8051 Development Boards

An alternative to Arduino Uno, for Studying Embedded Systems hands-on

Reference Manual

AMOTECH LABS

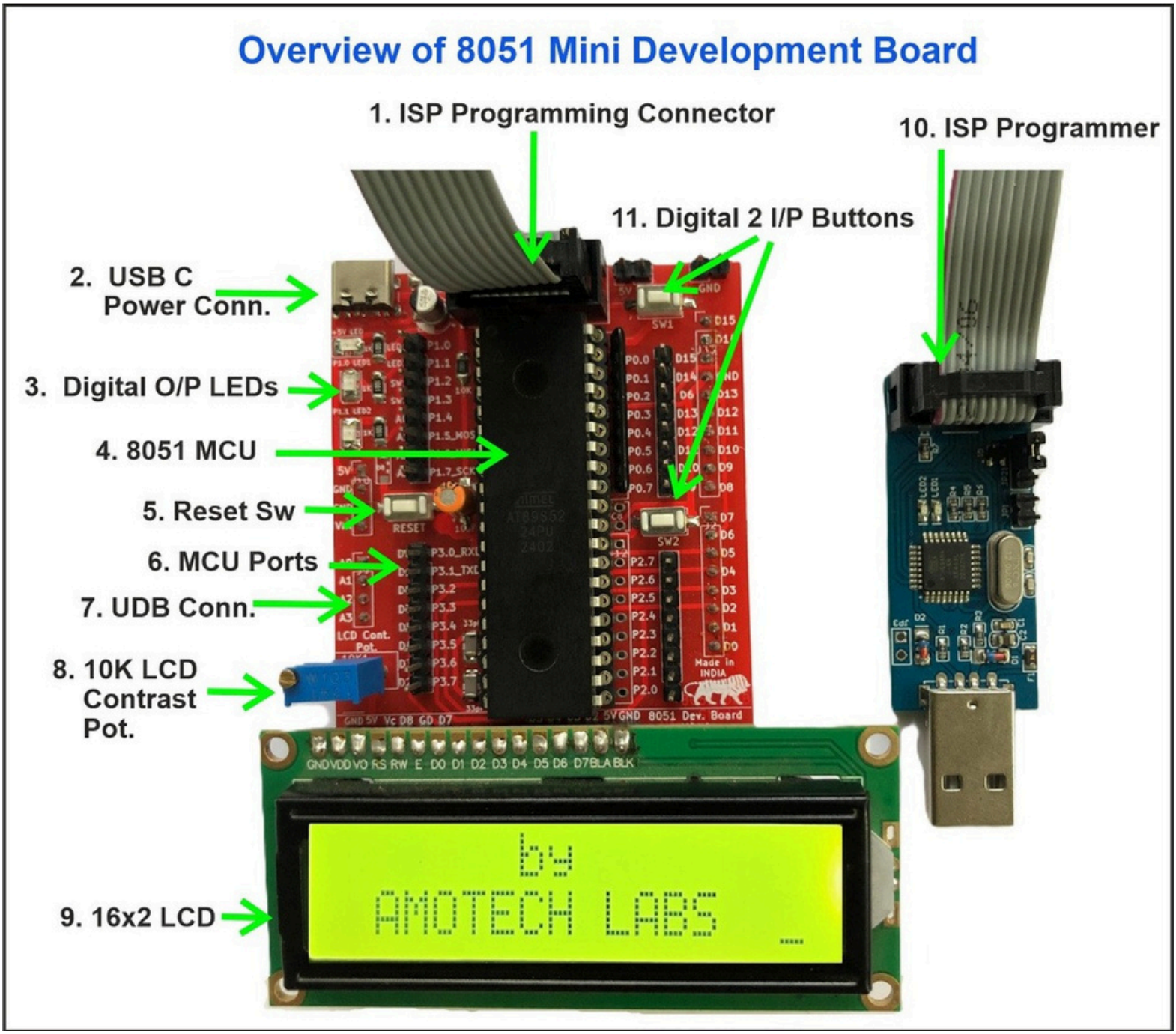
Embeddedsystems | IndustrialAutomation | Robotics

Scan to download Soft Copy of Manual & Online Purchase

Scan for Video Tutorials

Contents

Overview of '8051 Mini Development Board'.....3
Schematic of '8051 Mini Development Board'.....4
Overview of '8051 Development Board'.....5
Schematic of '8051 Development Board'.....;.....6
Description of 'Universal Development Board'.....;.....9
Introduction to AT89S52 Microcontroller.....10
Steps of Keil uvision for Programming.....11
Lab1. LED Blinking 17
Lab2. SSD Interfacing.....19
Lab3. LCD_Interfacing.....21
Lab4. Keypad_Interfacing.....25
Lab5. ADC 0804 Interfacing.....30
Lab6. ADC 0809 interfacing.....33
Lab7. Timer.....37
Lab8: Serial Communication.....42
Lab9: GSM Module (SIM900A) Interfacing.....48
Lab10: Relay Interfacing.....53
Lab11: Servo Motor Control Using PWM.....55
Lab12: DC Motor Control Using PWM.....61
Lab13: Interfacing of Stepper Motor.....66



1. ISP Programming Conn: A socket for the ISP Programmer, which allows easy insertion/removal of the in-system programmer.

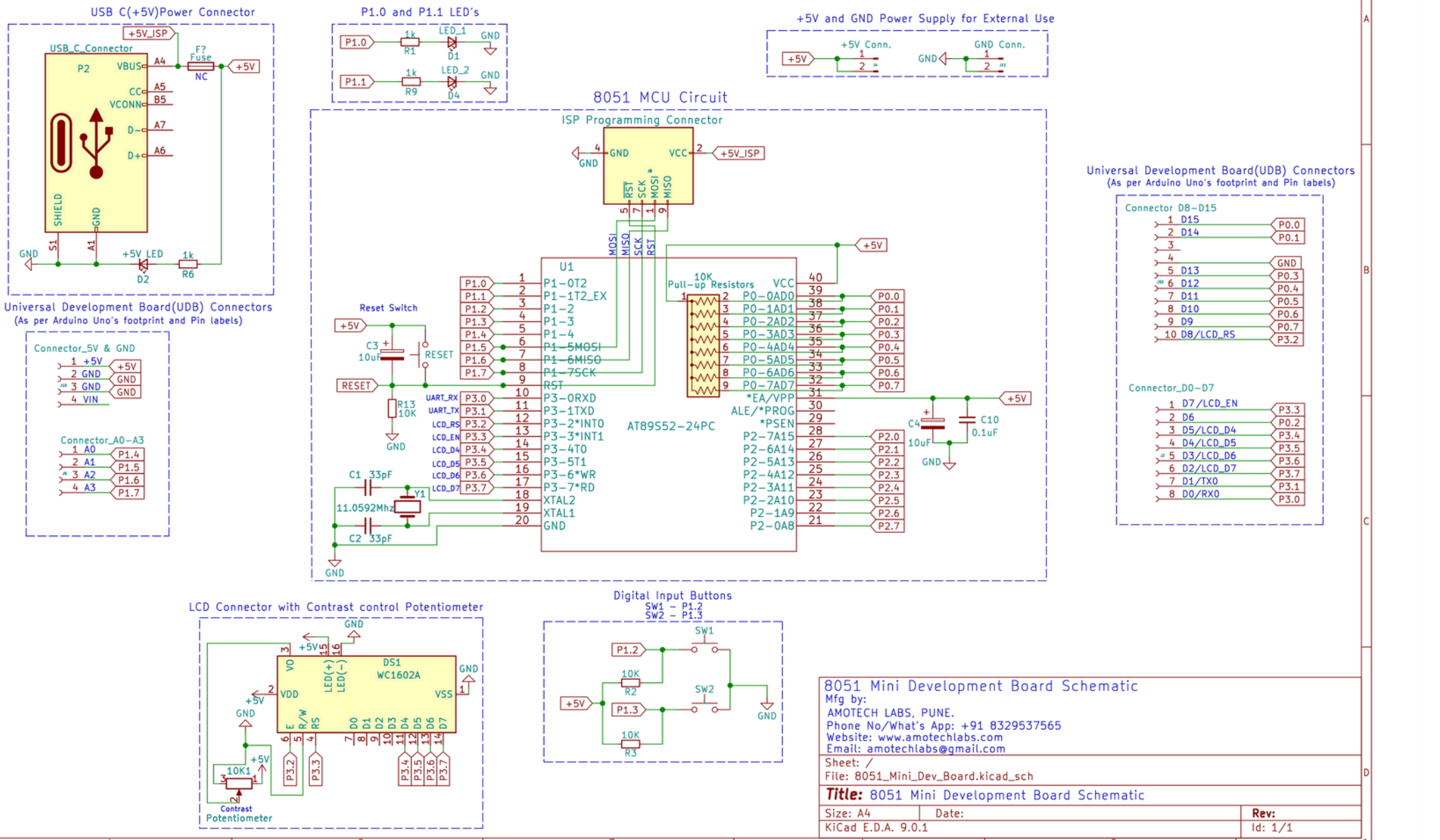
2. Power I/P (Power Input): This is the power supply input to the board, providing necessary +5V voltage from the USB Type-C connector to power the board when the programmer is not connected.

3. 8051 MCU: Any 8051-based 40-pin DIP IC can be used with this board. This board is equipped with AT89S52.

4. UDB Conn: These connectors are used to attach this kit to the Universal Development Board (UDB) by Amotech Labs. Check UDB details on our website.

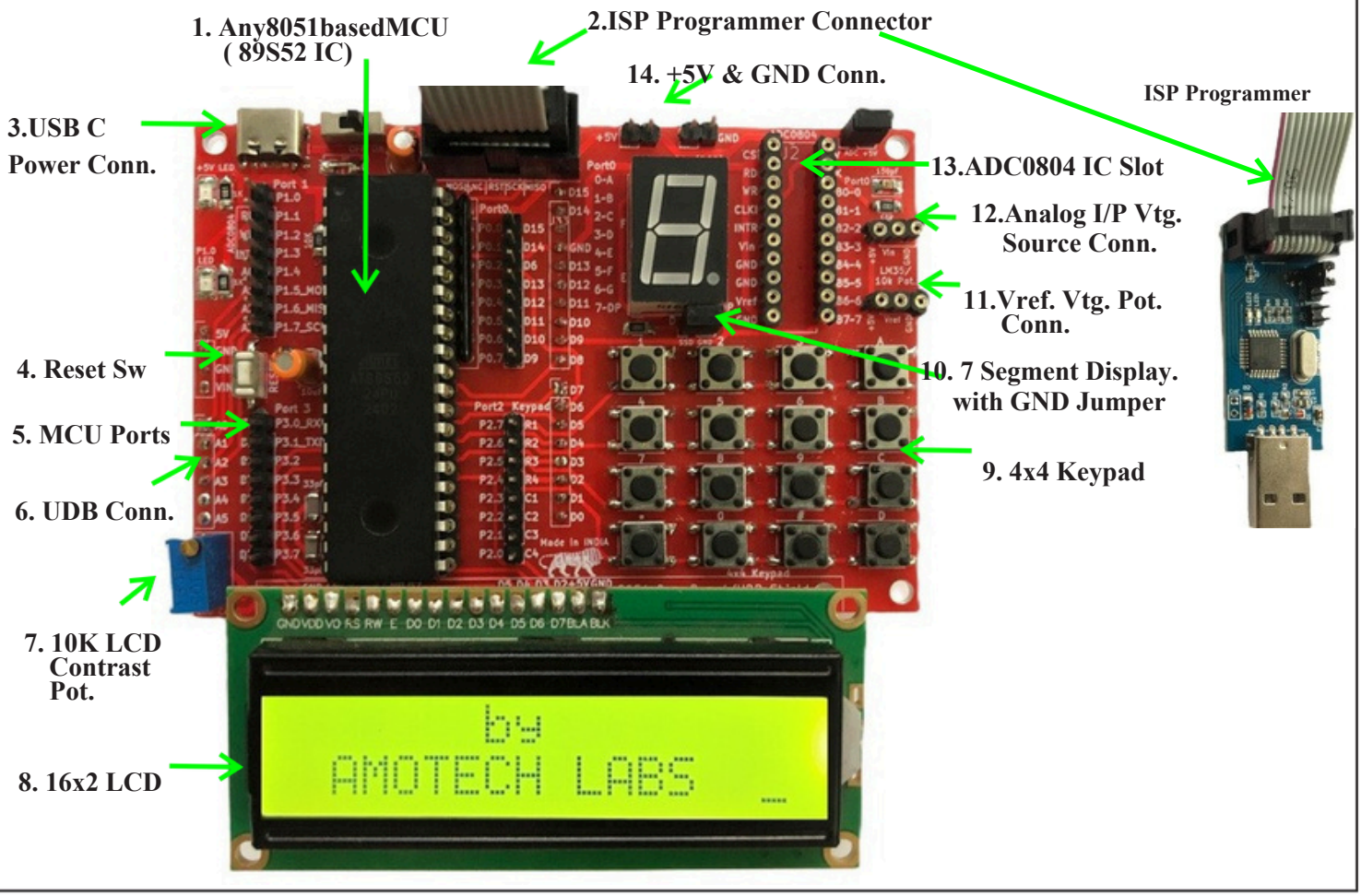
5. Digital 2 I/P Button Switches: Two push buttons serve as digital input switches connected to P1.2 and P1.3. When configured as inputs, they are pulled up at logic HIGH. On pressing, they are connected to ground, making the pin state LOW.

8051 Mini Development Board Schematic



8051 Mini Development Board Schematic	
Mfg by: AMOTECH LABS, PUNE.	
Phone No/What's App: +91 8329537565	
Website: www.amotechlabs.com	
Email: amotechlabs@gmail.com	
Sheet: /	
File: 8051_Mini_Dev_Board.kicad_sch	
Title: 8051 Mini Development Board Schematic	
Size: A4	Date:
KiCad E.D.A. 9.0.1	Id: 1/1

Overview of 8051 Development Board Kit:



Most of the above labels are self-explanatory. Below is the description needed for some of the labels. Also check the brief Introduction of 8051 (AT89S52) and Schematic of the Kit in the following pages for detailed understanding.

1. Microcontroller (AT89S52): This kit comes with AT89S52 IC, but it can support any 8051-based 40-pin DIP IC.

2. ISP Programmer Connector: It is used to connect a USB ISP Programmer for programming the microcontroller.

3. USB-C Power Connector: It is a USB Type-C connector. This board can be powered using a +5V USB-C adapter.

4. UDB Conn: These connectors are used to attach this kit to the Universal Development Board (UDB) by Amotech Labs. Check UDB details on our website.

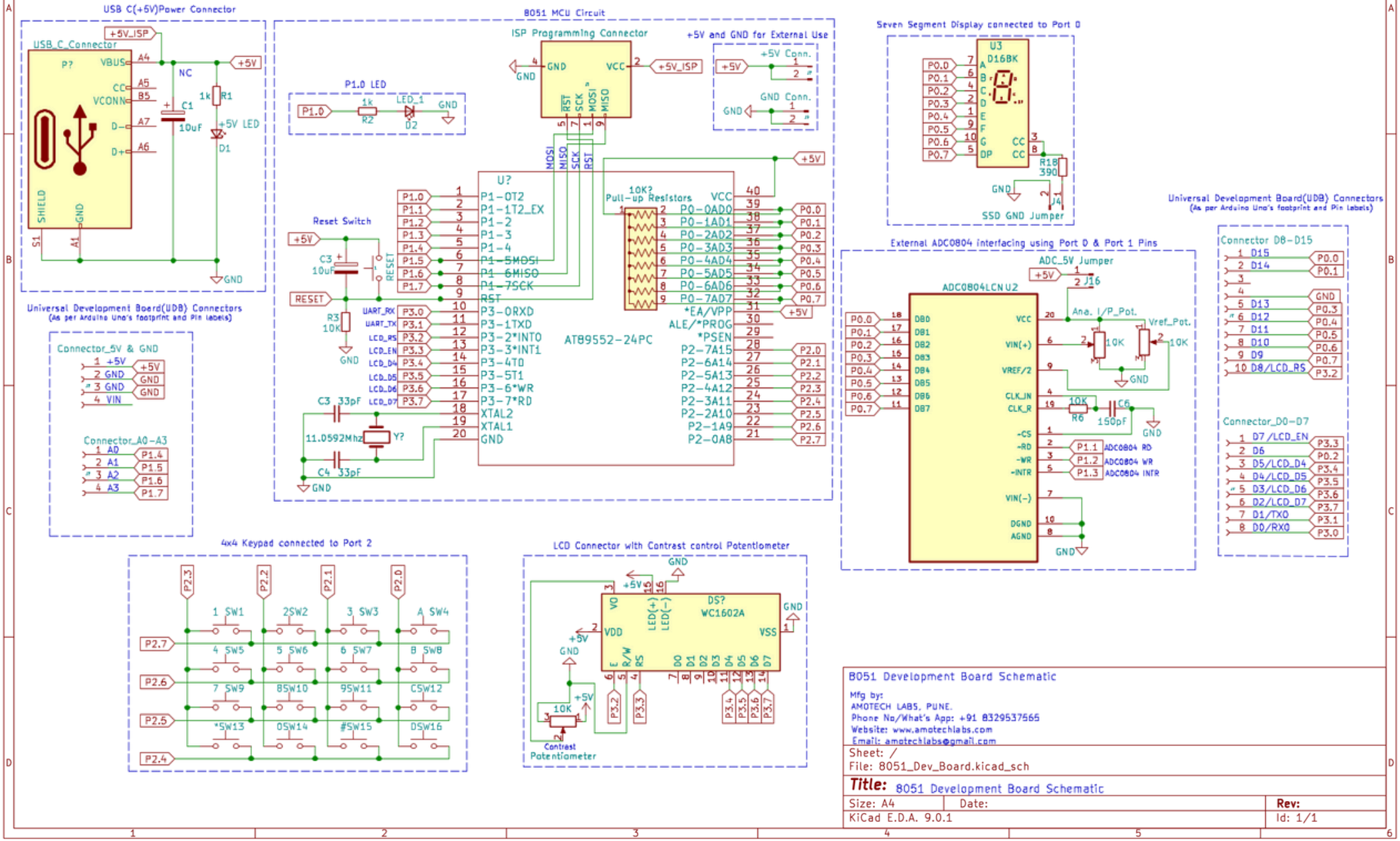
5. 4×4 Keypad: The 4×4 keypad is connected to Port 2 of the 8051 microcontroller.

6. 7-Segment Display: It is optionally connected to Port 0 of the MCU. Port 0 values are reflected on the display when its jumper is inserted. The jumper connects the common cathode of the 7-segment display to ground.

7. Vref Vtg. Pot.: This connector can be used to insert a 10 kΩ potentiometer to adjust the reference (Vref) voltage for ADC0804. When this potentiometer is not inserted, the default Vref is 5 V, provided no external Vref is applied.

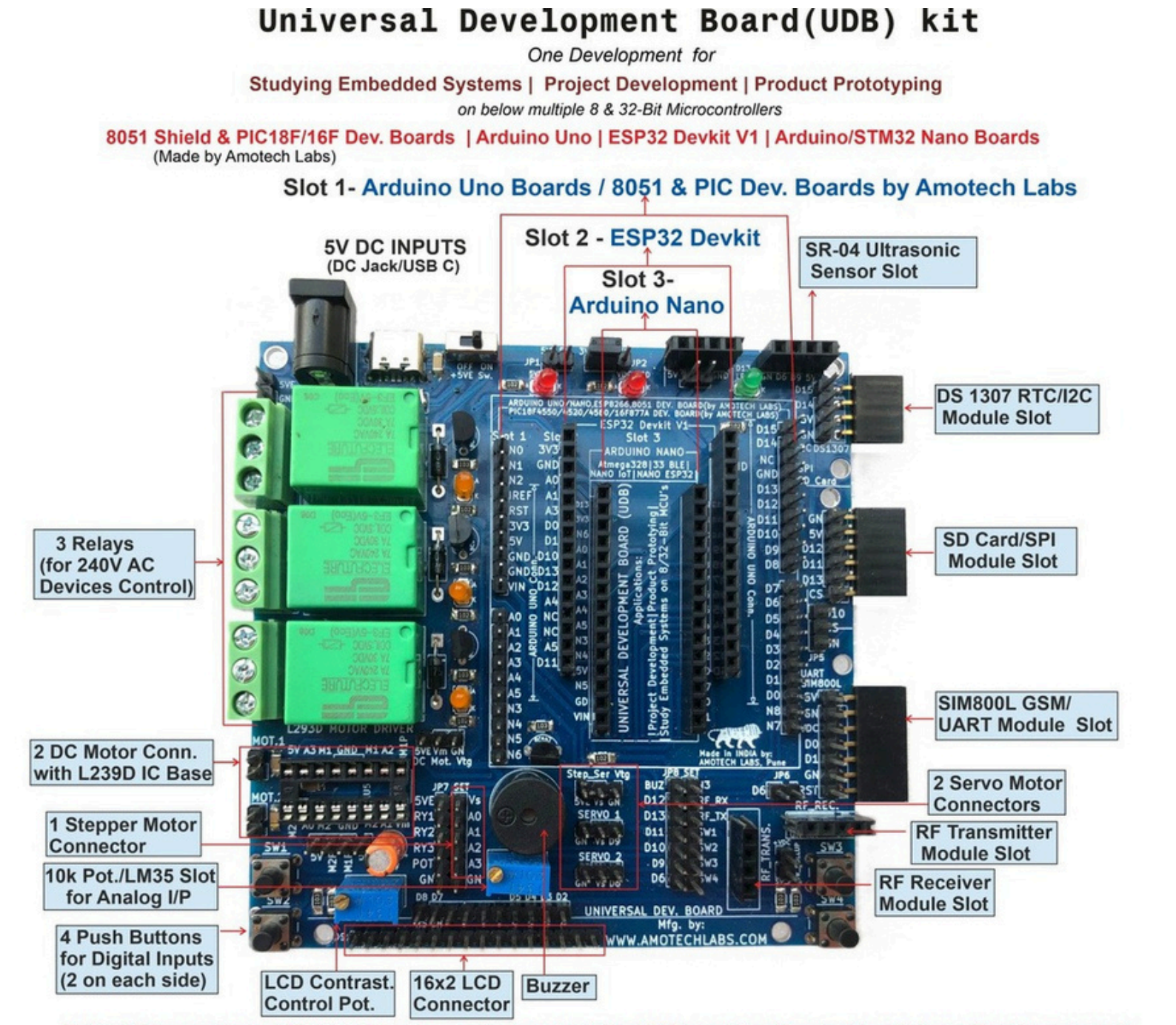
8. Analog I/P Vtg. Pot.: This connector allows insertion of a 10 kΩ potentiometer or LM35 sensor to provide a variable analog input voltage to ADC0804.

8051 Development Board Schematic

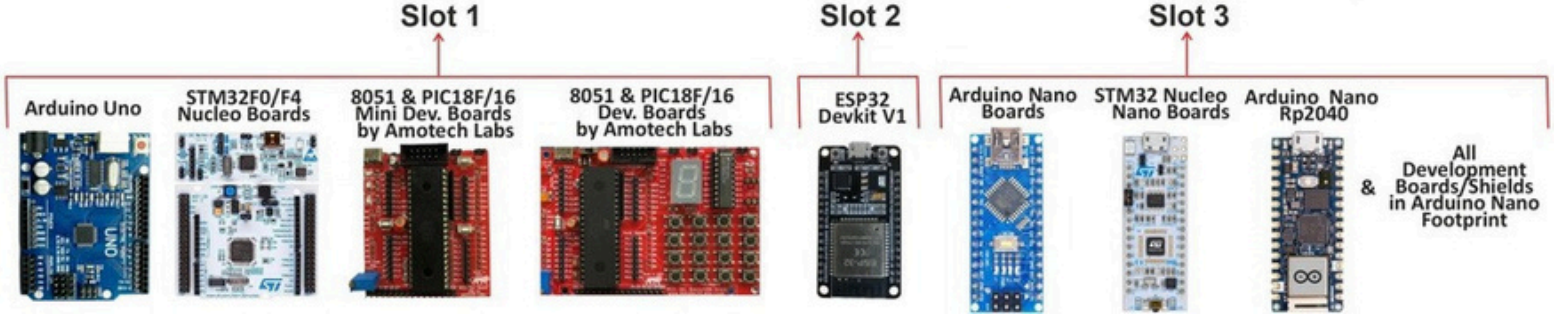


8051 Development Board Schematic	
Mfg by: AMOTECH LABS, PUNE. Phone No/What's App: +91 8329537565 Website: www.amotechlabs.com Email: amotechlabs@gmail.com	
Sheet: / File: 8051_Dev_Board.kicad_sch	
Title: 8051 Development Board Schematic	
Size: A4 Date:	**Rev:**
KiCad E.D.A. 9.0.1	Id: 1/1

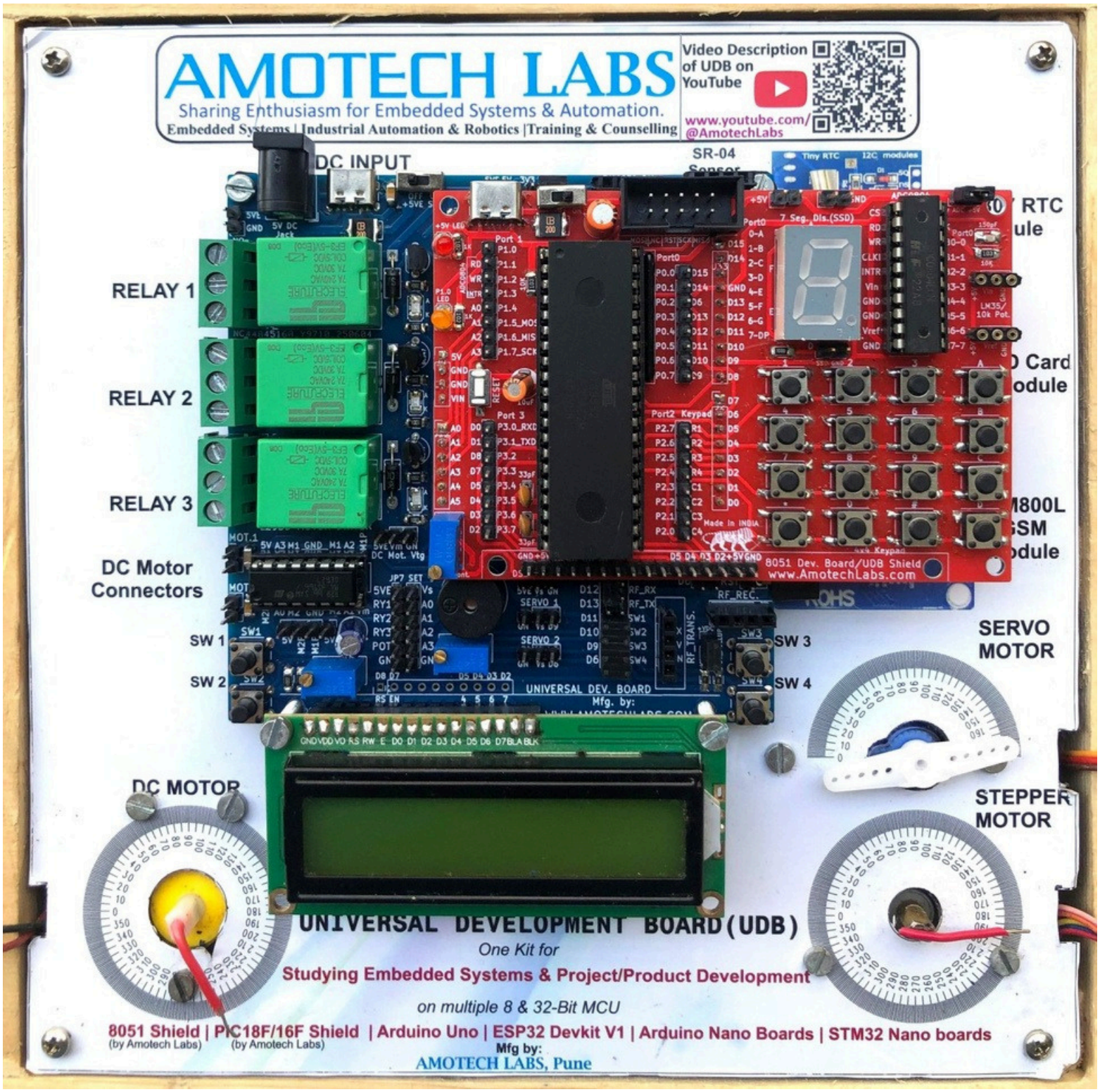
'8051 Development Board' can be mounted on below UDB Kit:



Below Micro-controller Shields/Boards can be inserted into UDB and interfaced with all above Peripherals on it

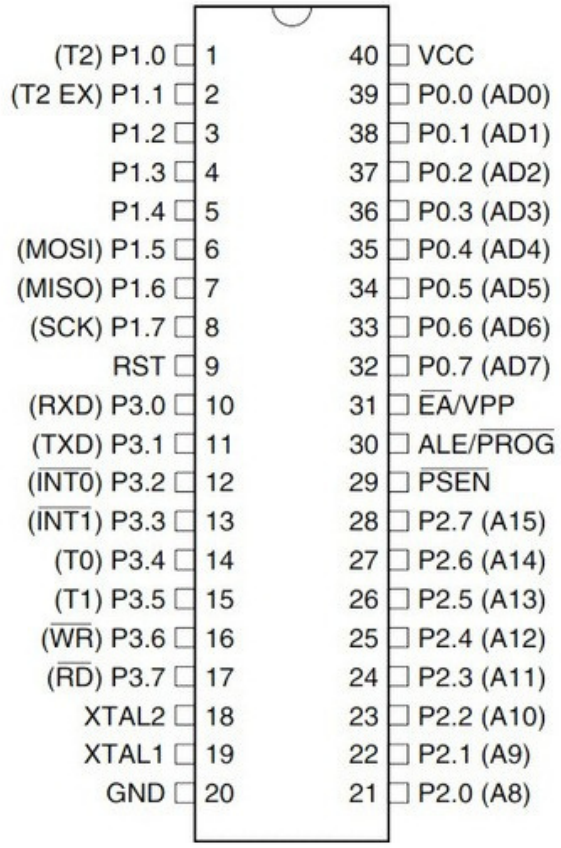


'8051 Development Board' mounted on UDB Kit's Slot 1 can be seen in the below UDB Kit which has DC, Servo and Stepper Motors mounted on-board.



Note: This Kit was assembled for College Lab use in Wooden Box Enclosure. That's why it is fitted on a Acrylic Sheet with Motors mounted on it for studying their interfacing using 8051 MCU & UDB. For Students, only UDB is also available to buy with 'UDB 2 Wheel Robot Acrylic Sheet'. Students can configure the UDB by adding the peripherals as per their application needs.

Introduction to AT89S52 Microcontroller



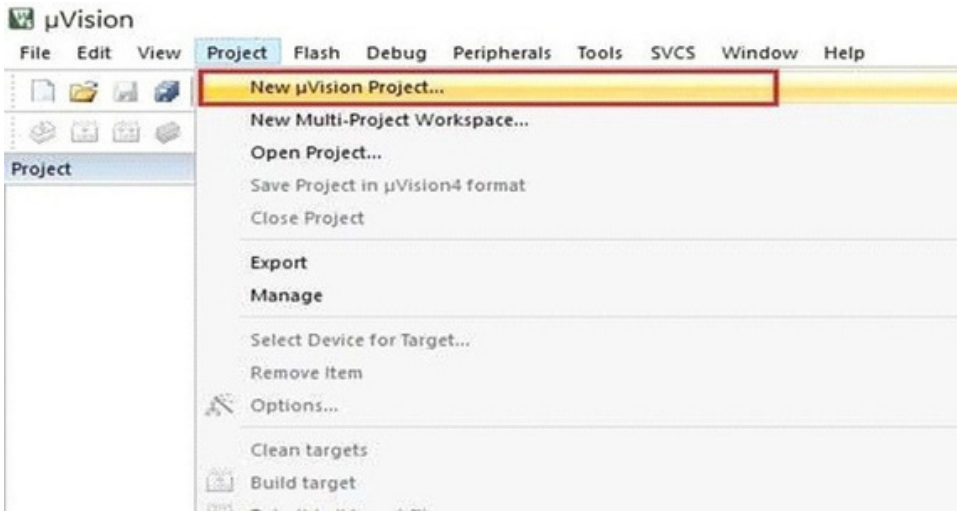
Features of AT89S52:

- Operating Frequency: DC to 33 MHz
- Program Memory Size: 8 KB Flash
- Program Memory (Instructions): 8192 instructions
- Data Memory (Internal RAM): 256 bytes
- Data EEPROM Memory: Not available
- Interrupt Sources: 6
- I/O Ports: 4 ports (Port 0, Port 1, Port 2, Port 3), total 32 I/O lines
- Timers/Counters: 3 timers, 16-bit each (Timer 0, Timer 1, Timer 2)
- Capture/Compare/PWM Modules: Not available
- Enhanced Capture/Compare/PWM Modules: Not available
- Serial Communication Interface: UART
- Universal Serial Bus (USB): Not available
- Streaming Parallel Port (SPP): Not available
- Analog Comparators: Not available
- Programmable Low-Voltage Detect: Not available
- Instruction Set Size: 255 instructions
- Supported Packages: 40-pin DIP, 44-pin PLCC, 44-pin TQFP

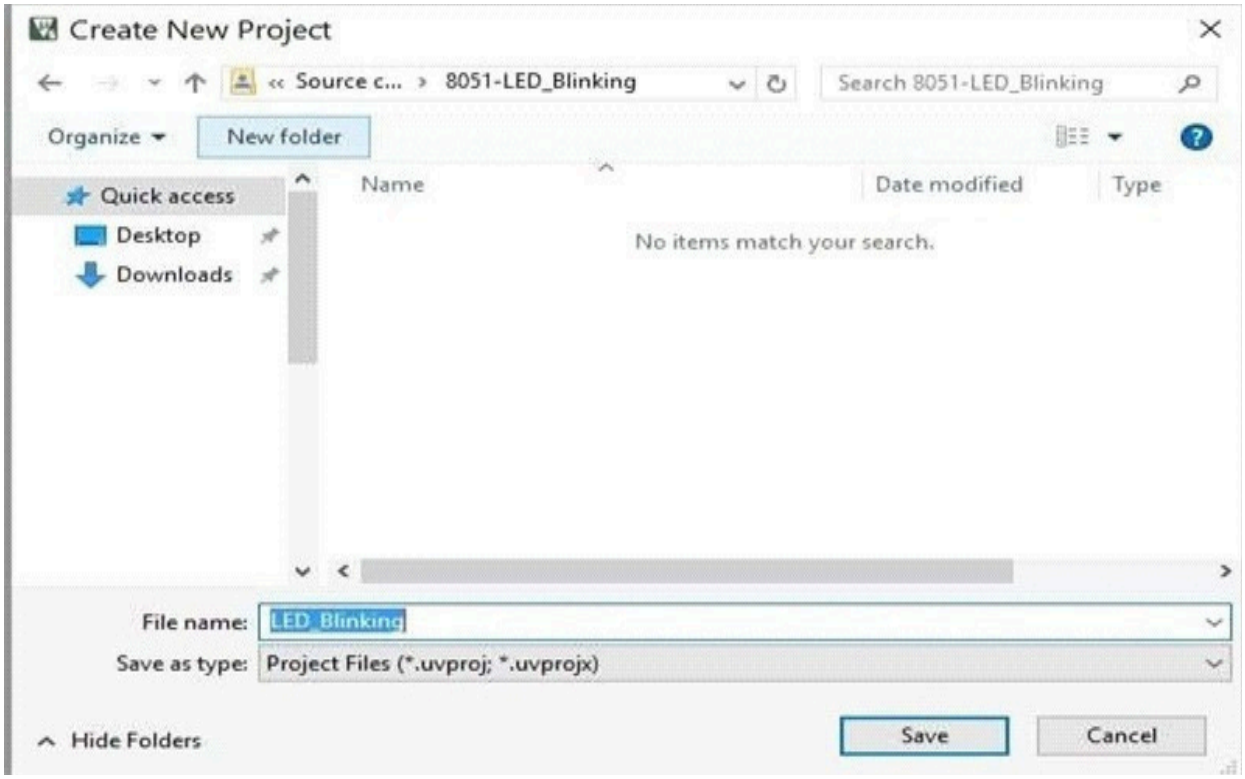
Keil IDE for Programming of 8051 (AT98S52) MC

Below are the steps to program any AT89S52 code

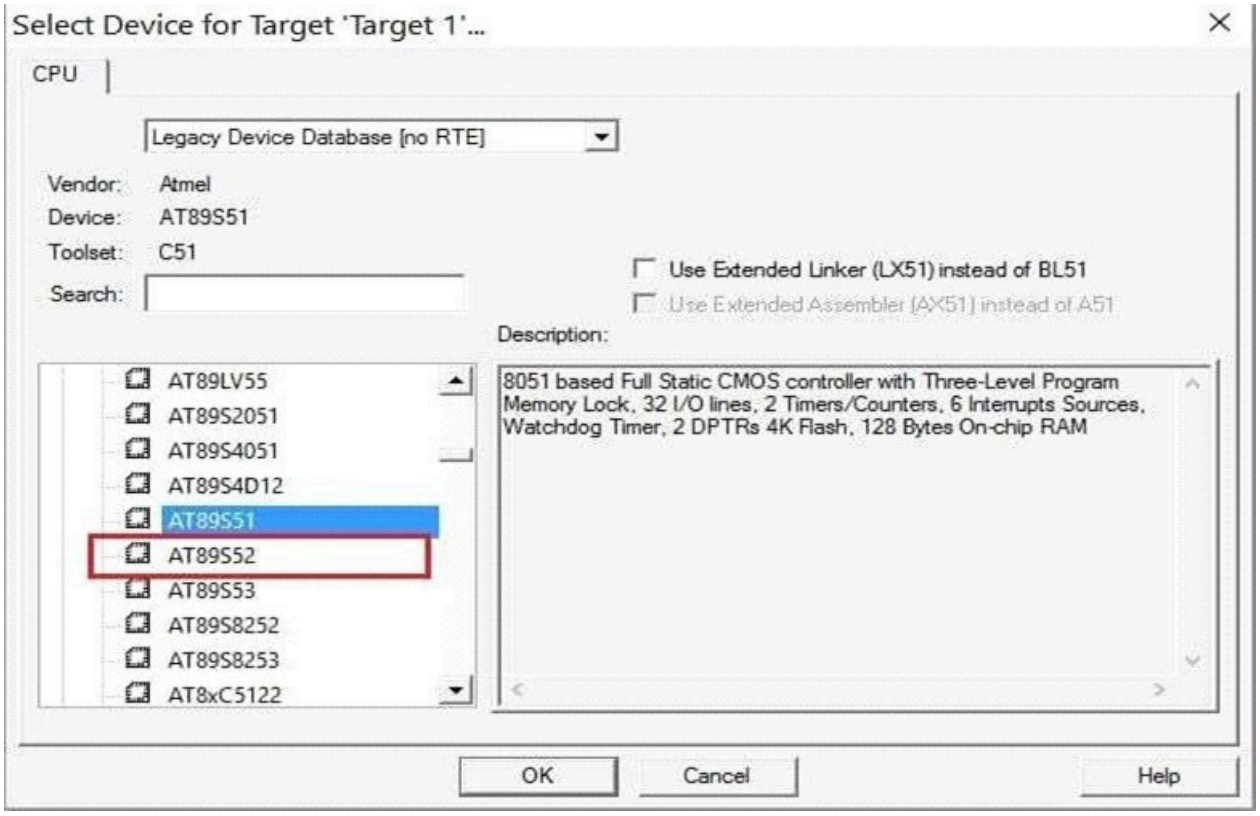
1.Select New µVision Project from the Project Menu.



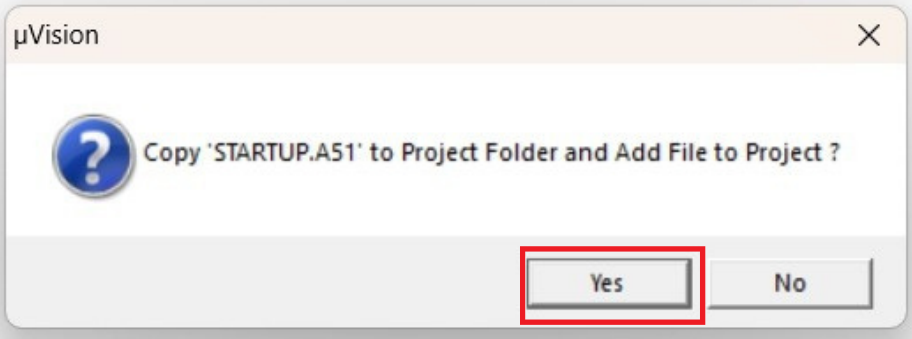
2. Create New Project" window will pop up, type a project name and location for the project and save.



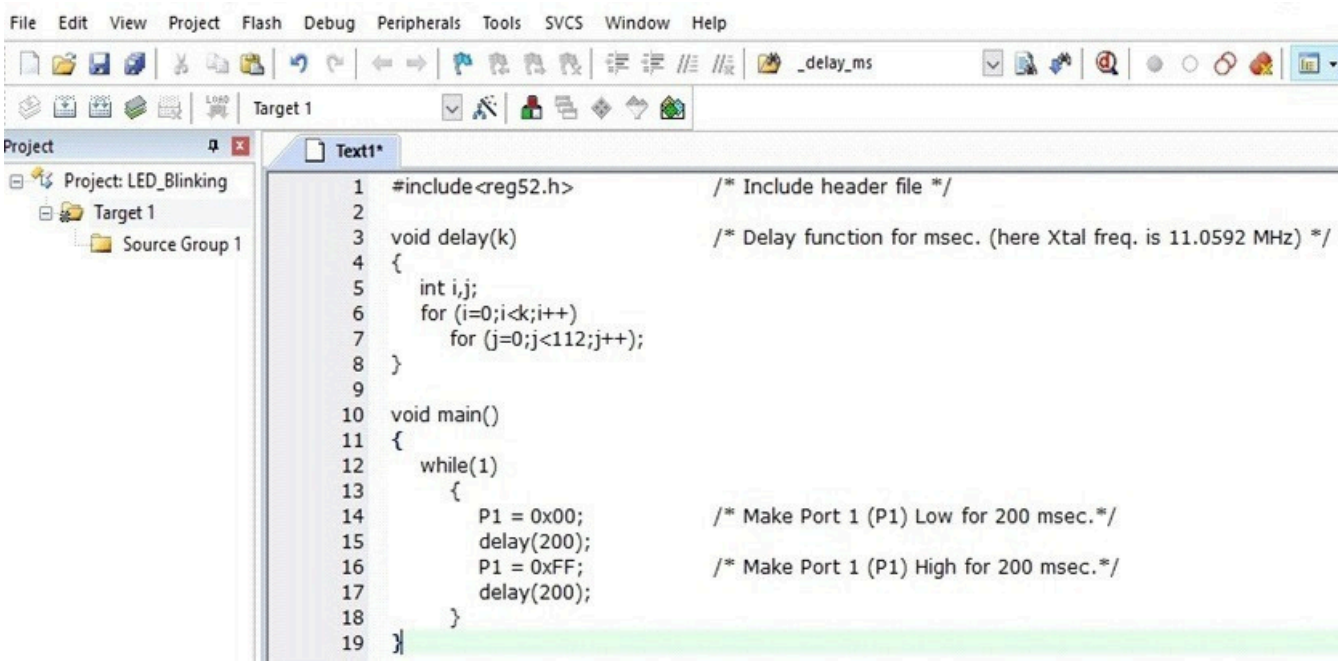
3.Select Device for Target" window will pop up, select your device (here we selecting AT89S52)



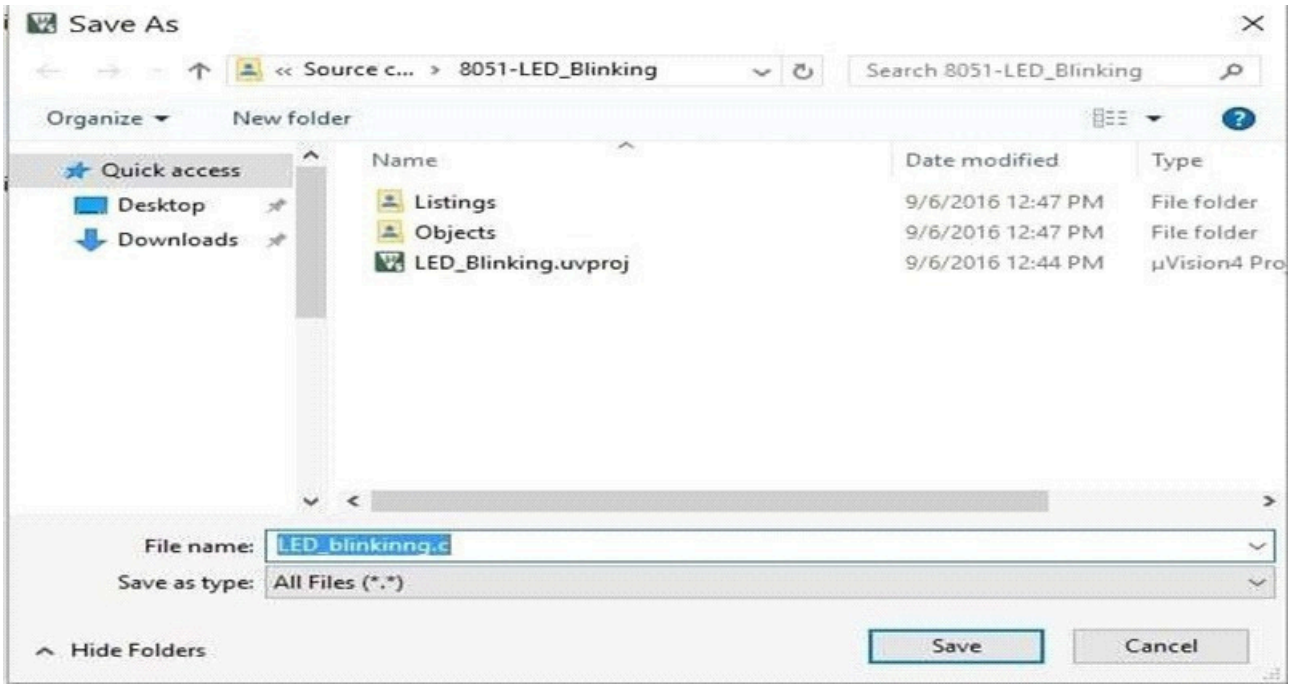
4."µVision" window will ask for copy STARTUP.A51 to the project folder and add a file to the project (If necessary you can select "No" but we recommended you to select "Yes" because it will help to run the program smoothly.)



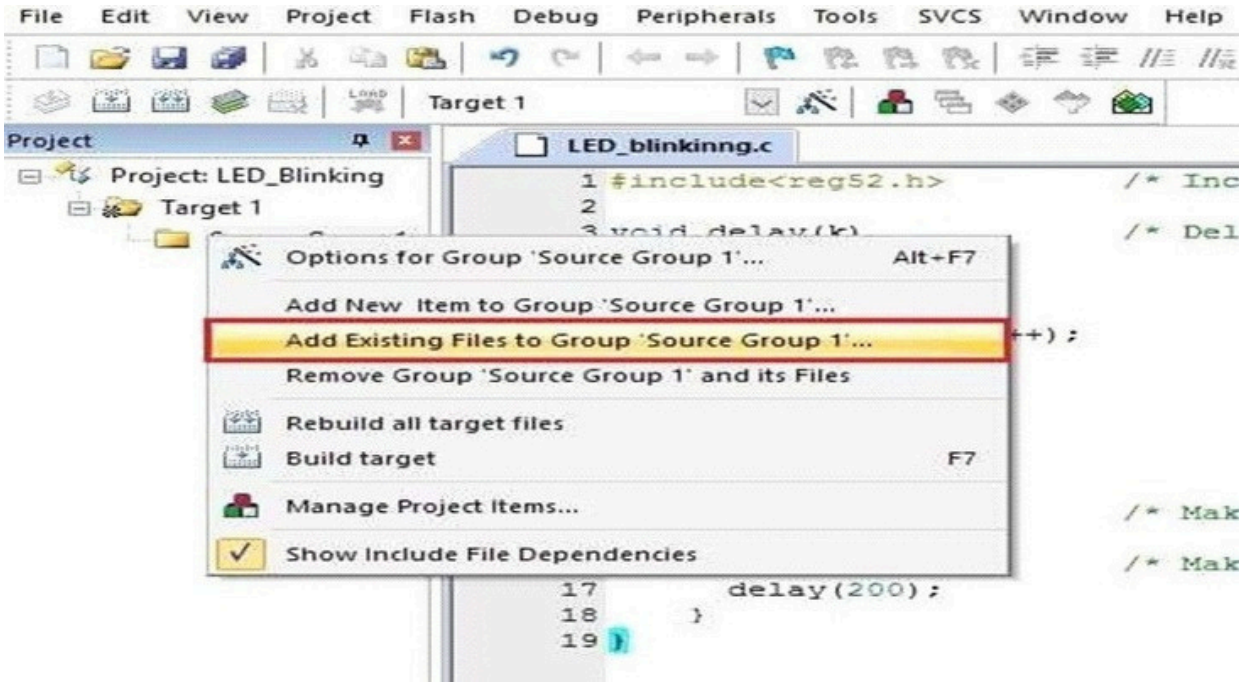
5.Now select New file from the File menu and type your program code (here we have typed LED blinkingprogram)



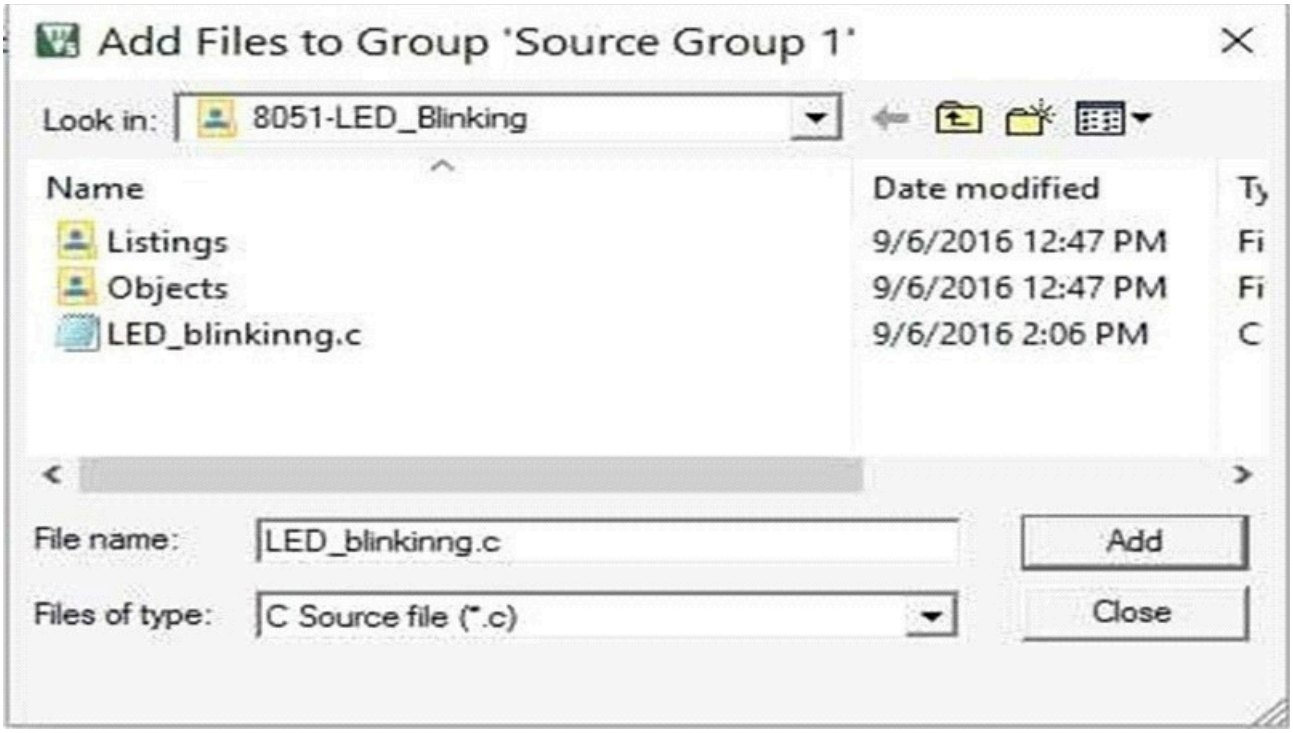
6.Save program code with ".c" extension (In case if you are using assembly language then save



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

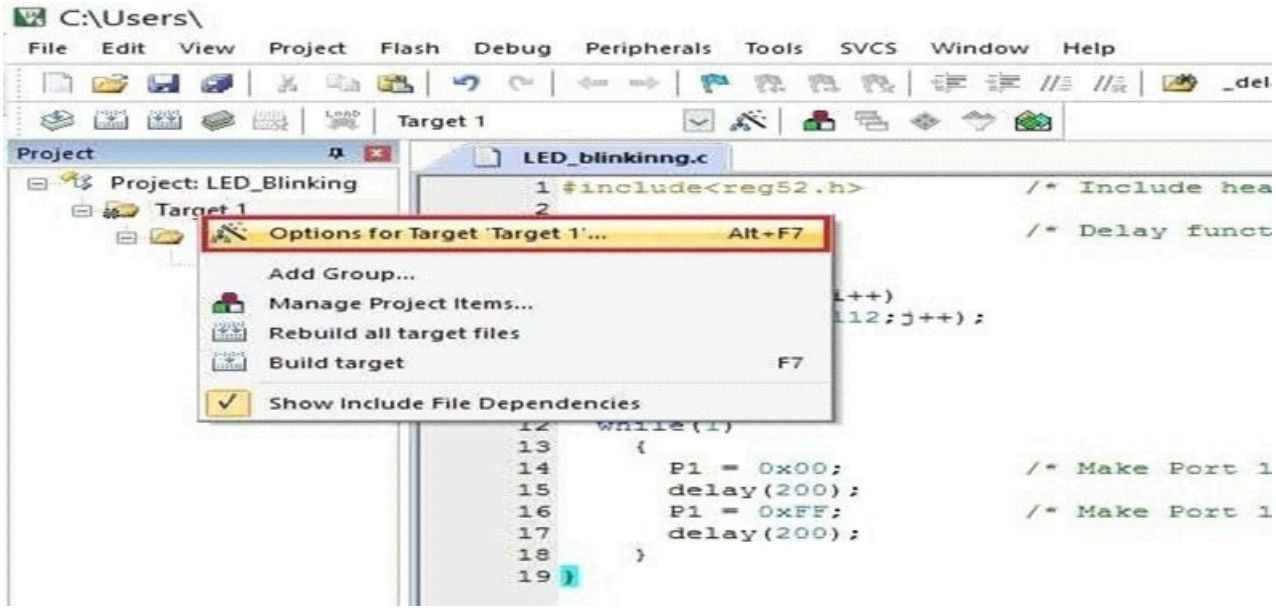


8.Select the program file saved with “.c “or “.asm” (in case of assembly language) and add it. Then close that window. You can see the added file in the “Source Group 1” folder in the left project window.

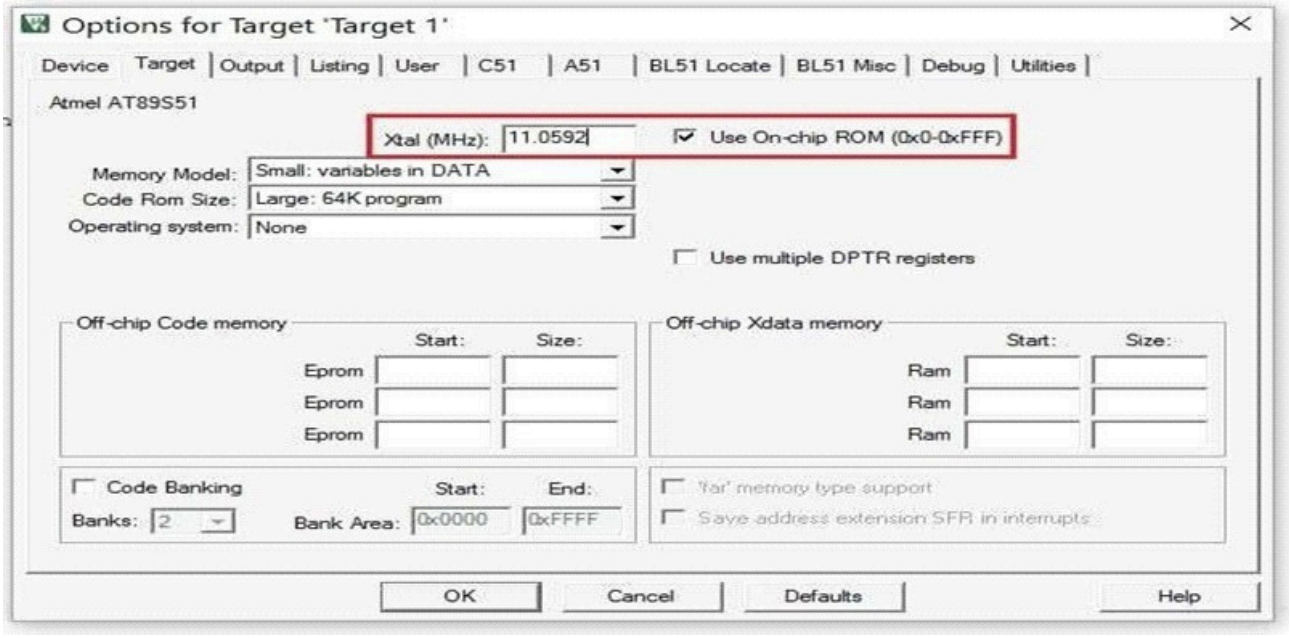


9..Now select the Project menu and click on “Build target”, it will build a project and give status in the Buildoutput window with Error and Warning count if any.

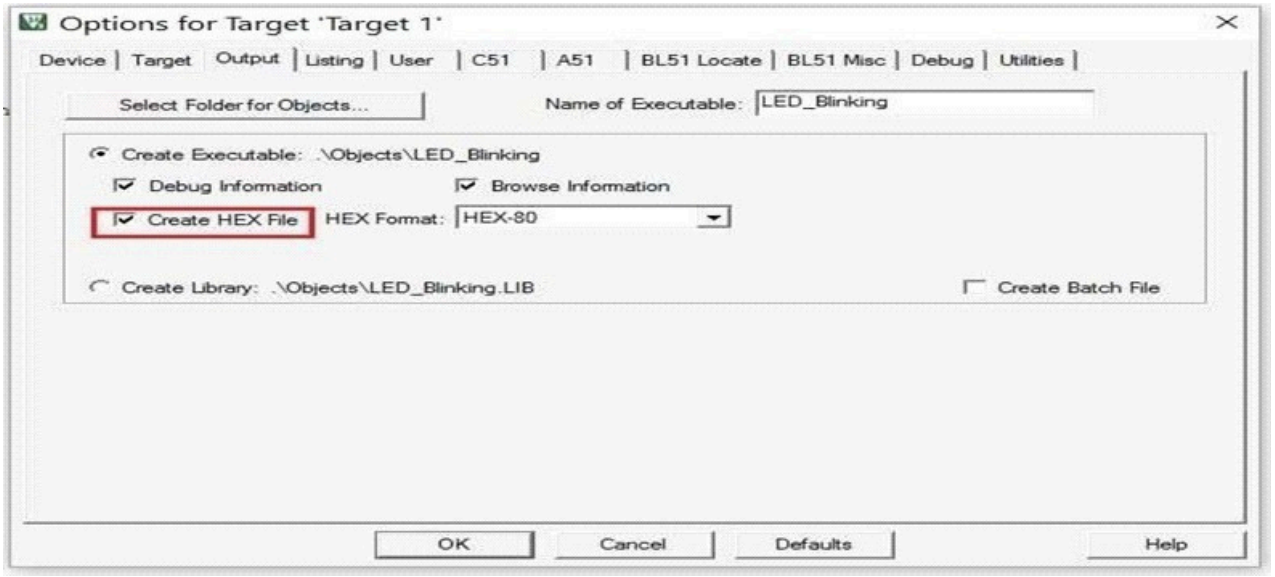
10.To create a Hex file right click on Target 1 and select Option for Target ‘Tar



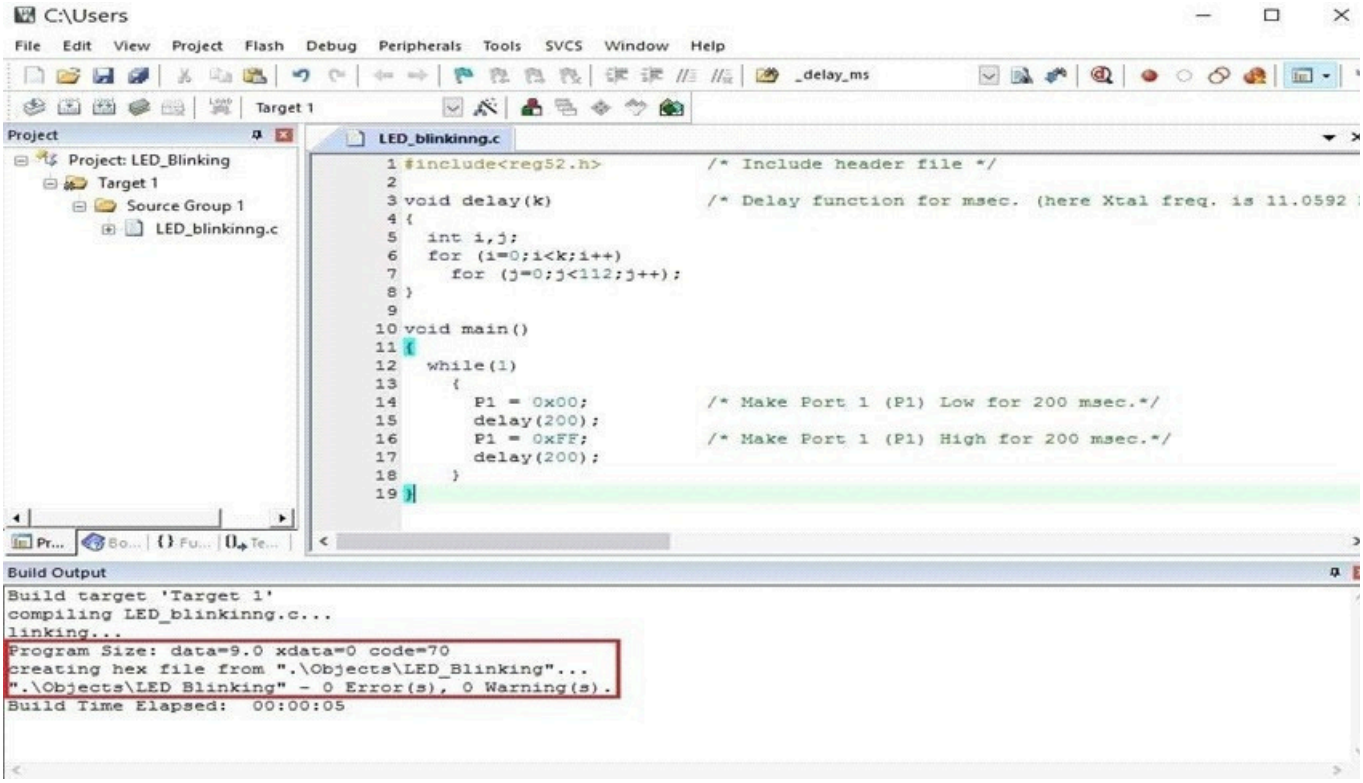
11.The target window will pop up, enter Xtal (MHz) frequency (here we used 11.0592 MHz), and tick mark in front of the “Use On-chip ROM” tag.



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



13 .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

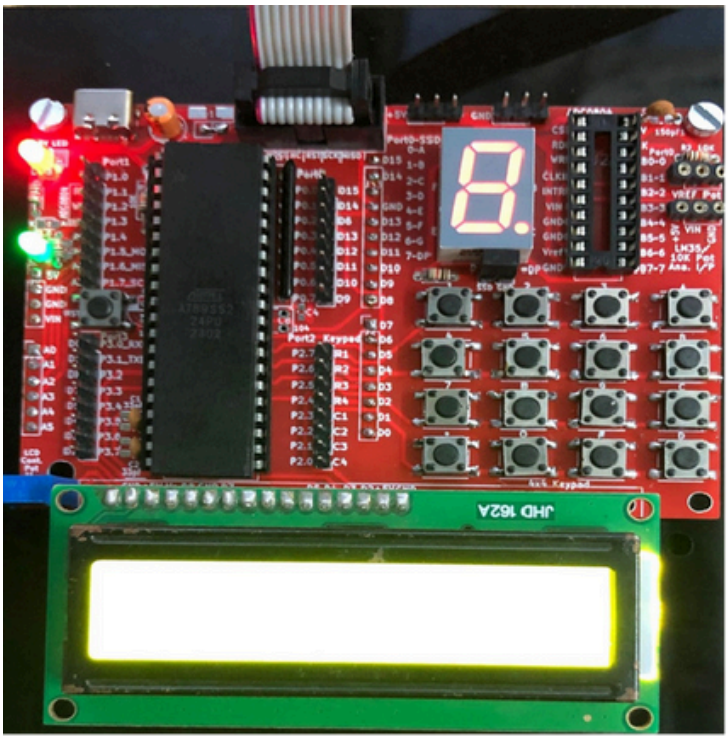


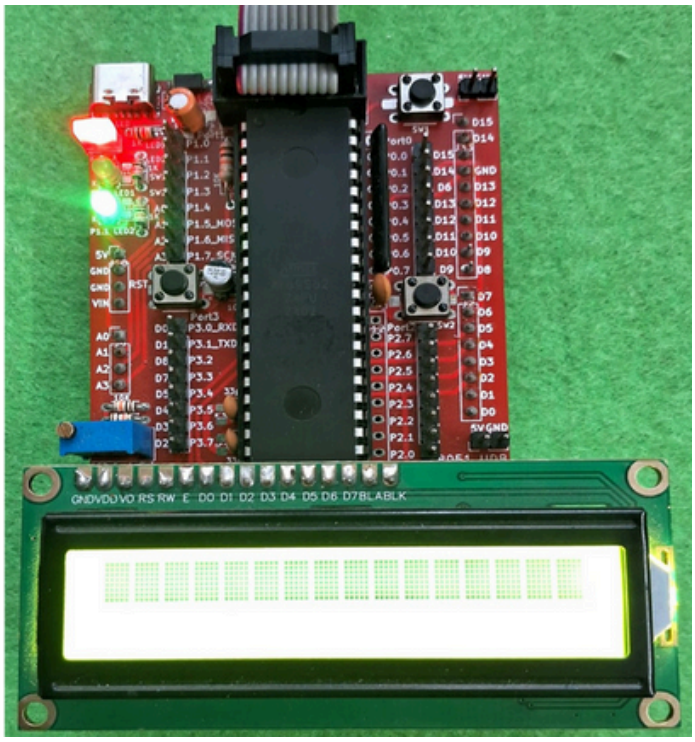
14.Check Video Tutorials on our YouTube Channel, for Instructions of Installing ISP Programmer Drivers, Compiling the program and Downloading the Hex File into our 8051 Development Boards. In Videos for '8051 Development Boards', under their description you will also find a Google Drive Link to download ISP Programmer, Drivers, Programmer Software Files, Sample Programs and Schematic.

YouTube Channel : https://www.youtube.com/@AmotechLabs

Lab 1 - LED Blinking

Aim: To study the LED's Interfacing with AT89S52 and blink them in different patterns with delay.





Procedure:

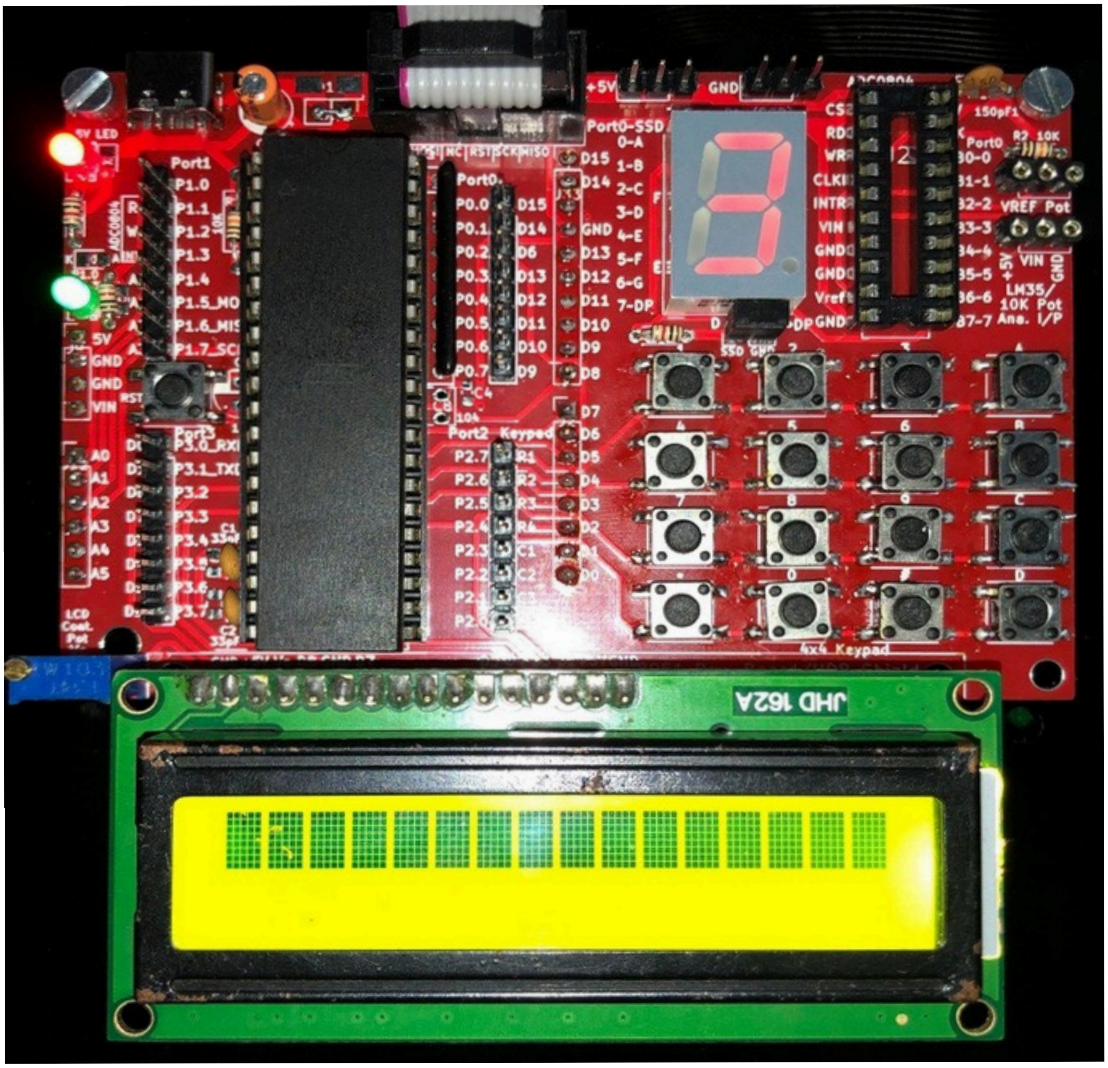
1.Read the LED Blink Program on the next page with all the comments explaining the code.
2.Open the Project folder in Keil µVision IDE or create a new project following the Keil manual.
3.Build the project and generate the HEX file.
4. Use a USB ISP programmer along with ProgISP software to download the Hex file into your 8051 based 89S52 Microcontroller.

Program

```
#include <reg52.h>

sbit LED1 = P1^0;

void delay(unsigned int count);

void main(void)
{
    while(1)    \
    {
        LED1 = 0;
        delay(1000);

        LED1 = 1;
        delay(1000);
    }
}

void delay(unsigned int count)
{
    unsigned int i, j;
    for(i = 0; i < count; i++)
        for(j = 0; j < 112; j++);
}
```

Lab 2 - SSD Counter

Aim: To study the Interfacing SSD(Seven Seg. Displays) with AT89S52 by doing SSD Counter Program.



Procedure:

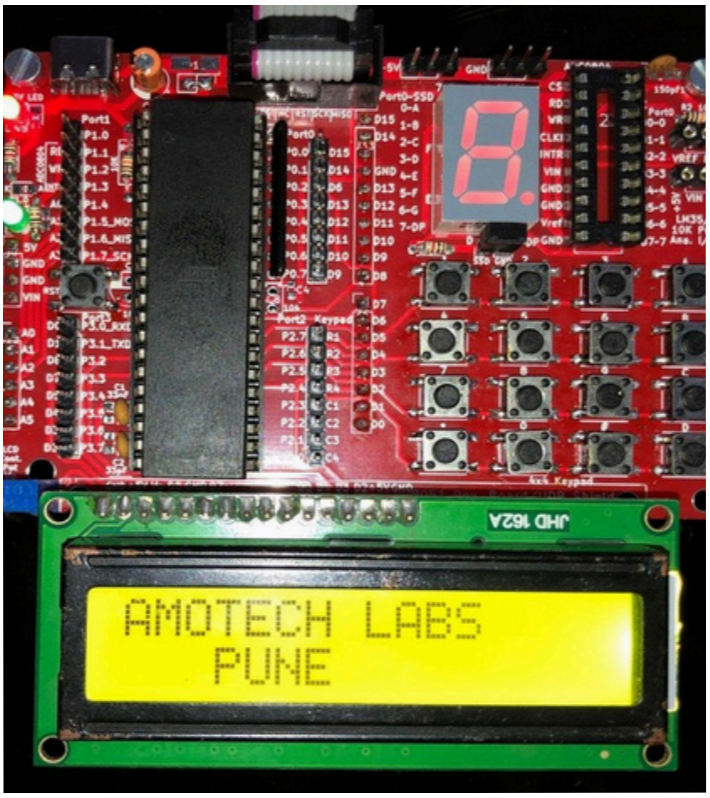
1.Read the SSD Counter Program with comments explaining the code.
2.Open the Project folder in Keil µVision IDE or create a new project following the Keil manual.
3.Build the project and generate the HEX file.Use a USB ISP programmer along with ProgISP software to download the Hex file into your 8051 based 89S52 Microcontroller.
4.Insert the SSD Ground jumper to provide ground to both SSDs.

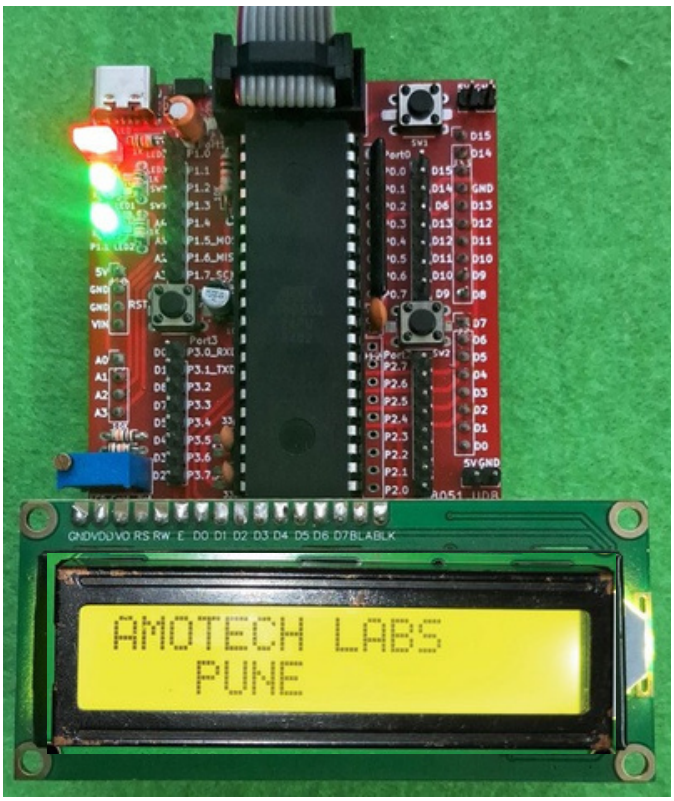
Program:

```
#include <reg52.h>          // special function register declarations
void delay(unsigned int count);
// Segment patterns for digits 0-9 (Common Cathode)
unsigned char segment_code[] = {
    0x3F,   // 0
    0x06,   // 1
    0x5B,   // 2
    0x4F,   // 3
    0x66,   // 4
    0x6D,   // 5
    0x7D,   // 6
    0x07,   // 7
    0x7F,   // 8
    0x6F    // 9
};

void main(void) {
    unsigned char i;
    while(1) {                      // infinite loop
        for(i = 0; i < 10; i++) {   // loop through digits 0-9
            P0 = segment_code[i];   // send pattern to Port 0
            delay(1000);            // delay for visibility
        }
    }
}

// Delay function (approximate millisecond delay)
void delay(unsigned int count) {
    unsigned int i, j;
    for(i = 0; i < count; i++) {
        for(j = 0; j < 112; j++);   // calibrated loop
    }
}
```

Lab 3 - LCD Interfacing

Aim: To study the 16x2 LCD Interfacing with AT89S52.





.**Procedure:** 1.Read the LCD Interfacing Program with all comments explaining the code.
2.Open Keil µVision, create/open a project for 8051 (AT89C51), write the program, and build it to generate the Hex file.
3.Use a USB ISP programmer along with ProgISP software to download the Hex file into your 8051 based 89S52 Microcontroller.
4.Adjust the LCD contrast using the 10k pot. Once programmed and disconnected, the expected content will appear on the LCD.

Program:

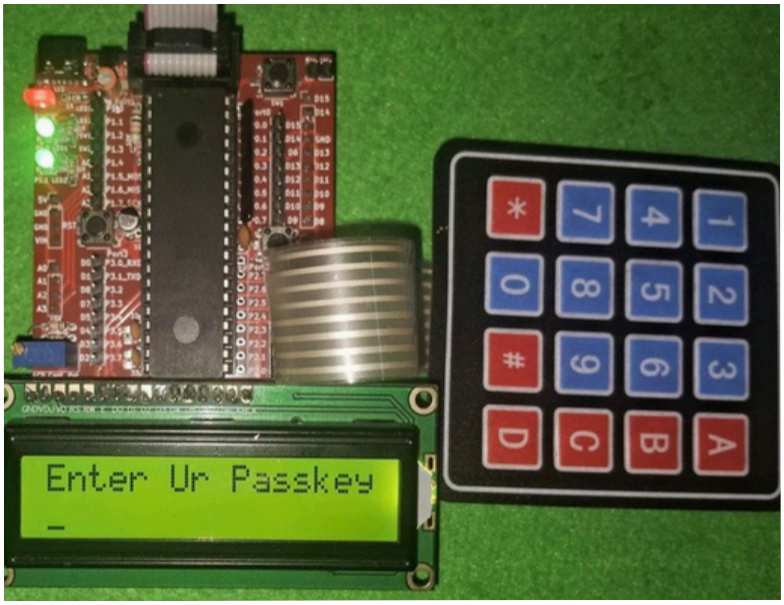
```
#include <reg52.h>

/* LCD Data pins (4-bit mode) */
sbit D7 = P3^7;   // LCD D7  -> Arduino D2 (example)
sbit D6 = P3^6;   // LCD D6  -> Arduino D3
sbit D5 = P3^5;   // LCD D5  -> Arduino D4
sbit D4 = P3^4;   // LCD D4  -> Arduino D5

/* LCD Control pins */
sbit rs = P3^2;   // Register Select -> Arduino D8
sbit en = P3^3;   // Enable          -> Arduino D7

#define LCD_Port P1

/* Delay function (approx 1ms for 11.0592 MHz crystal) */
void delay(unsigned int count)
{
    unsigned int i, j;
    for(i = 0; i < count; i++)
        for(j = 0; j < 112; j++);
}

/* LCD Command Function */
void LCD_Command(char cmnd)
{
    /* Send upper nibble */
    LCD_Port = (cmnd & 0xF0);
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;

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

    /* Send lower nibble */
    LCD_Port = (cmnd & 0x0F) << 4;
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;

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

/* LCD Character Display */
void LCD_Char(char char_data)
{
    /* Send upper nibble */
    LCD_Port = (char_data & 0xF0);
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;

    rs = 1;
    en = 1;
    delay(1);
```

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

    /* Send lower nibble */
    LCD_Port = (char_data & 0x0F) << 4;
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;

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

/* Display String */
void LCD_String(char *str)
{
    unsigned int i;
    for(i = 0; str[i] != 0; i++)
        LCD_Char(str[i]);
}

/* Display String at row & column */
void LCD_String_xy(char row, char pos, char *str)
{
    if(row == 0)
        LCD_Command((pos & 0x0F) | 0x80);
    else if(row == 1)
        LCD_Command((pos & 0x0F) | 0xC0);
    LCD_String(str);
    delay(1);
    en = 0;
    delay(2);

    /* Send lower nibble */
    LCD_Port = (char_data & 0x0F) << 4;
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;

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

/* Display String */
void LCD_String(char *str)
{
    unsigned int i;
    for(i = 0; str[i] != 0; i++)
        LCD_Char(str[i]);
}

/* Display String at row & column */
void LCD_String_xy(char row, char pos, char *str)
{
    if(row == 0)
        LCD_Command((pos & 0x0F) | 0x80);
    else if(row == 1)
        LCD_Command((pos & 0x0F) | 0xC0);
    LCD_String(str);
```

```
}

/* LCD Initialization */
void LCD_Init(void)
{
    delay(20);
    LCD_Command(0x02);  // 4-bit mode
    LCD_Command(0x28);  // 16x2 LCD
    LCD_Command(0x0C);  // Display ON, Cursor OFF
    LCD_Command(0x06);  // Auto increment cursor
    LCD_Command(0x01);  // Clear display
    LCD_Command(0x80);  // Cursor at home
}

/* Main Program */
void main()
{
    LCD_Init();

    LCD_String_xy(0, 0, "AMOTECH LABS");
    LCD_String_xy(1, 0, "PUNE");

    while(1);
}
```

Lab 4 - 4x4 Keypad Interfacing

Aim : To study the Interfacing of 4x4 Keypad with AT89S52.



Procedure:

1.Read the Keypad Interfacing Program on the next page with all the comments explaining the program.
2.Open the Project folder for the program in Keil µVision IDE or create a new project following the procedure explained in the Keil manual. Build the project and generate the HEX file.
3.Use a USB ISP programmer along with ProgISP software to download the Hex file into your 8051 based 89S52 Microcontroller.
4.If you entered the correct passkey then you will get acess otherwise you will get the message “Access denied”.

Program:

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

#define lcdport P3

/* LCD pins */
sbit rs = P3^2;
sbit en = P3^3;

/* Keypad */
sbit col1 = P2^3;
sbit col2 = P2^2;
sbit col3 = P2^1;
sbit col4 = P2^0;

sbit row1 = P2^7;
sbit row2 = P2^6;
sbit row3 = P2^5;
sbit row4 = P2^4;

/* LEDs */
sbit LED = P1^0;      // Access LED
#define ERR_LED P0    // Error LEDs

char pass[5];
int i;

/* Delay */
void delay(int itime)
{
    int i,j;
    for(i=0;i<itime;i++)
}

/* LCD enable */
void daten()
{
    rs = 1;
    en = 1;
    delay(5);
    en = 0;
}

void cmden()
{
    rs = 0;
    en = 1;
    delay(5);
    en = 0;
}

/* LCD data */
void lcddata(unsigned char ch)
{
    lcdport = ch & 0xF0;
    daten();
    lcdport = (ch<<4) & 0xF0;
    daten();
}
```

Program:

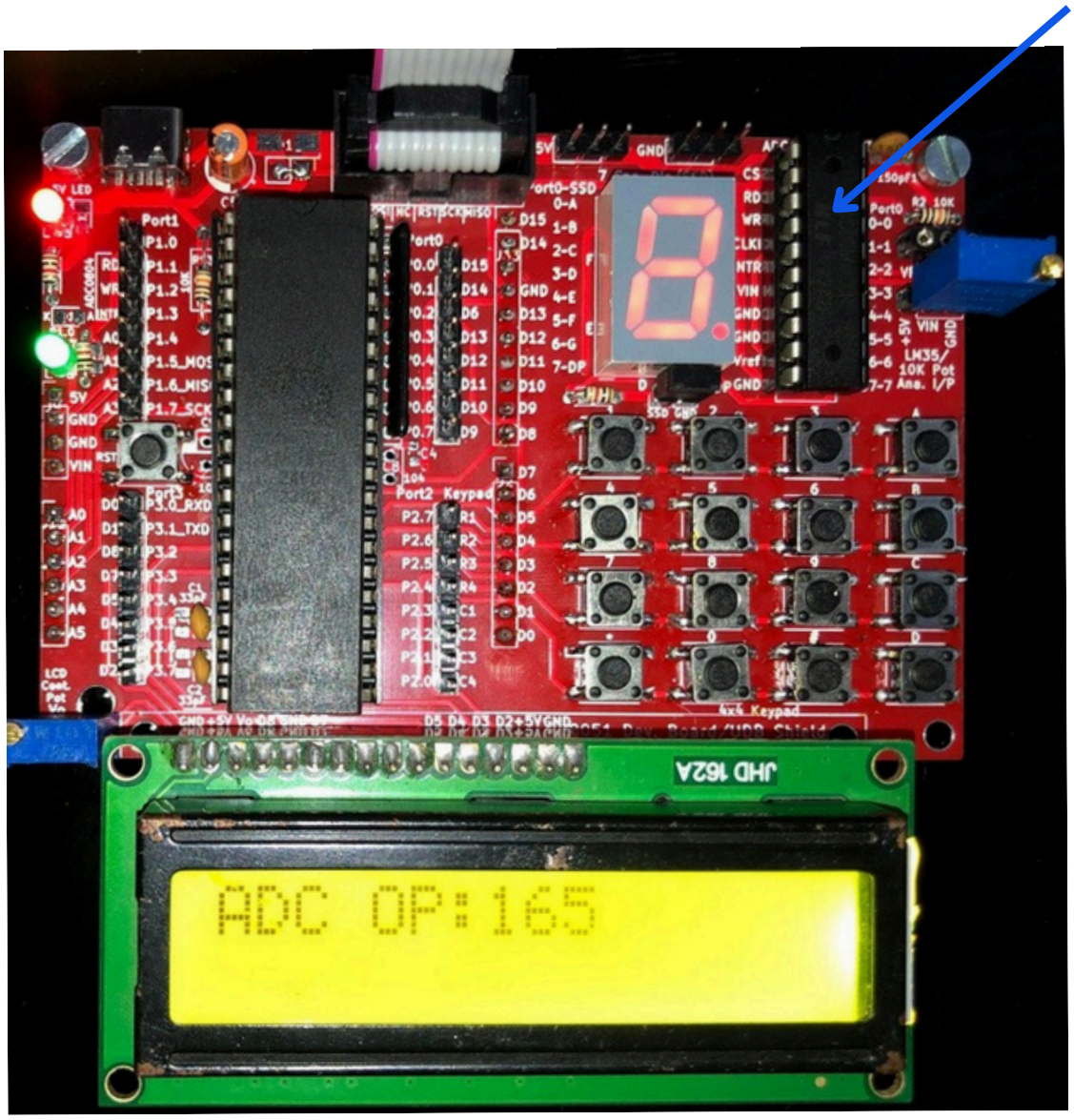
```
/* LCD command */
void lcdcmd(unsigned char ch)
{
    lcdport = ch & 0xF0;
    cmden();
    lcdport = (ch<<4) & 0xF0;
    cmden();
}

/* LCD string */
void lcdstring(char *str)
{
    while(*str)
        lcddata(*str++);
}

/* LCD init */
void lcd_init()
{
    lcdcmd(0x02);
    lcdcmd(0x28);
    lcdcmd(0x0E);
    lcdcmd(0x01);
}

/* ================= KEYPAD (ORIGINAL STYLE) ================= */
void keypad()
{
    int cursor = 0xC0, flag = 0;

    lcdcmd(1);
    lcdstring("Enter Ur Passkey");
    lcdcmd(0xC0);

  i = 0;

  while(i < 4)
  {
      flag = cursor;

      col1 = 0; col2 = col3 = col4 = 1;
      if(!row1){ lcddata('*'); pass[i++]='1'; cursor++; while(!row1); }
      else if(!row2){ lcddata('*'); pass[i++]='4'; cursor++; while(!row2); }
      else if(!row3){ lcddata('*'); pass[i++]='7'; cursor++; while(!row3); }
      else if(!row4){ lcddata('*'); pass[i++]='*'; cursor++; while(!row4); }

      col2 = 0; col1 = col3 = col4 = 1;
      if(!row1){ lcddata('*'); pass[i++]='2'; cursor++; while(!row1); }
      else if(!row2){ lcddata('*'); pass[i++]='5'; cursor++; while(!row2); }
      else if(!row3){ lcddata('*'); pass[i++]='8'; cursor++; while(!row3); }
      else if(!row4){ lcddata('*'); pass[i++]='0'; cursor++; while(!row4); }

      col3 = 0; col1 = col2 = col4 = 1;
      if(!row1){ lcddata('*'); pass[i++]='3'; cursor++; while(!row1); }
      else if(!row2){ lcddata('*'); pass[i++]='6'; cursor++; while(!row2); }
      else if(!row3){ lcddata('*'); pass[i++]='9'; cursor++; while(!row3); }
      else if(!row4){ lcddata('*'); pass[i++]='#'; cursor++; while(!row4); }

      col4 = 0; col1 = col2 = col3 = 1;
      if(!row1){ lcddata('*'); pass[i++]='A'; cursor++; while(!row1); }
      else if(!row2){ lcddata('*'); pass[i++]='B'; cursor++; while(!row2); }
      else if(!row3){ lcddata('*'); pass[i++]='C'; cursor++; while(!row3); }
      else if(!row4){ lcddata('*'); pass[i++]='D'; cursor++; while(!row4); }

      if(i > 0 && flag != cursor)
      {
```

```
            delay(100);
            lcdcmd(cursor-1);
            lcddata('*');
        }
    }

    pass[4] = '\0';
}

/* Show typed password for 1 sec */
void show_password()
{
    lcdcmd(1);
    lcdstring("Password:");
    lcdcmd(0xC0);
    lcdstring(pass);
    delay(100);
}

/* Blink P0 LEDs twice */
void blink_error()
{
    char k;
    for(k=0;k<2;k++)
    {
        ERR_LED = 0xFF;
        delay(100);
        ERR_LED = 0x00;
        delay(100);
    }
}

/* Accept */
void accept()
{
    LED = 1;
    ERR_LED = 0x00;
    lcdcmd(1);
    lcdstring("Access Granted");
    delay(300);
}

/* Wrong */
void wrong()
{
    LED = 0;
    lcdcmd(1);
    lcdstring("Access Denied");
    blink_error();
}

/* MAIN */
void main()
{
    LED = 0;
    ERR_LED = 0x00;

    lcd_init();
    lcdstring("AMOTECH LABS");
    lcdcmd(0xC0);
    lcdstring("   PUNE");
    delay(1000);
```

```
    lcdcmd(1);
    lcdstring("Electronic Code");
    lcdcmd(0xC0);
    lcdstring("Lock System");
    delay(400);

    while(1)
    {
        keypad();
        show_password();

        if(strncmp(pass,"4201",4)==0)
            accept();
        else
            wrong();
    }
}
```

Lab 5 - ADC 0804 Interfacing

Aim: To study the ADC Interfacing with AT895S2



Procedure:

1.Read the ADC Interfacing Program on the next page with all the comments explaining the program
2.Open the Project folder for the program in Keil µVision IDE or create a new project following the procedure explained in the Keil manual. Build the project and generate the HEX file.
3.Use a USB ISP programmer along with ProgISP software to download the Hex file into your 8051 based 89S52 Microcontroller.
4.Make sure that, a Analog input is given to ADC0804 by using 10k pot or a LM35 sensor which is inserted into the Analog Input slot.

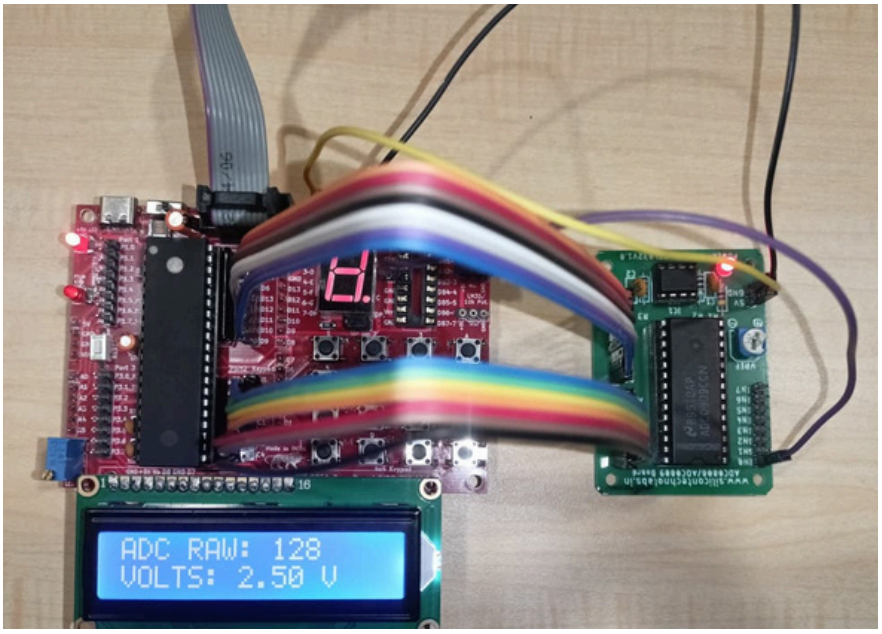
Program

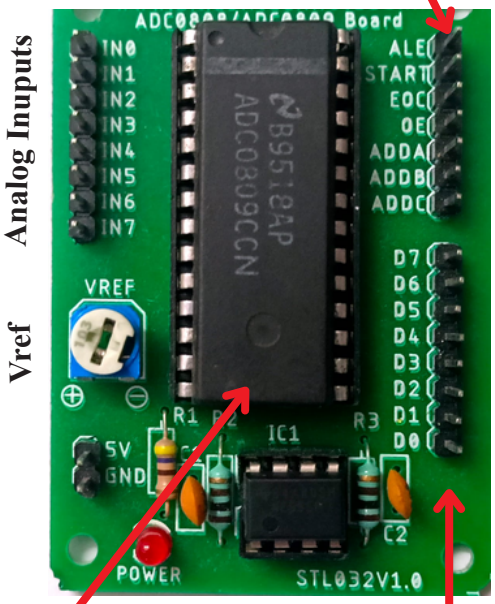
```
#include <reg51.h>

// LCD connections
sbit D7 = P3^7;   // LCD data pin D7
sbit D6 = P3^6;   // LCD data pin D6
sbit D5 = P3^5;   // LCD data pin D5
sbit D4 = P3^4;   // LCD data pin D4
sbit rs = P3^2;   // Register select pin
sbit en = P3^3;   // Enable pin

// ADC connections
sbit rd   = P1^1;   // ADC Read
sbit wr   = P1^2;   // ADC Write
sbit intr = P1^3;   // ADC Interrupt

unsigned char get_value, conv;
unsigned int i;
unsigned int LCD_Port;   // Temporary variable for LCD nibble transfer

// Delay function (~1ms with 11.0592 MHz crystal)
void delay(unsigned int count) {
    unsigned int i, j;
    for(i = 0; i < count; i++)
        for(j = 0; j < 112; j++);
}

// Send command to LCD
void LCD_Command(unsigned char cmnd) {
    LCD_Port = (cmnd & 0xF0);   // upper nibble
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;
    rs = 0;
    en = 1; delay(1); en = 0; delay(2);

    LCD_Port = (cmnd & 0x0F) << 4;   // lower nibble
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;
    rs = 0;
    en = 1; delay(1); en = 0; delay(5);
}

// Send data (character) to LCD
void LCD_Char(unsigned char char_data) {
    LCD_Port = (char_data & 0xF0);   // upper nibble
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;
    rs = 1;
    en = 1; delay(1); en = 0; delay(2);

    LCD_Port = (char_data & 0x0F) << 4;   // lower nibble
    D7 = (LCD_Port & 0x80) ? 1 : 0;
    D6 = (LCD_Port & 0x40) ? 1 : 0;
    D5 = (LCD_Port & 0x20) ? 1 : 0;
    D4 = (LCD_Port & 0x10) ? 1 : 0;
    rs = 1;
    en = 1; delay(1); en = 0; delay(5);
}
```

```
// Send string to LCD
void LCD_String(char *str) {
    int i;
    for(i = 0; str[i] != 0; i++) {
        LCD_Char(str[i]);
    }
}

// Initialize LCD
void LCD_Init(void) {
    delay(20);              // LCD power ON delay
    LCD_Command(0x02);      // 4-bit mode
    LCD_Command(0x28);      // 2 line, 5x7 matrix
    LCD_Command(0x0C);      // Display ON, Cursor OFF
    LCD_Command(0x06);      // Auto increment cursor
    LCD_Command(0x01);      // Clear display
}

// ADC read function
unsigned char adc() {
    wr = 0; delay(1); wr = 1;   // Start conversion
    while(intr == 1);           // Wait for conversion complete
    rd = 0;
    conv = P0;                  // Read ADC output from Port 0
    rd = 1;
    return conv;
}

// Main program
void main() {
    LCD_Init();
    LCD_String("ADC Output:");

    while(1) {
        get_value = adc();

        LCD_Command(0xC0);   // Move to second line
        LCD_Char((get_value/100) + 48);         // Hundreds
        LCD_Char(((get_value/10)%10) + 48);     // Tens
        LCD_Char((get_value%10) + 48);          // Units

        delay(200);
    }
}
```

Lab 6 - ADC 0809 Interfacing

Aim: To study the ADC 0809 Interfacing with AT895S2





Procedure:

1.Read the ADC Interfacing Program on the next page with all the comments explaining the program

2.Open the Project folder for the program in Keil µVision IDE or create a new project following the procedure explained in the Keil manual. Build the project and generate the HEX file.

3.Use a USB ISP programmer along with ProgISP software to download the Hex file into your 8051 based 89S52 Microcontroller.

4.The Microcontroller and external ADC0809 should be connected using connector wires as seen in above pic.

Program

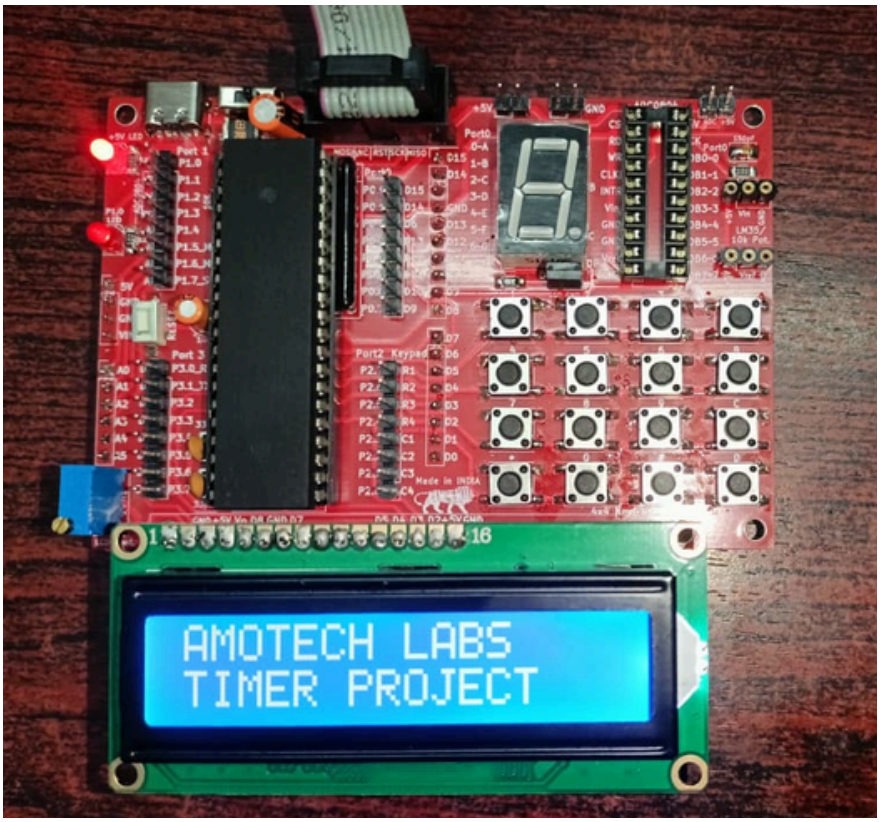
```
#include <reg51.h>

/* ================= LCD CONNECTION ================= */
#define lcdport P3
sbit rs = P3^2;
sbit en = P3^3;

/* ================= ADC0809 CONNECTION ============= */
#define adc_data P0
sbit ALE   = P2^0;
sbit START = P2^1;
sbit EOC   = P2^2;
sbit OE    = P2^3;
sbit ADDA  = P2^4;
sbit ADDB  = P2^5;
sbit ADDC  = P2^6;

/* ================= DELAY ========================== */
void delay(unsigned int t) {
    unsigned int i, j;
    for(i=0; i<t; i++)
        for(j=0; j<125; j++);
}

/* ================= LCD FUNCTIONS ================== */
void lcd_pulse() {
    en = 1;
    delay(1);
    en = 0;
    delay(1);
}

void lcdcmd(unsigned char cmd) {
    rs = 0;
    lcdport = (lcdport & 0x0F) | (cmd & 0xF0);
    lcd_pulse();
    lcdport = (lcdport & 0x0F) | ((cmd << 4) & 0xF0);
    lcd_pulse();
}

void lcddata(unsigned char dat) {
    rs = 1;
    lcdport = (lcdport & 0x0F) | (dat & 0xF0);
    lcd_pulse();
    lcdport = (lcdport & 0x0F) | ((dat << 4) & 0xF0);
    lcd_pulse();
}

void lcdstring(char *s) {
    while(*s)
        lcddata(*s++);
}

void lcd_print_voltage(unsigned int v) {
    lcddata((v / 100) + '0');
    lcddata('.');
    lcddata(((v / 10) % 10) + '0');
    lcddata((v % 10) + '0');
}
```

```
void lcd_init() {
    delay(20);
    lcdcmd(0x02);
    lcdcmd(0x28);
    lcdcmd(0x0C);
    lcdcmd(0x06);
    lcdcmd(0x01);
    delay(10);
}
/* ================= ADC READ (IN0) ================= */
unsigned char read_adc_IN0(void) {
    unsigned char value;

    /*
       ADC0809 CHANNEL SELECTION
       -------------------------
       Channel selection is done using address lines:

       ADDC  ADDB  ADDA    Selected Channel
        0     0     0      IN0
        0     0     1      IN1
        0     1     0      IN2
        0     1     1      IN3
        1     0     0      IN4
        1     0     1      IN5
        1     1     0      IN6
        1     1     1      IN7

       Since potentiometer is connected to IN0,
       we set ADDA = 0, ADDB = 0, ADDC = 0
    */

    ADDA = 0;    // Least Significant Address Bit
    ADDB = 0;    // Middle Address Bit
    ADDC = 0;    // Most Significant Address Bit

    /* Latch address and start conversion */
    ALE = 1;
    START = 1;
    delay(1);
    ALE = 0;
    START = 0;

    /* Wait until End Of Conversion goes HIGH */
    while(EOC == 0);

    /* Enable ADC output and read data */
    OE = 1;
    delay(1);
    value = adc_data;
    OE = 0;

    return value;
}

/* ================= MAIN ========================== */
void main() {
    unsigned char adc_val;
    unsigned int voltage;

    adc_data = 0xFF;      // Port 1 as input
    EOC = 1;

    lcd_init();
    lcdstring("ADC0809 IN0");
    delay(1000);
```

```
    lcdcmd(0x01);

    while(1) {
        adc_val = read_adc_IN0();

        // Voltage = (ADC * 5.00V * 100) / 255
        voltage = (unsigned int)(((unsigned long)adc_val * 500) / 255);

        lcdcmd(0x80);
        lcdstring("ADC RAW: ");
        lcddata((adc_val / 100) + '0');
        lcddata(((adc_val / 10) % 10) + '0');
        lcddata((adc_val % 10) + '0');

        lcdcmd(0xC0);
        lcdstring("VOLTS: ");
        lcd_print_voltage(voltage);
        lcdstring(" V");

        delay(300);
    }
}
```

Lab 7. Timer

Aim: To study the Timer operation of the AT89S52 microcontroller by configuring and using its internal timer to generate accurate time delays.



Procedure for User-Defined Timer (AT89S52)

- Study the user-defined timer program and understand the timer configuration and delay calculations given in the comments.
- Open Keil µVision IDE and create a new project or open the existing project for the timer program.
- Select the appropriate device (AT89S52) and add the source file to the project.
- Configure the required timer mode (Timer 0 / Timer 1) in the program and load the calculated initial count values. Build the project and generate the HEX file. Use a USB ISP programmer along with ProgISP software to download the Hex file into your 8051 based 89S52 Microcontroller.
- Power ON the kit and observe the timer operation (LED blinking / LCD display / output pin toggling).
- Change the user-defined delay value in the program and repeat the experiment to verify timer accuracy.

Program

```
#include <reg51.h>

/* ================ LCD (DAY-4 STYLE) ================ */
#define lcdport P3
sbit rs = P3^2;
sbit en = P3^3;

/* ================ KEYPAD ================ */
sbit col1 = P2^3;
sbit col2 = P2^2;
sbit col3 = P2^1;
sbit col4 = P2^0;

sbit row1 = P2^7;
sbit row2 = P2^6;
sbit row3 = P2^5;
sbit row4 = P2^4;

/* ================ FUNCTION PROTOTYPES ================ */
void delay(int);
void lcdcmd(unsigned char);
void lcddata(unsigned char);
void lcdstring(char *);
void lcd_init();
char getkey();
unsigned int read_number();
void timer_1ms(char);

/* ================ SOFTWARE DELAY ================ */
void delay(int d)
{
int i,j;
for(i=0;i<d;i++)
for(j=0;j<1275;j++);
}

/* ================ LCD ENABLE ================ */
void daten()
{
rs = 1;
en = 1;
delay(5);
en = 0;
}

void cmden()
{
rs = 0;
en = 1;
delay(5);
en = 0;
}

/* ================ LCD DATA ================ */
void lcddata(unsigned char ch)
{
lcdport = ch & 0xF0;          // P3.4-P3.7
daten();
lcdport = (ch << 4) & 0xF0;
daten();
}
```

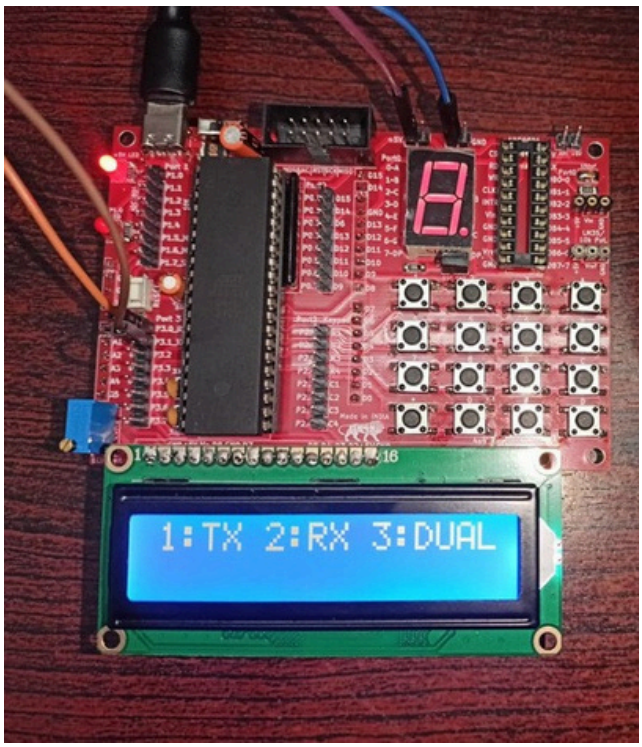
```
/* ================ LCD COMMAND ================ */
void lcdcmd(unsigned char ch)
{
lcdport = ch & 0xF0;
cmden();
lcdport = (ch << 4) & 0xF0;
cmden();
}

/* ================ LCD STRING ================ */
void lcdstring(char *s)
{
while(*s)
lcddata(*s++);
}

/* ================ LCD INIT ================ */
void lcd_init()
{
delay(20);
lcdcmd(0x02);   // 4-bit
lcdcmd(0x28);   // 2 line
lcdcmd(0x0C);   // Display ON
lcdcmd(0x01);   // Clear
}

/* ================ 1ms HARDWARE TIMER ================ */
void timer_1ms(char t)
{
if(t == 'A')            // TIMER 0
{
/* ================ 1ms HARDWARE TIMER ================ */
void timer_1ms(char t)
{
if(t == 'A')            // TIMER 0
{
TMOD &= 0xF0;
TMOD |= 0x01;
TH0 = 0xFC;
TL0 = 0x67;
TR0 = 1;
while(!TF0);
TR0 = 0;
TF0 = 0;
}
else                    // TIMER 1
{
TMOD &= 0x0F;
TMOD |= 0x10;
TH1 = 0xFC;
TL1 = 0x67;
TR1 = 1;
while(!TF1);
TR1 = 0;
TF1 = 0;
}
}

/* ================ READ NUMBER ( * = BACKSPACE ) ================ */
unsigned int read_number()
{
unsigned int n = 0;
char k;
```

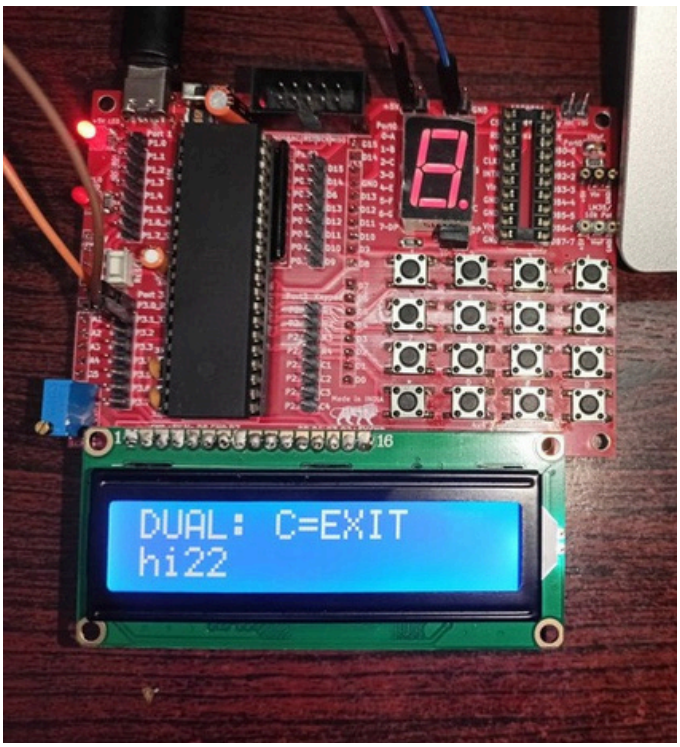
```
while(1)
{
k = getkey();

/* DIGITS */
if(k >= '0' && k <= '9')
{
if(n < 60000)                    // overflow safety
{
n = n * 10 + (k - '0');
lcddata(k);
}
}

/* BACKSPACE */
else if(k == '*')
{
if(n > 0)
{
n = n / 10;

lcdcmd(0x10);                // cursor left
lcddata(' ');                // erase
lcdcmd(0x10);                // cursor left again
}
}

/* ENTER */
else if(k == '#')
{
break;
}
}
return n;
}

/* ================ KEYPAD ================ */
char getkey()
{
while(1)
{
col1=0; col2=col3=col4=1;
if(!row1){ while(!row1); return '1'; }
if(!row2){ while(!row2); return '4'; }
if(!row3){ while(!row3); return '7'; }
if(!row4){ while(!row4); return '*'; }

col2=0; col1=col3=col4=1;
if(!row1){ while(!row1); return '2'; }
if(!row2){ while(!row2); return '5'; }
if(!row3){ while(!row3); return '8'; }
if(!row4){ while(!row4); return '0'; }

col3=0; col1=col2=col4=1;
if(!row1){ while(!row1); return '3'; }
if(!row2){ while(!row2); return '6'; }
if(!row3){ while(!row3); return '9'; }
if(!row4){ while(!row4); return '#'; }

col4=0; col1=col2=col3=1;
if(!row1){ while(!row1); return 'A'; }
if(!row2){ while(!row2); return 'B'; }
}
}
```

```
/* ================= MAIN ================= */
void main()
{
char timer_sel;
unsigned int delay_val;

P0 = 0x00;
lcd_init();

lcdstring("AMOTECH LABS");
lcdcmd(0xC0);
lcdstring("TIMER PROJECT");
delay(1000);

while(1)
{
lcdcmd(0x01);
lcdstring("A=T0  B=T1");
lcdcmd(0xC0);
lcdstring("Select Timer");

timer_sel = getkey();
if(timer_sel != 'A' && timer_sel != 'B')
continue;

lcdcmd(0x01);
lcdstring("Delay (ms):");
lcdcmd(0xC0);

delay_val = read_number();

lcdcmd(0x01);
lcdstring("Running...");
P0 = 0xFF;

while(delay_val--)
timer_1ms(timer_sel);

P0 = 0x00;
lcdcmd(0x01);
lcdstring("Done!");
delay(500);
}
}
```

Lab 8: Serial Communication

Aim: To study and implement USB to TTL serial communication between a PC and the AT89S52 (8051) microcontroller for bidirectional data transfer.



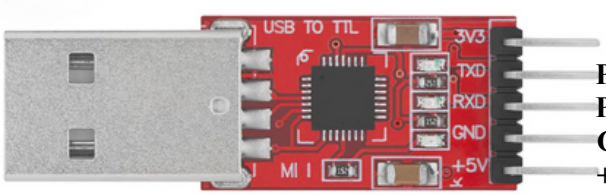


Note: Same code can be executed on '8051 Mini-development board.'

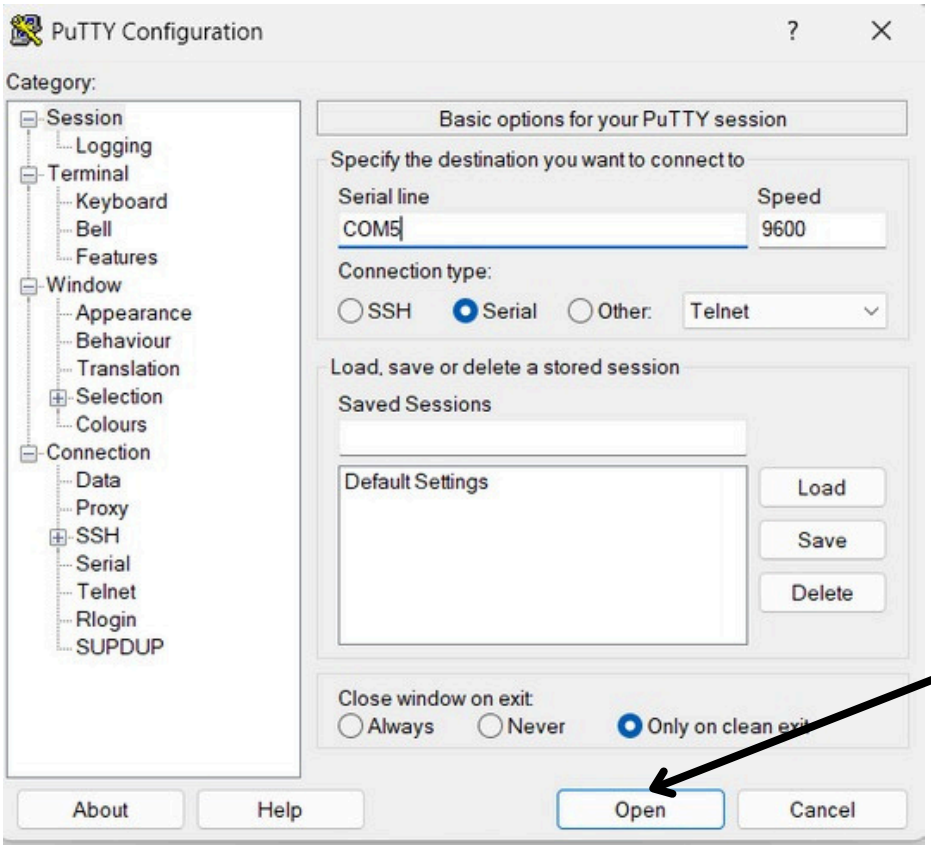
Procedure

- Study the USB to TTL serial communication program and understand UART initialization, baud rate setting, and data transmission/reception explained in the comments.
- Open Keil µVision IDE and create a new project or open the existing serial communication project.
- Select the AT89S52 microcontroller and add the source file to the project.
- Build the project and generate the HEX file without errors.
- Use a USB ISP programmer along with ProgISP software to download the Hex file into your 8051 based 89S52 Microcontroller.
- Connect the USB to TTL converter to the microcontroller UART pins (TXD and RXD) and connect the USB end to the PC.
- Open PuTTY serial terminal software (download link: https://apps.microsoft.com/detail/XPFNZKSKLBP7RJ?hl=en-IN&gl=IN&ocid=pdpshare) and configure the COM port and baud rate.
- Send data from the PC and observe the received data on LCD or terminal; verify data sent from the microcontroller is received correctly on the PC.

USB to TTL Modules for Serial communication

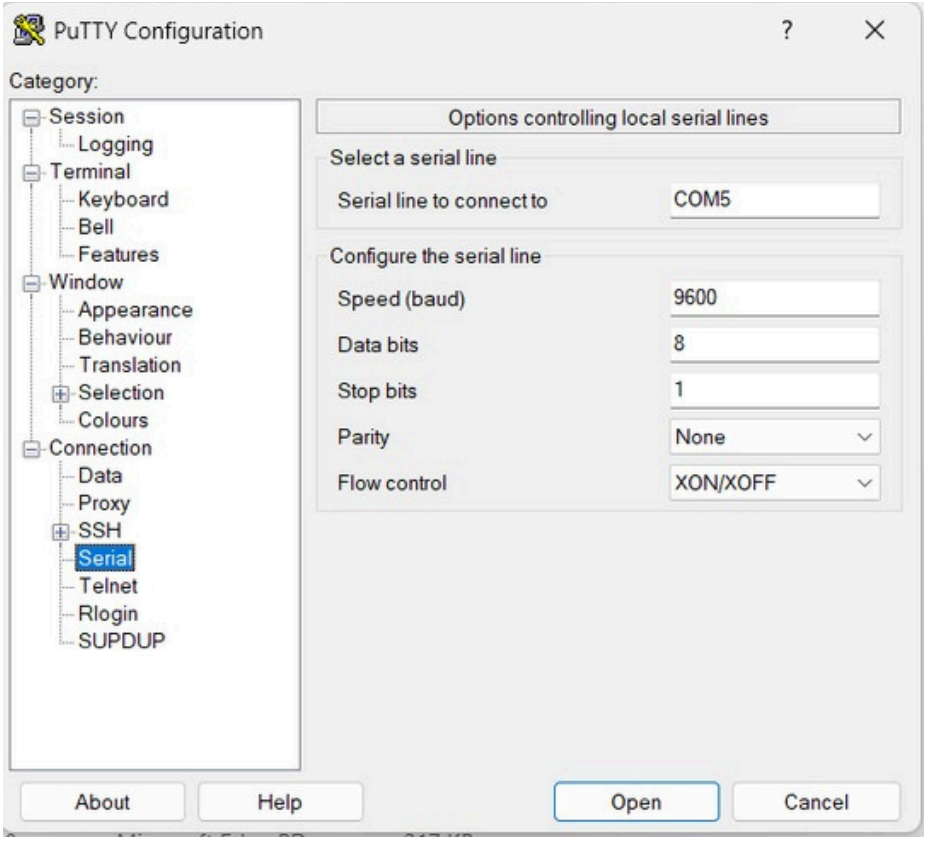


PuTTY configuration



Click Here

PuTTY configuration for serial communication



PuTTY Serial monitor to receive data transmitted by the 8051

Program

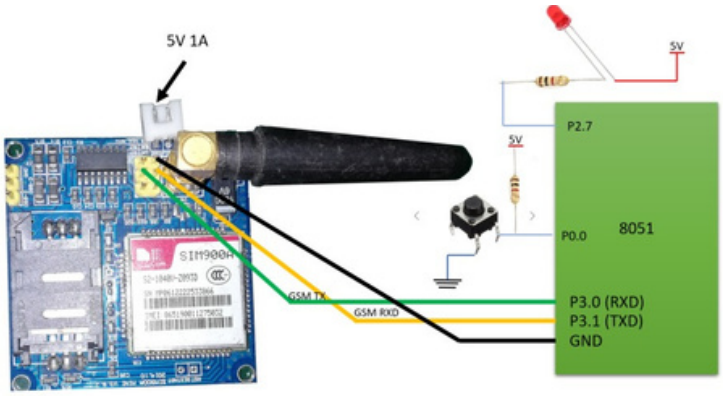
```
#include <reg51.h>

/* ================ HARDWARE PINS ================ */
#define lcdport P3
sbit rs = P3^2;
sbit en = P3^3;

sbit row1 = P2^7; sbit row2 = P2^6; sbit row3 = P2^5; sbit row4 = P2^4;
sbit col1 = P2^3; sbit col2 = P2^2; sbit col3 = P2^1; sbit col4 = P2^0;

/* ================ FUNCTION PROTOTYPES ================ */
void delay_ms(unsigned int);
void lcdcmd(unsigned char);
void lcddata(unsigned char);
void lcd_init(void);
void lcdstring(char *);
void uart_init(void);
void uart_tx(unsigned char);
unsigned char keypad_scan(void);

/* ================ DELAY ================ */
void delay_ms(unsigned int ms) {
    unsigned int i, j;
    for(i=0; i<ms; i++)
        for(j=0; j<120; j++);
}

/* ================ LCD LOGIC (4-BIT) ================ */
void pulse_en() { en=1; delay_ms(1); en=0; delay_ms(1); }

void lcdcmd(unsigned char ch) {
    rs=0;
    lcdport = (lcdport & 0x0F) | (ch & 0xF0); pulse_en();
}

void lcddata(unsigned char ch) {
    rs=1;
    lcdport = (lcdport & 0x0F) | (ch & 0xF0); pulse_en();
    lcdport = (lcdport & 0x0F) | (ch << 4); pulse_en();
}

void lcd_init() {
    delay_ms(20);
    lcdcmd(0x02); // Return Home
    lcdcmd(0x28); // 4-bit, 2-line
    lcdcmd(0x0C); // Display ON
    lcdcmd(0x01); // Clear
}

void lcdstring(char *s) { while(*s) lcddata(*s++); }

/* ================ UART LOGIC ================ */
void uart_init() {
    TMOD = 0x20; // Timer 1, Mode 2
    TH1 = 0xFD;  // 9600 Baud @ 11.0592MHz
    SCON = 0x50; // 8-bit, REN=1
    TR1 = 1;
}

void uart_tx(unsigned char ch) {
    SBUF = ch;
    while(TI == 0);
    TI = 0;
}
```

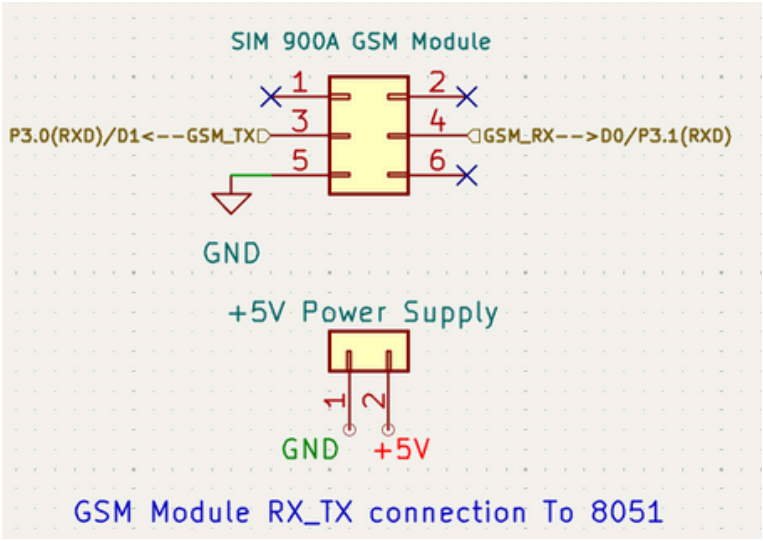
```
/* ================ KEYPAD WITH EXIT STABILITY ================ */
unsigned char keypad_scan() {
    unsigned char key = 0;

    // Scan Columns
    col1=0; col2=1; col3=1; col4=1;
    if(!row1) key = '1'; if(!row2) key = '4'; if(!row3) key = '7';

    col1=1; col2=0; col3=1; col4=1;
    if(!row1) key = '2'; if(!row2) key = '5'; if(!row3) key = '8';

    col1=1; col2=1; col3=0; col4=1;
    if(!row1) key = '3'; if(!row2) key = '6'; if(!row3) key = '9';

    col1=1; col2=1; col3=1; col4=0;
    if(!row3) key = '#';

    if(key != 0) {
        delay_ms(200);// Debounce
        while(!row1 || !row2 || !row3 || !row4); // WAIT FOR RELEASE (Prevents multiple sends)
        delay_ms(100); // Buffer delay
    }
    return key;
}

/* ================ MAIN PROGRAM ================ */
void main() {
    unsigned char ch;
    lcd_init();
    uart_init();
lcdstring("AMOTECH LABS");
lcdcmd(0xC0);
lcdstring("   PUNE");
delay_ms(1000);
lcdcmd(1);
lcdstring("Serial comm");
lcdcmd(0xC0);
lcdstring("using 8051");
  delay_ms(1000);
    while(1) {
        lcdcmd(0x01); // Clear Screen
        lcdstring("1:TX 2:RX 3:DUAL");

        // Wait specifically for a valid menu press
        do {
            ch = keypad_scan();
        } while(ch != '1' && ch != '2' && ch != '3');

        if(ch == '1') { // SEND MODE
            lcdcmd(0x01);
            lcdstring("TX: C=EXIT");
            lcdcmd(0xC0);
            while(1) {
                ch = keypad_scan();
                if(ch == '#') break; // Exit logic
                if(ch) {
                    lcddata(ch);
                    uart_tx(ch);
                    uart_tx(' '); // Space separator for Serial Monitor
                }
            }
        }
        else if(ch == '2') { // RECEIVE MODE
            lcdcmd(0x01);
            lcdstring("RX: C=EXIT");
            lcdcmd(0xC0);
```

```
            while(1) {
                if(RI) {
                    ch = SBUF; RI = 0;
                    if(ch > 31) { // Filter out junk/newlines
                        lcddata(ch);
                    }
                }
                // Check if user pressed # on keypad to exit
                if(keypad_scan() == '#') break;
            }
        }
        else if(ch == '3') { // BOTH MODE
            lcdcmd(0x01);
            lcdstring("DUAL: C=EXIT");
            lcdcmd(0xC0);
            while(1) {
                if(RI) {
                    ch = SBUF; RI = 0;
                    if(ch > 31) lcddata(ch);
                }
                ch = keypad_scan();
                if(ch == '#') break;
                if(ch) {
                    uart_tx(ch);
                    lcddata(ch);
                }
            }
        }
    }
}
```

Lab 9: GSM Module (SIM900A) Interfacing

Aim: To study and implement GSM-based wireless communication using the AT89S52 (8051) microcontroller to receive SMS commands and control external devices, enabling remote monitoring and control.





Procedure:

- Study the GSM interfacing program to understand UART initialization, baud rate setting (9600 bps), and AT command operation.
- Create or open the project in Keil µVision, select the AT89S52 microcontroller, add the GSM source file, build the project, and generate the HEX file.
- Program the AT89S52 using a USB ISP programmer and ProgISP software.
- Connect the GSM module, LCD (4-bit mode), PIR sensor, and LED to the 8051 microcontroller as per the given pin configuration.
- Power the GSM module and initialize it in TEXT mode using AT commands, then continuously monitor UART data.
- Receive SMS commands (ON/OFF) to control the LED and send alert SMS when motion is detected, while displaying the status on the LCD.

Program in c to send SMS from the GSM module.

```
#include <reg51.h>
sbit PIR = P2^0;   // PIR sensor input
void delay(int itime)
{
    int i, j;
    for(i = 0; i < itime; i++)
        for(j = 0; j < 1275; j++);
}

void Serialbegin()
{
    TMOD = 0x20;   // Timer1, Mode2 (8-bit auto reload)
    SCON = 0x50;   // Serial mode 1, 8-bit UART
    TH1  = 0xFD;   // 9600 baud rate @11.0592MHz
    TR1  = 1;      // Start Timer1
}

void Serialwrite(char dat)
{
    SBUF = dat;
    while(!TI);
    TI = 0;
}

void Serialprintln(char *p)
{
    while(*p)
    {
        Serialwrite(*p);
        p++;
    }
    Serialwrite(0x0D);  // Carriage return
}
void main()
{
    Serialbegin();
    Serialprintln("ATE0");
    delay(500);

    while(1)
    {
        if(PIR == 0)    // Motion detected
        {
            delay(1000);

            Serialprintln("AT+CMGF=1");
            delay(200);

            Serialprintln("AT+CMGS=\"XXXXXXXXXX\""); //Enter Your mobile number in place of X
            delay(200);

            Serialprintln("Someone is Enter in your Place.");
            delay(100);

            Serialwrite(26);   // CTRL+Z to send SMS
            delay(3000);
        }
    }
}
```

Program in C to receive SMS from the GSM module

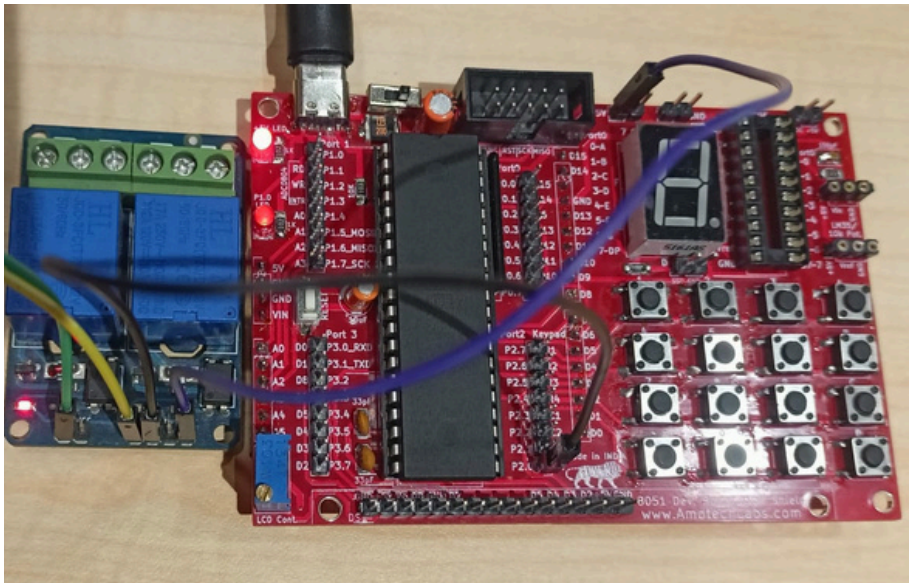
```
#include<reg51.h>
sbit D7=P2^4;  //Ard D2
sbit D6=P2^5;  //Ard D3
sbit D5=P2^6;  //Ard D4
sbit D4=P2^7;  //Ard D5
sbit rs=P3^6; /* Register select pin */  //Ard D8
sbit en=P3^7; /* Enable pin */           //Ard D7
sbit PIR=P2^0;
sbit LED = P1^1; //LED pin
int LCD_Port ;
int k=0;
int m=0;
unsigned char str[75];
unsigned char msg[25];

/* Function to provide delay Approx 1ms with 11.0592 Mhz crystal*/
void delay(int itime)
{
 int i,j;
 for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);
}
void LCD_Command (char cmnd) /* LCD16x2 command funtion */
{
LCD_Port = (cmnd & 0xF0)>>4;/* Send upper nibble */
D7=LCD_Port&0X08;
D6=LCD_Port&0X04;
D5=LCD_Port&0X02;
D4=LCD_Port&0X01;
rs=0; /* Command reg. */
// rw=0; /* Write operation */
en=1;
delay(1);
en=0;
delay(2);
LCD_Port = (cmnd & 0x0F);/* Send lower nibble */
D7=LCD_Port&0X08;
D6=LCD_Port&0X04;
D5=LCD_Port&0X02;
D4=LCD_Port&0X01;
rs=0;
en=1; /* Enable pulse */
delay(1);
en=0;
delay(5);
}
void LCD_Char (char char_data) /* LCD data write function */
{
LCD_Port =(char_data & 0xF0)>>4;/* Send upper nibble */
D7=LCD_Port&0X08;
D6=LCD_Port&0X04;
D5=LCD_Port&0X02;
D4=LCD_Port&0X01;
rs=1; /*Data reg.*/
// rw=0; /*Write operation*/
en=1;
delay(1);
en=0;
delay(2);
LCD_Port = (char_data & 0x0F);/* Send lower nibble */
D7=LCD_Port&0X08;
D6=LCD_Port&0X04;
D5=LCD_Port&0X02;
D4=LCD_Port&0X01;
en=1; /* Enable pulse */
delay(1);
```

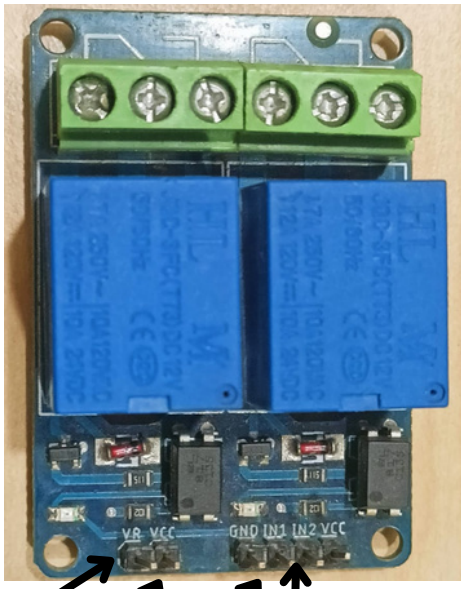
```
en=0;
delay(5);
}
void LCD_String (char *str) /* Send string to LCD function */
{
int i;
for(i=0;str[i]!=0;i++) /* Send each char of string till the NULL */
{
LCD_Char (str[i]); /* Call LCD data write */
}
}
void LCD_String_xy (char row, char pos, char *str) /* Send string to LCD function */
{
if (row == 0)
LCD_Command((pos & 0x0F)|0x80);
else if (row == 1)
LCD_Command((pos & 0x0F)|0xC0);
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 Serialbegin()
{
 TMOD=0x20; // use Timer 1, 8 bit mode
 SCON=0x50; // Serial mode 1, 8-bit data, 1 stop bit, 1 start bit

 TH1=0xfd; // 9600 Baud Rate
 TR1=1; //start the timer alloted for serial comm.
}
void Serialwrite(char dat)
{
SBUF=dat;
while(!TI);
TI=0;
}
void Serialprintln(char *p)
{
while(*p)
      {
      Serialwrite(*p);
      p++;
      }
      Serialwrite(0x0d);
      }
      void GSM_read()     // Function to read the response from GSM
      {
      while(RI==0);   // Wait until the byte received
      str[k]=SBUF;    //storing byte in str array
      RI=0;           //clear RI to receive next byte
      }
      void main()
      {
k=0;
       LCD_Init(); /* Initialization of LCD*/
LCD_String("  AMOTECH LABS"); /* write string on 1st line of LCD*/
       LCD_Command(0xC0); /* Go to 2nd line*/
Serialbegin();
Serialprintln("ATE0");
       delay(50);
```

```
delay(500);
Serialprintln("AT+CMGF=1"); // Configuring TEXT mode
    delay(500);
    Serialprintln("AT+CNMI=1,2,0,0,0"); // Decides how newly arrived SMS messages should be handled
    delay(500);
    while(1)
    {
    GSM_read();
  if(str[k-1]=='O' && str[k]=='N')
  {
  LED = 1;
    LCD_Char(str[k-1]);
    LCD_Char(str[k]);
  delay(200);
  k=0;
  LCD_Command(0xC0); /* Go to 2nd line*/
  }
  if(str[k-1]=='O' && str[k]=='F')
  {
  LED = 0;
    LCD_Char(str[k-1]);
   LCD_Char(str[k]);
  delay(200);
  k=0;
     LCD_Command(0xC0); /* Go to 2nd line*/
  }
    k=k+1;
  }
  }
```

Lab 10: Relay Interfacing

Aim: To interface a relay with the 8051 microcontroller and control its ON and OFF operation.





Relay Coil voltage
+12v/+5v
(from external power supply)

+5v GND Input from P2.0
(From 8051 Development Board)

Procedure:

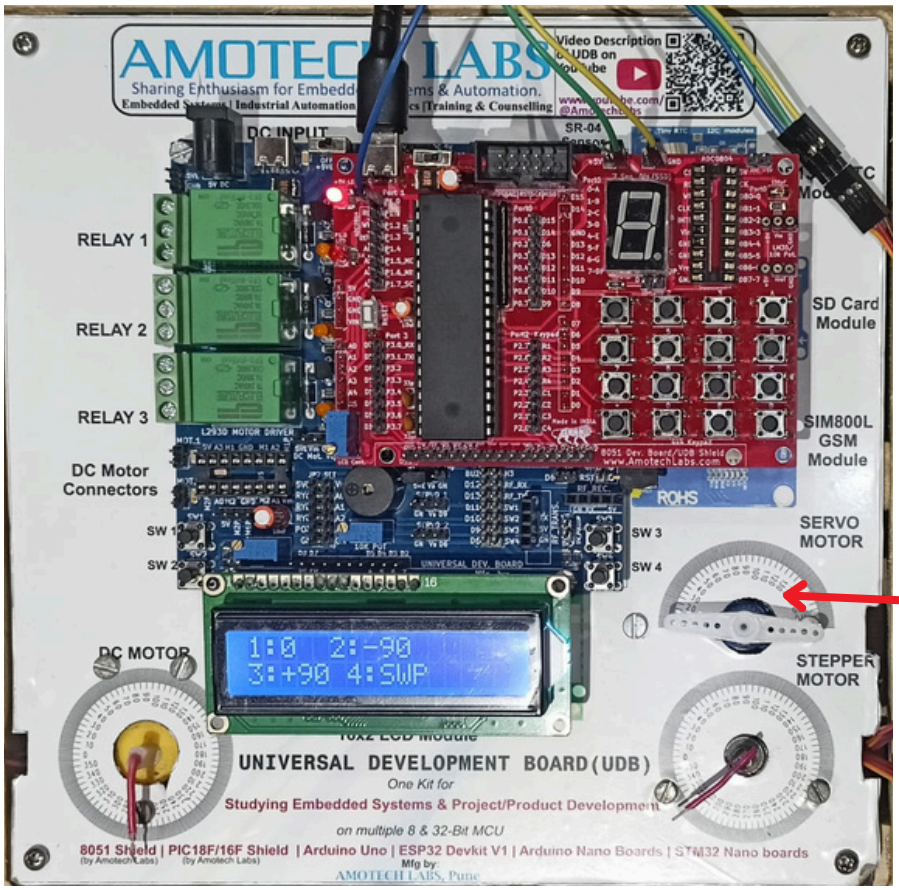
- Study the relay interfacing program to understand digital output control of the relay using Port P2.0 and software delay generation.
- Create or open the project in Keil µVision, select the AT89S52 microcontroller, add the relay interfacing source file, build the project, and generate the HEX file.
- Program the AT89S52 microcontroller using a USB ISP programmer and ProgISP software.
- Connect a 12 V active-LOW relay module to Port P2.0 of the 8051 through a proper relay driver circuit; alternatively, a 5 V relay module with optocoupler isolation can also be used.
- Power ON the system and observe the relay turning ON when logic LOW is applied and OFF when logic HIGH is applied at regular time intervals.
- Verify relay operation by connecting an external load and confirming correct switching behavior.

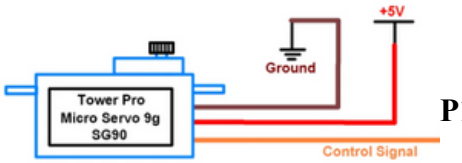
Program

```
#include <reg52.h>

sbit RELAY = P2^0;   // Relay connected to P2.0

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

void main()
{P0=0x00;
    while(1)
    {
        RELAY = 1;   // Relay OFF
        delay(1000);

        RELAY = 0;   // Relay ON
        delay(1000);
    }
}
```

Lab 11: Servo Motor Control Using PWM

Aim: To control the position of a servo motor using PWM generated by the AT89S52 (8051) microcontroller and operate it through a keypad interface, with status and menu display on a 16×2 LCD.





Servo Motor used inside ***TOWER PRO SG90***

Procedure

- Study the principle of PWM generation using Timer-0 interrupt of the AT89S52 microcontroller for servo motor control.
- Create a new project in Keil µVision, select the AT89S52, write the embedded C program, and generate the HEX file.
- Program the AT89S52 microcontroller using a USB ISP programmer and ProgISP software.
- Interface the servo motor to Port P1.0, 4×4 keypad to Port-2, and 16×2 LCD in 4-bit mode to Port-3 as per the circuit diagram.
- Power ON the system using an external 5 V supply for the servo motor and observe the LCD displaying the project title and menu options.
- Press the keypad keys to control the servo motor at 0°, -90°, +90°, and sweep mode, and verify smooth and stable operation.

Program

```
#include <reg52.h>

/* ================= SERVO PWM ================= */
#define PWM_Period 0xB7FE        // 20 ms @ 11.0592 MHz
sbit Servo = P1^0;

unsigned int ON_Period, OFF_Period;

/* ===== SWEEP CONTROL VARIABLES ===== */
bit sweep_mode = 0;              // sweep enable flag
signed int sweep_pos = 1290;     // start at center (0°)
signed char sweep_dir = 1;       // +1 or -1 direction

/* ================= LCD ================= */
#define lcdport P3
sbit rs = P3^2;
sbit en = P3^3;

/* ================= KEYPAD (PORT 2) ===== */
sbit col1 = P2^3;
sbit col2 = P2^2;
sbit col3 = P2^1;
sbit col4 = P2^0;

sbit row1 = P2^7;
sbit row2 = P2^6;
sbit row3 = P2^5;
sbit row4 = P2^4;

/* ================= BASIC DELAY ================= */
void delay(unsigned int ms)
{
    unsigned int i,j;
    for(i=0;i<ms;i++)
        for(j=0;j<1275;j++);
}

/* ================= LCD DRIVER (STABLE) ================= */
void lcd_delay_us(unsigned int us)
{
    unsigned int i;
    while(us--)
        for(i=0;i<12;i++);
}

void lcd_en_pulse()
{
    en = 1;
    lcd_delay_us(10);
    en = 0;
}

void lcd_cmd(unsigned char cmd)
{
    EA = 0;
    rs = 0;

    lcdport = (lcdport & 0x0F) | (cmd & 0xF0);
    lcd_en_pulse();
```

```
    lcdport = (lcdport & 0x0F) | ((cmd<<4) & 0xF0);
    lcd_en_pulse();

    EA = 1;
    lcd_delay_us(50);
}

void lcd_data(unsigned char dat)
{
    EA = 0;
    rs = 1;

    lcdport = (lcdport & 0x0F) | (dat & 0xF0);
    lcd_en_pulse();

    lcdport = (lcdport & 0x0F) | ((dat<<4) & 0xF0);
    lcd_en_pulse();

    EA = 1;
    lcd_delay_us(50);
}

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

void lcd_init()
{
    delay(20);
    lcd_cmd(0x02);
    lcd_cmd(0x28);
    lcd_cmd(0x0C);
    lcd_cmd(0x06);
    lcd_cmd(0x01);
    delay(2);
}

/* ================ KEYPAD READ ================ */
char keypad_read()
{
    P2 = 0xF0;

    col1=0; col2=col3=col4=1;
    if(!row1){ while(!row1); return '1'; }

    col2=0; col1=col3=col4=1;
    if(!row1){ while(!row1); return '2'; }

    col3=0; col1=col2=col4=1;
    if(!row1){ while(!row1); return '3'; }

    col4=0; col1=col2=col3=1;
    if(!row1){ while(!row1); return '4'; }

    return 0;
}
```

```
/* ================= TIMER-0 PWM ================= */
/*
   Timer-0 is configured in Mode-1 (16-bit timer).
   It is used to generate a 20 ms PWM period required by a servo motor.

   Crystal frequency = 11.0592 MHz
   Machine cycle = 12 / 11.0592 MHz ~1.085 µs

   PWM_Period = 0xB7FE = 47102
   Timer counts = 65535 - 47102 = 18433 counts
   Time = 18433 × 1.085 µs ~20 ms (50 Hz)
*/

void Timer_init()
{
    TMOD = 0x01;              // Timer0 Mode-1 (16-bit)
    TH0  = PWM_Period >> 8;   // Load high byte of 20 ms value
    TL0  = PWM_Period;        // Load low byte of 20 ms value
    TF0  = 0;                 // Clear Timer0 overflow flag
    TR0  = 1;                 // Start Timer0
}

/*
   Timer0 ISR:
   This ISR generates PWM by TOGGLING the servo pin.

   Logic:
   - Each interrupt toggles Servo pin state
   - If Servo pin becomes HIGH  ON time starts
   - If Servo pin becomes LOW   OFF time starts
   - Timer reload value decides duration of ON or OFF time
*/
void Timer0_ISR(void) interrupt 1
{
    TF0 = 0;                  // Clear interrupt flag explicitly

    Servo = !Servo;           // Toggle servo output pin

    if(Servo)
    {
        // Servo pin HIGH  load ON-time duration
        TH0 = ON_Period >> 8;
        TL0 = ON_Period;
    }
    else
    {
        // Servo pin LOW  load OFF-time duration
        TH0 = OFF_Period >> 8;
        TL0 = OFF_Period;
    }
}

/* ================ SERVO POSITIONS ================ */

/*
   Servo pulse width requirements (20 ms period):

   -90°  ~0.54 ms   497 timer counts
    0°   ~1.40 ms   1290 timer counts
   +90°  ~2.40 ms   2212 timer counts

   These values are PRE-CALCULATED to avoid floating-point math
   (important for 8051 stability).
*/
```

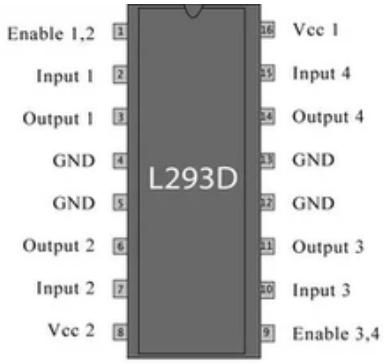
```
void Set_Servo_Position(char key)
{
    unsigned int period = 65535 - PWM_Period;  // Total PWM period counts
    unsigned int pos;                          // ON-time counts

    if(key=='3')          pos = 497;    // -90° position
    else if(key=='1')     pos = 1290;   // 0° (center)
    else if(key=='2')     pos = 2212;   // +90° position
    else return;                        // Ignore invalid keys

    sweep_pos = pos;      // Synchronize sweep start position

    /*
       ON_Period  = time servo pin remains HIGH
       OFF_Period = remaining time servo pin remains LOW

       OFF_Period = Total period - ON_Period
    */
    ON_Period  = pos;
    OFF_Period = period - ON_Period;

    /*
       Timer reload values:
       Timer counts UP from loaded value to 65535.
       Therefore actual reload = 65535 - desired counts.
    */
    ON_Period  = 65535 - ON_Period;
    OFF_Period = 65535 - OFF_Period;
}

/* ================= NON-BLOCKING SWEEP ================= */
/*
   Sweep is implemented as a STATE MACHINE, not a blocking loop.

   Advantages:
   - Keypad remains responsive
   - No CPU lock-up
   - Smooth and stable servo motion
*/

void Servo_Sweep_Step()
{
    unsigned int period = 65535 - PWM_Period;

    /*
       Update PWM ON/OFF based on current sweep position
    */
    ON_Period  = sweep_pos;
    OFF_Period = period - ON_Period;

    ON_Period  = 65535 - ON_Period;
    OFF_Period = 65535 - OFF_Period;

    /*
       Move servo gradually:
       sweep_dir = +1  move right
       sweep_dir = -1  move left
       Step size = 20 counts  controls sweep speed
    */
    sweep_pos += (sweep_dir * 20);

    /*
       Limit checking:
    */
    if(sweep_pos >= 2212)
    {
        sweep_pos = 2212;      // Upper limit (+90°)
        sweep_dir = -1;        // Reverse direction
    }
    else if(sweep_pos <= 497)
    {
        sweep_pos = 497;       // Lower limit (-90°)
        sweep_dir = 1;         // Reverse direction
    }

    delay(5);   // Small delay to control sweep speed (not blocking PWM)
}
```

```
/* ================= MAIN ================= */
void main()
{
    char key;

    P1 = 0x00;
    P2 = 0xF0;
    P3 = 0x00;

    lcd_init();

    lcd_string("AMOTECH LABS");
    lcd_cmd(0xC0);
    lcd_string("PUNE");
    delay(500);

    lcd_cmd(0x01);
    lcd_string("SERVO CONTROL");
    lcd_cmd(0xC0);
    lcd_string("USING PWM");
    delay(500);
     lcd_cmd(0x01);
     lcd_string("1:0  2:-90");
     lcd_cmd(0xC0);
     lcd_string("3:+90 4:SWP");

     Set_Servo_Position('1');   // start at center

     EA  = 1;
     ET0 = 1;
     Timer_init();

     while(1)
     {
         key = keypad_read();

         if(key)
         {
             sweep_mode = 0;

             if(key=='1' || key=='2' || key=='3')
                 Set_Servo_Position(key);
             else if(key=='4')
                 sweep_mode = 1;
         }

         if(sweep_mode)
             Servo_Sweep_Step();
     }
}
```

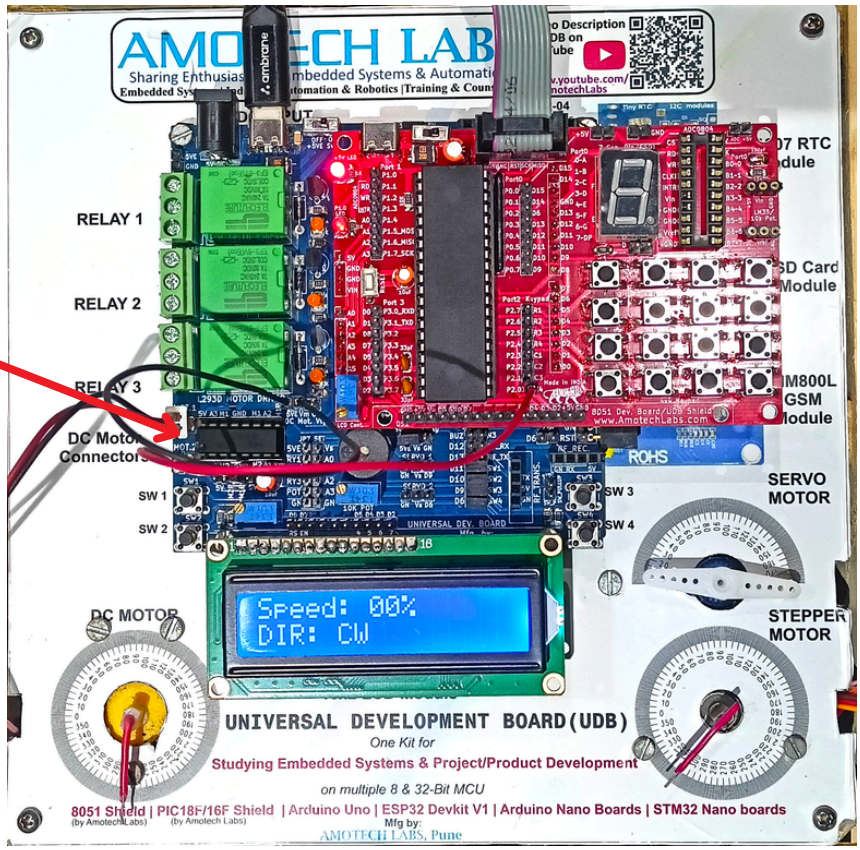
Lab 12: DC Motor Control Using PWM

Aim: To control the speed and direction of a DC motor using PWM generated by the AT89S52 microcontroller and display the motor status on a 16×2 LCD.

L293D Motor Driver



BO DC Motor
(5V)



Procedure:

- Study the principle of DC motor operation and PWM generation using Timer-0 interrupt of the AT89S52 microcontroller for speed control.
- Create a new project in Keil µVision, select the AT89S52, write the embedded C program for DC motor speed and direction control using PWM, and generate the HEX file.
- Program the AT89S52 microcontroller using a USB ISP programmer and ProgISP software.
- Interface the DC motor to the microcontroller through a motor driver circuit, connect the PWM enable pin to Port P2.0, motor direction control pins to Port P1.6 and P1.7, push-button switches to Port-0, and the 16×2 LCD in 4-bit mode to Port-3 as per the circuit diagram.
- Power ON the system using an external DC supply for the motor and verify that the LCD displays the motor speed (in %) and direction (CW/CCW).
- Press the increase and decrease switches to vary the motor speed in steps using PWM, press the direction switch to change the direction of rotation, and press the stop switch to stop the motor.
- Observe and verify smooth speed variation, proper direction change, and stable operation of the DC motor for all switch operations.

Program

```
#include <reg52.h>

/* ================ PWM ================ */
#define PWM_Period 0xFC18       // ~1 kHz PWM @ 11.0592 MHz
#define MIN_SPEED  40           // minimum duty for motor start

sbit PWM_EN = P2^0;             // ENABLE pin of motor driver

/* ================ SWITCHES (ACTIVE LOW) ================ */
sbit SW_INC  = P0^5;
sbit SW_DEC  = P0^6;
sbit SW_DIR  = P0^7;
sbit SW_STOP = P0^2;

/* ================ MOTOR DIRECTION ================ */
sbit M1 = P1^6;
sbit M2 = P1^7;

/* ================ LCD (DAY-3 VERIFIED CONNECTION) ================ */
sbit D7 = P3^7;
sbit D6 = P3^6;
sbit D5 = P3^5;
sbit D4 = P3^4;
sbit rs = P3^2;
sbit en = P3^3;

/* ================ VARIABLES ================ */
unsigned int ONP, OFFP;
unsigned char Speed = 0;
bit Direction = 0;
bit pwm_enable = 0;

/* ================ DELAY ================ */
void delay(unsigned int d)
{
    unsigned int i,j;
    for(i=0;i<d;i++)
        for(j=0;j<112;j++);
}

/* ================ LCD COMMAND ================ */
void LCD_Command(char cmnd)
{
    D7 = (cmnd & 0x80); D6 = (cmnd & 0x40);
    D5 = (cmnd & 0x20); D4 = (cmnd & 0x10);
    rs = 0; en = 1; delay(1); en = 0;

    D7 = (cmnd & 0x08); D6 = (cmnd & 0x04);
    D5 = (cmnd & 0x02); D4 = (cmnd & 0x01);
    rs = 0; en = 1; delay(1); en = 0;

    delay(2);
}

/* ================ LCD DATA ================ */
void LCD_Data(char dat)
{
    D7 = (dat & 0x80); D6 = (dat & 0x40);
    D5 = (dat & 0x20); D4 = (dat & 0x10);
    rs = 1; en = 1; delay(1); en = 0;

    D7 = (dat & 0x08); D6 = (dat & 0x04);
    D5 = (dat & 0x02); D4 = (dat & 0x01);
    rs = 1; en = 1; delay(1); en = 0;

    delay(2);
```

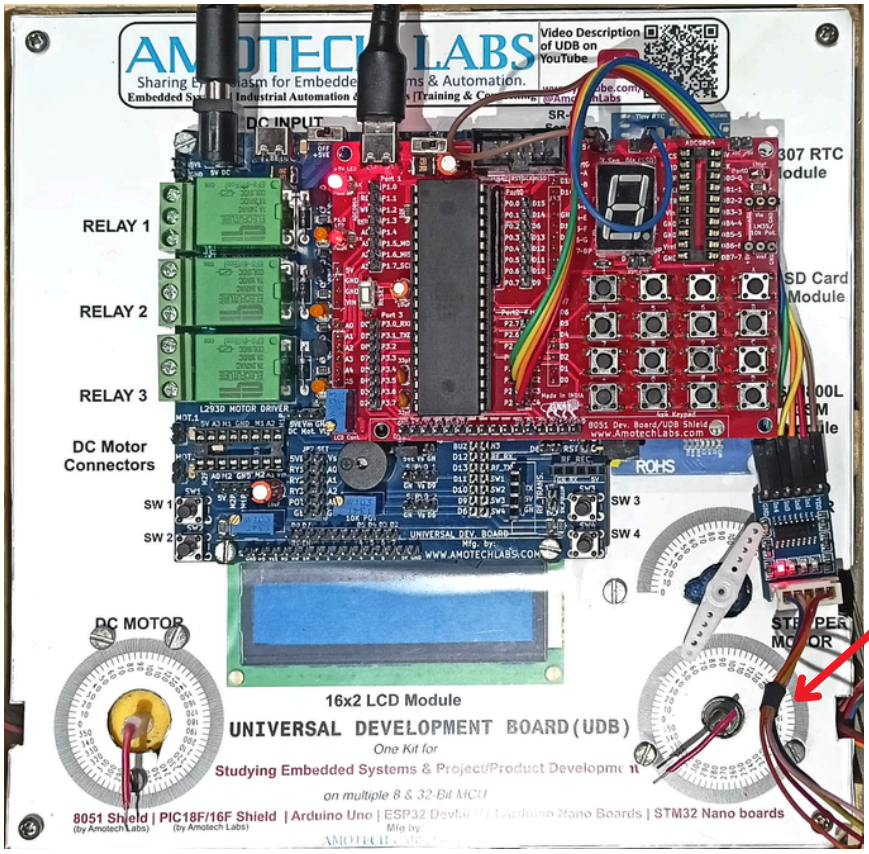
```
}

void LCD_String(char *s)
{
    while(*s) LCD_Data(*s++);
}

void LCD_Init()
{
    delay(20);
    LCD_Command(0x02);
    LCD_Command(0x28);
    LCD_Command(0x0C);
    LCD_Command(0x01);
}

/* ================= PWM LOGIC ================= */
void set_duty(unsigned char d)
{
    unsigned int p = 65535 - PWM_Period;
    ONP  = 65535 - (p * d) / 100;
    OFFP = 65535 - (p - (p * d) / 100);
}

void timer0_isr(void) interrupt 1
{
    if(!pwm_enable)
    {
        PWM_EN = 0;
        TH0 = PWM_Period >> 8;
        TL0 = PWM_Period;
        return;
    }
    PWM_EN = !PWM_EN;

    if(PWM_EN)
    {
        TH0 = ONP >> 8;
        TL0 = ONP;
    }
    else
    {
        TH0 = OFFP >> 8;
        TL0 = OFFP;
    }
}

/* ================= LCD UPDATE ================= */
void LCD_Update()
{
    EA = 0;   // protect LCD timing

    LCD_Command(0x80);
    LCD_String("Speed: ");
    LCD_Data((Speed/10) + '0');
    LCD_Data((Speed%10) + '0');
    LCD_String("%   ");

    LCD_Command(0xC0);
    if(Direction)
        LCD_String("DIR: CCW ");
    else
        LCD_String("DIR: CW  ");

    EA = 1;
```

```
}

/* ================ MAIN ================ */
void main()
{
    P0 = 0xFF;          // pull-ups for switches

    /* SAFE START */
    PWM_EN = 0;
    pwm_enable = 0;
    Speed = 0;
    Direction = 0;

    M1 = 1; M2 = 0;     // default direction

    LCD_Init();
    LCD_Update();

    TMOD = 0x01;
    TH0  = PWM_Period >> 8;
    TL0  = PWM_Period;
    ET0  = 1;
    EA   = 1;
    TR0  = 1;

    while(1)
    {
        /* SPEED INCREASE */
        if(!SW_INC)
        {
            if(Speed < 100)
            {
                if(Speed == 0)
                 Speed = MIN_SPEED;
             else
                 Speed += 10;
        }

        if(Speed >= 100)
        {
            pwm_enable = 0;
            PWM_EN = 1;       // full speed
        }
        else
        {
            set_duty(Speed);
            pwm_enable = 1;
        }

        LCD_Update();
        while(!SW_INC);
        delay(150);
    }

    /* SPEED DECREASE */
    if(!SW_DEC)
    {
        if(Speed > MIN_SPEED)
            Speed -= 10;
        else
            Speed = 0;

        if(Speed == 0)
        {
            pwm_enable = 0;
            PWM_EN = 0;
```

```
            }
            else
            {
                set_duty(Speed);
                pwm_enable = 1;
            }

            LCD_Update();
            while(!SW_DEC);
            delay(150);
        }

        /* DIRECTION CHANGE */
        if(!SW_DIR)
        {
            Direction = !Direction;
            M1 = Direction;
            M2 = !Direction;

            LCD_Update();
            while(!SW_DIR);
            delay(150);
        }

        /* STOP */
        if(!SW_STOP)
        {
            Speed = 0;
            pwm_enable = 0;
            PWM_EN = 0;

            LCD_Update();
            while(!SW_STOP);
            delay(150);
        }
    }
}
```

Lab 13: Interfacing of Stepper Motor

Aim: To control the speed and direction of a stepper motor using the AT89S52 microcontroller by generating precise step sequences with Timer-0–based delays.



Stepper Motor used:
28BYJ-48 (5V DC)

Stepper Motor Driver used:
ULN2003

Procedure:

- Study the principle of stepper motor operation, including full-step and half-step excitation methods, and understand how precise delays control speed and direction using Timer-0 of the AT89S52 microcontroller.
- Create a new project in Keil µVision, select the AT89S52 device, write the embedded C program to generate stepper motor drive sequences (half-step and full-step) with Timer-0–based delay, and compile the program to generate the HEX file.
- Program the AT89S52 microcontroller with the generated HEX file using a USB ISP programmer and ProgISP software.
- Interface the stepper motor to the microcontroller through a ULN2003 driver circuit , connect the stepper motor phases to Port P2, and ensure proper power supply connections for both logic and motor.
- Power ON the system and verify that the stepper motor rotates clockwise using half-step sequence and anticlockwise using full-step sequence as defined in the program.
- Vary the delay value in the program to change the speed of rotation, observe accurate step movement, correct direction control, and stable operation of the stepper motor for both stepping modes.

Program

```
#include <reg52.h>

#define Stepper_Port P2      /* Stepper motor connected to Port 2 */

/* ================= TIMER-0 DELAY FUNCTION =================
   Crystal = 11.0592 MHz
   Machine cycle = 1.085 µs
   For 1 ms delay  921 counts
------------------------------------------------------------*/
void delay_ms(unsigned int ms)
{
    unsigned int i;
    for(i = 0; i < ms; i++)
    {
        TMOD &= 0xF0;          // Clear Timer0 bits
        TMOD |= 0x01;          // Timer0 Mode-1 (16-bit)

        TH0 = 0xFC;            // Load for 1 ms delay
        TL0 = 0x66;

        TR0 = 1;               // Start Timer0
        while(TF0 == 0);       // Wait for overflow
        TR0 = 0;               // Stop Timer0
        TF0 = 0;               // Clear overflow flag
    }
}

/* ================= MAIN PROGRAM ================= */
int main(void)
{
    int i;
    int period = 2;           //  Reduce value for higher speed (try 2-10)

    while(1)

     {
        /* ===== Clockwise Rotation (Half-Step) ===== */
        for(i = 0; i < 12; i++)
        {
            Stepper_Port = 0x09; delay_ms(period);
            Stepper_Port = 0x08; delay_ms(period);
            Stepper_Port = 0x0C; delay_ms(period);
            Stepper_Port = 0x04; delay_ms(period);
            Stepper_Port = 0x06; delay_ms(period);
            Stepper_Port = 0x02; delay_ms(period);
            Stepper_Port = 0x03; delay_ms(period);
            Stepper_Port = 0x01; delay_ms(period);
        }

        Stepper_Port = 0x09;
        delay_ms(500);

        /* ===== Anticlockwise Rotation (Full-Step) ===== */
        for(i = 0; i < 12; i++)
        {
            Stepper_Port = 0x09; delay_ms(period);
            Stepper_Port = 0x03; delay_ms(period);
            Stepper_Port = 0x06; delay_ms(period);
            Stepper_Port = 0x0C; delay_ms(period);
        }

        Stepper_Port = 0x09;
        delay_ms(500);
    }
}
```