CERTIFICATE

Certified that Mr. Rohit Mohanlal Yadav

Class M.Sc. Computer Science (II) (Sem-4) Roll No: 4729 has satisfactorily completed the required number of Practical / Tutorial assignments in the subject of Real Time & Embedded Systems in the department of Computer Science during the academic year 2024-2025.

Lecture Incharge	External Examiner	Head of Department

Date:

INDEX

Practical 1

Aim:

Interfacing LCD.

Theory:

This code is written for an 8051 microcontroller to control a 16x2 LCD display using the 4-bit interface method. The microcontroller interacts with the LCD to display the text "ROHIT" on the screen. The LCD is controlled through a combination of commands and data instructions sent to it by the microcontroller.

Here’s an explanation of the key components and functions:

Ports and Pins:

The **rs, rw**, and en pins are connected to the control pins of the LCD:

rs (Register Select) is used to switch between data and command modes.

rw (Read/Write) determines whether data is being written to or read from the LCD.

en (Enable) is a pulse that enables the LCD to read the data or command.

Port P2 (data lines D0 to D7) is used to send 8-bit data or commands to the LCD.

lcdcmd Function:

This function sends a command byte to the LCD. It sets the data bus (Port 2) to the desired command and uses the control pins (rs=0, rw=0, en=1, then en=0) to properly latch the data into the LCD.

lcddata Function:

This function sends a character to the LCD. It works similarly to lcdcmd but with the rs pin set to 1 to indicate data mode.

delay Function:

This function generates a simple delay to allow the LCD enough time to process the commands or data. It uses a for-loop to generate a delay.

Main Function:

Initializes the LCD display and continuously writes "ROHIT" on the screen. The commands used include:

0x38: Initialize the LCD in 8-bit mode with a 5x7 matrix font.

0x01: Clear the display screen.

0x10: Enable cursor blinking.

0x0c: Display the content on the screen (turn on the display).

0x81: Force the cursor to the first line, first position.

The characters 'R', 'O', 'H','I' and 'T' are written one by one.

Components:

1. AT89C51
2. LM016L (LED DISPLAY)

Steps and Procedure:

1. Software: Keil uVision5

1. Project > New Project > AT89C51

2. File name: Choose a proper file name and save the project in the desired folder. Make sure the file extension is .c.

3. Right-click on the Source Group, then add an existing file → select the file to be added.

4. Rebuild the project by pressing F7 or check for errors and solve them.

5. Right-click on Target, go to Target Options, set the Clock Frequency to 11.0592 MHz.

6. Select ROM Output and choose to create HEX file.

7. Rebuild the code again. The HEX file will be generated in the Objects folder. Minimize the window.

2. Software: Proteus 8.13 SP0 Pro

8. File → Project File: In the project file window, choose the appropriate options → create PCB/Schematics.

9. Create a Firmware Project for the 8051 AT89C51 microcontroller.

10. Complete the setup by selecting the appropriate options, and finish setting up the firmware project.

11. Select all required components from the toolbar (which is P in bluebox). Type the component names and select them. Click OK.

12. Select the required components from the list and click on the schematic screen to place them.

13. To add more ports, click on the appropriate component to add it to the schematic.

14. Hover your mouse over the pin and click to connect the components with wires, forming the circuit diagram.

15. For Power and Ground, select the terminal component for power (Vcc) and ground (GND), and connect them properly.

16. After a successful connection, double-click on the controller → select Program → navigate to your folder, select the HEX file, and click OK.

17. Set the Clock Frequency to 11.0592 MHz and click OK.

18. At the bottom (Downside), click Play to run the program.

19. If the output is not available, perform the previous steps again to ensure everything is set up correctly.

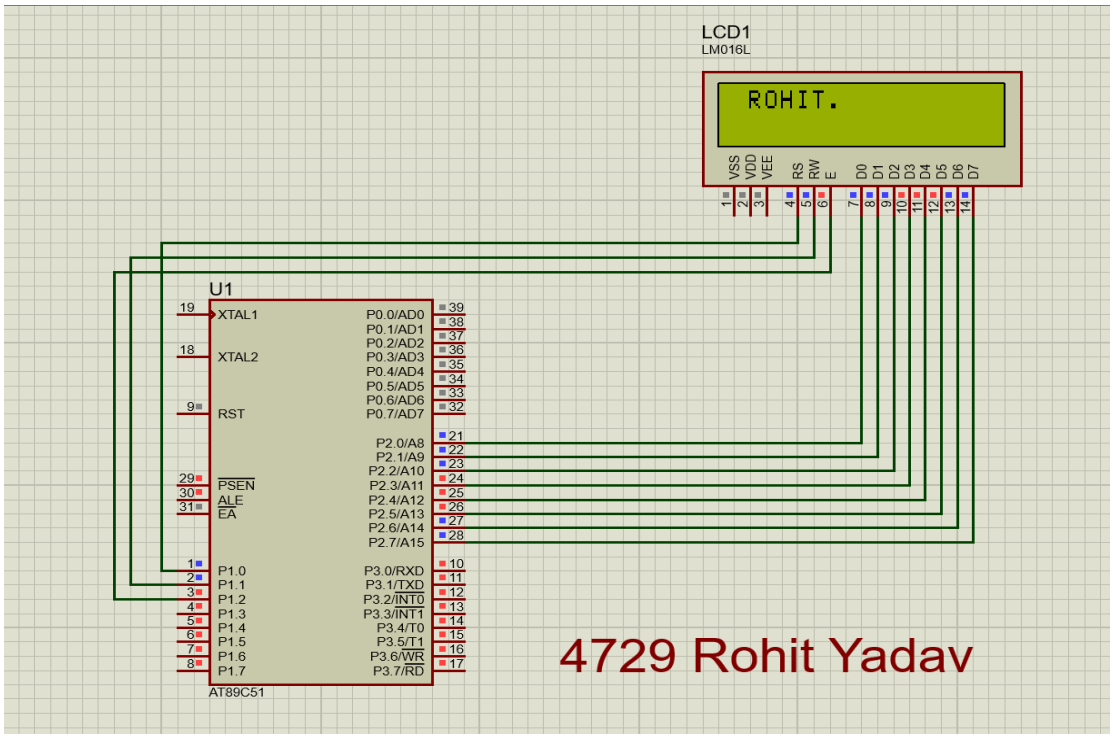
Code:

```
#include<reg51.h>
sbit rs= P1^0;
sbit rw= P1^1;
sbit en= P1^2;
void lcdcmd(unsigned char);
void lcddata(unsigned char);
void delay();
void main()
{
        P2=0x00; // output declaration, data lines d0-d7 connected with p2
        while(1) //finite loop
        {
        lcdcmd(0x38); //   5x7 matrix crystal
        delay();
        lcdcmd(0x01);         // clear screen
        delay();
        lcdcmd(0x10); // cursor blinking
        delay();
        lcdcmd(0x0c); // display on
        delay();
        lcdcmd(0x81); // force cursor 1st line 1st position
        delay();
        lcddata('N');
        delay();
```

```
        lcddata('A');
        delay();
        lcddata('M');
        delay();
        lcddata('E');
        delay();
        }
        }
void lcdcmd(unsigned char val)
{
P2=val;
rs=0;
rw=0;
en=1;
delay();
en=0;
}

void lcddata(unsigned char val)
{
P2= val;
rs= 1;
rw=0;
en= 1;
delay();
en=0;
}
void delay()
{
unsigned int i;
```

```
for(i=0; i<12000; i++);
}
```

Output:



Conclusion:

This program demonstrates how to interface an 8051 microcontroller with a 16x2 LCD using basic commands to display a fixed message ("ROHIT"). The program initializes the LCD, clears the screen, enables the cursor, and continuously displays the word "ROHIT" at the start of the first line of the screen.

The key concepts demonstrated here are:

Sending commands and data to the LCD.

Managing the control pins (rs, rw, en).

Using delays to ensure proper timing for the LCD to process the commands.

This program can be expanded to display different text, scroll text, or interact with external input devices.

Practical 2

Aim:

Traffic Control System

Theory:

This code is an implementation for a **traffic light controller** using an **8051 microcontroller.** The program controls two sets of traffic lights (each with red, yellow, and green LEDs) to simulate the behavior of a typical traffic light system.

The microcontroller is connected to the traffic light LEDs via Port 1 of the microcontroller, and each LED (Red, Yellow, and Green) is controlled by a specific pin. Here's a breakdown of the components and logic:

Traffic Light Pins:

r1, y1, g1: Control the red, yellow, and green LEDs for the first set of traffic lights (set 1).

r2, y2, g2: Control the red, yellow, and green LEDs for the second set of traffic lights (set 2).

Logic for Traffic Light Sequences:

The traffic light sequence is controlled in a finite loop (while(1)), where the lights for both sets of traffic lights change according to a predefined pattern.

For each state in the traffic light system, appropriate LEDs are turned on or off to simulate the red, yellow, and green phases.

Traffic Light Timing and Logic

The traffic lights operate in the following pattern:

Red for both lights (set 1 and set 2):

r1 = 1; r2 = 1; (both red lights are turned ON).

The system waits for a delay to simulate the red light duration.

Green for the second set (set 2):

g2 = 1; r2 = 0; (green for the second set, red for the first set).

The system waits for a delay to simulate the green light for set 2.

Yellow for the first set (set 1):

r1 = 0; y1 = 1; (yellow for set 1, red for set 2).

The system waits for a delay to simulate the yellow light duration for set 1.

Green for the first set (set 1):

g1 = 1; r1 = 0; (green for set 1, red for set 2).

The system waits for a delay to simulate the green light for set 1.

Red for the second set (set 2):

r2 = 1; g2 = 0; (red for set 2, green for set 1).

The system waits for a delay to simulate the red light for set 2.

Repeat the cycle:

The loop repeats, alternating between green and red phases for both sets of traffic lights.

Delay Mechanism

The delay is created using two **for-loops (for(i=0; i<60000; i++);**). These loops don't perform any specific task but simply count up to a certain number, creating a time delay. The purpose of the delay is to simulate how long each light stays on (i.e., how long each phase of the traffic light lasts).

Components:

1. AT89C51
2. TRAFFIC LIGHT (x2)

Steps and Procedure:

1. Software: Keil uVision5

1. Project > New Project > AT89C51

2. File name: Choose a proper file name and save the project in the desired folder. Make sure the file extension is .c.

3. Right-click on the Source Group, then add an existing file → select the file to be added.

4. Rebuild the project by pressing F7 or check for errors and solve them.

5. Right-click on Target, go to Target Options, set the Clock Frequency to 11.0592 MHz.

6. Select ROM Output and choose to create HEX file.

7. Rebuild the code again. The HEX file will be generated in the Objects folder. Minimize the window.

2. Software: Proteus 8.13 SP0 Pro

1. File → Project File: In the project file window, choose the appropriate options → create PCB/Schematics.

2. Create a Firmware Project for the 8051 AT89C51 microcontroller.

3. Complete the setup by selecting the appropriate options, and finish setting up the firmware project.

4. Select all required components from the toolbar (which is P in bluebox). Type the component names and select them. Click OK.

5. Select the required components from the list and click on the schematic screen to place them.

6. To add more ports, click on the appropriate component to add it to the schematic.

7. Hover your mouse over the pin and click to connect the components with wires, forming the circuit diagram.

8. For Power and Ground, select the terminal component for power (Vcc) and ground (GND), and connect them properly.

9. After a successful connection, double-click on the controller → select Program → navigate to your folder, select the HEX file, and click OK.

10. Set the Clock Frequency to 11.0592 MHz and click OK.

11. At the bottom (Downside), click Play to run the program.

12. If the output is not available, perform the previous steps again to ensure everything is set up correctly.

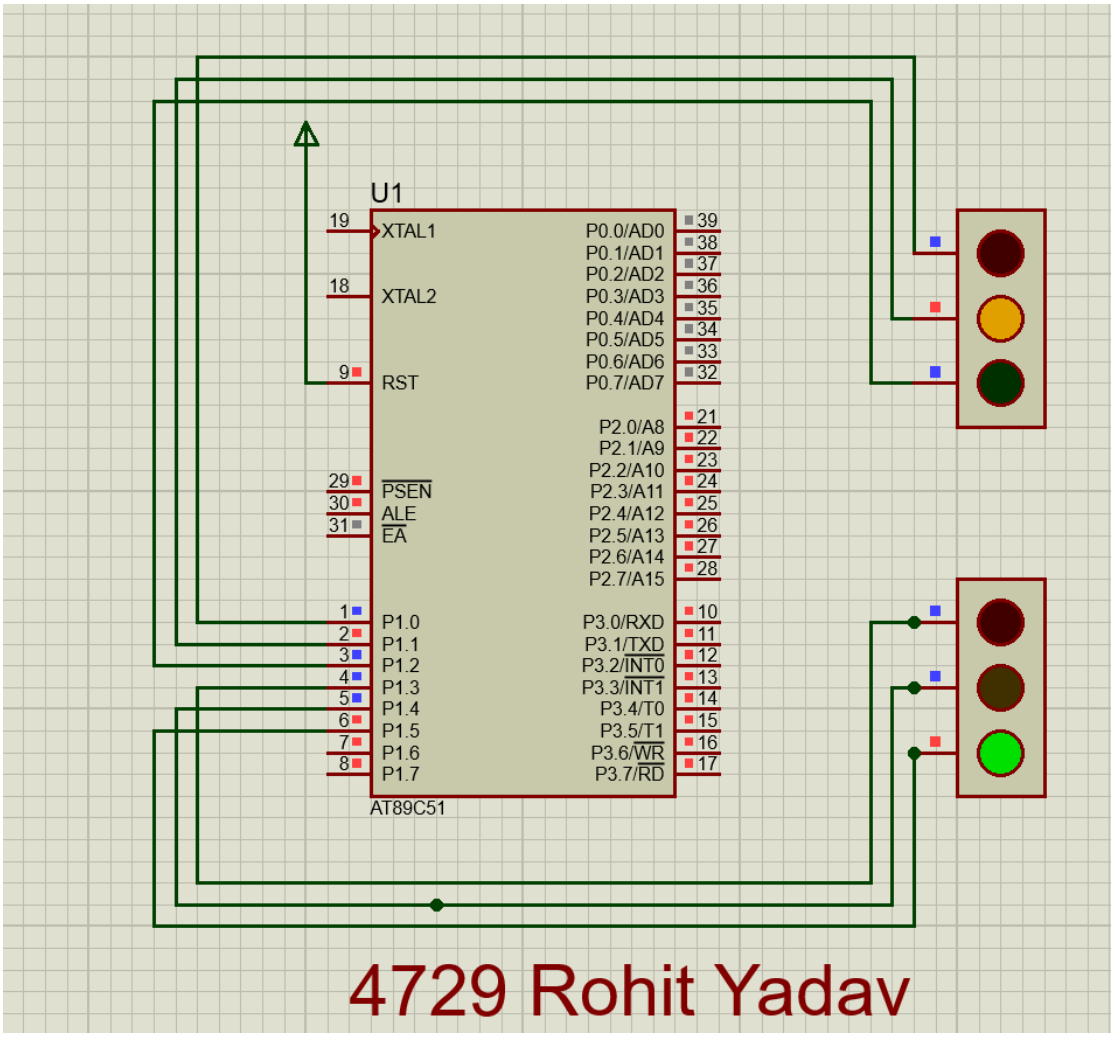
Code:

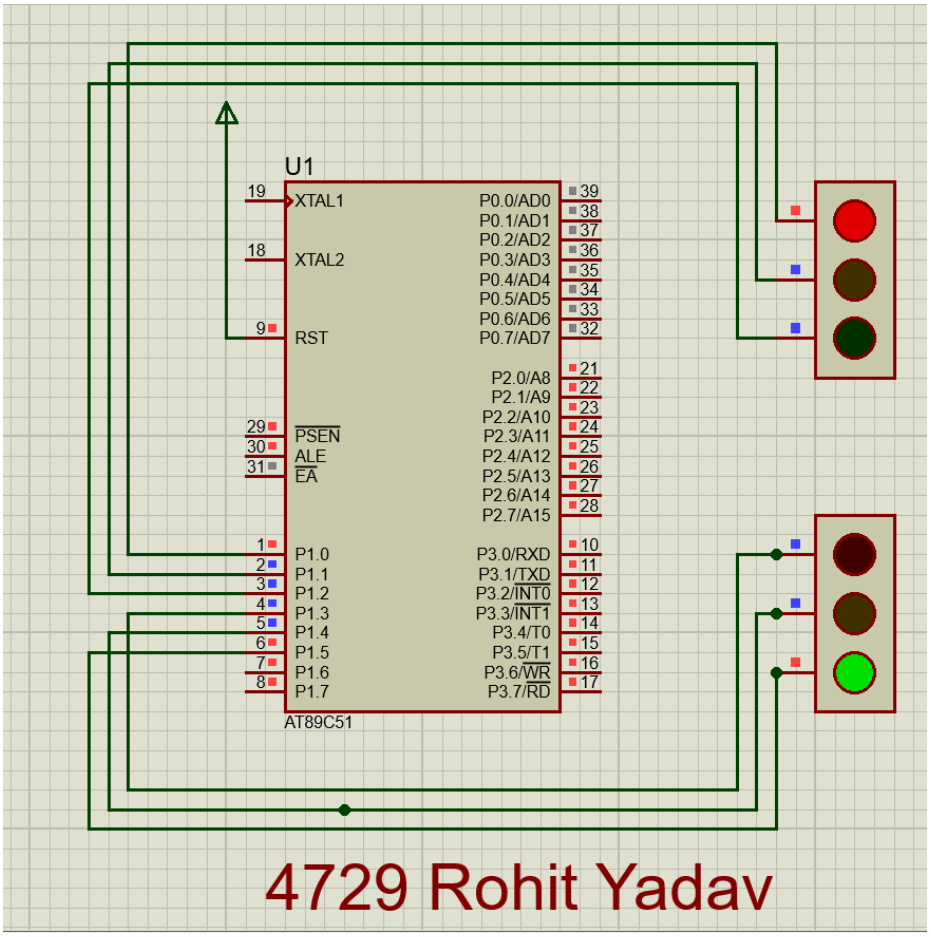
```
#include<reg51.h>

sbit r1=P1^0;

sbit y1=P1^1;

sbit g1=P1^2;


sbit r2=P1^3;

sbit y2=P1^4;
```

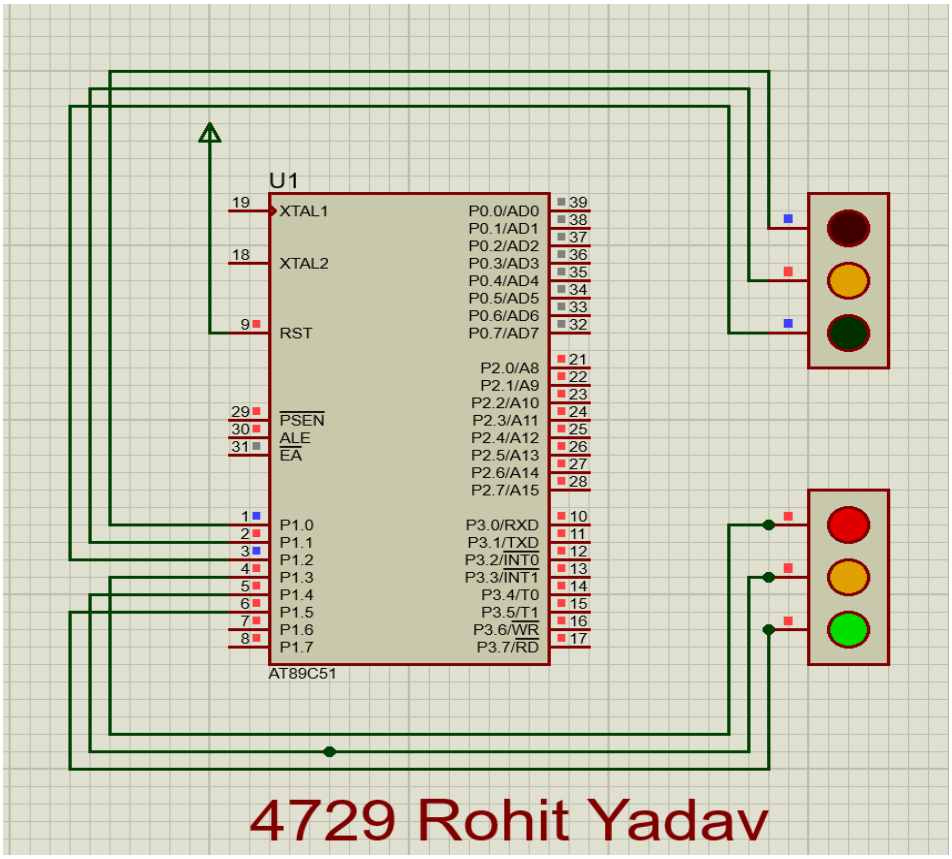
```
sbit g2=P1^5;
void main()
{
unsigned int i;
r1=y1=g1=0;
r2=y2=g2=0;
while(1)
{
r1=1;
r2=1;
y1=0;
g1=0;
y2=0;
g2=0;
for(i=0;i<60000;i++);
for(i=0;i<60000;i++);
g2=1;
r2=0;
for(i=0;i<60000;i++);
for(i=0;i<60000;i++);
r1=0;
y1=1;
for(i=0;i<60000;i++);
g1=1;
y1=0;
r1=0;
r2=1;
g2=0;
y2=0;
for(i=0;i<60000;i++);
```

```
r1=1;
r2=0;
y1=0;
y2=0;
g1=0;
g2=1;
}
}
```

Output:




U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
AT89C51
4729 Rohit Yadav


U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
AT89C51
4729 Rohit Yadav

Conclusion:

This program simulates the working of a traffic light system using the 8051 microcontroller. It controls two sets of traffic lights and cycles through various light sequences: red, yellow, and green. The microcontroller controls the state of each light using Port 1's pins, and the light changes are managed by a sequence of commands within the while(1) loop.

The sequence works in the following order:

Both lights show red.

Set 2 shows green, and Set 1 stays red.

Set 1 shows yellow while Set 2 stays green.

Set 1 shows green, and Set 2 stays red.

Set 2 shows red while Set 1 stays green.

This process repeats indefinitely. The use of simple delays and bit manipulation provides an effective solution for controlling the traffic lights. The program can be enhanced further by using timers or interrupts to handle more precise timing or by adding more traffic light sets.

Practical 3

Aim:

Interface matrix keyboard with microcontroller and display the key pressed on seven segment display.

Theory:

This program is for a 4x3 keypad interfaced with an 8051 microcontroller. It is used to detect which key is pressed and display the corresponding 7-segment display value. The program utilizes Port 1 for scanning the rows and columns of the keypad, and Port 2 is used to control the 7-segment display. The main function continuously scans the keypad and displays the corresponding 7-segment code on the display.

Here's an explanation of how the program works:

Keypad Configuration:

The keypad consists of 4 rows (R0, R1, R2, R3) and 3 columns (C0, C1, C2).

The microcontroller uses the scanning method to detect which key is pressed. It sets one row low (activating it), then checks each column to see if it is low, indicating the corresponding key is pressed.

Keypad Scanning Logic:

Set all rows (R0, R1, R2, R3) to high to initialize them to a neutral state.

Scan each row by setting one row to low at a time, and then check the columns.

If a column is low (C0, C1, or C2), this indicates a key has been pressed at that particular row and column. The corresponding value for the key is then sent to the 7-segment display using the seg() function.

Keypad to 7-Segment Mapping:

The program uses the following mappings for the 7-segment display values:

0xF9 corresponds to the number 1.

0xA4 corresponds to the number 4.

0xB0 corresponds to the number 7.

0x99 corresponds to the number 2.

0x92 corresponds to the number 5.

0x82 corresponds to the number 8.

0xB8 corresponds to the number 3.

0x80 corresponds to the number 6.

0x98 corresponds to the number 9.

0xC0 corresponds to the number 0.

These values are used to display the corresponding digit on the 7-segment display. Each value is a 7-segment display encoding in hexadecimal that lights up the appropriate segments to form the correct number.

Working of the Main Function:

The program repeatedly scans each of the rows (R0, R1, R2, R3), one at a time, by setting each row to low while keeping others high.

After setting a row low, the program checks the columns (C0, C1, C2) to see if any of them are low, which indicates that a key in that row and column is pressed.

Once a key is detected, the corresponding 7-segment code is sent to Port 2 using the seg() function.

seg() Function:

The seg() function receives a hexadecimal value (ch) representing the 7-segment display encoding for a number. It writes this value to Port 2 to update the 7-segment display.

Components:

1. AT89C51
2. KEYPAD PHONE
3. 7 Segment Display (7SEG-MPX1-CA)
4. VCC (POWER)

Steps and Procedure:

1. Software: Keil uVision5

1. Project > New Project > AT89C51

2. File name: Choose a proper file name and save the project in the desired folder. Make sure the file extension is .c.

3. Right-click on the Source Group, then add an existing file → select the file to be added.

4. Rebuild the project by pressing F7 or check for errors and solve them.

5. Right-click on Target, go to Target Options, set the Clock Frequency to 11.0592 MHz.

6. Select ROM Output and choose to create HEX file.

8. Rebuild the code again. The HEX file will be generated in the Objects folder. Minimize the window.

2. Software: Proteus 8.13 SP0 Pro

1. File → Project File: In the project file window, choose the appropriate options → create PCB/Schematics.

2. Create a Firmware Project for the 8051 AT89C51 microcontroller.

3. Complete the setup by selecting the appropriate options, and finish setting up the firmware project.

4. Select all required components from the toolbar (which is P in bluebox). Type the component names and select them. Click OK.

5. Select the required components from the list and click on the schematic screen to place them.

6. To add more ports, click on the appropriate component to add it to the schematic.

7. Hover your mouse over the pin and click to connect the components with wires, forming the circuit diagram.

8. For Power and Ground, select the terminal component for power (Vcc) and ground (GND), and connect them properly.

9. After a successful connection, double-click on the controller → select Program → navigate to your folder, select the HEX file, and click OK.

10. Set the Clock Frequency to 11.0592 MHz and click OK.

11. At the bottom (Downside), click Play to run the program.

12. If the output is not available, perform the previous steps again to ensure everything is set up correctly.

Code:

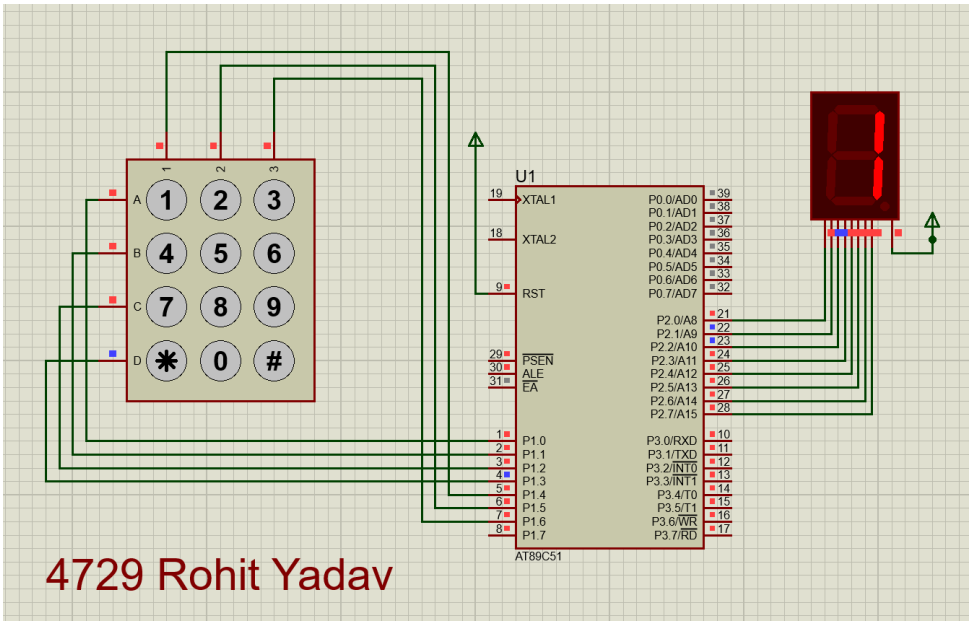
```
#include<reg51.h>
sbit R0 = P1^0;
sbit R1 = P1^1;
sbit R2 = P1^2;
sbit R3 = P1^3;
```

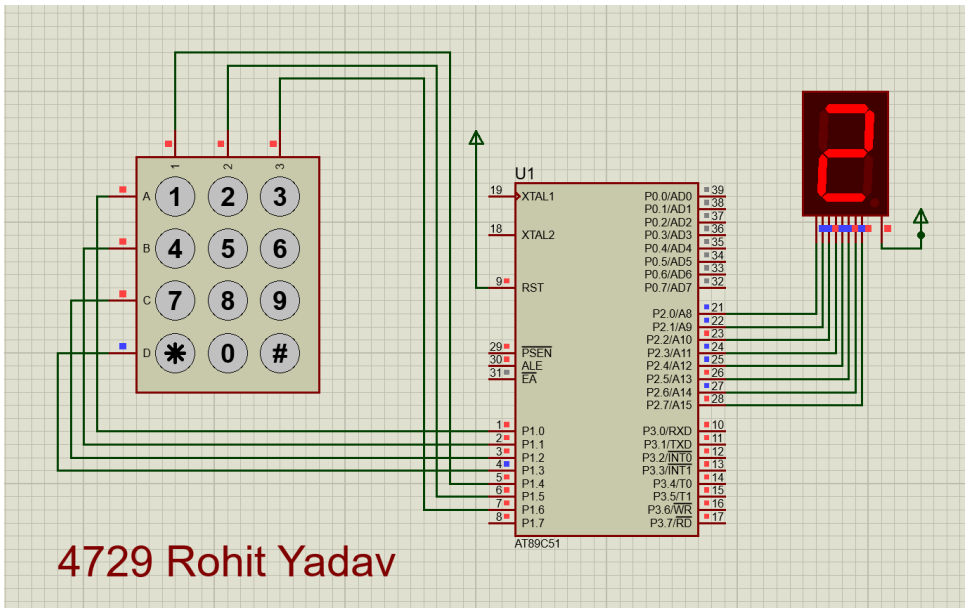
```
sbit C0 = P1^4;
sbit C1 = P1^5;
sbit C2 = P1^6;

void seg(unsigned int);
void main()
{
R0 = R1 = R2 = R3 = 1;
R0 = 0;
if(C0 == 0)
seg(0xF9);

R0 = R1 = R2 = R3 = 1;
R0 = 0;
if(C1 == 0)
seg(0xA4);

R0 = R1 = R2 = R3 = 1;
R0 = 0;
if(C2 == 0)
seg(0xB0);

R0 = R1 = R2 = R3 = 1;
R1 = 0;
if(C0 == 0)
seg(0x99);

R0 = R1 = R2 = R3 = 1;
R1 = 0;
if(C1 == 0)
```

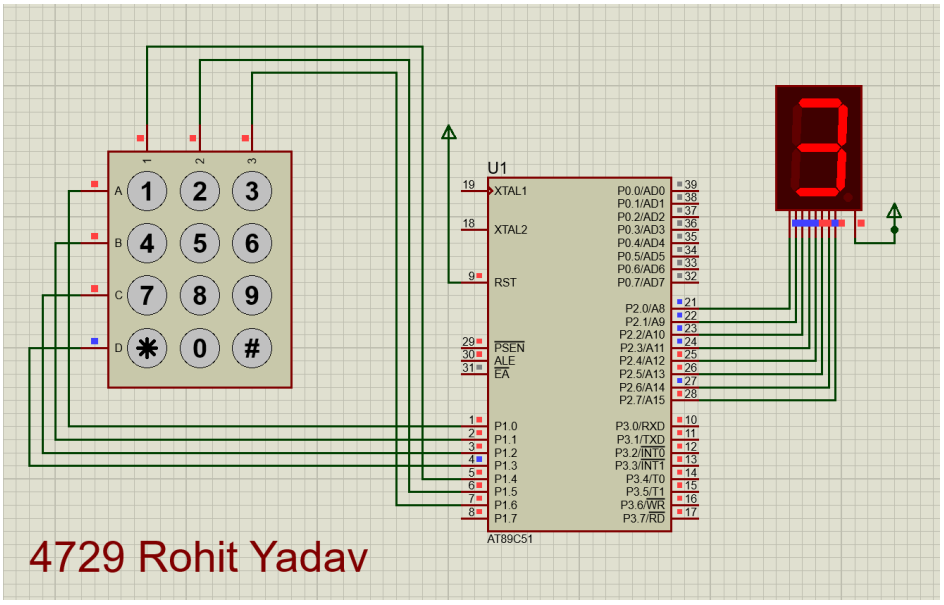
```
seg(0x92);

R0 = R1 = R2 = R3 = 1;
R1 = 0;
if(C2 == 0)
seg(0x82);

R0 = R1 = R2 = R3 = 1;
R2 = 0;
if(C0 == 0)
seg(0xB8);

R0 = R1 = R2 = R3 = 1;
R2 = 0;
if(C1 == 0)
seg(0x80);

R0 = R1 = R2 = R3 = 1;
R2 = 0;
if(C2 == 0)
seg(0x98);

R0 = R1 = R2 = R3 = 1;
R3 = 0;
if(C1 == 0)
seg(0xC0);
}

void seg(unsigned int ch)
{
```

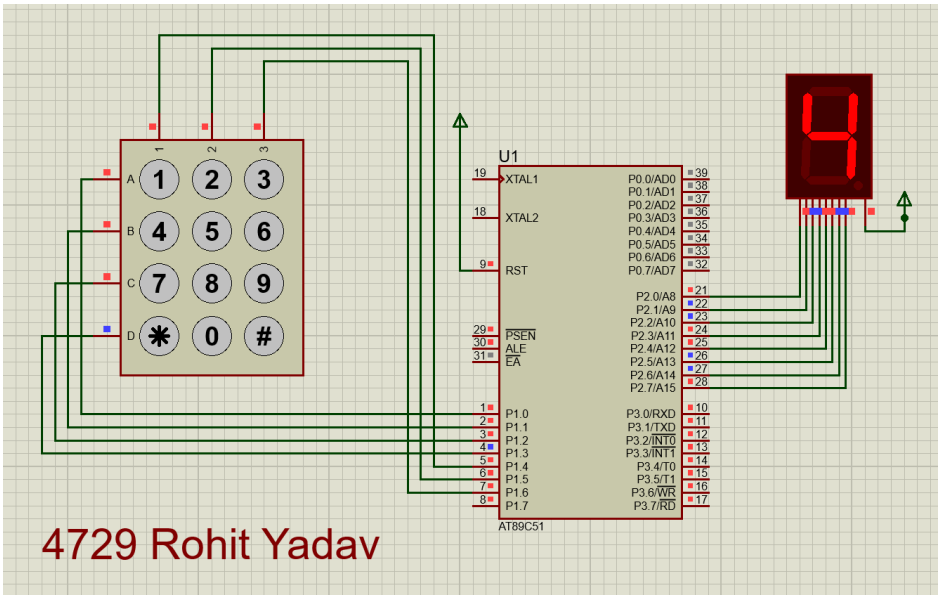
```
P2 = 0x00;
P2 = ch;
}
```

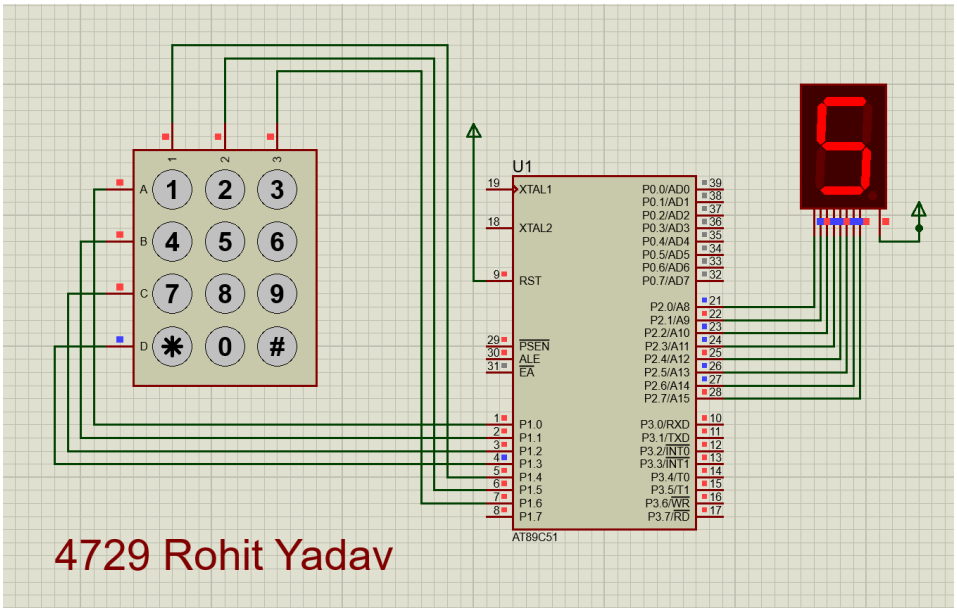
Output:

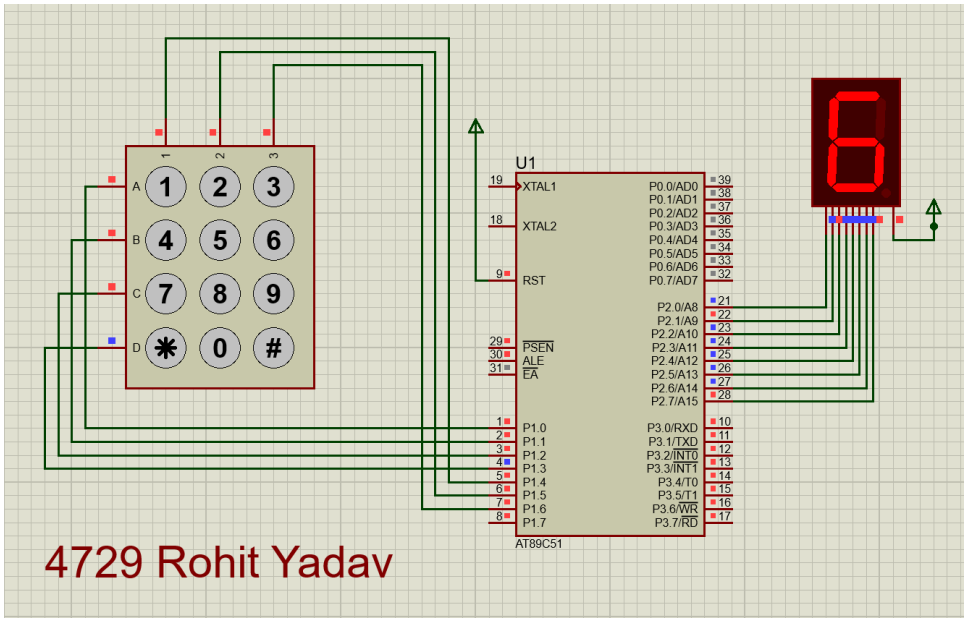


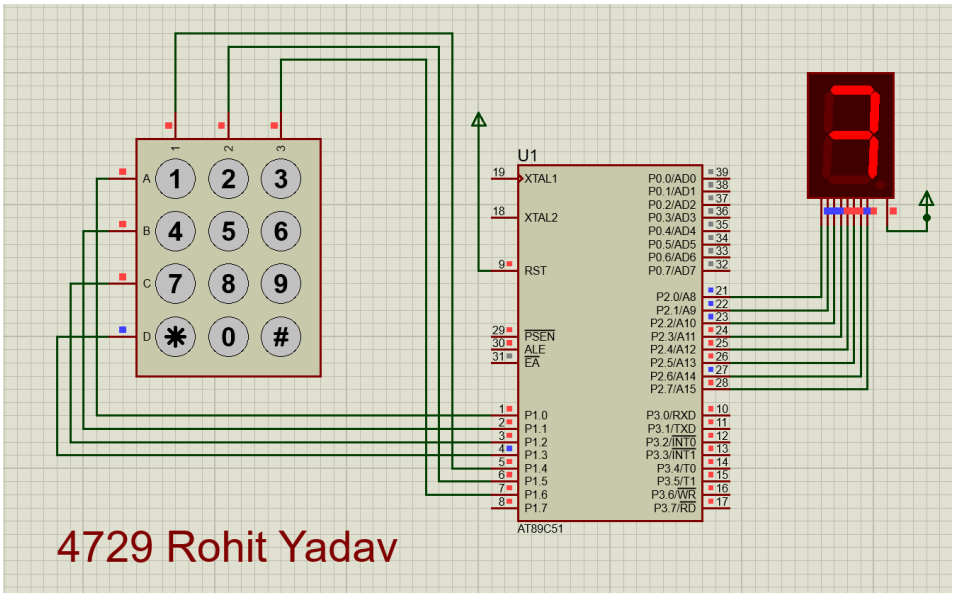


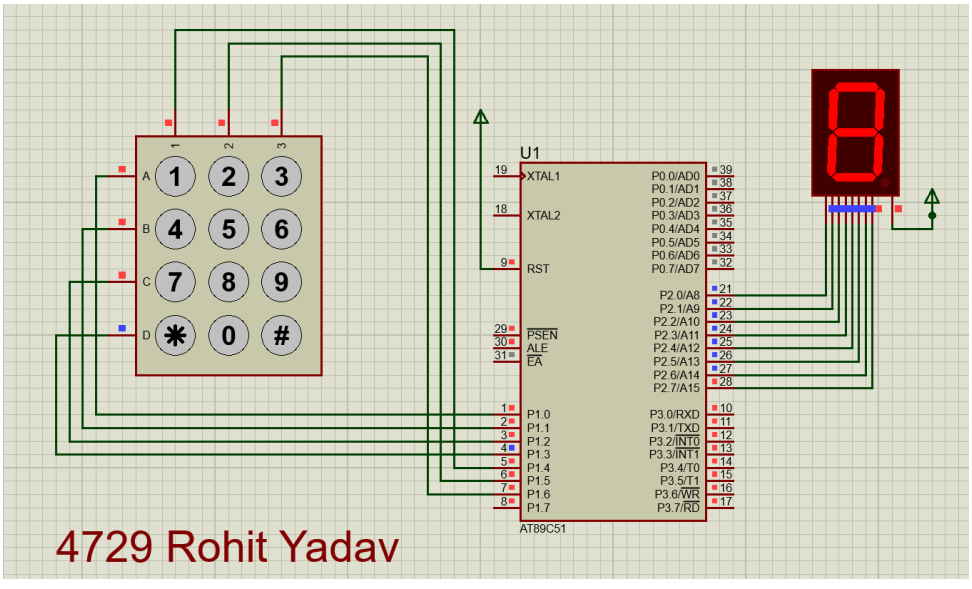

U1
AT89C51
4729 Rohit Yadav

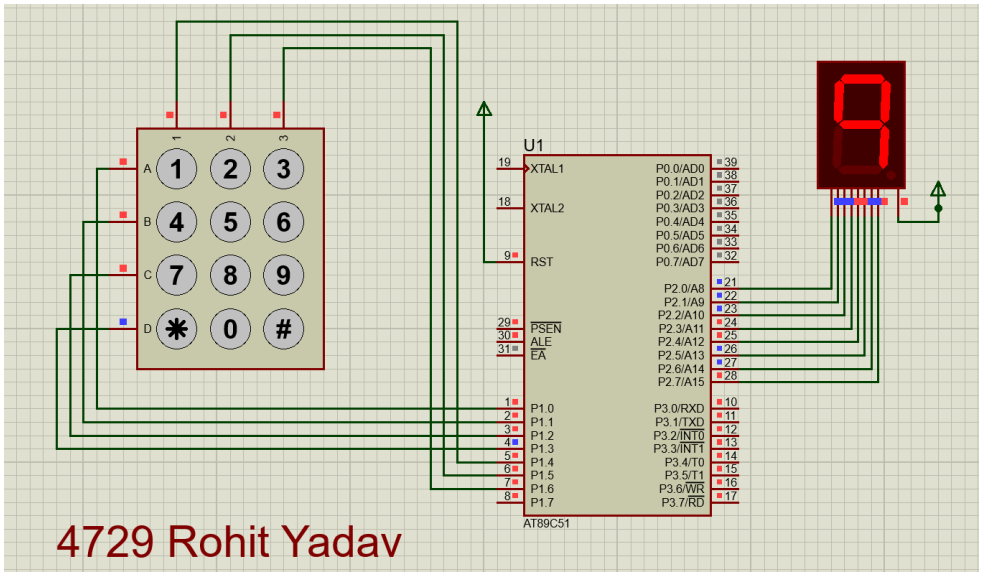

U1
AT89C51
4729 Rohit Yadav

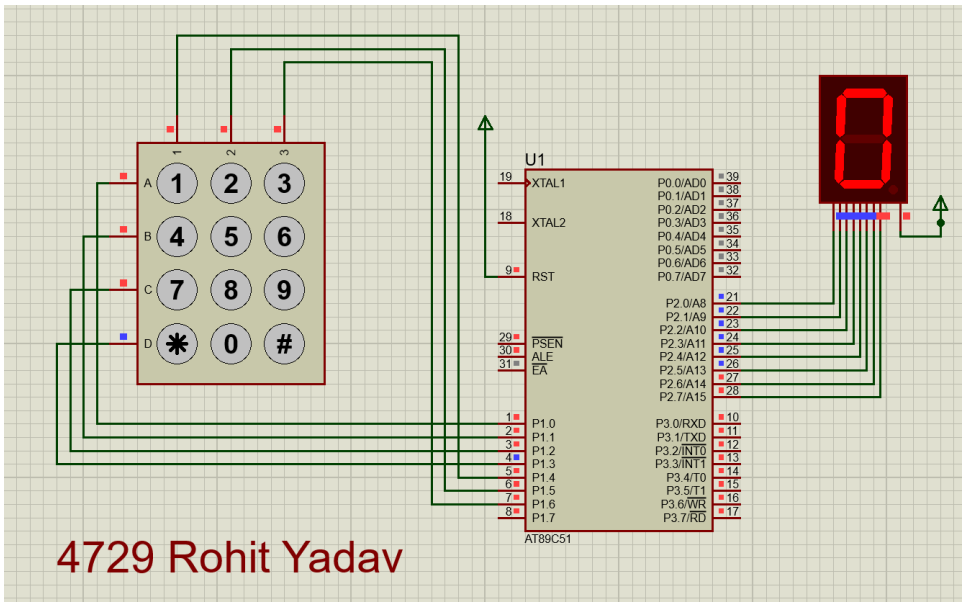

U1
AT89C51
4729 Rohit Yadav


U1
AT89C51
4729 Rohit Yadav


U1
AT89C51
4729 Rohit Yadav


U1
AT89C51
4729 Rohit Yadav





Conclusion:

This program demonstrates the use of a 4x3 keypad with the 8051 microcontroller to display the corresponding numbers on a 7-segment display. The microcontroller scans the keypad by toggling rows and checking columns to detect key presses. Once a key is detected, the program converts the key press into a 7-segment code and sends it to Port 2, which is connected to the 7-segment display.

This setup can be extended by adding more functionality, such as handling multi-digit input or adding additional keypad keys for more complex operations. The program can also be modified to handle debouncing if necessary for more accurate key detection.

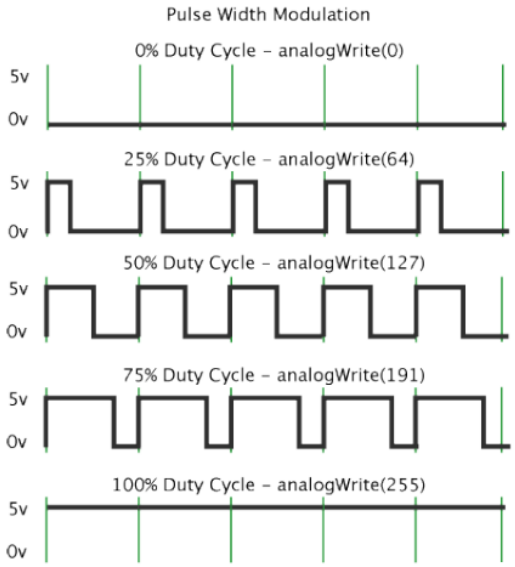
Practical 4

Aim:

PWM (Pulse Width Modulation) based Motor Control

Theory:

This code is designed to control the operation of a DC motor using a microcontroller (8051). The motor is connected to Port 2, and its speed is controlled based on the input received from three switches connected to Port 3. The motor's speed is varied according to whether a "low," "average," or "high" signal is detected.



Duty Cycle :- T on / T off %

Key Components:

Motor Control Pin:

mot: This pin controls the ON/OFF state of the DC motor connected to P2^0. When mot = 1, the motor is turned on, and when mot = 0, the motor is turned off.

Switches for Speed Control:

low: This switch is connected to P3^0. It corresponds to the "low" speed setting for the motor.

avg: This switch is connected to P3^1. It corresponds to the "average" speed setting for the motor.

high: This switch is connected to P3^2. It corresponds to the "high" speed setting for the motor.

Motor Speed Control Logic:

The motor’s speed is controlled by how long the motor is turned ON and OFF, which is managed using the delay() function. The delay() function introduces a delay in the operation, and by adjusting the duration of the ON and OFF states of the motor, we can control its speed. The delays are adjusted based on which switch (low, average, or high) is pressed.

Low Speed (low == 0):

When the low switch is pressed, the motor is turned ON for a short duration (10 units of time) and then turned OFF for a longer duration (90 units of time).

This results in a low speed as the motor is on for a shorter time.

Average Speed (avg == 0):

When the avg switch is pressed, the motor is turned ON for a medium duration (50 units of time) and turned OFF for the same medium duration (50 units of time).

This results in a medium speed for the motor.

High Speed (high == 0):

When the high switch is pressed, the motor is turned ON for a longer duration (90 units of time) and turned OFF for a shorter duration (10 units of time).

This results in a high speed for the motor.

Delay Function:

The delay() function is used to introduce a time delay between switching the motor ON and OFF. The delay() function uses a nested loop to create a delay, where the value d1 specifies how many iterations the loop will execute.

Working of the Main Function:

Initialization:

The motor is initially turned off by setting mot = 0.

Main Loop:

The program continuously checks the states of the switches (low, avg, and high).

Depending on which switch is pressed (low, average, or high), the motor is turned ON and OFF with different timing to achieve different speeds.

When no switches are pressed, the motor remains off.

Components:

1. AT89C51
2. BUTTON (x3)
3. VCC (POWER)
4. GND (GROUND)
5. DC MOTOR
6. L293D (MOTOR DRIVER)

Steps and Procedure:

1. Software: Keil uVision5

1. Project > New Project > AT89C51

7. File name: Choose a proper file name and save the project in the desired folder. Make sure the file extension is .c.

8. Right-click on the Source Group, then add an existing file → select the file to be added.

9. Rebuild the project by pressing F7 or check for errors and solve them.

10. Right-click on Target, go to Target Options, set the Clock Frequency to 11.0592 MHz.

11. Select ROM Output and choose to create HEX file.

9. Rebuild the code again. The HEX file will be generated in the Objects folder. Minimize the window.

2. Software: Proteus 8.13 SP0 Pro

1. File → Project File: In the project file window, choose the appropriate options → create PCB/Schematics.

2. Create a Firmware Project for the 8051 AT89C51 microcontroller.

3. Complete the setup by selecting the appropriate options, and finish setting up the firmware project.

4. Select all required components from the toolbar (which is P in bluebox). Type the component names and select them. Click OK.

5. Select the required components from the list and click on the schematic screen to place them.

6. To add more ports, click on the appropriate component to add it to the schematic.

7. Hover your mouse over the pin and click to connect the components with wires, forming the circuit diagram.

8. For Power and Ground, select the terminal component for power (Vcc) and ground (GND), and connect them properly.

9. After a successful connection, double-click on the controller → select Program → navigate to your folder, select the HEX file, and click OK.

10. Set the Clock Frequency to 11.0592 MHz and click OK.

11. At the bottom (Downside), click Play to run the program.

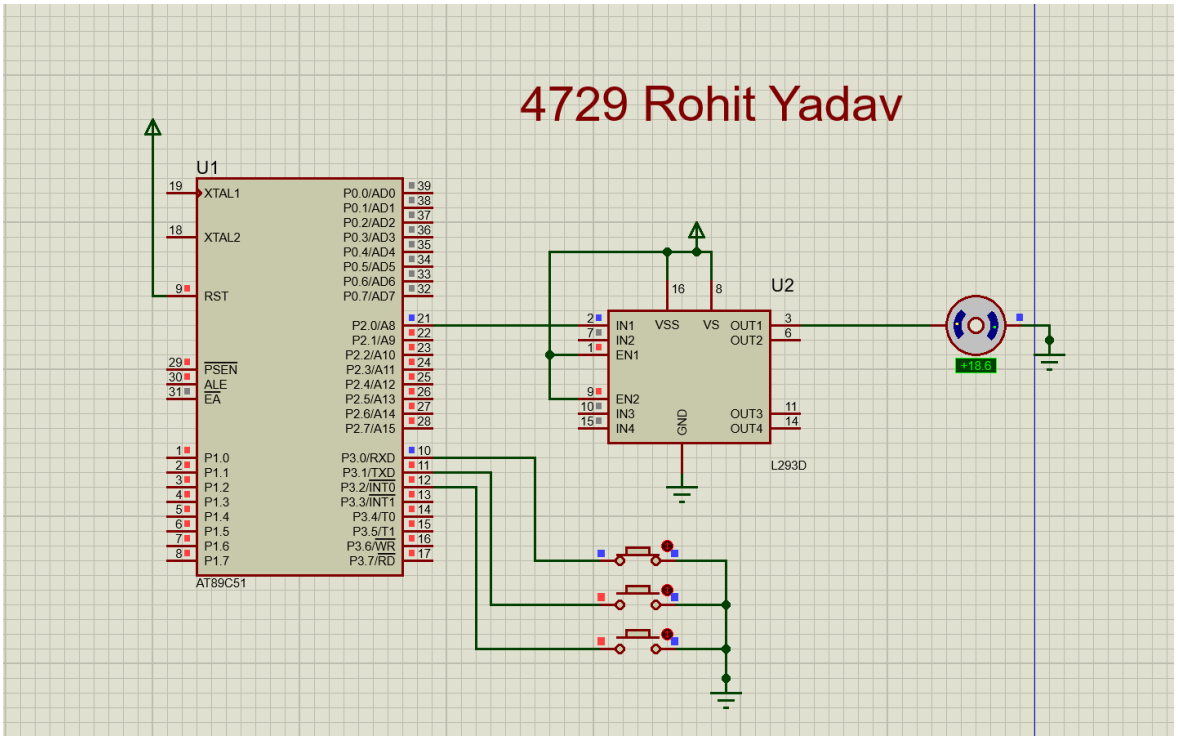
12. If the output is not available, perform the previous steps again to ensure everything is set up correctly.

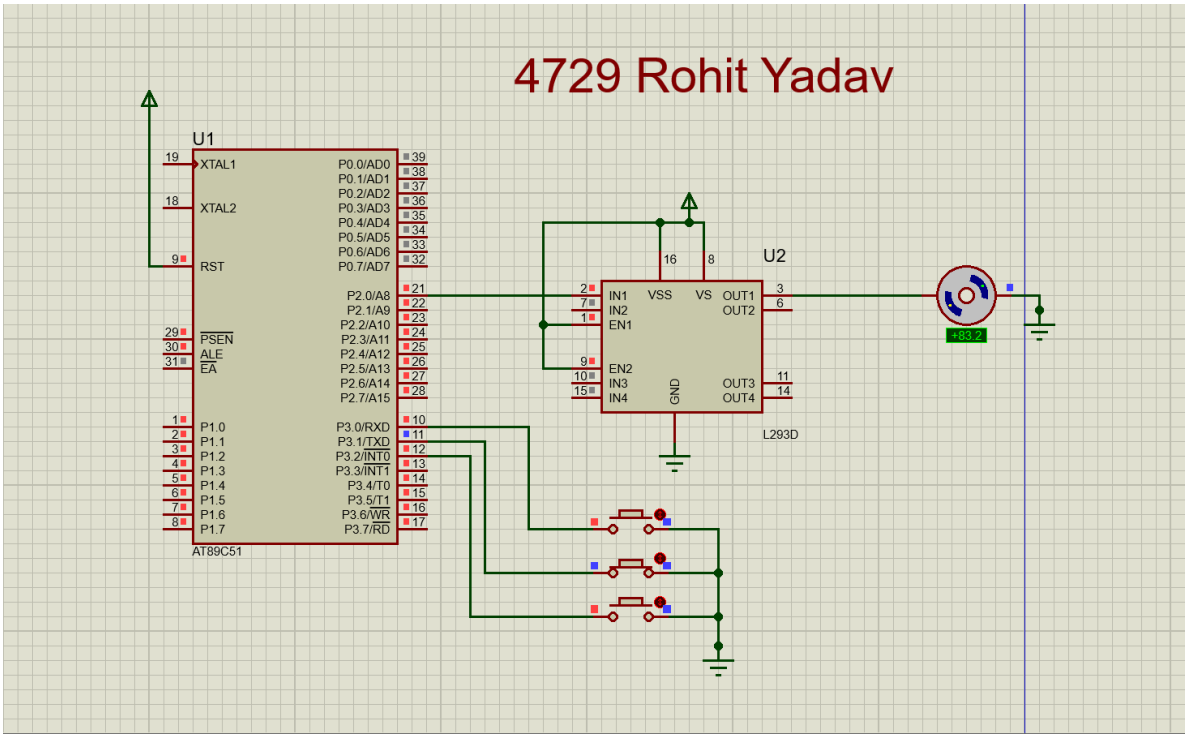
Code:

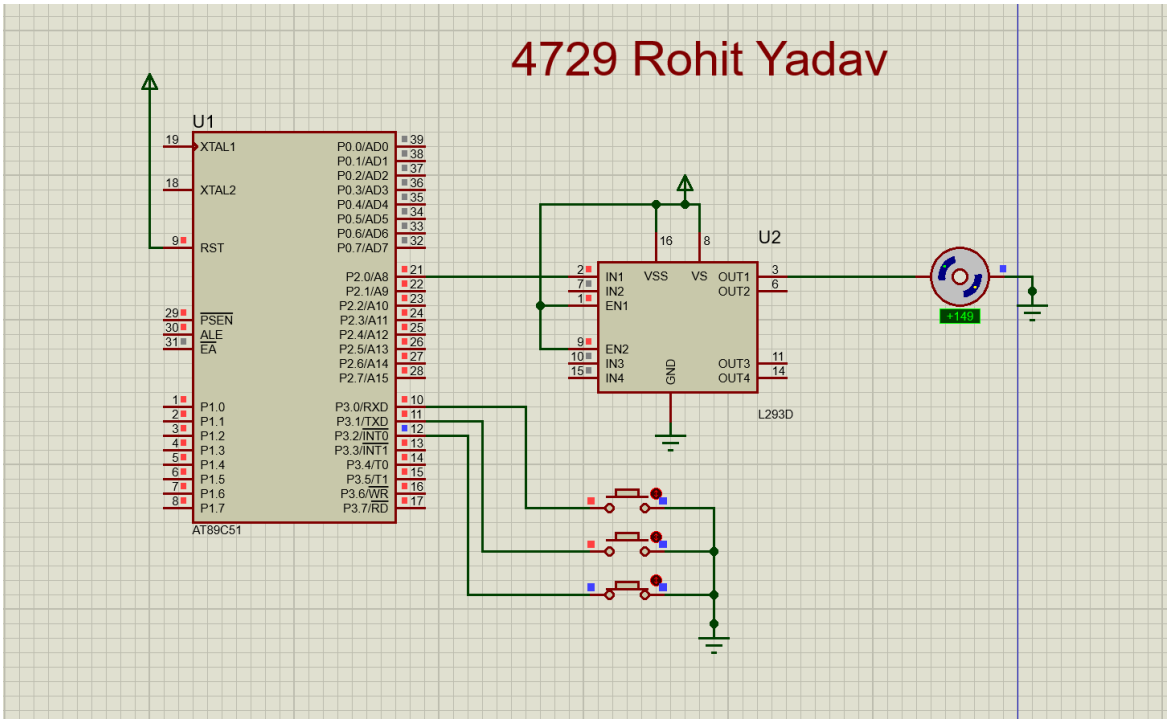
```
#include<reg51.h>
sbit mot = P2^0;
sbit low = P3^0;
sbit avg = P3^1;
sbit high = P3^2;

void delay(unsigned int);
void main()
{
mot = 0;
while(1)
{
if(low == 0)
{
mot = 1;
delay(10);
mot = 0;
delay(90);
```

```
}

if(avg == 0)
{
mot = 1;
delay(50);
mot = 0;
delay(50);
}

if(high == 0)
{
mot = 1;
delay(90);
mot = 0;
delay(10);
}
}
}

void delay(unsigned int d1)
{
unsigned int d2;
for(d2=0;d2<d1;d2++);
}
```

Output:


4729 Rohit Yadav
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
AT89C51
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
U2
IN1
IN2
EN1
EN2
IN3
IN4
VSS
VS
GND
OUT1
OUT2
OUT3
OUT4
L293D
+18.6


4729 Rohit Yadav
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
AT89C51
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
U2
IN1
IN2
EN1
EN2
IN3
IN4
VSS
VS
GND
OUT1
OUT2
OUT3
OUT4
L293D
+83.2



Conclusion:

This program demonstrates how to control the speed of a DC motor based on user input using an 8051 microcontroller. The motor's speed is determined by which of the three switches (low, avg, high) is pressed. The delay() function is used to control how long the motor stays on and off, thereby controlling the speed. By adjusting the duration of the motor's ON and OFF states, we can simulate different speed levels.

The system can be extended by adding more speed levels or by using PWM (Pulse Width Modulation) for more precise control over the motor speed. This program offers a simple and effective method of motor control based on user input through switches.

Practical 5

Aim:

Serial Communication

Theory:

This code is a simple example of serial communication (UART) on an 8051 microcontroller. It transmits the character 'M' through the serial port using the UART protocol. The program configures the UART, sets up the baud rate, and then continuously transmits the character 'M'. Below, we will go over the key components, the baud rate, and the clock frequency used.

Key Components:

SCON (Serial Control Register):

The SCON register is used to configure the serial communication settings (such as mode, enable transmission/reception, etc.).

SCON = 0x50; configures the serial port for mode 1 (8-bit data, 1 stop bit) and enables the receiver (by setting the REN bit) but does not enable interrupts.

TMOD (Timer Mode Register):

The TMOD register is used to configure timers. TMOD = 0x20; sets Timer 1 to Mode 2 (8-bit auto-reload mode).

Timer 1 is used to generate the clock signal for the serial communication.

TH1 (Timer 1 High Byte):

TH1 = -3; sets the high byte of Timer 1 to -3 (0xFD). This value is used to generate the correct baud rate for serial communication.

This specific value is calculated to achieve a baud rate of 9600 baud with a system clock frequency of 11.0592 MHz.

TR1 (Timer 1 Run Control):

TR1 = 1; starts Timer 1, enabling the timer to count and generate the necessary timing for the serial communication.

SBUF (Serial Buffer Register):

SBUF is the data register for transmitting and receiving serial data. Here, we load it with the character 'M' to send over the UART.

SBUF = 'M'; writes the ASCII character 'M' to the transmit buffer.

TI (Transmit Interrupt Flag):

TI is the Transmit Interrupt Flag. It is set to 1 when the data in the transmit register has been shifted out and is ready for new data to be written.

The while (TI == 0); loop waits for the transmission to complete before writing new data to SBUF. After transmission, TI is cleared by setting TI = 0; to indicate that the buffer is ready to transmit again.

Baud Rate Calculation:

The baud rate is controlled by the Timer 1 settings and the system clock frequency.

System Clock Frequency:

The system clock frequency is typically 11.0592 MHz for 8051 microcontrollers, commonly used in UART communication for a standard baud rate.

Baud Rate Formula:

For UART communication, the baud rate can be calculated using the following formula:

Baud Rate = 2 x Timer Overflow Rate / 32 x (256 - TH1)

Where:

Timer Overflow Rate = System Clock Frequency / 12

(since Timer 1 uses a clock divided by 12)

TH1 = 0xFD (value set in the code)

Given:

System Clock Frequency = 11.0592 MHz

TH1 = 0xFD = 253 (decimal)

$$\text{Timer Overflow Rate} = 11.0592 \times 10^6 / 12 = 921{,}600 \text{ Hz}$$

Now, calculate the baud rate:

$$\text{Baud Rate} = 2 \times 921{,}600 / 32 \times (256 - 253) = 1{,}843{,}200 / 1024 = 9600 \text{ baud}$$

So, the program is configured to transmit data at a baud rate of 9600.

Components:

1. AT89C51
2. VIRTUAL TERMINAL

Steps and Procedure:

1. Software: Keil uVision5

1. Project > New Project > AT89C51

2. File name: Choose a proper file name and save the project in the desired folder. Make sure the file extension is .c.

3. Right-click on the Source Group, then add an existing file → select the file to be added.

4. Rebuild the project by pressing F7 or check for errors and solve them.

5. Right-click on Target, go to Target Options, set the Clock Frequency to 11.0592 MHz.

6. Select ROM Output and choose to create HEX file.

7. Rebuild the code again. The HEX file will be generated in the Objects folder. Minimize the window.

2. Software: Proteus 8.13 SP0 Pro

1. File → Project File: In the project file window, choose the appropriate options → create PCB/Schematics.

2. Create a Firmware Project for the 8051 AT89C51 microcontroller.

3. Complete the setup by selecting the appropriate options, and finish setting up the firmware project.

4. Select all required components from the toolbar (which is P in bluebox). Type the component names and select them. Click OK.

5. Select the required components from the list and click on the schematic screen to place them.

6. To add more ports, click on the appropriate component to add it to the schematic.

7. To add virtual terminal click on Virtual instrument mode.

8. Hover your mouse over the pin and click to connect the components with wires, forming the circuit diagram.

9. For Power and Ground, select the terminal component for power (Vcc) and ground (GND), and connect them properly.

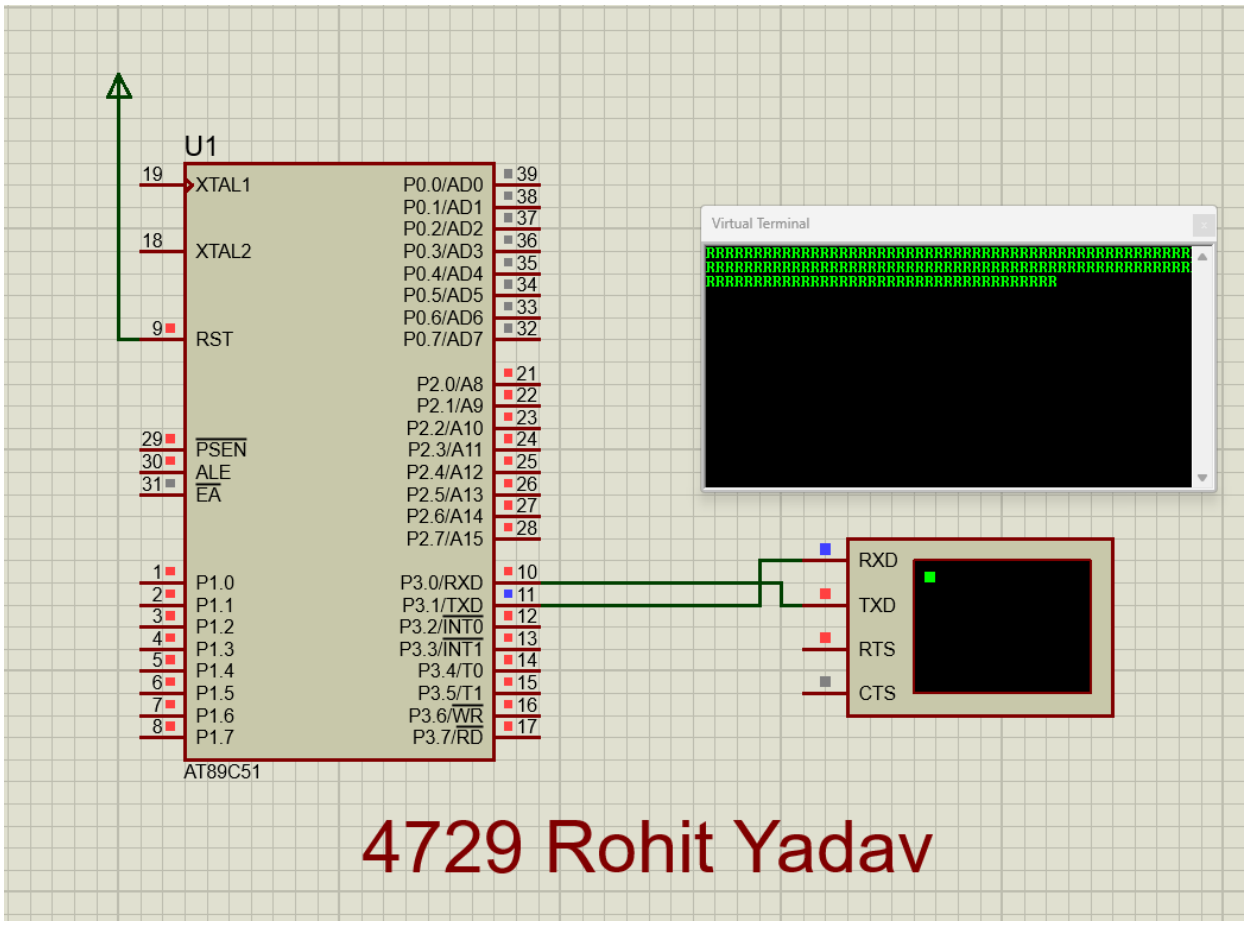
10. After a successful connection, double-click on the controller → select Program → navigate to your folder, select the HEX file, and click OK.

11. Set the Clock Frequency to 11.0592 MHz and click OK.

12. At the bottom (Downside), click Play to run the program.

13. If the output is not available, perform the previous steps again to ensure everything is set up correctly.

Code:



Conclusion:

This program demonstrates how to configure the 8051 microcontroller for serial communication (UART) to transmit a character ('M'). The program:

Configures the UART for 8-bit data, 1 stop bit (mode 1).

Sets up Timer 1 in Mode 2 to generate the correct baud rate (9600 baud).

Continuously transmits the character 'M' over the serial port by writing it to SBUF and waiting for the transmission to complete using the TI flag.

Key Points:

Baud rate: 9600 baud.

System clock frequency: 11.0592 MHz.

Timer used: Timer 1 in Mode 2 (auto-reload mode)

Practical 6

Aim:

Program to read analog voltage applied at the input and display

Theory:

This code is a simple embedded system program designed for the 8051 microcontroller to control two output ports (Port 1 and Port 2) using a binary pattern with delays to alternate the outputs. It can be used for controlling LEDs or any digital devices connected to the microcontroller's I/O pins.

The two output ports (P1 and P2) are continuously toggled between two specific states in the program, creating an alternating visual effect or pattern. A delay function is used to control the timing between the two states, allowing the user to see the change from one state to another.

Components of the Code:

Delay Function: The delay() function is implemented using nested loops. These loops create a delay that controls how long the outputs on Port 1 and Port 2 stay in each state.

The outer loop runs time times, and for each iteration of the outer loop, the inner loop runs 1275 times.

The number of total iterations determines how long the program waits before changing the state of the output ports.

By passing different values to the delay() function, the time between the changes in output state can be adjusted.

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

Port Manipulation (P1 and P2):

The Port 1 (P1) and Port 2 (P2) are controlled using bitwise assignments.

P1 = 0xAA; and P2 = 0xAA; set both ports to the alternating pattern 10101010 in binary (where each bit alternates between 1 and 0).

P1 = 0x55; and P2 = 0x55; set both ports to the alternating pattern 01010101 in binary (the inverse of 0xAA).

This toggling pattern causes the connected digital devices (like LEDs) to blink in a specific alternating sequence.

Infinite Loop:

The infinite while(1) loop ensures the program continuously executes the pattern changes. Once one pattern is displayed for the set time, it switches to the next pattern and repeats.

```
while(1) {
    P1 = 0xAA;
    P2 = 0xAA;
    delay(500);
    P1 = 0x55;
    P2 = 0x55;
    delay(500);
}
```

Port 1 and Port 2 will continuously alternate between the two patterns:

0xAA (10101010): The outputs alternate between high and low for each pin, with 1 on every even pin and 0 on every odd pin.

0x55 (01010101): The outputs alternate again, but this time 1 appears on odd pins and 0 appears on even pins.

This alternating pattern can create visual effects, like blinking LEDs connected to the microcontroller, where the LEDs blink in a regular alternating pattern.

Delay:

The delay is controlled by the time parameter passed to the delay() function. The number of iterations in the inner loop and outer loop determines the overall duration of the delay. Since the delay is proportional to time * 1275, this affects how fast the pattern alternates between 0xAA and 0x55.

In the main loop, delay(500) is used, meaning the program waits for 500 iterations before changing the pattern. Depending on the microcontroller's clock speed, the

actual delay time will vary, but it will provide a visible delay between the toggling patterns.

Components:

1. AT89C51
2. LOGIC PROBE (BIG)
3. LED BIBY (x8)
4. RES (RESISTOR) (x8 330 Ohm)
5. GND (GROUND)

Steps and Procedure:

1. Software: Keil uVision5

1. Project > New Project > AT89C51

2. File name: Choose a proper file name and save the project in the desired folder. Make sure the file extension is .c.

3. Right-click on the Source Group, then add an existing file → select the file to be added.

4. Rebuild the project by pressing F7 or check for errors and solve them.

5. Right-click on Target, go to Target Options, set the Clock Frequency to 11.0592 MHz.

6. Select ROM Output and choose to create HEX file.

7. Rebuild the code again. The HEX file will be generated in the Objects folder. Minimize the window.

2. Software: Proteus 8.13 SP0 Pro

1. File → Project File: In the project file window, choose the appropriate options → create PCB/Schematics.

2. Create a Firmware Project for the 8051 AT89C51 microcontroller.

3. Complete the setup by selecting the appropriate options, and finish setting up the firmware project.

4. Select all required components from the toolbar (which is P in bluebox). Type the component names and select them. Click OK.

5. Select the required components from the list and click on the schematic screen to place them.

6. To add more ports, click on the appropriate component to add it to the schematic.

7. Hover your mouse over the pin and click to connect the components with wires, forming the circuit diagram.

8. For Power and Ground, select the terminal component for power (Vcc) and ground (GND), and connect them properly.

9. After a successful connection, double-click on the controller → select Program → navigate to your folder, select the HEX file, and click OK.

10. Set the Clock Frequency to 11.0592 MHz and click OK.

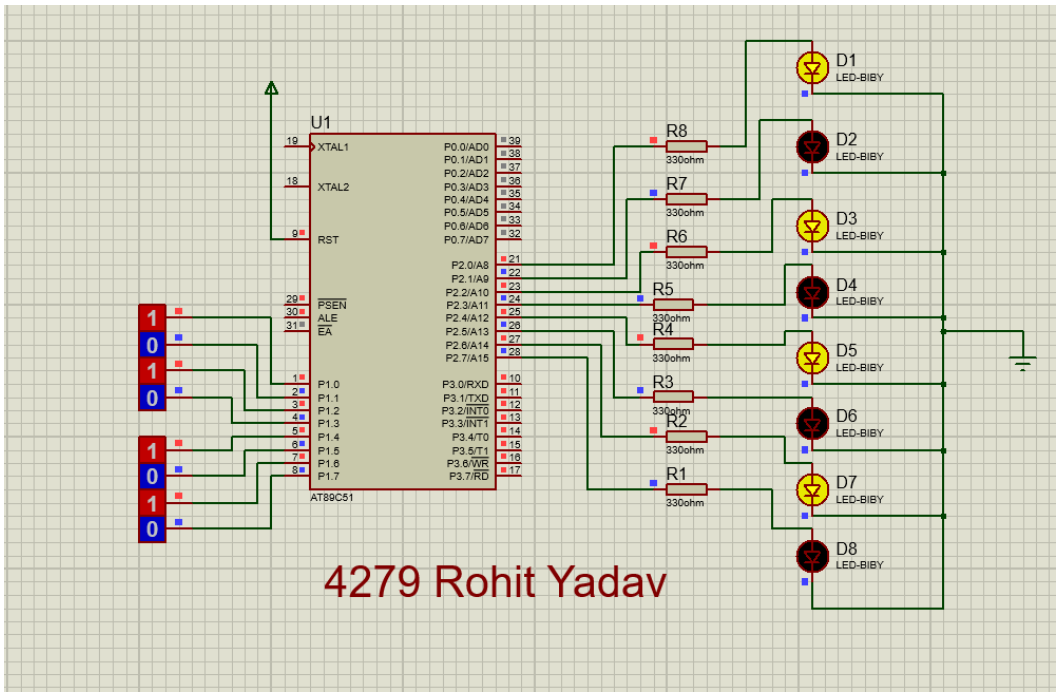
11. At the bottom (Downside), click Play to run the program.

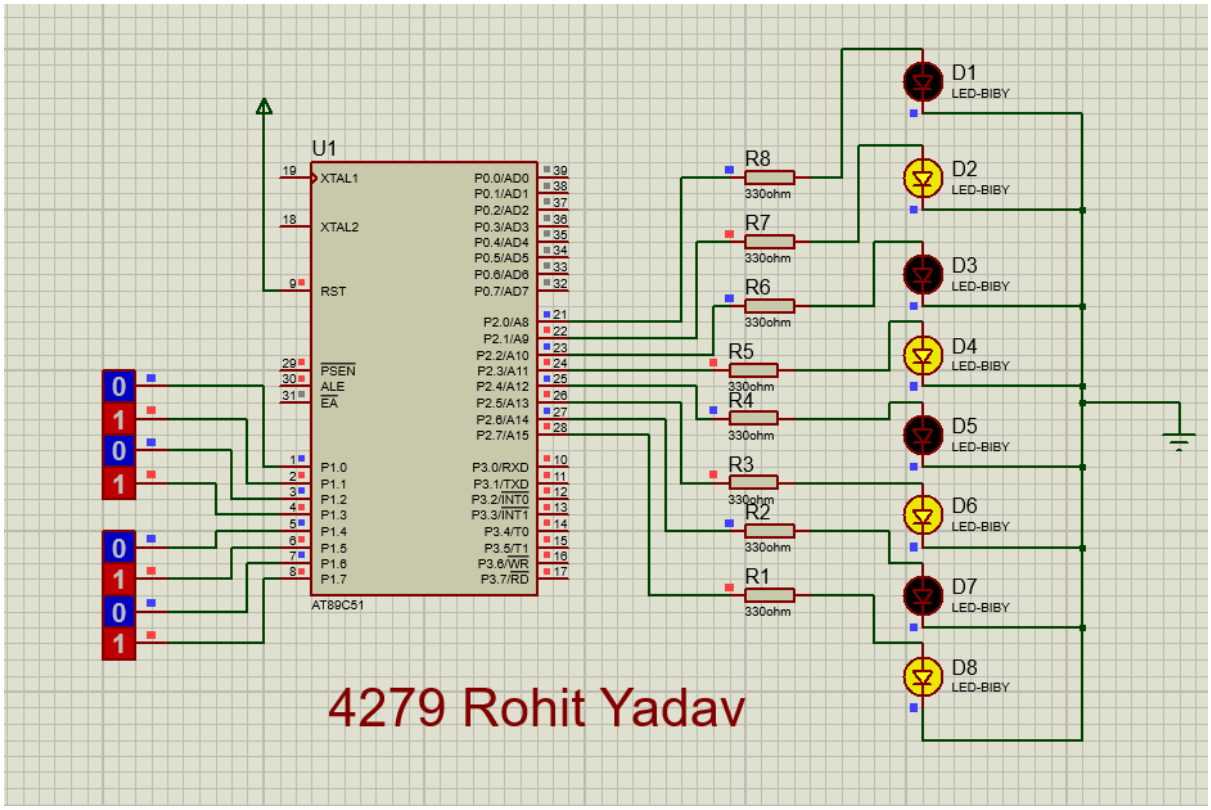
12. If the output is not available, perform the previous steps again to ensure everything is set up correctly.

Code:

```
#include <reg51.h>
void delay(unsigned int time) {
    unsigned int i, j;
    for(j = 0; j < time; j++) {
        for(i = 0; i < 1275; i++);
    }
}
void main() {
    while(1) {
        P1 = 0xAA;
        P2 = 0xAA;
        delay(500);
        P1 = 0x55;
        P2 = 0x55;
        delay(500);
    }
}
```

Output:





Conclusion:

This program demonstrates the use of Port 1 and Port 2 on an 8051 microcontroller to control digital outputs using a simple alternating pattern. The key concepts explored in the code are:

Port manipulation to set bit patterns on the microcontroller's I/O pins.

Delay implementation using nested loops to control the timing between output changes.

Infinite loop for continuous execution of the alternating patterns.

This type of pattern is commonly used in embedded systems for visual effects, such as controlling LEDs or other digital outputs. The program is a basic demonstration of how to work with digital I/O and create timed effects using simple software techniques.

Practical 7

Aim:

Implement a prototype for elevators

Theory:

This program is designed to control a motorized elevator system using an 8051 microcontroller. The elevator can be moved up or down depending on the user input, which is provided via Port 0 (P0) of the microcontroller. The elevator's position is controlled using Port 2 (P2), and the motor's steps are controlled by Port 3 (P3).

The program consists of several functions:

A delay function that introduces a time delay to control the speed of the motor.

Two movement functions, up() and down(), which control the elevator's movement in the upward or downward direction by manipulating the motor's steps.

A control function that adjusts the elevator’s position based on input, either moving the elevator up or down to the target position.

The main function continuously checks Port 0 (P0) for user input, and depending on the input, moves the elevator to a specific position.

The elevator’s position is controlled in increments, and the program allows the elevator to be positioned based on different values of input on Port 0.

Components:

1. AT89C51
2. MOTOR STEPPER UNIPOLAR
3. 7447
4. BUTTON (x6)
5. VCC (x3)
6. ULN2003A
7. & Segment Display (7SEG-COM-ANODE)

Steps and Procedure:

1. Software: Keil uVision5

1. Project > New Project > AT89C51

2. File name: Choose a proper file name and save the project in the desired folder. Make sure the file extension is .c.

3. Right-click on the Source Group, then add an existing file → select the file to be added.

4. Rebuild the project by pressing F7 or check for errors and solve them.

5. Right-click on Target, go to Target Options, set the Clock Frequency to 11.0592 MHz.

6. Select ROM Output and choose to create HEX file.

7. Rebuild the code again. The HEX file will be generated in the Objects folder. Minimize the window.

2. Software: Proteus 8.13 SP0 Pro

1. File → Project File: In the project file window, choose the appropriate options → create PCB/Schematics.

2. Create a Firmware Project for the 8051 AT89C51 microcontroller.

3. Complete the setup by selecting the appropriate options, and finish setting up the firmware project.

4. Select all required components from the toolbar (which is P in bluebox). Type the component names and select them. Click OK.

5. Select the required components from the list and click on the schematic screen to place them.

6. To add more ports, click on the appropriate component to add it to the schematic.

7. Hover your mouse over the pin and click to connect the components with wires, forming the circuit diagram.

8. For Power and Ground, select the terminal component for power (Vcc) and ground (GND), and connect them properly.

9. After a successful connection, double-click on the controller → select Program → navigate to your folder, select the HEX file, and click OK.

10. Set the Clock Frequency to 11.0592 MHz and click OK.

11. At the bottom (Downside), click Play to run the program.

12. If the output is not available, perform the previous steps again to ensure everything is set up correctly.

Code:

```
#include<REG51.H>
#include<stdio.h>
int p,q,r;
q=10; // for motor freezing count
r=10; //Rotatioin of teh motor 0 min 32000 max
delay(c)
{
        int i,j;
        if(c==0)
        {
```

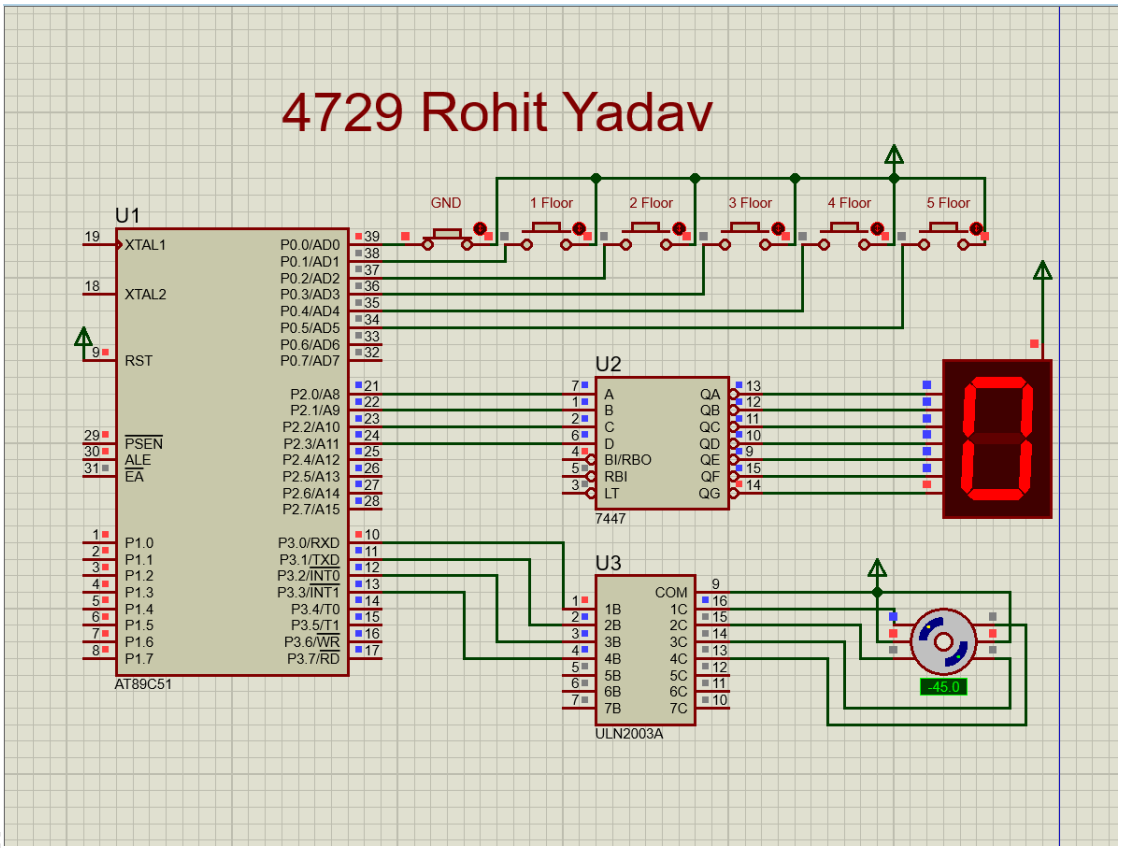
```
                for(i=0; i<500; i++)
                {
                        for(j=0; j<r; j++);
                }
        }
return(c);
}
//elevator in up direction
up(b)
{
int i,j;
for(i=1;i<=b;i++)
        {
          for(j=0; j<=10;j++)
                {
                P3=1;
                delay(0);
                P3=2;
                delay(0);
                P3=4;
                delay(0);
                P3=8;
                delay(0);
                P3=16;
                delay(0);
                }
        P2=p+i;
        }
p=p+b;
return b;
```

```
}
// elevator going down
down(b)
{
int i,j;
for(i=1;i<=b;i++)
        {
          for(j=0; j<=q;j++)
                {
                P3=16;
                delay(0);
                P3=8;
                delay(0);
                P3=4;
                delay(0);
                P3=2;
                delay(0);
                P3=1;
                delay(0);
                }
        P2=p-i;
        }
p=p-b;
return b;
}
control(a)
{
int diff;
if(a>p)
        {
```

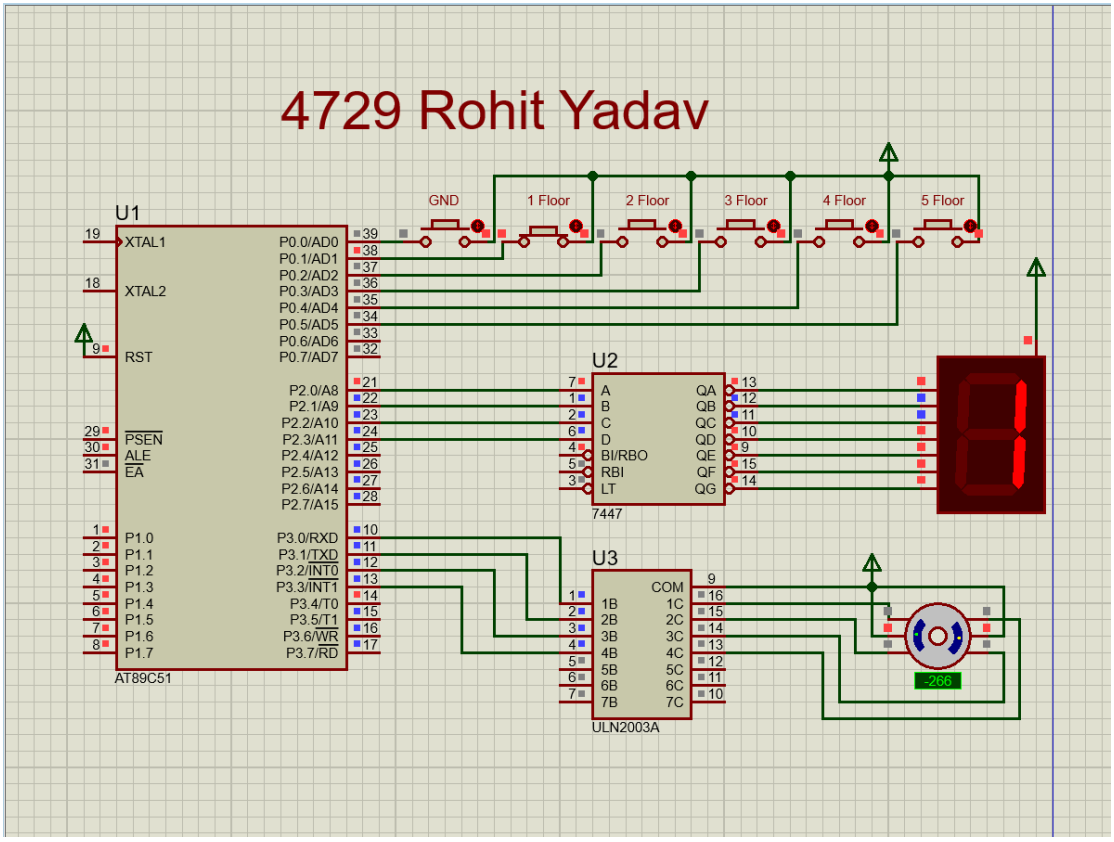
```
        diff=a-p;
        up(diff);
        }
if(a<p)
        {
        diff=p-a;
        down(diff);
        }
return a;
}
main()
{
int p1;
p=0;
P2=p;
while(1)
        {
        if(P0==2)
                {
                        p1=1;
                        control(1);
                }
        if(P0==4)
                {
                        p1=2;
                        control(2);
                }
                if(P0==8)
                {
                        p1=3;
```

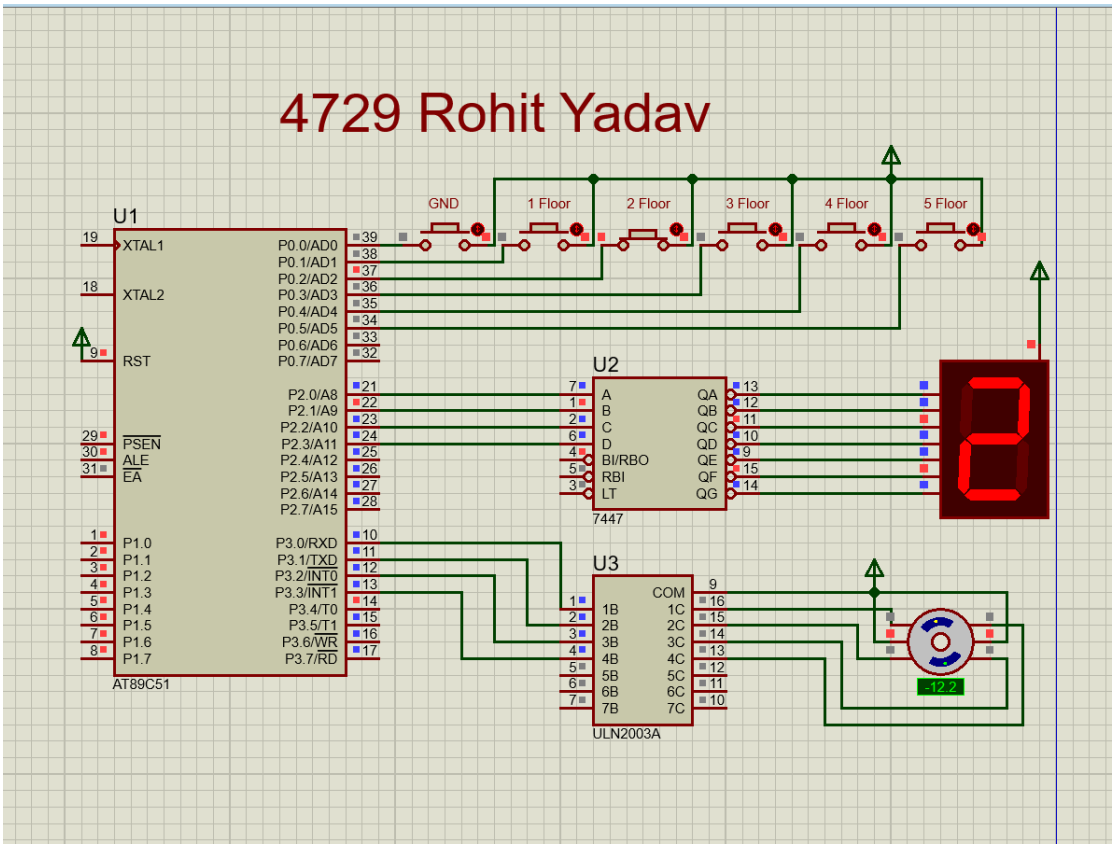
```
                control(3);
            }
            if(P0==16)
            {
                p1=4;
                control(4);
            }
            if(P0==32)
            {
                p1=5;
                control(5);
            }
            if(P0==1)
            {
                p1=0;
                control(0);
            }
        }
}
```

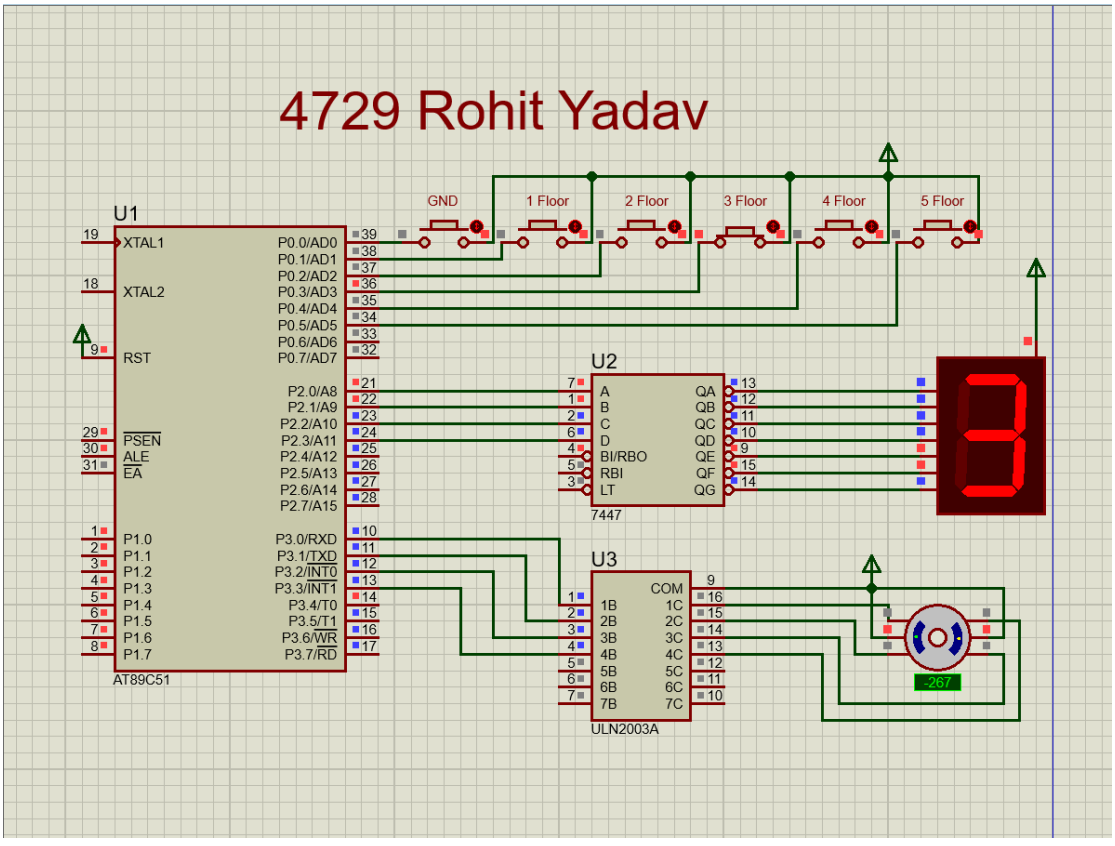
Output:

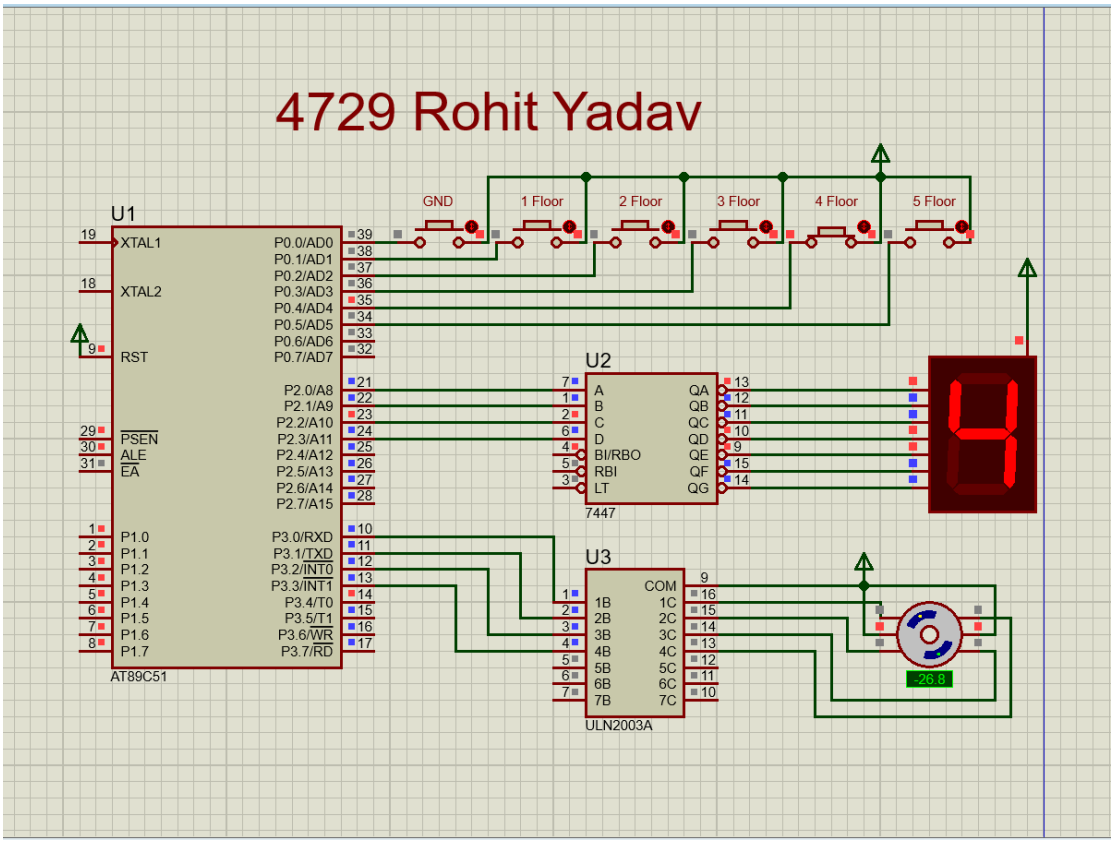

4729 Rohit Yadav
GND
1 Floor
2 Floor
3 Floor
4 Floor
5 Floor
U1
AT89C51
U2
7447
U3
ULN2003A
-45.0

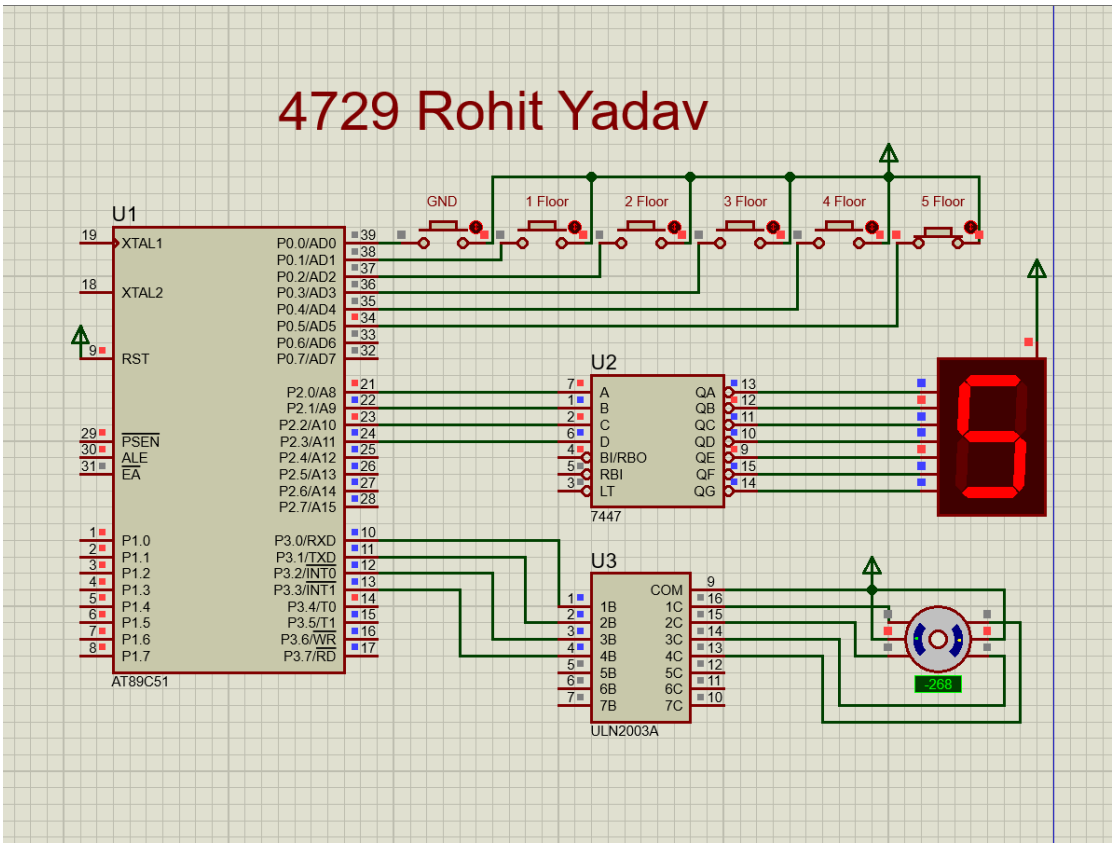
:


4729 Rohit Yadav
GND
1 Floor
2 Floor
3 Floor
4 Floor
5 Floor
U1
AT89C51
U2
7447
U3
ULN2003A
-266


4729 Rohit Yadav
GND
1 Floor
2 Floor
3 Floor
4 Floor
5 Floor
U1
AT89C51
U2
7447
U3
ULN2003A
-12.2


4729 Rohit Yadav
GND
1 Floor
2 Floor
3 Floor
4 Floor
5 Floor
U1
AT89C51
U2
7447
U3
ULN2003A
-267


4729 Rohit Yadav
GND
1 Floor
2 Floor
3 Floor
4 Floor
5 Floor
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
AT89C51
U2
A
B
C
D
BI/RBO
RBI
LT
QA
QB
QC
QD
QE
QF
QG
7447
U3
COM
1B
2B
3B
4B
5B
6B
7B
1C
2C
3C
4C
5C
6C
7C
ULN2003A
-26.8


4729 Rohit Yadav
GND
1 Floor
2 Floor
3 Floor
4 Floor
5 Floor
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
AT89C51
U2
A
B
C
D
BI/RBO
RBI
LT
QA
QB
QC
QD
QE
QF
QG
7447
U3
COM
1B
2B
3B
4B
5B
6B
7B
1C
2C
3C
4C
5C
6C
7C
ULN2003A
-268

Conclusion:

This embedded system program provides basic elevator control functionality using the 8051 microcontroller. The system adjusts the position of the elevator based on inputs received from Port 0 and controls the motor via Port 3. The system can move the elevator up or down to specific positions and can be useful in automated systems that require stepper motor control, such as small-scale elevator systems or other position-controlled applications. The delay function helps regulate the motor speed, providing smooth movements for the elevator.