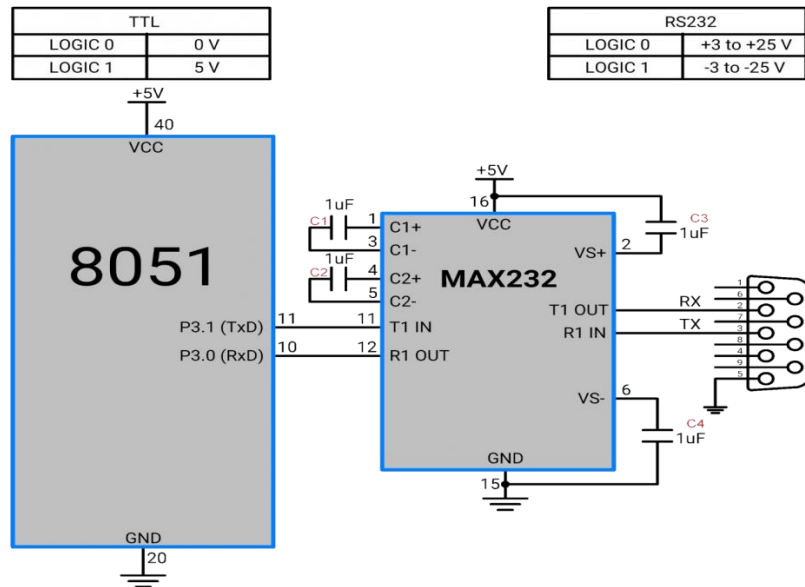
### 8051 Serial Frame Structure

#### Data transmission rate

The data transmission rate is measured in bits per second (bps). In the binary system, it is also called a baud rate (number of signal changes per second). Standard baud rates supported are 1200, 2400, 4800, 19200, 38400, 57600, and 115200. Normally most of the time 9600 bps is used when speed is not a big issue.

#### Interface standard

- 8051 serial communication has TTL voltage level which are 0 v for logic 0 and 5 v for logic 1.
- In computers and most of the old devices for serial communication, RS232 protocol with DB9 connector is used. RS232 serial communication has different voltage levels than 8051 serial communication. i.e. +3 v to +25 v for logic zero and -3 v to -25 v for logic 1.
- So to communicate with RS232 protocol, we need to use a voltage level converter like MAX232 IC.
- Although there are 9 pins in the DB9 connector, we don't need to use all the pins. Only 2nd Tx(Transmit), 3rd Rx(Receive), and 5th GND pin need to be connected.

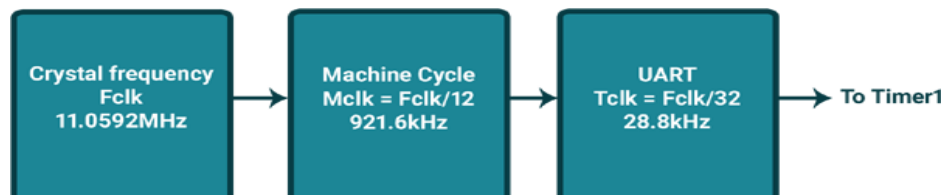


### 8051 Serial Interface

#### 8051 UART Programming

##### Baud Rate calculation:

- To meet the standard baud rates generally crystal with 11.0592 MHz is used.
- As we know, 8051 divides crystal frequency by 12 to get a machine cycle frequency of 921.6 kHz.
- The internal UART block of 8051 divides this machine cycle frequency by 32, which gives the frequency of 28800 Hz which is used by UART.
- To achieve a baud rate of 9600, again 28800 Hz frequency should be divided by 3.
- This is achieved by using Timer1 in mode-2 (auto-reload mode) by putting 253 in TH1 (8-bit reg.)
- So 28800 Hz will get divided by 3 as the timer will overflow after every 3 cycles.
- we can achieve different baud rates by putting the division factor in the TH1 register.



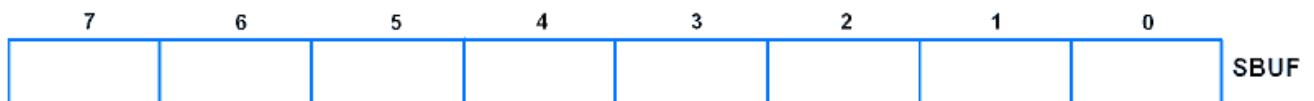
### Division factor to achieve different baud rates

| Baud Rate | TH1 (Hex) |
|-----------|-----------|
| 9600      | FD        |
| 4800      | FA        |
| 2400      | F4        |
| 1200      | E8        |

### Serial communication Registers

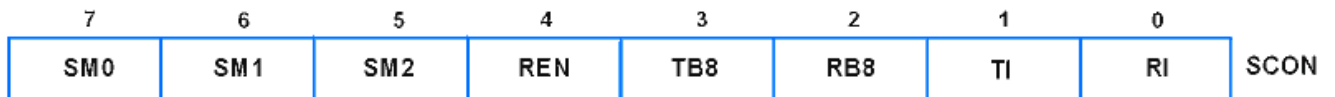
#### SBUF: Serial Buffer Register

This is the serial communication data register used to transmit or receive data through it.



#### SCON: Serial Control Register

Serial control register SCON is used to set serial communication operation modes. Also it is used to control transmit and receive operations.



#### Bit 7:6 - SM0:SM1: Serial Mode Specified

| Mode | SM0 | SM1 | Mode  |
|------|-----|-----|---|
| 0    | 0   | 0   | 1/12 of Osc frequency shift register mode fixed baud rate |
| 1    | 0   | 1   | 8-bit UART with timer 1 determined baud rate              |
| 2    | 1   | 0   | 9-bit UART with 1/32 of Osc fixed baud rate               |
| 3    | 1   | 1   | 9-bit UART with timer 1 determined baud rate              |

Normally mode-1 (SM0 =0, SM1=1) is used with 8 data bits, 1 start bit, and 1 stop bit.

**Bit 5 - SM2:** for Multiprocessor Communication

This bit enables a multiprocessor communication feature in mode 2 & 3.

**Bit 4 - REN:** Receive Enable

1 = Receive enable

0 = Receive disable

**Bit 3 - TB8:** 9th Transmit Bit

This is the 9th bit which is to be transmitted in mode 2 & 3 (9-bit mode)

**Bit 2 - RB8:** 9th Receive Bit

This is the 9th received bit in mode 2 & 3 (9-bit mode), whereas in mode 1 if SM2 = 0 then RB8 hold the stop bit that received

**Bit 1 - TI:** Transmit Interrupt Flag

This bit indicates the transmission is complete and gets set after transmitting the byte from the buffer. Normally TI (Transmit Interrupt Flag) is set by hardware at the end of the 8th bit in mode 0 and at the beginning of stop bit in other modes.

**Bit 0 – RI:** Receive Interrupt Flag

This bit indicates reception is complete and gets set after receiving the complete byte in the buffer. Normally RI (Receive Interrupt Flag) is set by hardware in receiving mode at the end of the 8th bit in mode 0 and at the stop bit receive time in other modes.

**Programming steps**

1. Configure Timer 1 in auto-reload mode.
2. Load TH1 with value as per required baud rate e.g. for 9600 baud rate load 0xFD. (-3 in decimal)
3. Load SCON with serial mode and control bits. e.g. for mode 1 and enable reception, load 0x50.
4. Start timer1 by setting TR1 bit to 1.
5. Load transmitting data in the SBUF register.
6. Wait until loaded data is completely transmitted by polling the TI flag.
7. When the TI flag is set, clear it, and repeat from step 5 to transmit more data.

## Example

Let's Program 8051 (here AT89C51) to send character data “test” serially at 9600 baud rate in mode 1    Program for serial data transmit

```
#include <reg51.h>          /* Include x51 header file */
void UART_Init()
{
    TMOD = 0x20;           /* Timer 1, 8-bit auto reload mode */
    TH1 = 0xFD;            /* Load value for 9600 baud rate */
    SCON = 0x50;           /* Mode 1, reception enable */
    TR1 = 1;               /* Start timer 1 */
}
void Transmit_data(char tx_data)
{
    SBUF = tx_data;        /* Load char in SBUF register */
    while (TI==0);         /* Wait until stop bit transmit */
    TI = 0;                /* Clear TI flag */
}
void String(char *str)
{
    int i;
    for(i=0;str[i]!=0;i++) /* Send each char of string till the NULL */
    {
        Transmit_data(str[i]); /* Call transmit data function */
    }
}
void main()
{
    UART_Init();           /* UART initialize function */
    String("test");        /* Transmit 'test' */
    while(1);
}
```

## 8051 serial interrupt

8051 UART has a serial interrupt. Whenever data is transmitted or received, serial interrupt flags TI and RI are activated respectively.

8051 serial interrupt has a vector address (0023H) where it can jump to serve ISR (Interrupt service routine) if the global and serial interrupt is enabled.

Let's see how the serial interrupt routine will be used in serial communication programming.

### Programming steps

1. Set timer 1 in auto-reload mode.
2. Load TH1 with value as per required baud rate e.g. for 9600 baud rate load 0xFD.
3. Load SCON with serial mode and control bits. e.g. for mode 1 and enable reception load 0x50.
4. Start timer1 by setting TR1 bit to 1.
5. Enable Global and serial interrupt bit, i.e. EA = 1 and ES = 1.
6. Now whenever data is received or transmitted, the interrupt flag will set and the controller will jump to serial ISR.
7. Note that, TI/RI flag must be cleared by software in ISR.

Note: For transmission and reception interrupt, the same interrupt vector address is assigned, so when the controller jumps to the ISR, we have to check whether it is Tx interrupt or Rx interrupt by TI and RI bits status.

### Example

Let's Program 8051 (here AT89C51) to receive character data serially at 9600 baud rate in mode 1 and send received data on port 1 using a serial interrupt.

- In this example, after a byte is received, the RI flag is set which will generate a serial interrupt.
- After the interrupt, 8051 will jump to the serial ISR routine.
- In the ISR routine, we will send data to port 1.

Program for serial interrupt

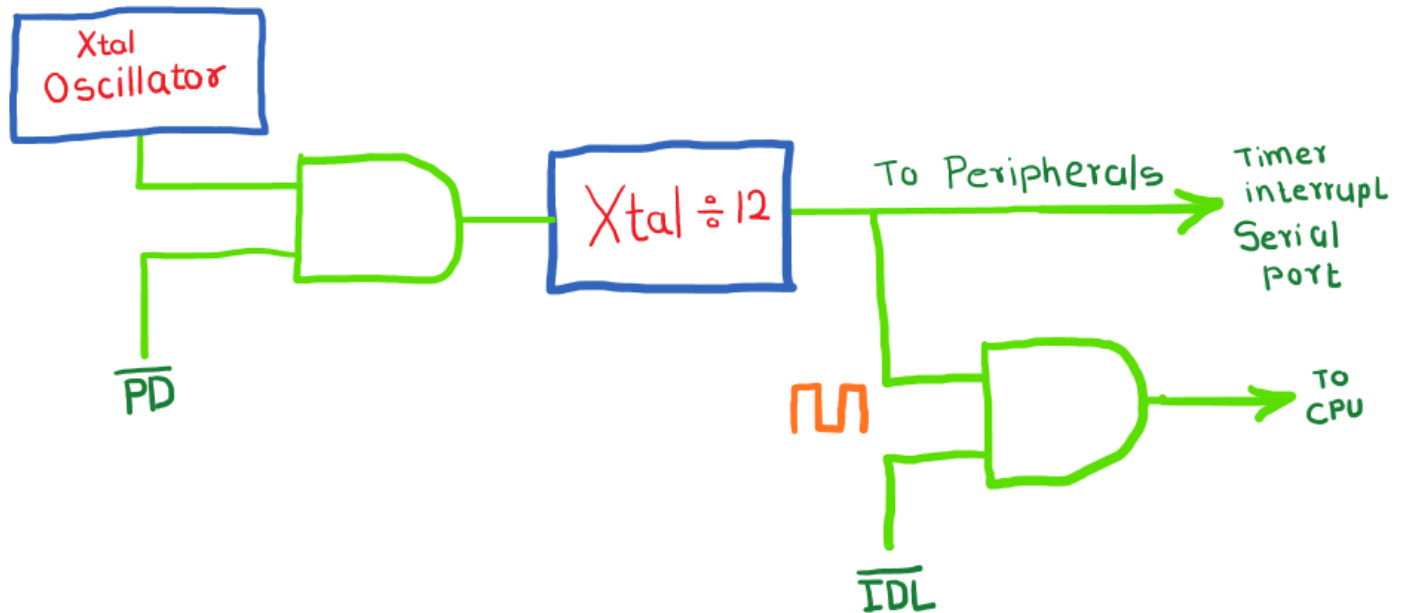
```
#include <reg51.h>          /* Include x51 header file */
void Ext_int_Init()
{
    EA = 1;                 /* Enable global interrupt */
    ES = 1;                 /* Enable serial interrupt */
}
void UART_Init()
{
    TMOD = 0x20;            /* Timer 1, 8-bit auto reload mode */
    TH1 = 0xFD;             /* Load value for 9600 baud rate */
    SCON = 0x50;            /* Mode 1, reception enable */
    TR1 = 1;               /* Start timer 1 */
}
void Serial_ISR() interrupt 4
{
    P1 = SBUF;              /* Give received data on port 1 */
    RI = 0;                 /* Clear RI flag */
}
void main()
{
    P1 = 0x00;              /* Make P1 output */
    Ext_int_Init();         /* Call Ext. interrupt initialize */
    UART_Init();
    while(1);
}
```



# 8051 Power down and idle mode

## Introduction

Power down and idle mode features are used to save power in microcontrollers. 8051 has an inbuilt power-saving feature which is useful in embedded applications where power consumption is the main constraint.



**8051 Power Control Logic**

8051 has two power-saving modes,

- Power Down Mode
- Idle Mode

## Difference between Power down & Idle Mode

As shown in the above figure of 8051 power control logic, two control bits are there, IDL and PD, which are used for Idle and Power-down mode respectively.

In **Power Down** mode, the oscillator clock provided to the system is OFF i.e. CPU and peripherals clock remains inactive in this mode.

In **Idle Mode**, only the clock provided to the CPU gets deactivated, whereas the peripherals clock will remain active in this mode.

Hence power saved in power-down mode is more than in idle mode.

The below table shows the power supply current required for 8051 family controllers in Normal (Active), Idle and Power-down mode.

| <b>8051<br/>Controllers</b> | <b>Operating<br/>Oscillator<br/>Frequency<br/><br/>Fosc.</b> | <b>Current required<br/>in Normal mode</b> | <b>Current<br/>required in Idle<br/>mode</b> | <b>Current required<br/>in Power Down<br/>mode</b>         |
|-----------------------------|--|--|--|--|
| AT89S51                     | 12 MHz   | 25mA                                       | 6.5mA  | 50uA   |
| P89V51RD2                   | 12 MHz   | 11.5mA                                     | 8.5mA  | 80-90uA  |
| DS80C323                    | 18 MHz   | 10mA                                       | 6mA  | 0.1uA for BGR<br>enabled.<br><br>40uA for BGR<br>disabled. |

As per the above table, it is clear that power consumption in power-down mode is less than in Normal or idle mode.

8051 has a power control register for power control. Let's see the power control register.

#### **PCON Register: Power control register**

PCON (Power Control) register is used to force the 8051 microcontrollers into power-saving mode. The power control register of 8051 contains two power-saving mode bits and one serial baud rate control bit.

| 7    | 6   | 5   | 4   | 3   | 2   | 1  | 0   |      |
|------|-----|-----|-----|-----|-----|----|-----|------|
| SMOD | --- | --- | --- | GF1 | GF0 | PD | IDL | PCON |

#### **Bit 7 – SMOD**

**1** = Baud rate is doubled in UART mode 1, 2 and 3.

**0** = No effect on Baud rate.

#### **Bit 3:2 – GF1 & GF0:**

These are general purpose bit for user.

#### **Bit 1 – PD: Power Down**

**1** = Enable Power-Down mode. In this mode, the Oscillator clock turned OFF and both CPU and peripherals clock stopped. Hardware reset can cancel this mode.

**0** = Disable Power-down mode.

#### **Bit 0 – IDL: Idle**

**1** = Enable Idle mode. CPU clock turned off whereas internal peripheral module such as a timer, serial port, interrupts works normally. Interrupt and H/W reset can cancel this mode.

**0** = Disable Idle mode.

#### Example

Let's program the AT89C51 microcontroller to toggle Pin 0 of port 1 and force the microcontroller in Idle (sleep) mode by external interrupt1. We wake the microcontroller in normal mode by external interrupt0.

#### Programming steps

1. Enable global and external0 and external1 interrupt. EA = 1, EXx = 1.
2. Select interrupt type i.e. here interrupt on falling edge is selected ITx = 0.
3. Set priority for interrupt if required through the IP register.
4. The controller sleep mode can also be canceled by reset pin.
5. Enable and disable idle (sleep) mode through the PCON register.

Here, sleep mode is enabled in external1 interrupt ISR and disabled in external0 interrupt ISR.

Program for Idle (sleep) mode

```
#include <reg51.h>          /* Include x51 header file */
sbit test = P1^0;
void delay(k)               /* mSecond Delay function for Xtal 11.0592 MHz */
{
    int i,j;
    for (i=0;i<k;i++)
        for (j=0;j<112;j++);
}
void ExtInt_Init()          /* External interrupt initialize */
{
    IT0 = 1;                /* Interrupt0 on falling edge */
    EX0 = 1;                /* Enable External interrupt0 */
    IT1 = 1;                /* Interrupt1 on falling edge */
    EX1 = 1;                /* Enable External interrupt1 */
    EA = 1;                 /* Enable global interrupt */
    IP = 0x01;              /* Set highest priority for Ext. interrupt0 */
}
void External0_ISR() interrupt 0 /* External int0 ISR */
{
    PCON = 0x00;            /* Disable Idle & Power Down mode */
}
void External1_ISR() interrupt 2 /* External int1 ISR */
{
    PCON = 0x01;            /* Enable Idle mode */
                           /* Enable Power Down mode by PCON = 0x02; */
}
void main()
{
    ExtInt_Init();
    while(1)                /* Toggle P1.0 continuous */
    {
        test = 0;
        delay(30);
        test = 1;
        delay(30);
    }
}
```

### Power down Mode:

To enable Power-down mode, set PD bit i.e. PCON = 0x02. Also, note that only hardware reset can cancel this mode.

Note: according to Intel's MCS-51 family user manual:

“The only exit from Power Down for the 80C51 is hardware reset. Reset redefines all the SFRs, but does not change the on-chip RAM.”

But according to AT89s51 datasheet:

“Exit from Power-down mode can be initiated either by a hardware reset or by activation of an enabled external interrupt (INT0 or INT1). Reset redefines the SFRs but does not change the on-chip RAM.”

So AT89s51 controller can exit from power-down mode by reset as well as by external interrupts also.

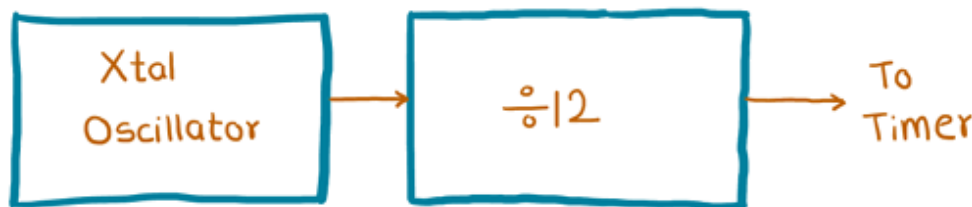
# 8051 Timers

## Introduction

8051 microcontrollers have two timers/counters which work on the clock frequency. Timer/counter can be used for time delay generation, counting external events, etc.

## Clock

Every Timer needs a clock to work, and 8051 provides it from an external crystal which is the main clock source for Timer. The internal circuitry in the 8051 microcontrollers provides a clock source to the timers which are 1/12th of the frequency of crystal attached to the microcontroller, also called Machine cycle frequency.



**8051 Timer Clock**

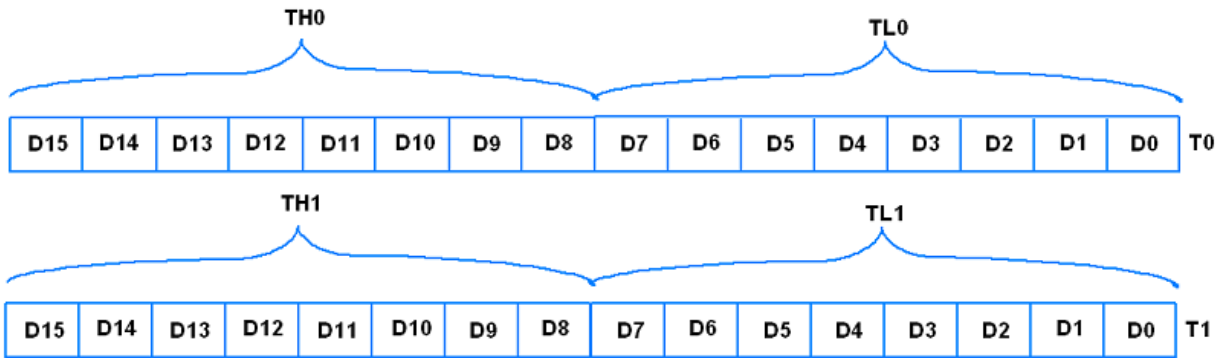
For example, suppose we have a crystal frequency of 11.0592 MHz then the microcontroller will provide 1/12th i.e.

$$\text{Timer clock frequency} = (\text{Xtal Osc.frequency})/12 = (11.0592 \text{ MHz})/12 = 921.6 \text{ KHz}$$

$$\text{Period } T = 1 / (921.6 \text{ kHz}) = 1.085 \mu\text{S}$$

## Timer

8051 has two timers Timer0 (T0) and Timer1 (T1), both are 16-bit wide. Since 8051 has 8-bit architecture, each of these is accessed by two separate 8-bit registers as shown in the figure below. These registers are used to load timer count.

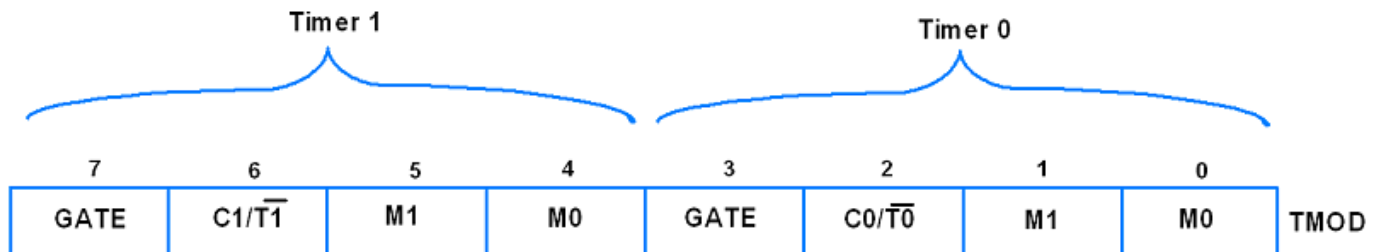


8051 has a Timer Mode Register and Timer Control Register for selecting a mode of operation and controlling purpose.

Let's see these registers,

### TMOD register

TMOD is an 8-bit register used to set timer mode of timer0 and timer1.



Its lower 4 bits are used for Timer0 and the upper 4 bits are used for Timer1

#### Bit 7,3 – GATE:

**1** = Enable Timer/Counter only when the INT0/INT1 pin is high and TR0/TR1 is set.

**0** = Enable Timer/Counter when TR0/TR1 is set.

#### Bit 6,2 - C/T (Counter/Timer): Timer or Counter select bit

**1** = Use as Counter

**0** = Use as Timer

#### Bit 5:4 & 1:0 - M1:M0: Timer/Counter mode select bit

These are Timer/Counter mode select bit as per the below table

| M1 | M0 | Mode                       | Operation  |
|----|----|----------------------------|--|
| 0  | 0  | 0 (13-bit timer mode)      | 13-bit timer/counter, 8-bit of THx & 5-bit of TLx  |
| 0  | 1  | 1 (16-bit timer mode)      | 16-bit timer/counter, THx cascaded with TLx  |
| 1  | 0  | 2 (8-bit auto-reload mode) | 8-bit timer/counter (auto-reload mode), TLx reload with the value held by THx each time TLx overflow |
| 1  | 1  | 3 (split timer mode)       | Split the 16-bit timer into two 8-bit timers i.e. THx and TLx like two 8-bit timer                   |

### TCON Register

|     |     |     |     |     |     |     |     |      |
|-----|-----|-----|-----|-----|-----|-----|-----|------|
| 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |      |
| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 | TCON |

TCON is an 8-bit control register and contains a timer and interrupt flags.

#### Bit 7 - TF1: Timer1 Overflow Flag

**1** = Timer1 overflow occurred (i.e. Timer1 goes to its max and roll over back to zero).

**0** = Timer1 overflow not occurred.

It is cleared through software. In the Timer1 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

#### Bit 6 - TR1: Timer1 Run Control Bit

**1** = Timer1 start.

**0** = Timer1 stop.

It is set and cleared by software.

#### Bit 5 – TF0: Timer0 Overflow Flag

**1** = Timer0 overflow occurred (i.e. Timer0 goes to its max and roll over back to zero).

**0** = Timer0 overflow not occurred.

It is cleared through software. In the Timer0 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.



**Bit 4 – TR0: Timer0 Run Control Bit**

**1** = Timer0 start.

**0** = Timer0 stop.

It is set and cleared by software.

**Bit 3 - IE1: External Interrupt1 Edge Flag**

**1** = External interrupt1 occurred.

**0** = External interrupt1 Processed.

It is set and cleared by hardware.

**Bit 2 - IT1: External Interrupt1 Trigger Type Select Bit**

**1** = Interrupt occurs on falling edge at INT1 pin.

**0** = Interrupt occur on a low level at the INT1 pin.

**Bit 1 – IE0: External Interrupt0 Edge Flag**

**1** = External interrupt0 occurred.

**0** = External interrupt0 Processed.

It is set and cleared by hardware.

**Bit 0 – IT0: External Interrupt0 Trigger Type Select Bit**

**1** = Interrupt occurs on falling edge at INT0 pin.

**0** = Interrupt occur on a low level at INT0 pin.

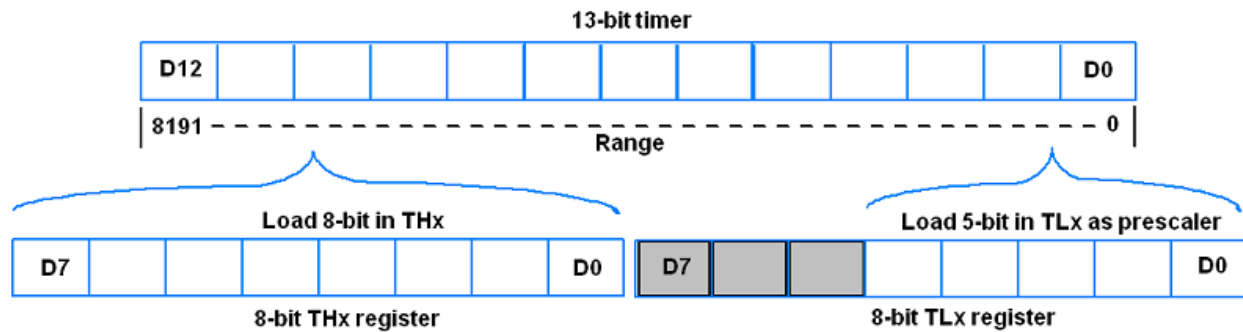
Let's see the timer's modes

**Timer Modes**

Timers have their operation modes which are selected in the TMOD register using M0 & M1 bit combinations.

Mode 0 (13-bit timer mode)

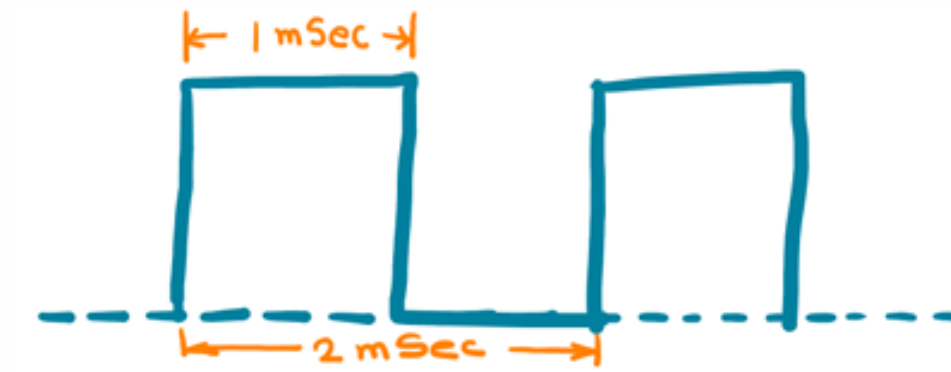
Mode 0 is a 13-bit timer mode for which 8-bit of THx and 5-bit of TLx (as Prescaler) are used. It is mostly used for interfacing possible with old MCS-48 family microcontrollers.



As shown in the above figure, 8-bit of THx and lower 5-bit of TLx used to form a total 13-bit timer. Higher 3-bits of TLx should be written as zero while using timer mode0, or it will affect the result.

### Example

Let's generate a square wave of 2mSec period using an AT89C51 microcontroller with timer0 in mode0 on the P1.0 pin of port1. Assume xtal oscillator frequency of 11.0592 MHz.



As the Xtal oscillator frequency is 11.0592 MHz we have a machine cycle of 1.085uSec. Hence, the required count to generate a delay of 1mSec. is,

$$\text{Count} = (1 \times 10^{-3}) / (1.085 \times 10^{-6}) \approx 921$$

The maximum count of Mode0 is  $2^{13}$  (0 - 8191) and the Timer0 count will increment from 0 – 8191. So we need to load value which is 921 less from its maximum count i.e. 8191. Also, here in the below program, we need an additional 13 MC (machine cycles) from call to return of delay function. Hence value needed to be loaded is,

$$\text{Value} = (8191 - \text{Count}) + \text{Function\_MCycles} + 1 = 7284 = 0x1C74$$

So we need to load 0x1C74 value in Timer0.

1C74 = 0001 **1100 0111 0100** b, now load lower 5-bit in TL0 and next 8-bit in TH0

so here we get,

TL0 = 0001 **0100** = 0x14 and TH0 = 1110 0011 = 0xE3

Programming steps for delay function

1. Load Tmod register value i.e. TMOD = 0x00 for Timer0/1 mode0 (13-bit timer mode).
2. Load calculated THx value i.e. here TH0 = 0xE3.
3. Load calculated TLx value i.e. here TL0 = 0x14.
4. Start the timer by setting a TRx bit. i.e. here TR0 = 1.
5. Poll TFx flag till it does not get set.
6. Stop the timer by clearing TRx bit. i.e. here TR0 = 0.
7. Clear timer flag TFx bit i.e. here TF0 = 0.
8. Repeat from step 1 to 7 for the delay again.

Program for timer mode0

```
#include <reg51.h>                /* Include x51 header file */

sbit test = P1^0;                 /* set test pin 0 of port1 */

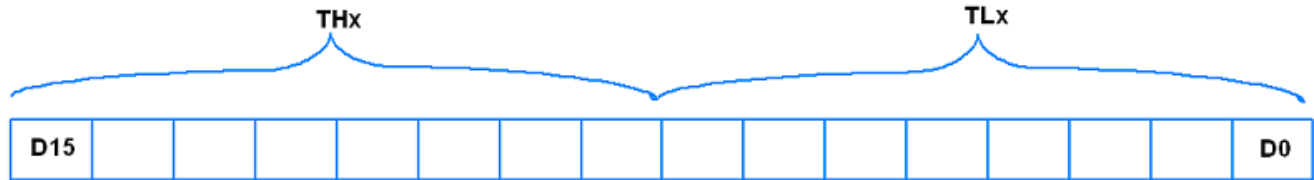
void timer_delay()                /* Timer0 delay function */
{
    TH0 = 0xE3;                   /* Load 8-bit in TH0 (here Timer0 used) */
    TL0 = 0x14;                   /* Load 5-bit in TL0 */

    TR0 = 1;                      /* Start timer0 */
    while(TF0 == 0);              /* Wait until timer0 flag set */
    TR0 = 0;                      /* Stop timer0 */
    TF0 = 0;                      /* Clear timer0 flag */
}

void main()
{
    TMOD = 0x00;                  /* Timer0/1 mode0 (13-bit timer mode) */
    while(1)
    {
        test = ~test;            /* Toggle test pin */
        timer_delay();           /* Call timer0 delay */
    }
}
```

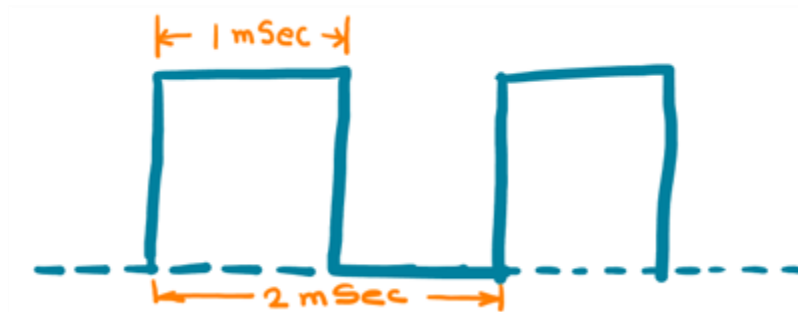
### Model (16-bit timer mode)

Mode 1 is a 16-bit timer mode used to generate a delay, it uses 8-bit of THx and 8-bit of TLx to form a total 16-bit register.



### Example

Let's generate a square wave of 2mSec time period using an AT89C51 microcontroller with timer0 in mode1 on the P1.0 pin of port1. Assume Xtal oscillator frequency of 11.0592 MHz.



As Xtal is 11.0592 MHz we have a machine cycle of 1.085uSec.

Hence, the required count to generate a delay of 1mSec. is,

$$\text{Count} = (1 \times 10^{-3}) / (1.085 \times 10^{-6}) \approx 921$$

And mode1 has a max count is  $2^{16}$  (0 - 65535) and it increments from 0 to 65535 so we need to load value which is 921 less from its max. count i.e. 65535. Also, here in the below program, we need an additional 13 MC (machine cycles) from call to return of delay function. Hence value needed to be loaded is,

$$\text{Value} = (65535 - \text{Count}) + \text{Function\_MCycles} + 1 = 64615 = (\text{FC74})_{\text{Hex}}$$

So we need to load FC74 Hex value higher byte in TH0 and lower byte in TL0 as,

TH0 = 0xFC & TL0 = 0x74

### Programming steps for delay function

1. Load Tmod register value i.e. TMOD = 0x01 for Timer0 mode1 (16-bit timer mode).
2. Load calculated THx value i.e. here TH0 = 0xFC.
3. Load calculated TLx value i.e. here TL0 = 0x74.
4. Start the timer by setting a TRx bit. i.e. here TR0 = 1.
5. Poll TFx flag till it does not get set.
6. Stop the timer by clearing TRx bit. i.e. here TR0 = 0.
7. Clear timer flag TFx bit i.e. here TF0 = 0.
8. Repeat from step 1 to 7 for the delay again.

Program for timer mode1

```
#include <reg51.h>          /* Include x51 header file */

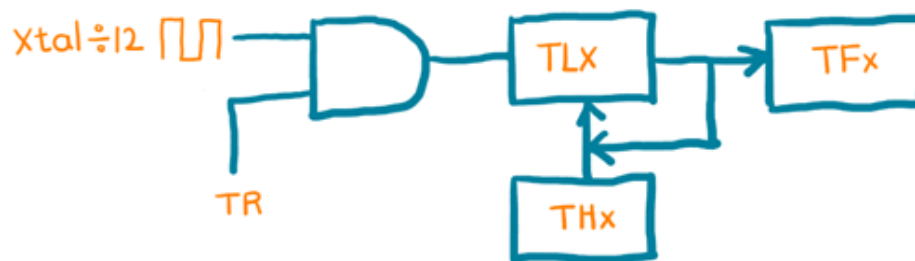
sbit test = P1^0;          /* set test pin0 of port1 */

void timer_delay()          /* Timer0 delay function */
{
    TH0 = 0xFC;             /* Load higher 8-bit in TH0 */
    TL0 = 0x74;             /* Load lower 8-bit in TL0 */
    TR0 = 1;                /* Start timer0 */
    while(TF0 == 0);        /* Wait until timer0 flag set */
    TR0 = 0;                /* Stop timer0 */
    TF0 = 0;                /* Clear timer0 flag */
}

void main()
{
    TMOD = 0x01;            /* Timer0 mode1 (16-bit timer mode) */
    while(1)
    {
        test = ~test;       /* Toggle test pin */
        timer_delay();      /* Call timer0 delay */
    }
}
```

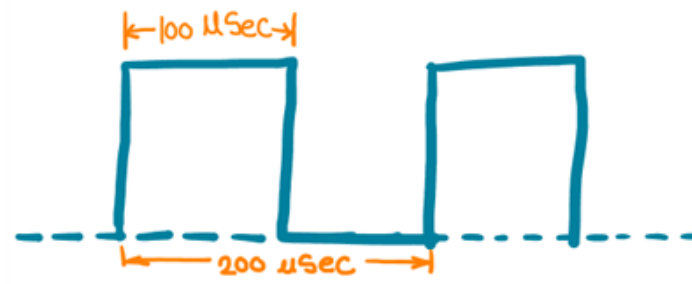
### Mode2 (8-bit auto-reload timer mode)

Mode 2 is an 8-bit auto-reload timer mode. In this mode, we have to load the THx-8 bit value only. When the Timer gets started, the THx value gets automatically loaded into the TLx and TLx starts counting from that value. After the value of TLx overflows from the 0xFF to 0x0, the TFX flag gets set and again value from the THx gets automatically loaded into the TLx register. That's why this is called the auto-reload mode.



### Example

Here we are generating a square wave on PORT1.0 with 200µSec. time period using Timer1 in mode2. We will use 11.0592 MHz Xtal oscillator frequency.



As Xtal is 11.0592 MHz we have a machine cycle of 1.085µSec. Hence, the required count to generate a delay of 1mSec. is,

$$\text{Count} = (100 \times 10^{-6}) / (1.085 \times 10^{-6}) \approx 92$$

And mode2 has a max count is  $2^8$  (0 - 255) and it increment from 0 – 255 so we need to load value which is 92 less from its max. count i.e. 255. Hence value need to be load is,

$$\text{Value} = (255 - \text{Count}) + 1 = 164 = 0xA4$$

So we need to load A4 Hex value in a higher byte as,

$$\text{TH1} = 0xA4$$



### Programming steps for delay function

1. Load Tmod register value i.e. TMOD = 0x20 for Timer1 mode2 (8-bit timer auto reload mode).
2. Load calculated THx value i.e. here TH1 = 0xA4.
3. Load same value for TLx i.e. here TL1 = 0xA4.
4. Start the timer by setting a TRx bit. i.e. here TR1 = 1.
5. Poll TFx flag till it does not get set.
6. Clear timer flag TFx bit i.e. here TF1 = 0.
7. Repeat from step 5 and 6 for the delay again.

### Program for timer mode2

```
#include <reg51.h>                /* Include x51 header file */

sbit test = P1^0;                 /* set test pin0 of port1 */

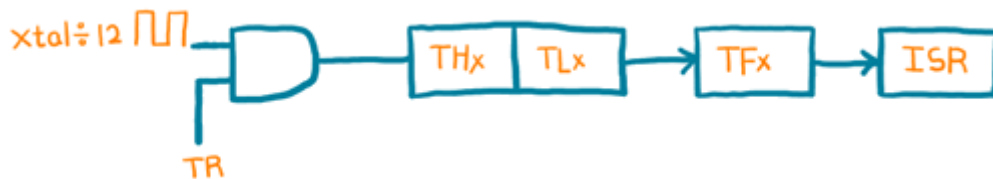
void main()
{
    TMOD = 0x20;                  /* Timer1 mode2 (8-bit auto reload timer mode) */
    TH1 = 0xA4;                   /* Load 8-bit in TH1 */
    TL1 = 0xA4;                   /* Load 8-bit in TL1 once */
    TR1 = 1;                      /* Start timer1 */
    while(1)
    {
        test = ~test;            /* Toggle test pin */
        while(TF1 == 0);         /* Wait until timer1 flag set */
        TF1 = 0;                 /* Clear timer1 flag */
    }
}
```

## Timer interrupt in 8051

8051 has two timer interrupts assigned with different vector address. When Timer count rolls over from its max value to 0, it sets the timer flag  $TF_x$ . This will interrupt the 8051 microcontroller to serve ISR (interrupt service routine) if global and timer interrupt is enabled.

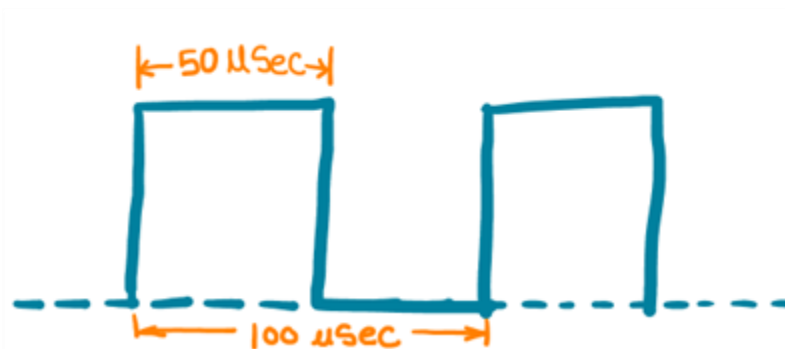
| Interrupt source       | Vector address |
|------------------------|----------------|
| Timer 0 overflow (TF0) | 000BH          |
| Timer 1 overflow (TF1) | 001BH          |

The timer overflow interrupt assigned with the vector address shown in the table. 8051 microcontroller jumps directly to the vector address on the occurrence of a corresponding interrupt.



### Example

Here we will generate a square wave of 10Hz on PORT1.0 using Timer0 interrupt. We will use Timer0 in mode1 with 11.0592 MHz oscillator frequency.



As Xtal is 11.0592 MHz we have a machine cycle of 1.085uSec. Hence, the required count to generate a delay of 50mSec. is,

$$\text{Count} = (50 \times 10^{-3}) / (1.085 \times 10^{-6}) \approx 46080$$

And mode1 has a max count is  $2^{16}$  (0 - 65535) and it increment from 0 – 65535 so we need to load value which is 46080 less from its max. count i.e. 65535. Hence value need to be load is,

$$\text{Value} = (65535 - \text{Count}) + 1 = 19456 = (4C00)\text{Hex}$$

So we need to load 4C00 Hex value in a higher byte and lower byte as,

$$\text{TH0} = 0x4C \ \& \ \text{TL0} = 0x00$$

Note that the TF0 flag no need to clear by software as a microcontroller clears it after completing the ISR routine.

**Program for the timer interrupt**

```

#include<reg51.h>          /* Include x51 header file */

sbit test = P1^0;          /* set test pin0 of port1 */

void Timer_init()
{
    TMOD = 0x01;           /* Timer0 mode1 */

    TH0 = 0x4C;            /* 50ms timer value */

    TL0 = 0x00;

    TR0 = 1;              /* Start timer0 */
}

void Timer0_ISR() interrupt 1 /* Timer0 interrupt service routine (ISR) */
{
    test = ~test;          /* Toggle port pin */

    TH0 = 0x4C;            /* 50ms timer value */

    TL0 = 0x00;
}

int main(void)
{
    EA = 1;               /* Enable global interrupt */

    ET0 = 1;              /* Enable timer0 interrupt */

    Timer_init();

    while(1);
}

```

## 8051 Interrupts

### Introduction

An interrupt is an event that occurs randomly in the flow of continuity. It is just like a call you have when you are busy with some work and depending upon call priority you decide whether to attend or neglect it.

The same thing happens in microcontrollers. 8051 architecture handles 5 interrupt sources, out of which two are internal (Timer Interrupts), two are external and one is a serial interrupt. Each of these interrupts has its interrupt vector address. The highest priority interrupt is the Reset, with vector address 0x0000.

**Vector Address:** This is the address where the controller jumps after the interrupt to serve the ISR (interrupt service routine).

| Interrupt          | Flag  | Interrupt vector address |
|--------------------|-------|--------------------------|
| Reset              | -     | 0000H                    |
| INT0 (Ext. int. 0) | IE0   | 0003H                    |
| Timer 0            | TF0   | 000BH                    |
| INT1 (Ext. int. 1) | IE1   | 0013H                    |
| Timer 1            | TF1   | 001BH                    |
| Serial             | TI/RI | 0023H                    |

### Reset

Reset is the highest priority interrupt, upon reset 8051 microcontroller start executing code from 0x0000 addresses.

### Internal interrupt (Timer Interrupt)

8051 has two internal interrupts namely timer0 and timer1. Whenever timer overflows, timer overflow flags (TF0/TF1) are set. Then the microcontroller jumps to their vector address to serve the interrupt. For this, global and timer interrupt should be enabled.

### Serial interrupt

8051 has a serial communication port and have related serial interrupt flags (TI/RI). When the last bit (stop bit) of a byte is transmitted, the TI serial interrupt flag is set, and when the last bit (stop bit) of the receiving data byte is received, the RI flag gets set.

### IE register: Interrupt Enable Register

IE register is used to enable/disable interrupt sources.



#### Bit 7 – EA: Enable All Bit

**1** = Enable all interrupts

**0** = Disable all interrupts

#### Bit 6,5 – Reserved bits

#### Bit 4 – ES: Enable Serial Interrupt Bit

**1** = Enable serial interrupt

**0** = Disable serial interrupt

#### Bit 3 – ET1: Enable Timer1 Interrupt Bit

**1** = Enable Timer1 interrupt

**0** = Disable Timer1 interrupt

#### Bit 2 – EX1: Enable External1 Interrupt Bit

**1** = Enable External1 interrupt

**0** = Disable External1 interrupt

#### Bit 1 – ET0: Enable Timer0 Interrupt Bit

**1** = Enable Timer0 interrupt

**0** = Disable Timer0 interrupt

#### Bit 0 – EX0: Enable External0 Interrupt Bit

**1** = Enable External0 interrupt

**0** = Disable External0 interrupt

Interrupt priority

Priority to the interrupt can be assigned by using the **interrupt priority register (IP)**

**Interrupt priority after Reset:**

| Priority | Interrupt source     | Intr. bit / flag |
|----------|----------------------|------------------|
| 1        | External Interrupt 0 | INT0             |
| 2        | Timer Interrupt 0    | TF0              |
| 3        | External Interrupt 1 | INT1             |
| 4        | Timer Interrupt 1    | TF1              |
| 5        | Serial interrupt     | (TI/RI)          |

In the table, interrupts priorities upon reset are shown. As per 8051 interrupt priorities, the lowest priority interrupts are not served until the microcontroller is finished with higher priority ones. In a case when two or more interrupts arrive microcontroller queues them according to priority.

**IP Register: Interrupt priority register**

8051 has an interrupt priority register to assign priority to interrupts.

| 7   | 6   | 5   | 4  | 3   | 2   | 1   | 0   |    |
|-----|-----|-----|----|-----|-----|-----|-----|----|
| --- | --- | --- | PS | PT1 | PX1 | PT0 | PX0 | IP |

**Bit 7,6,5** – Reserved bits.

**Bit 4 – PS:** Serial Interrupt Priority Bit

**1** = Assign a high priority to serial interrupt.

**0** = Assign low priority to serial interrupt.

**Bit 3 – PT1:** Timer1 Interrupt Priority Bit

**1** = Assign high priority to Timer1 interrupt.

**0** = Assign low priority to Timer1 interrupt.

**Bit 2 – PX1: External Interrupt 1 Priority Bit**

**1** = Assign high priority to External1 interrupt.

**0** = Assign low priority to External1 interrupt.

**Bit 1 – PT0: Timer0 Interrupt Priority Bit**

**1** = Assign high priority to Timer0 interrupt.

**0** = Assign low priority to Timer0 interrupt.

**Bit 0 – PX0: External0 Interrupt Priority Bit**

**1** = Assign high priority to External0 interrupt.

**0** = Assign low priority to External0 interrupt.

**External interrupts in 8051**

- 8051 has two external interrupt INT0 and INT1.
- 8051 controller can be interrupted by external Interrupt, by providing level or edge on external interrupt pins PORT3.2, PORT3.3.
- External peripherals can interrupt the microcontroller through these external interrupts if global and external interrupts are enabled.
- Then the microcontroller will execute current instruction and jump to the Interrupt Service Routine (ISR) to serve to interrupt.
- In the polling, method the microcontroller has to continuously check for a pulse by monitoring pin, whereas, in the interrupt method, the microcontroller does not need to poll. Whenever an interrupt occurs microcontroller serves the interrupt request.

External interrupt has two types of activation level

1. Edge triggered (Interrupt occur on rising/falling edge detection)
2. Level triggered (Interrupt occur on high/low-level detection)

In 8051, two types of activation levels are used. These are,

Low level triggered



Whenever a low level is detected on the INT0/INT1 pin while global and external interrupts are enabled, the controller jumps to interrupt service routine (ISR) to serve interrupt.

Falling edge triggered

Whenever falling edge is detected on the INT0/INT1 pin while global and ext. interrupts are enabled, the controller jumps to interrupt service routine (ISR) to serve interrupt.

There are lower four flag bits in the **TCON register** required to select and monitor the external interrupt type and ISR status.

#### **TCON: Timer/ counter Register**

| 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|
| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |

#### **Bit 3- IE1:**

External Interrupt 1 edge flag, set by hardware when interrupt on INT1 pin occurred and cleared by hardware when interrupt get processed.

#### **Bit 2- IT1:**

This bit selects the external interrupt event type on INT1 pin,

1= sets interrupt on falling edge

0= sets interrupt on low level

#### **Bit 1- IE0:**

Interrupt0 edge flag, set by hardware when interrupt on INT0 pin occurred and cleared by hardware when an interrupt is processed.

#### **Bit 0 - IT0:**

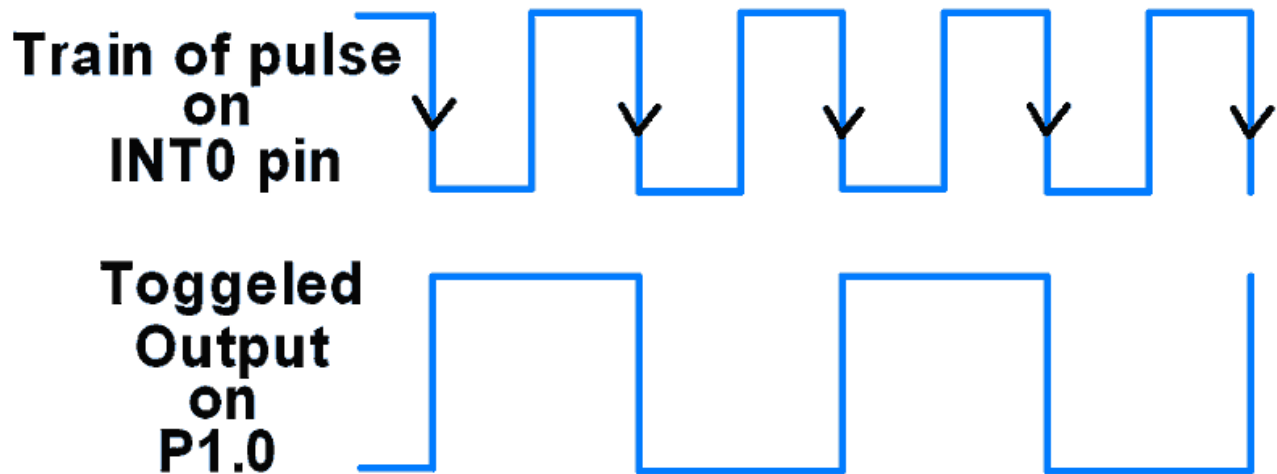
This bit selects the external interrupt event type on the INT0 pin.

1= sets interrupt on falling edge

0= sets interrupt on low level

### Example

Let's program the external interrupt of AT89C51 such that, when falling edge is detected on the INT0 pin then the microcontroller will toggle the P1.0 pin.



### Programming steps

1. Enable global interrupt i.e.  $EA = 1$
2. Enable external interrupt i.e.  $EX0 = 1$
3. Enable interrupt trigger mode i.e. whether interrupt is edge triggered or level triggered, here we will use falling edge trigger interrupt, so make  $IT0 = 1$ .

### Program

```
#include <reg51.h>    /* Include x51 header file */
sbit LED = P1^0;      /* set LED on port1 */

void Ext_int_Init()
{
    EA = 1;           /* Enable global interrupt */
    EX0 = 1;           /* Enable Ext. interrupt0 */
    IT0 = 1;           /* Select Ext. interrupt0 on falling edge */
}

void External0_ISR() interrupt 0
{
    LED = ~LED; /* Toggle pin on falling edge on INT0 pin */
}

void main()
{
    Ext_int_Init(); /* Call Ext. interrupt initialize */
    while(1);
}
```

Note: For **level triggered interrupt** IT0 needs to be cleared i.e. IT0 = 0.

# **LCD16x2 interfacing in 4-bit mode with 8051**

## **Introduction**

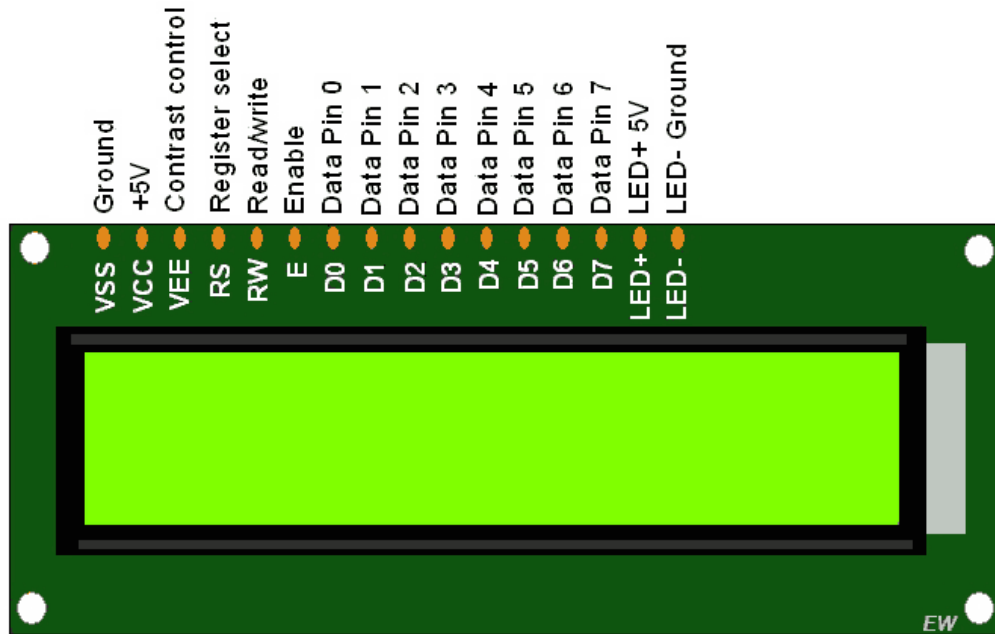
LCDs (Liquid Crystal Displays) are used for displaying status or parameters in embedded systems.

LCD 16x2 is 16 pin device which has 8 data pins (D0-D7) and 3 control pins (RS, RW, EN). The remaining 5 pins are for supply and backlight for the LCD.

The control pins help us configure the LCD in command mode or data mode. They also help configure read mode or write mode and also when to read or write.

LCD 16x2 can be used in 4-bit mode or 8-bit mode depending on the requirement of the application. In order to use it, we need to send certain commands to the LCD in command mode and once the LCD is configured according to our need, we can send the required data in data mode.

For more information about LCD 16x2 and how to use it, refer to the topic LCD 16x2 display module in the sensors and modules section.

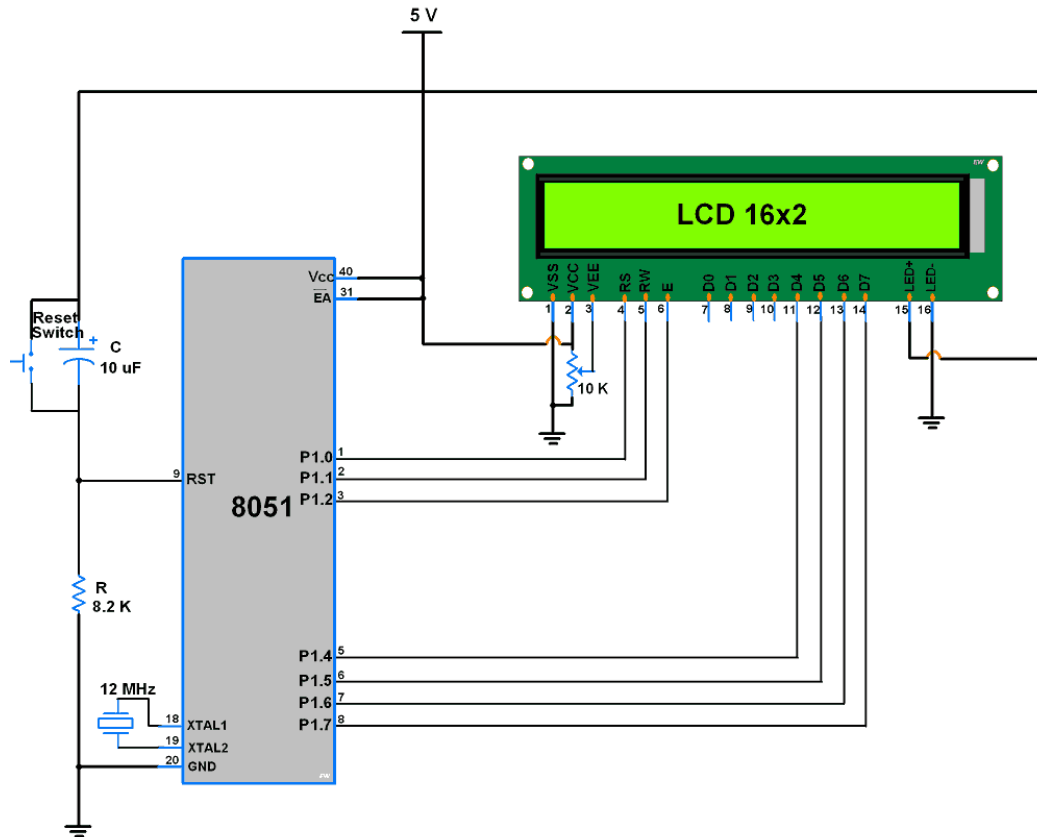


**16x2 LCD Display**

### 4-bit Mode

- In 4-bit mode, data/command is sent in a 4-bit (nibble) format.
- To do this 1st send Higher 4-bit and then send lower 4-bit of data/command.
- Only 4 data (D4 - D7) pins of 16x2 of LCD are connected to the microcontroller and other control pins RS (Register select), RW (Read/write), E (Enable) is connected to other GPIO Pins of the controller.
- Therefore, due to such connections, we can save four GPIO pins which can be used for another application.

### Interfacing Diagram



**LCD 16 x 2 Interfaces with 8051**

Programming LCD16x2 4-bit mode

### Initialization

1. Wait for 15ms, Power-on initialization time for LCD16x2.
2. Send 0x02 command which initializes LCD 16x2 in 4-bit mode.
3. Send 0x28 command which configures LCD in 2-line, 4-bit mode, and 5x8 dots.
4. Send any Display ON command (0x0E, 0x0C)
5. Send 0x06 command (increment cursor)

```

void LCD_Init (void)                /* LCD Initialize function */

{
    delay(20);                      /* LCD Power ON Initialization time >15ms */
    LCD_Command (0x02);             /* 4bit mode */
    LCD_Command (0x28);             /* Initialization of 16X2 LCD in 4bit
mode */
    LCD_Command (0x0C);             /* Display ON Cursor OFF */
    LCD_Command (0x06);             /* Auto Increment cursor */
    LCD_Command (0x01);             /* Clear display */
    LCD_Command (0x80);             /* Cursor at home position */
}

```

Now we successfully initialized LCD & it is ready to accept data in 4-bit mode to display.

To send command/data to 16x2 LCD we have to send a higher nibble followed by a lower nibble. As 16x2 LCD's D4 - D7 pins are connected as data pins, we have to shift the lower nibble to the right by 4 before transmitting.

### Command write function

1. First, send a Higher nibble of command.
2. Make RS pin low, RS=0 (command reg.)
3. Make RW pin low, RW=0 (write operation) or connect it to ground.
4. Give High to Low pulse at Enable (E).
5. Send lower nibble of command.
6. Give High to Low pulse at Enable (E).

```

void LCD_Command (char cmnd)        /* LCD16x2 command funtion */

void LCD_Command (char cmnd) /* LCD16x2 command funtion */
{
    LCD_Port =(LCD_Port & 0x0F) | (cmnd & 0xF0); /* Send upper nibble */
    rs=0;                      /* Command reg. */
    rw=0;                      /* Write operation */
    en=1;
    delay(1);
    en=0;
}

```

```

    delay(2);
    LCD_Port = (LCD_Port & 0x0F) | (cmnd << 4);/* Send lower nibble */
    en=1;                /* Enable pulse */
    delay(1);
    en=0;
    delay(5);
}

```

### Data write function

1. First, send a Higher nibble of data.
2. Make RS pin high, RS=1 (data reg.)
3. Make RW pin low, RW=0 (write operation) or connect it to ground.
4. Give High to Low pulse at Enable (E).
5. Send lower nibble of data.
6. Give High to Low pulse at Enable (E).

```

void LCD_Char (char char_data) /* LCD data write function */
{
    LCD_Port =(LCD_Port & 0x0F) | (char_data & 0xF0);/* Send upper nibble */
    rs=1;                /* Data reg.*/
    rw=0;                /* Write operation*/
    en=1; delay(1); en=0; delay(2);
    LCD_Port = (LCD_Port & 0x0F) | (char_data << 4);/* Send lower nibble */
    en=1;                /* Enable pulse */
    delay(1); en=0; delay(5);
}

```

### Program

```

#include<reg51.h>

sfr LCD_Port=0x90;                /* P1 port as data port */

sbit rs=P1^0;                    /* Register select
pin */

sbit rw=P1^1;                    /* Read/Write pin
*/

```



```
sbit en=P1^2; /* Enable pin */
```

```
/* Function to provide delay Approx 1ms with 11.0592 Mhz crystal*/
```

```
void delay(unsigned int count)
```

```
{
```

```
    int i,j;
```

```
    for(i=0;i<count;i++)
```

```
        for(j=0;j<112;j++);
```

```
}
```

```
void LCD_Command (char cmnd) /* LCD16x2 command funtion */
```

```
{
```

```
    LCD_Port =(LCD_Port & 0x0F) | (cmnd & 0xF0); /* sending upper nibble */
```

```
    rs=0;
```

```
    /* command reg. */
```

```
    rw=0;
```

```
    /* Write operation */
```

```
    en=1;
```

```
        delay(1);
```

```
        en=0;
```

```
        delay(2);
```

```
    LCD_Port = (LCD_Port & 0x0F) | (cmnd << 4); /* sending lower nibble */
```

```
    en=1; /* enable pulse */
```

```
        delay(1);
```

```
        en=0;
```

```
        delay(5);
```

```
}
```

```
void LCD_Char (char char_data) /* LCD data write function */
```

```
{
```

```
    LCD_Port =(LCD_Port & 0x0F) | (char_data & 0xF0); /* sending upper nibble */
```

```

rs=1;                                     /*Data reg.*/
rw=0;                                     /*Write operation*/

    en=1;

    delay(1); en=0; delay(2);

    LCD_Port = (LCD_Port & 0x0F) | (char_data << 4); /* sending lower nibble */

    en=1;                                 /* enable pulse */

    delay(1); en=0; delay(5);

}

void LCD_String (char *str)               /* Send string to LCD function */
{
    int i;

    for(i=0;str[i]!=0;i++)                /* Send each char of string till the NULL */
    {
        LCD_Char (str[i]);               /* Call LCD data write */
    }
}

void LCD_String_xy (char row, char pos, char *str) /* Send string to LCD function */
{
    if (row == 0)

        LCD_Command((pos & 0x0F)|0x80);    /* Command of first row and required position<16 */

    else if (row == 1)

        LCD_Command((pos & 0x0F)|0xC0);    /* Command of first row and required position<16 */

    LCD_String(str);                       /* Call LCD string function */
}

void LCD_Init (void)                     /* LCD Initialize function */
{
    delay(20);                            /* LCD Power ON Initialization time >15ms */
}

```

```

LCD_Command (0x02);          /* 4bit mode */

LCD_Command (0x28);          /* Initialization of 16X2 LCD in 4bit mode */

LCD_Command (0x0C);          /* Display ON Cursor OFF */

LCD_Command (0x06);          /* Auto Increment cursor */

LCD_Command (0x01);          /* clear display */

LCD_Command (0x80);          /* cursor at home position */

}

void main()

{

LCD_Init();                  /* initialization of LCD*/

LCD_String("ElectronicWINGS"); /* write string on 1st line of LCD*/

LCD_Command(0xc0);           /*go to 2nd line*/

LCD_String_xy(1,1,"Hello World"); /*write string on 2nd line*/

while(1);                    /* infinite loop. */

}

```

# HC-05 Bluetooth Module Interfacing with 8051

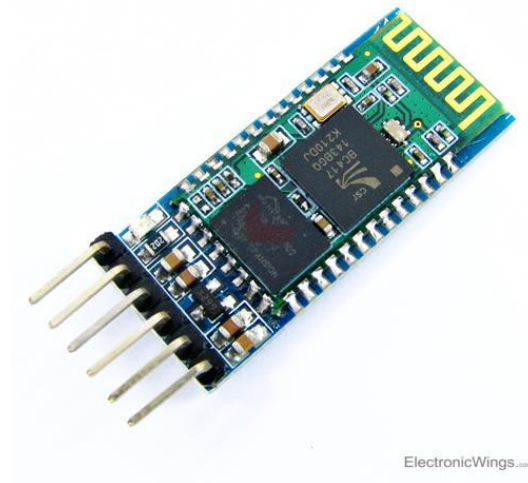
## Introduction

HC-05 is a Bluetooth device used for wireless communication. It works on serial communication (UART).

- It is a 6 pin module.
- The device can be used in 2 modes; data mode and command mode.
- The data mode is used for data transfer between devices whereas command mode is used for changing the settings of the Bluetooth module.
- AT commands are required in command mode.
- The module works on 5V or 3.3V. It has an onboard 5V to 3.3V regulator.
- As the HC-05 Bluetooth module has a 3.3 V level for RX/TX and the microcontroller can detect 3.3 V level, so, no need to shift the transmit level of the HC-05 module. But we need to shift the transmit voltage level from the microcontroller to RX of the HC-05 module.

For more information about the HC-05 Bluetooth module and how to use it, refer to the topic Bluetooth module HC-05 in the sensors and modules section.

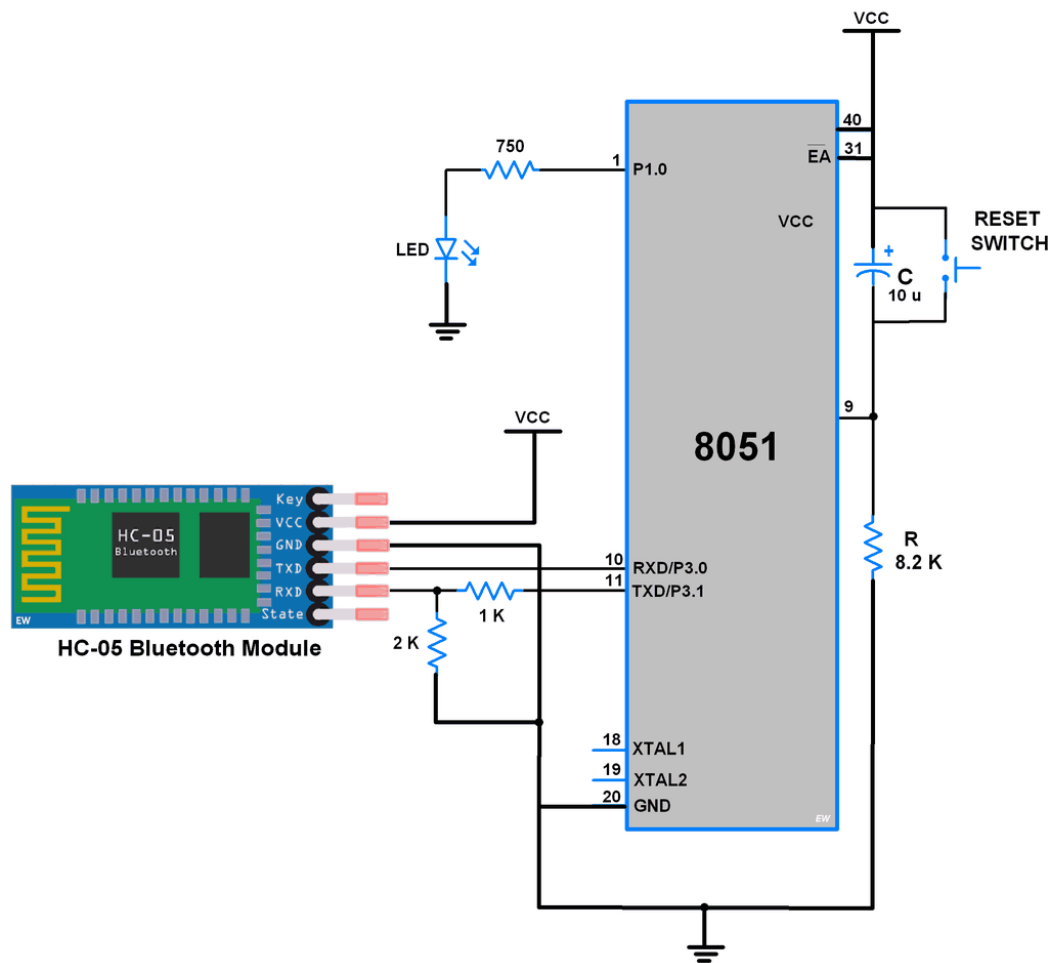
For information on UART in 8051 and how to use it, refer to the topic on UART in 8051 in the 8051 inside section.



ElectronicWings.com

**HC-05 Bluetooth Module**

## Interfacing Diagram



### HC-05 Bluetooth Module Interface with 8051

#### Example

Here let's develop a small application in which we can control LED ON-OFF through smartphones.

This is done by interfacing 8051 with the HC-05 Bluetooth module. Data from HC-05 is received/ transmitted serially by 8051.

In this application when 1 is sent from the smartphone, LED will turn ON and if 2 is sent LED will get Turned OFF. If received data is other than 1 or 2, it will return a message to mobile that select the proper option.

#### Program

1. Initialize 8051 UART communication.
2. Receive data from the HC-05 Bluetooth module.
3. Check whether it is '1' or '2' and take respective controlling action on the LED.

```

#include <reg51.h>
#include "UART_H_file.h" /* Include UART library */

sbit LED=P1^0;

void main()
{
    char Data_in;
    UART_Init();          /* Initialize UART */
    P1 = 0;                /* Clear port initially */
    LED = 0;               /* Initially LED turn OFF */
    while(1)
    {
        Data_in = UART_RxChar(); /* Receive char serially */
        if(Data_in == '1')
        {
            LED = 1; /* Turn ON LED */
            UART_SendString("LED_ON"); /* Send status of LED*/
        }
        else if(Data_in == '2')
        {
            LED = 0; /* Turn OFF LED */
            UART_SendString("LED_OFF"); /* Send status of LED*/
        }
        else
            UART_SendString("Select proper option");
    }
}

```