Faridpur Engineering College

Department of Computer Science & Engineering

Microcontroller Lab

Course Code: *CSE-3116*

Submitted To	**Submitted By**
Md. Rany Ahmed Lecturer, Dept. of Computer Science & Engineering Faridpur Engineering College	Ahatashamul Haque Radim Reg. No: 994 Exam Roll: 4065 Session: 2020-2021

INDEX

Experiment No.: 01

Name of the Experiment: Design and implement a circuit to Turning on LED using a Microcontroller.

Objectives:

- Design and build a circuit to turn on an LED using a microcontroller.
- Implement a simple program to set an output pin to a high state.
- Verify circuit functionality through simulation and breadboard testing.
- Understand the basic principles of microcontroller I/O configuration and control.

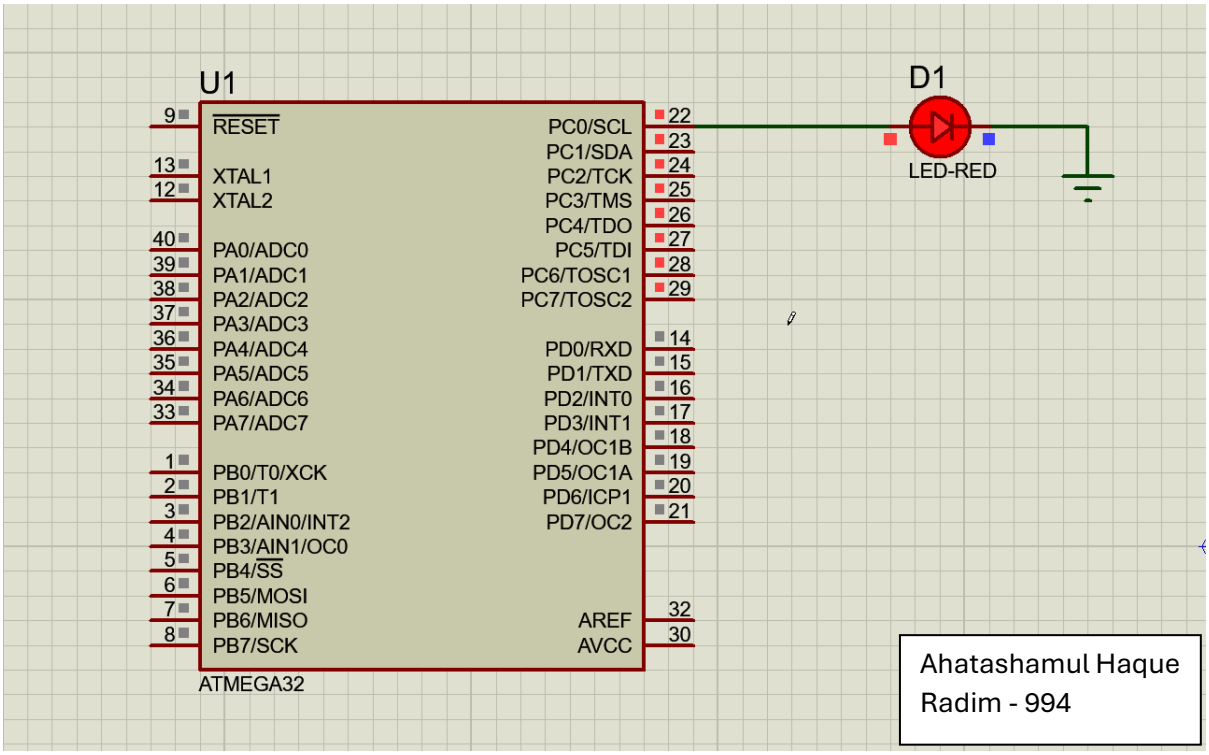
Necessary Equipment:

- Breadboard,
- Microcontroller Board (e.g., ATmega328P development board) LED (Light-Emitting Diode),
- Current-Limiting Resistor (value based on LED specifications and power supply voltage),
- Jumper Wires,
- Power Supply (voltage compatible with the microcontroller and LED),
- Computer with Microcontroller Development Software (e.g., Atmel Studio).

AVR Program:

```
#include <avr/io.h>

#include <util/delay.h>

int main(void)

{

        DDRC = 0xFF;

        PORTC = 0xFF;

}
```

Proteus Simulation:



The Proteus simulation should depict the microcontroller, LED, current-limiting resistor, power supply, and connecting wires. The microcontroller program code can be loaded into the simulation environment to verify that the LED turns on as expected.

Conclusion:

This experiment successfully demonstrated the design and implementation of a circuit to turn on an LED using a microcontroller. You gained experience in configuring

microcontroller I/O pins as outputs and writing basic control programs to set pin states. This is a fundamental step for controlling various electronic components using microcontrollers.

Experiment No.: 02

Name of the Experiment: Design and implement a circuit to Blinking LEDs using a Microcontroller.

Objectives:

- Design and build a circuit to blink an LED using a microcontroller.
- Understand the basic principles of microcontroller I/O configuration and control.
- Implement a simple program to control the LED's on/off state.
- Verify circuit functionality through simulation and breadboard testing.

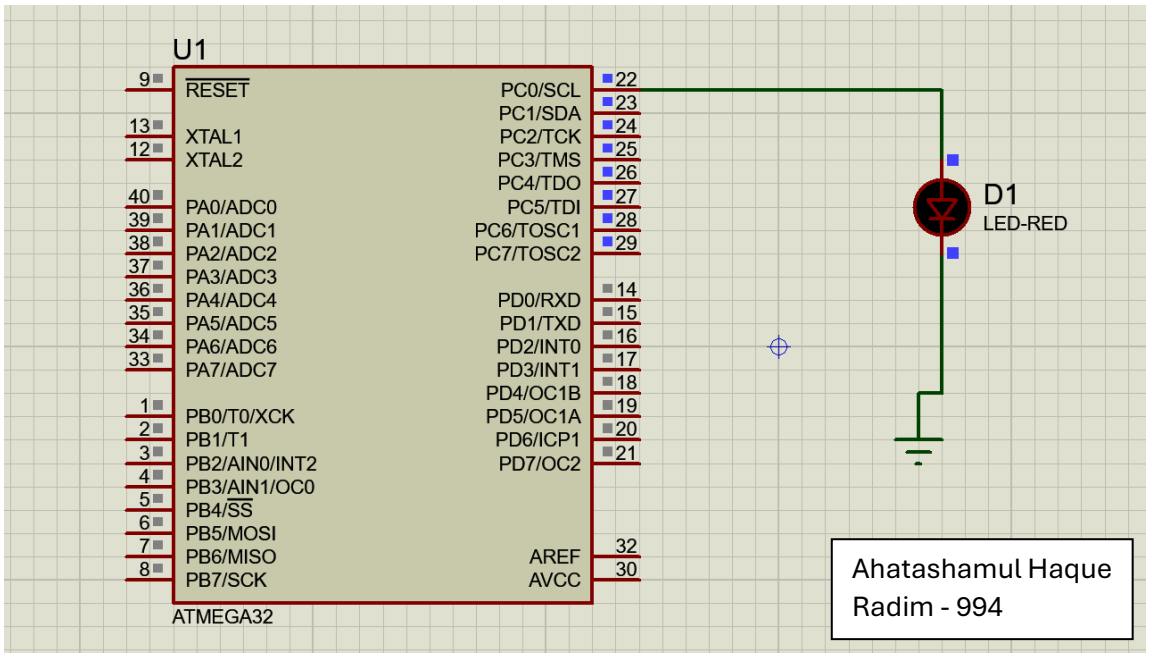
Necessary Equipment:

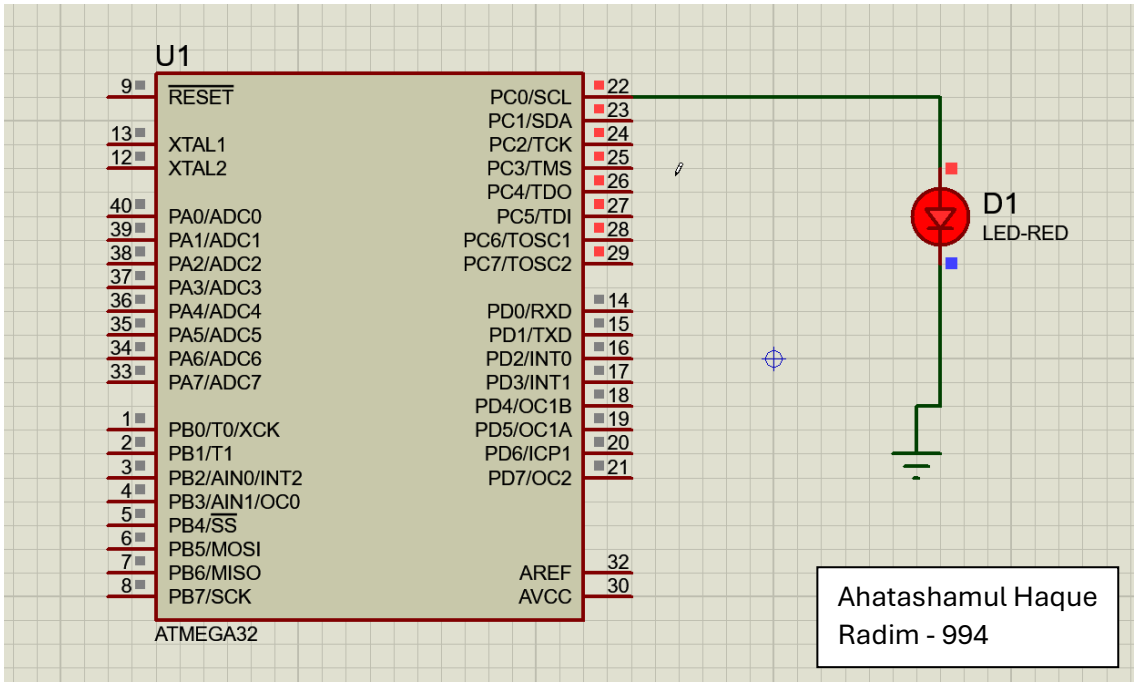
- Microcontroller Board (ATmega32 development board),
- LED (Light-Emitting Diode),
- Current-Limiting Resistor (value based on LED specifications and power supply voltage),
- Breadboard,
- Jumper Wires,
- Power Supply (voltage compatible with the microcontroller and LED),
- Computer with Microcontroller Development Software (e.g., Atmel Studio).

AVR Program:

```
#ifndef F_CPU
#define F_CPU 8000000UL
#endif
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
        DDRC = 0xFF; while(1)
        {
                PORTC = 0xFF;
                _delay_ms(100);
                PORTC = 0x00;
            _delay_ms(100);
        }
}
```

Proteus Simulation:





The Proteus simulation should depict the microcontroller, LED, current-limiting resistor, power supply, and connecting wires. The microcontroller program code can be loaded into the simulation environment to verify functionality before building the actual circuit.

Conclusion:

This experiment successfully demonstrated the design and implementation of a circuit to blink an LED using a microcontroller. You gained practical experience in configuring microcontroller I/O pins, writing basic control programs, and verifying circuit behavior through simulation. This forms a fundamental building block for more complex microcontroller-based 2

Experiment No.: 03

Name of the Experiment: Design and implement 7-segment LED display.

Microcontroller Objectives:

- Design and build a circuit to control multiple 7-segment LED displays using an ATmega32 microcontroller.
- Understand and implement the multiplexing technique to efficiently control multiple displays.
- Develop and test firmware to display digits 0-9 on the 7-segment displays.
- Verify circuit functionality through simulation in Proteus and hardware testing on a breadboard.

Necessary Equipment:

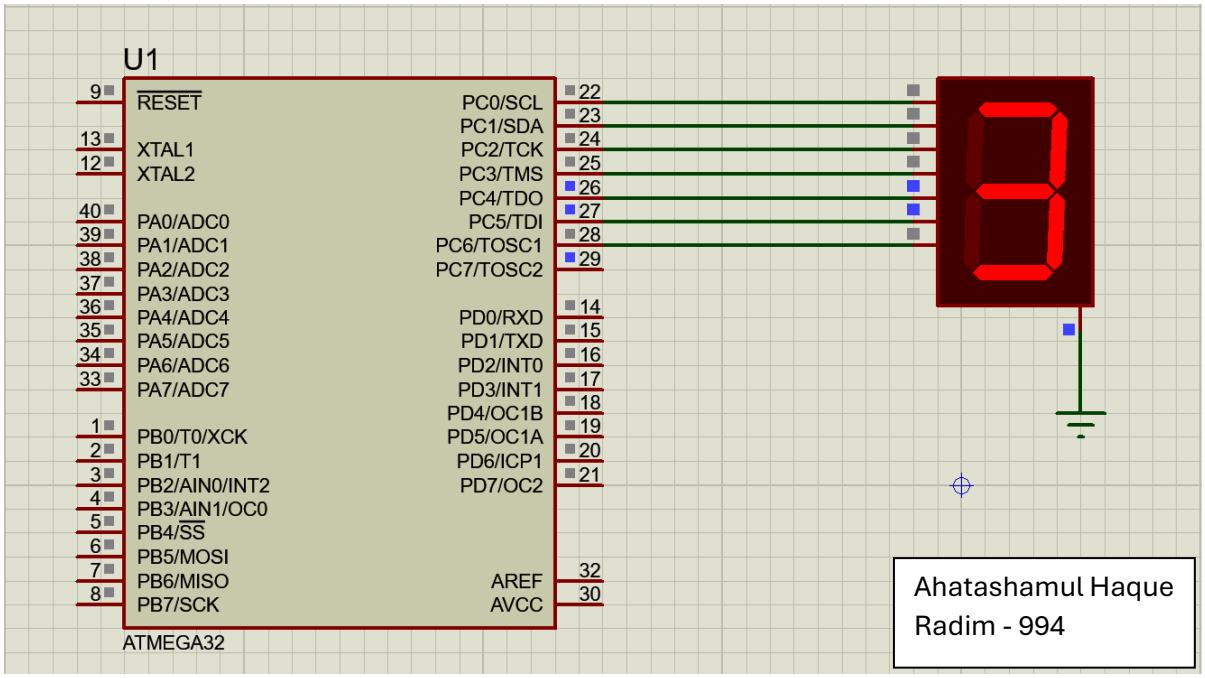
- ATmega32 Microcontroller Board,
- Two 7-Segment LED Displays (common cathode),
- Bread Board,
- Jumper wires,
- Current-Limiting Resistors (220Ω),
- Power Supply (voltage compatible with the ATmega32 and LED displays),
- Computer with Microcontroller Development Software (e.g., Microchip Studio, Proteus for simulation).

AVR Program:

```
#include<avr/io.h>
#include<util/delay.h>
void main()
{
    DDRC=0XFF; while(1)
    {
        PORTC=0b00111111;
        _delay_ms(1000);

        PORTC=0b00000110;
        _delay_ms(1000);

        PORTC=0b01011011;
        _delay_ms(1000);
```

```
            PORTC=0b01001111;
            _delay_ms(1000);

            PORTC=0b01100110;
            _delay_ms(1000);

            PORTC=0b01101101;
            _delay_ms(1000);

            PORTC=0b11111101;
            _delay_ms(1000);

            PORTC=0b00000111;
            _delay_ms(1000);

            PORTC=0B01111111;
            _delay_ms(1000);

            PORTC=0b11101111;
            _delay_ms(1000);
        }
}
```

Simulation:



The Proteus simulation should depict the microcontroller, LED, current-limiting resistor, power supply, and connecting wires. The microcontroller program code can be loaded into the simulation environment to verify functionality before building the actual circuit.

Conclusion:

This experiment successfully demonstrated the design and implementation of a circuit to blink an LED using a microcontroller. You gained practical experience in configuring microcontroller I/O pins, writing basic control programs, and verifying circuit behavior through simulation. This forms a fundamental building block for more complex microcontroller-based 2

Experiment No.: 04

Name of the Experiment: Design and implement 7-segment LED display with two outputs.

Objectives:

- Design a circuit to interface two 7-segment LED displays with an ATmega32 microcontroller in Proteus.
- Implement firmware to display different digits on each of the two 7-segment displays.
- Validate the functionality of the circuit by simulating in Proteus.
- Ensure accurate representation of digits 0-9 on each display through multiplexing.

Necessary Equipment:

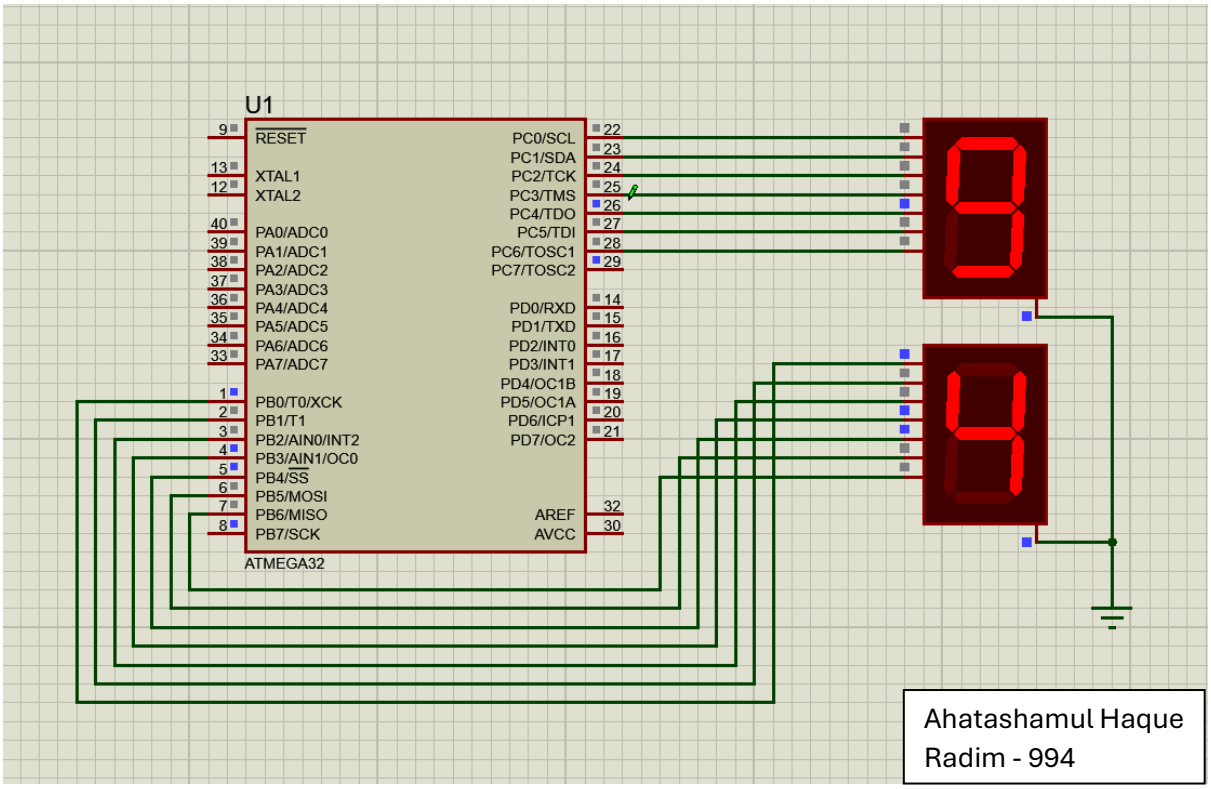
- Two 7-Segment LED Displays (common cathode).
- ATmega32 Microcontroller Board.
- Jumper Wires.
- Current limiting register (220Ω),
- Power Supply (voltage compatible with the ATmega32 and LED displays).
- Computer with Proteus for simulation.
- Breadboard.

AVR Program:

```
#include <avr/io.h>
#include <util/delay.h>
const uint8_t digits[10] = {
        0b00111111, // 0
        0b00000110, // 1
        0b01011011, // 2
        0b01001111, // 3
        0b01100110, // 4
        0b01101101, // 5
        0b01111101, // 6
        0b00000111, // 7
        0b01111111, // 8
        0b01101111  // 9
};
```

```
void init_ports() {
        DDRB = 0xFF;
        DDRC = 0xFF;
        PORTB = 0x00;
        PORTC = 0x00;
}
int main(void) {
        init_ports();
        while (1) {
                for (uint8_t i = 0; i <= 9; i++) {
                        PORTC = digits[i];
                        for (uint8_t i = 0; i <= 9; i++){
                                PORTB = digits[i];
                                _delay_ms(500);
                        }
                }
        }
        return 0;
}
```

Simulation:



The Proteus simulation should depict the microcontroller, LED, current-limiting resistor, power supply, and connecting wires. The microcontroller program code can be loaded into the simulation environment to verify functionality before building the actual circuit.

Conclusion:

This experiment successfully demonstrated the design and implementation of a circuit to blink an LED using a microcontroller. You gained practical experience in configuring microcontroller I/O pins, writing basic control programs, and verifying circuit behavior through simulation. This forms a fundamental building block for more complex microcontroller-based 2

Experiment No.: 05

Experiment Name: Implement 16x2 LCD Display to print “Welcome to Bangladesh 2.0” using Arduino simulation in proteus.

Objective:

- Learning how to interface an Arduino board with a 16x2 LCD display.
- Learning how to simulate a circuit before implementing in the real world using proteus simulation.
- The experiment helps understand the data communication between a microcontroller (Arduino) and an LCD, particularly how to send characters in the correct format to be displayed properly.
- Writing the code to display a specific string requires knowing how to use the “LiquidCrystal” library in Arduino, setting up pin configurations, initializing the display, and managing cursor positions for proper text output.

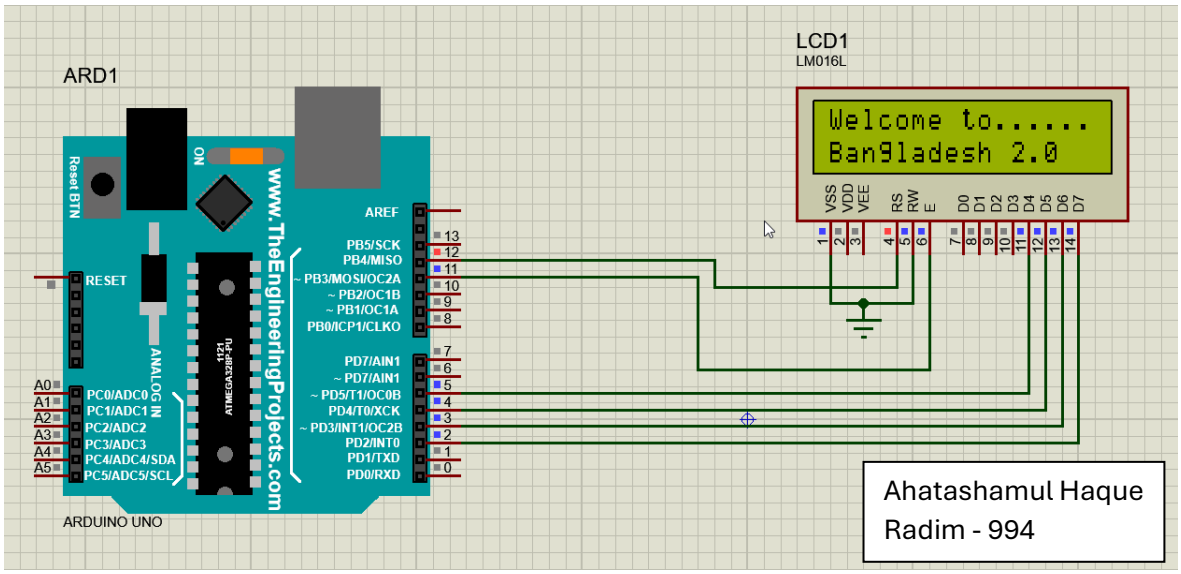
Necessary Equipment:

- Computer,
- Proteus simulation software,
- Arduino IDE,
- Necessary Library file,
- 16x2 LCD Display,
- Arduino,
- Breadboard,
- Connecting wire.

Code:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  lcd.begin(16, 2);
}
void loop() {
  lcd.clear();
  delay(500);
  lcd.setCursor(0, 0);
  lcd.print("Welcome to.....");
  delay(250);
  lcd.setCursor(0, 1);
  lcd.print("Bangladesh 2.0");
  delay(2000);
}
```

Proteus simulation:



The Proteus simulation should depict the microcontroller, LED, current-limiting resistor, power supply, and connecting wires. The microcontroller program code can be loaded into the simulation environment to verify functionality before building the actual circuit.

Conclusion: Arduino successfully interfaces with the 16x2 LCD display to output the custom message "Welcome to Bangladesh 2.0" It demonstrates the correct wiring, code implementation, and functionality of displaying text on the LCD. Additionally, it validates the use of Proteus simulation for verifying circuit behavior and Arduino programming before real-world application.

Experiment No.: 06

Experiment Name: Implement a Servo motor rotating using Arduino simulation in proteus.

Objective:

- Learn how to control the position and rotation of a servo motor using Arduino, including setting the desired angles and understanding the principles behind Pulse Width Modulation.
- Interfacing Arduino with servo motor.
- Practicing circuit simulation with Proteus.
- Explore real-world applications of servo motors in robotics, automation, and control systems by rotating the motor to predefined angles.

Necessary Equipment:

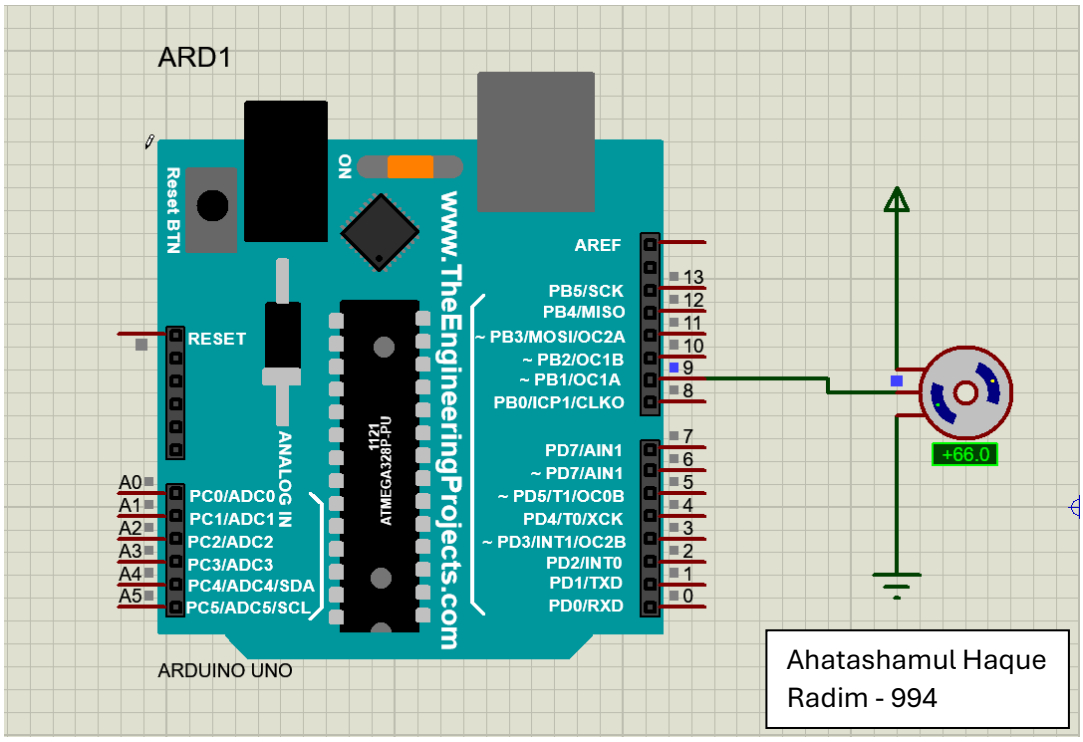
- Computer,
- Proteus simulation software,
- Arduino IDE,
- Necessary Library file,
- Arduino,
- Servo motor,
- Connecting wires,
- Power supply.

Code:

```
#include <Servo.h>
Servo myServo;
void setup() {
  myServo.attach(9);
}
void loop() {
  myServo.write(0);
  delay(500);
  myServo.write(90);
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
  void setup() {
    lcd.begin(16, 2);
  }
  void loop() {
    lcd.clear();
    delay(500);
    lcd.setCursor(0, 0);
```

```
  lcd.print("Welcome to.....");
  delay(250);
  lcd.setCursor(0, 1);
  lcd.print("Bangladesh 2.0");
  delay(2000);
 }

 delay(500);
 myServo.write(0);
 delay(500);
 myServo.write(180);
}
```

Proteus simulation:



The Proteus simulation should depict the microcontroller, LED, current-limiting resistor, power supply, and connecting wires. The microcontroller program code can be loaded into the simulation environment to verify functionality before building the actual circuit.

Conclusion: Arduino successfully controls the rotation of a servo motor using the Servo library, demonstrating the proper interfacing and programming to adjust the motor's position. The Proteus simulation validates the system's ability to control servo movement through PWM signals, providing a foundation for applications in robotics and automation.

Experiment No.: 07

Experiment Name: Implement a 4x4 matrix keyboard using a microcontroller.

Objectives:

- Design and build a circuit to turn on an LED using a microcontroller.
- Implement a simple program to set an output pin to a high state.
- Verify circuit functionality through simulation and breadboard testing.
- Understand the basic principles of microcontroller I/O configuration and control.

Necessary Equipment:

- Breadb oard
- Microcontroller Board (e.g., ATmega32 development board) LED (Light-Emitting Diode)
- Current-Limiting Resistor (value based on LED specifications and power supply voltage)
- Jumper Wires
- Power Supply (voltage compatible with the microcontroller and LED)
- Computer with Microcontroller Development Software (e.g., Atmel Studio)

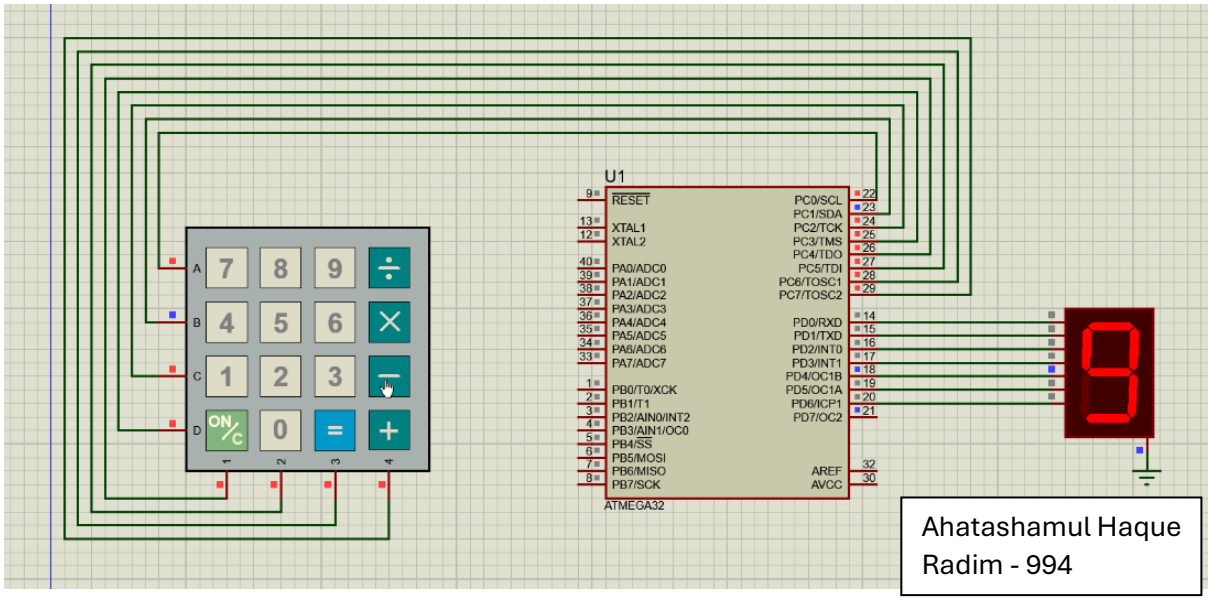
AVR Program:

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
const uint8_t segment_display[10] = {
        0b00111111, // 0
        0b00000110, // 1
        0b01011011, // 2
        0b01001111, // 3
        0b01100110, // 4
        0b01101101, // 5
        0b01111101, // 6
        0b00000111, // 7
        0b01111111, // 8
        0b01101111  // 9
};
const char keypad_mapping[4][4] = {
        {'7', '8', '9', '/'},
        {'4', '5', '6', '*'},
        {'1', '2', '3', '-'},
        {'C', '0', '=', '+'}
};
void init_peripherals() {
        DDRD = 0xFF;
        PORTD = 0x00;
```

```
	DDRC = 0x0F;
	PORTC = 0xF0;
char read_keypad() {
	for (uint8_t row = 0; row < 4; row++) {

		PORTC = ~(1 << row);
		for (uint8_t col = 0; col < 4; col++) {
			if (!(PINC & (1 << (col + 4)))) {
				_delay_ms(50);
				if (!(PINC & (1 << (col + 4)))) {
					return keypad_mapping[row][col];
				}
			}
		}
	}
	return '\0';
}
void display_digit(uint8_t digit) {
	PORTD = segment_display[digit];
}
uint8_t perform_calculation(uint8_t operand1, uint8_t operand2, char operator) {
	switch (operator) {
		case '+': return operand1 + operand2;
		case '-': return operand1 - operand2;
		case '*': return operand1 * operand2;
		case '/': return (operand2 != 0) ? operand1 / operand2 : 0;
		default: return 0;
	}
}

int main() {
	init_peripherals();

	uint8_t operand1 = 0, operand2 = 0, result = 0;
	char operator = '\0', key;

	while (1) {
		key = read_keypad();

		if (key != '\0') {
			if (key >= '0' && key <= '9') {
				if (operator == '\0') {
					operand1 = operand1 * 10 + (key - '0');
					display_digit(operand1 % 10);
					} else {
					operand2 = operand2 * 10 + (key - '0');
					display_digit(operand2 % 10);
				}
				} else if (key == '+' || key == '-' || key == '*' || key == '/') {
				operator = key;
```

```
                } else if (key == '=') {
                result = perform_calculation(operand1, operand2, operator);
                operand1 = result;
                operand2 = 0;
                operator = '\0';
                display_digit(result % 10);
                } else if (key == 'C') {
                operand1 = operand2 = result = 0;
                operator = '\0';
                PORTD = 0x00;
            }
            _delay_ms(200);
        }
    }
}
```

Proteus simulation:



The Proteus simulation should depict the microcontroller, LED, current-limiting resistor, power supply, and connecting wires. The microcontroller program code can be loaded into the simulation environment to verify functionality before building the actual circuit.

Conclusion: Arduino successfully controls the rotation of a servo motor using the Servo library, demonstrating the proper interfacing and programming to adjust the motor's position. The Proteus simulation validates the system's ability to control servo movement through PWM signals, providing a foundation for applications in robotics and automation.