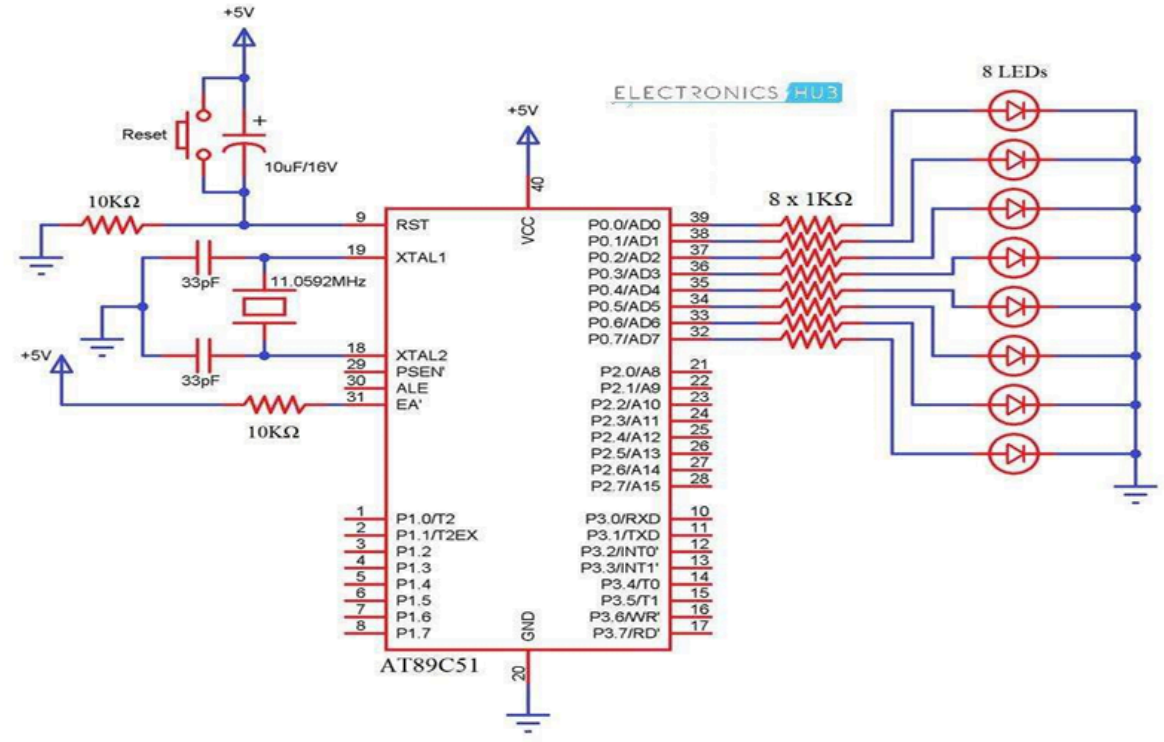
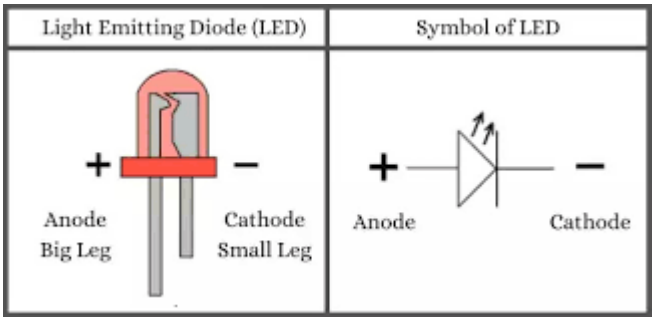
MCA

Tags

Exp 1 LED

LED INTERFACING DIAGRAM:





▼ LED ALTERNATE

```
#include <reg51.h>   // Header file for 8051 microcontroller

#define LED_PORT P2   // 8 LEDs connected to Port 2

void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for(i = 0; i < ms; i++)
        for(j = 0; j < 1275; j++);   // Roughly 1 ms delay for 12 MHz crystal
}

void main() {
    while(1) {
        // Glow even-numbered LEDs: 2, 4, 6, 8 (binary 10101010)
        LED_PORT = 0xAA;      // 0xAA = 10101010b
        delay_ms(500);

        // Glow odd-numbered LEDs: 1, 3, 5, 7 (binary 01010101)
        LED_PORT = 0x55;      // 0x55 = 01010101b
        delay_ms(500);
    }
}
```

▼ LED ALGO

Algorithm: LED Interfacing using Port (Delay Method)

- Configure **Port 2** as output for LEDs.
- In the main loop:
    - Send **0xAA (10101010b)** to Port 2 to turn ON even-numbered LEDs.
    - Call **delay_ms(500)** to keep LEDs ON for a short time.
    - Send **0x55 (01010101b)** to Port 2 to turn ON odd-numbered LEDs.
    - Call **delay_ms(500)** again to maintain delay.

- Repeat the above steps continuously to create an alternating LED blinking pattern.

▼ LED Timer ALGO

Algorithm: LED Interfacing using Timer Interrupt

- Configure **Port 2** as output for LEDs.
- Configure **Timer0** in **Mode 1 (16-bit)** using **TMOD register**.
- Load **TH0** and **TL0** with initial count values for delay.
- Enable **Timer0 interrupt (ET0)** and **global interrupt (EA)**.
- Start **Timer0** by setting **TR0 = 1**.
- In the **ISR (Interrupt Service Routine):**
  - Toggle all LEDs connected to **Port 2**.
  - Reload **TH0** and **TL0** for next delay.
- Main loop remains idle; LED toggling is handled by interrupt.

▼ LED Interfacing using Timer with interrupt

```
#include <reg51.h>     // Header file for 8051 microcontroller

#define LED_PORT P2     // LEDs connected to Port 2

// Function declarations
void Timer0_Init(void);

// Interrupt Service Routine (ISR) for Timer0
void Timer0_ISR(void) interrupt 1
{
   LED_PORT = ~LED_PORT;   // Toggle all LEDs
      // Reload timer values for next overflow (for delay)
   TH0 = 0x3C;
   TL0 = 0xB0;           // Together gives approx 50ms delay at 12MHz
}
```

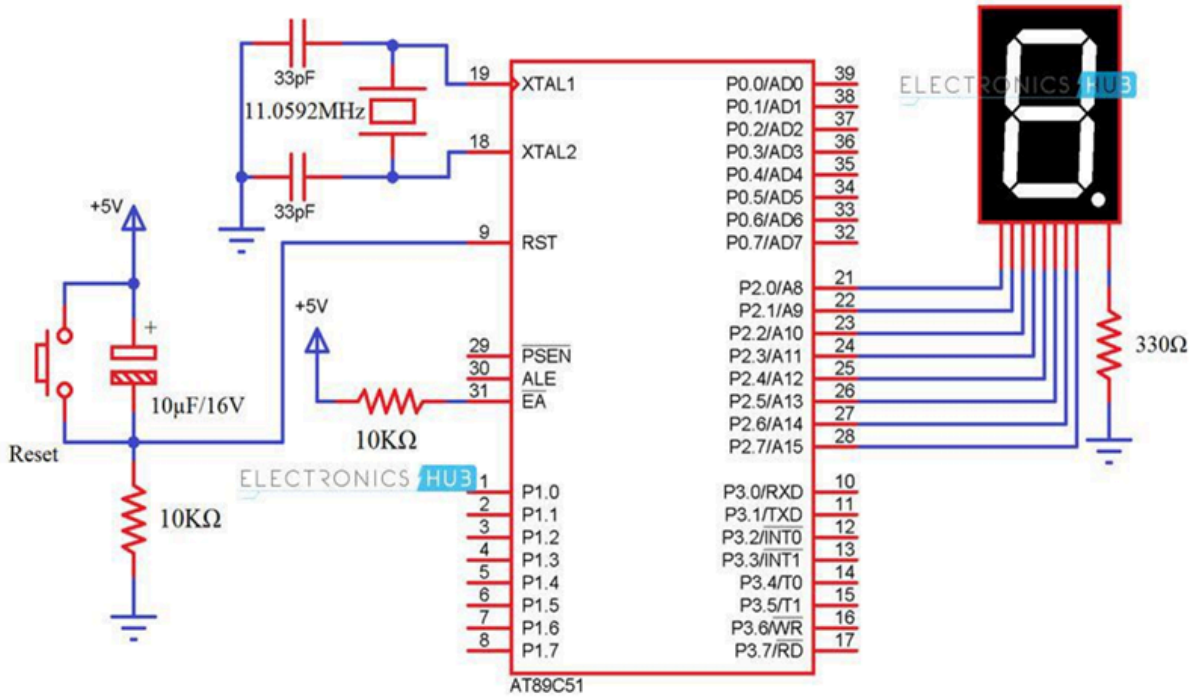
```
void main()
{
   LED_PORT = 0x00;       // Initially turn OFF all LEDs
   Timer0_Init();         // Initialize Timer0
   while(1);              // Infinite loop, everything handled by interrupt
}

void Timer0_Init(void)
{
   TMOD = 0x01;           // Timer0 in Mode1 (16-bit timer)
   TH0 = 0x3C;            // Load high byte for 50ms delay
   TL0 = 0xB0;            // Load low byte for 50ms delay
   ET0 = 1;               // Enable Timer0 interrupt
   EA  = 1;               // Enable global interrupt
   TR0 = 1;               // Start Timer0
   // When Timer0 overflows → interrupt flag TF0 = 1.
}
```

Timer Mode Setup
TMOD = 0x01; → Timer0, Mode 1 (16-bit timer).
16-bit timer counts from 0000H to FFFFH, then overflows.

Delay Calculation
For 12 MHz clock → each machine cycle = 1 µs.
For ~50ms delay:
65536 – 50000 = 15536 = 3CB0H → TH0 = 3CH, TL0 = B0H.
So timer overflows every 50ms.

Interrupt Concept
When Timer0 overflows → interrupt flag TF0 = 1.
CPU automatically jumps to ISR at vector address 000BH.
ISR executes, toggles LEDs, reloads TH0/TL0, and returns.

Exp 2 7seg



▼ 7 Seg Code

```
#include <reg51.h>

#define DATAPORT P0

sbit DISP1   = P3^4;

void msdelay (unsigned int time) //Function to generate delay
{
unsigned int i, j;
  for (i = 0; i < time; i++)
```

```
    for (j = 0; j < 113; j++);//Calibrated for a 1 ms delay in MPLAB
}

int main()
{
    unsigned char  seg_code[]= {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x8
2,
    0xF8,0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
    int cnt;

    while (1)
    {
        for (cnt = 0; cnt <= 9; cnt++) // loop to display 0-9999
        {
            DISP1 = 0;
            DATAPORT = seg_code[cnt];
            msdelay(500);
        }
    }
}
```

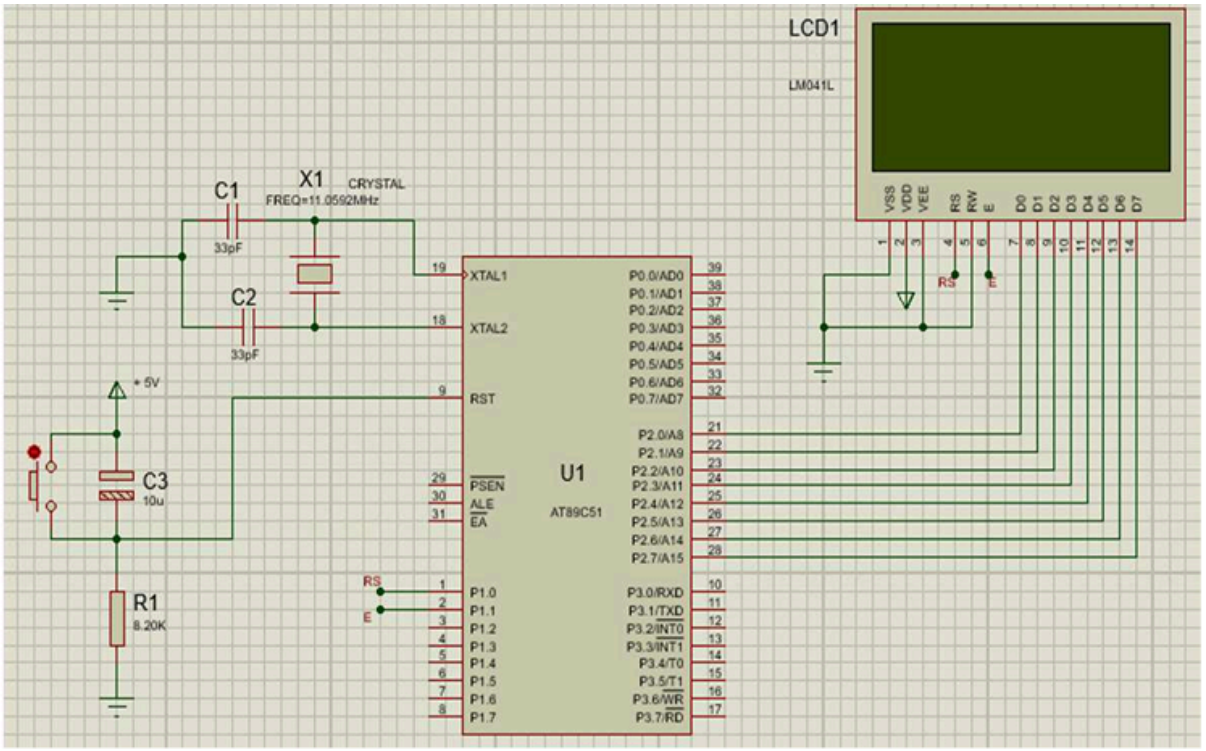
▼ 7Seg Algo

Algorithm

- Initialize port **P2** as output.
- Create a lookup table in code with segment patterns for digits **0–9**.
- Output the pattern for the desired digit to port **P2**.
- Hold the display long enough for it to be visible (~500ms), then change to next digit (if cycling).

Exp 3 LCD

Common LCD Commands:
0x38 – Initialize LCD in 8-bit, 2-line, 5×7 format
0x0C – Display ON, Cursor OFF
0x01 – Clear Display
0x06 – Increment Cursor
0x80 – Move cursor to first line
0xC0 – Move cursor to second line



15. LED+ = Backlight Anode
16. LED – = Backlight Cathode

▼ LCD Code

```
#include <reg51.h>

#define LCD P2

sbit RS = P3^0;
sbit RW = P3^1;
sbit EN = P3^2;
```

```
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for(i = 0; i < ms; i++)
        for(j = 0; j < 1275; j++);
}

void lcd_cmd(unsigned char cmd) {
    LCD = cmd;
    RS = 0;
    RW = 0;
    EN = 1;
    delay_ms(1);
    EN = 0;
    delay_ms(2);
}

void lcd_data(unsigned char data_byte) {
    LCD = data_byte;
    RS = 1;
    RW = 0;
    EN = 1;
    delay_ms(1);
    EN = 0;
    delay_ms(2);
}

void lcd_init() {
    lcd_cmd(0x38);
    lcd_cmd(0x0C);
    lcd_cmd(0x01);
    lcd_cmd(0x06);
    lcd_cmd(0x80);
}

void lcd_string(char *str) {
    while(*str) {
        lcd_data(*str++);
    }
```

```
}

void main() {
    lcd_init();
    lcd_string("Roll No T171000");
    while(1);
}
```

▼ LCD Algo

Algorithm:

- **Initialize LCD:**
  - Wait for LCD power-on delay (~15ms).
  - Send function set command (8-bit, 2 lines, 5×7 format).
  - Send display ON command.
  - Clear display.
  - Set entry mode (increment cursor).
- **Main loop:**
  - Send commands to set cursor.
  - Send data to display characters.
- **For each command or data byte:**
  - Set RS, RW pins.
  - Put byte on data bus.
  - Pulse EN pin.

Exp 4 Stepper Motor

A stepper motor is an electromechanical device that converts electrical pulses into discrete mechanical steps. The rotation angle of the motor is proportional

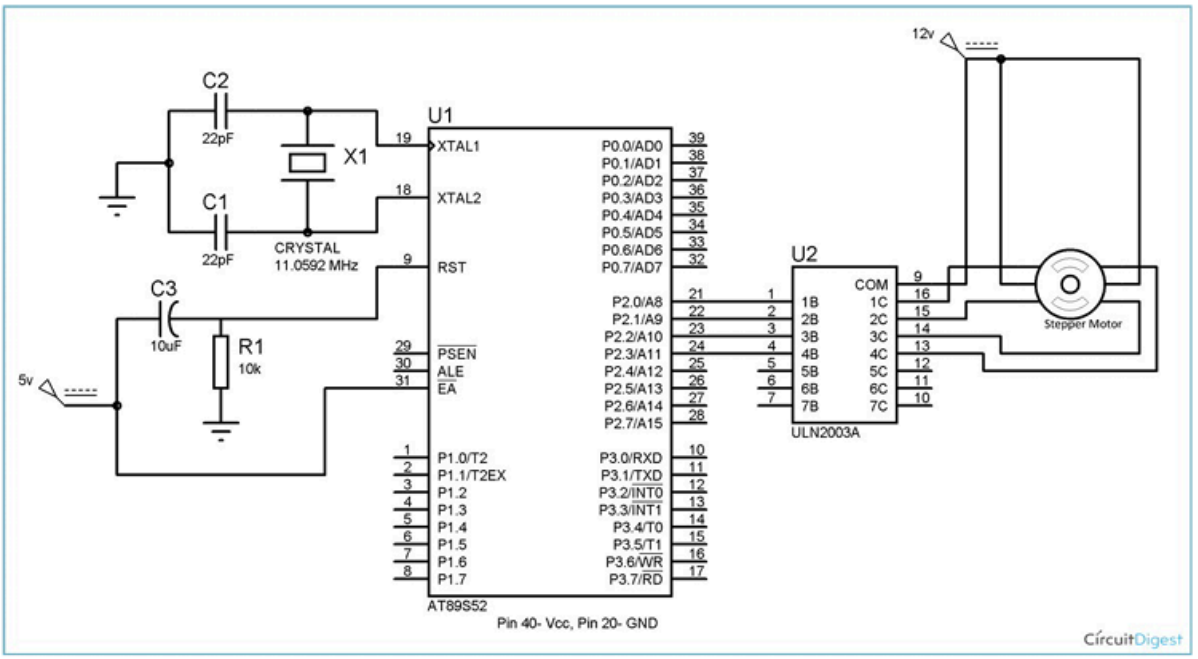
to the number of input pulses. Stepper motors are used in precise motion control applications such as robotics, printers, and CNC machines.

Principle of electromagnetism

Unipolar Stepper Motor – Each winding has a center tap. It is easier to drive using ULN2003.

The motor driver receives signals from the microcontroller and energizes the stepper coils accordingly. By continuously changing the energizing pattern, the rotor moves step by step, achieving controlled rotation.

7) Stepper Motor Interfacing Diagram:



▼ Motor Algo

Algorithm: Stepper Motor Control using 8051

1. *Start*
2. Configure the control port (e.g., P2) as output for the stepper motor coils.
3. *Define* the excitation sequence for the stepper motor (e.g., 0x09, 0x0C, 0x06, 0x03 for 4-step sequence).
4. *Send* one pattern to the motor port.

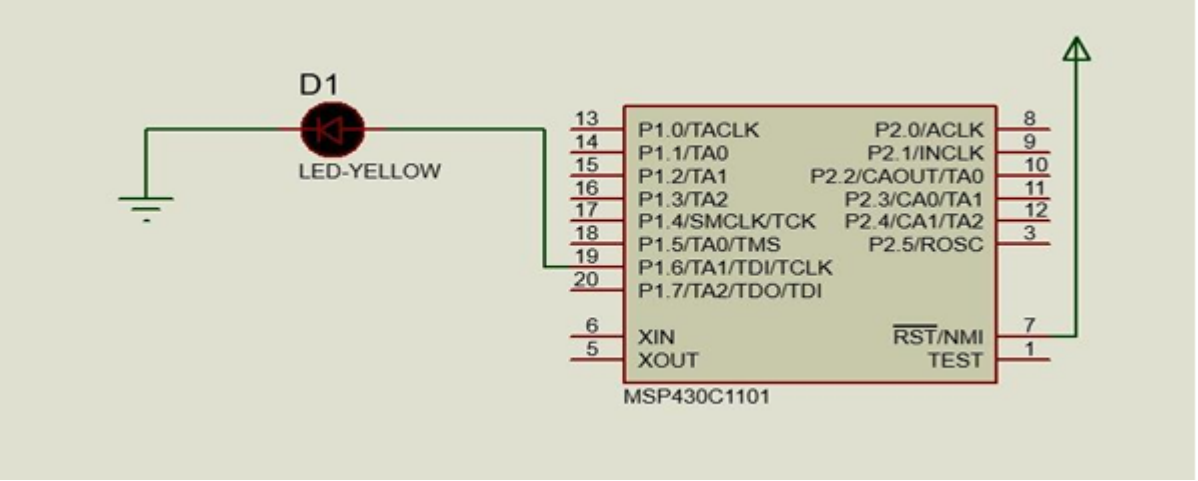
5. *Call delay* to allow the motor to step properly.
6. *Send* the next pattern in sequence to the motor port.
7. *Repeat* steps 5–6 to rotate motor continuously in one direction.
8. For reverse rotation, send the sequence in **reverse order*.
9. *Stop* when desired rotation is complete.
10. *End*

EXP 6 MSP

Introduction to MSP430 and GPIO:
The MSP430 is a 16-bit, ultra-low-power microcontroller designed by Texas Instruments for embedded
applications requiring low power and high performance. It features several peripherals, including timers,
ADCs, communication interfaces, and general-purpose input/output (GPIO) ports. GPIOs are versatile
pins that can function as either input or output depending on the configuration of their registers.

Each GPIO pin is controlled by specific registers that define its direction, output value, and function. The main registers used are PxDIR, PxOUT, PxIN, PxREN, PxSEL, and PxSEL2.
• PxDIR: Determines the direction of the pin (0 = Input, 1 = Output).
• PxOUT: Controls the output value

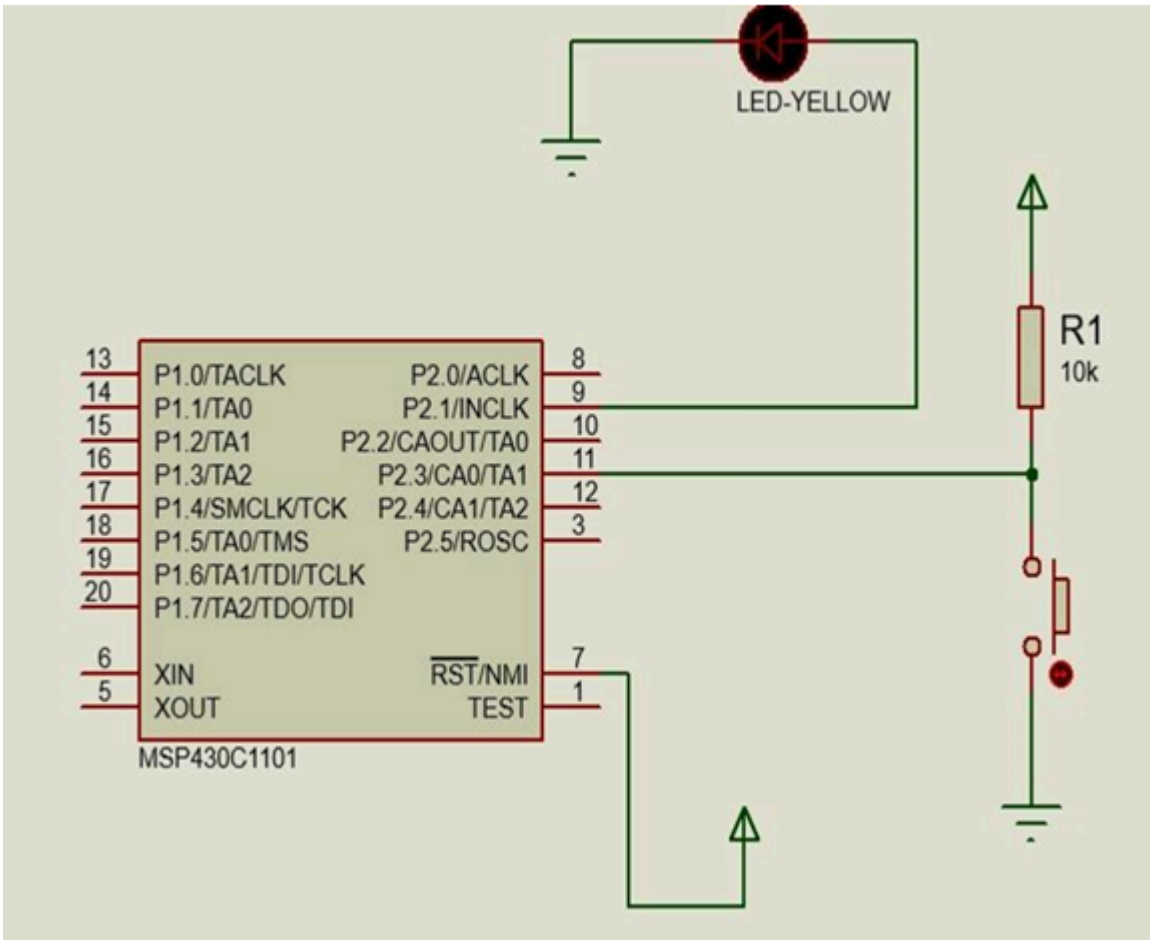
LED BLINKING



Algorithm: Blink On-board LED (MSP430 P1.6)

- Configure pin P1.6 as output.
- Repeat forever:
  - Set P1.6 HIGH to turn LED ON.
  - Wait 500 ms.
  - Set P1.6 LOW to turn LED OFF.
  - Wait 500 ms.

PUSH BUTTON



Algorithm: LED Control using Switch (MSP430 P2.1 & P2.3)

- Configure **P2.1** as **output** to drive the LED.
- Configure **P2.3** as **input** to read the switch status.
- Repeat forever:
  - Read the logic level from **P2.3** (switch).
  - If input is **LOW (0)** → switch is pressed → turn **LED ON**.
  - Else → switch is released → turn **LED OFF**.
  - Wait for a short delay to stabilize switch input.

EXP 7 IR

Infrared (IR) sensors are electronic devices that emit and/or detect infrared radiation to sense objects or
measure distances. An IR sensor module typically consists of an IR LED (emitter) and a photodiode
(receiver). The photodiode detects reflected IR light from nearby objects. When an object comes within
the range, the reflected IR radiation falls on the photodiode, changing its output voltage.

There are two main types of IR sensors:
• Active IR Sensor: Has both emitter and detector in a single module (used for obstacle/intruder detection).
• Passive IR Sensor (PIR): Detects IR radiation from human bodies.

In this experiment, an active IR sensor is used. It gives a digital output — logic HIGH (1) when no
obstacle is present and logic LOW (0) when an obstacle/intruder is detected.

Typical connections:
• IR Sensor Output → P1.3 (Input)
• Buzzer → P1.0 (Output)
• VCC → 5V, GND → Ground