Course No.: EEE4706
Course Name: Microcontroller Based System Design Lab

Lab Project

Submitted by

1. Kazi Raine Raihan
   Student ID: 180021102
2. Abdur Rahman Akib
   Student ID: 180021122
3. Ahnaf Tahmid Bin Siddique
   Student ID: 180021130
4. Minhaz Uddin Ahmed
   Student ID: 180021136

Section: A2

Date of Submission: 23.11.2022

Objective:

The main objectives of this project are –

- Construct a system using AT89S52 microcontroller to run a stepper motor according to the angle given as input.
- Proper control over the use of stepper motor
- Use of motor driver.

Introduction:

Stepper motors are characterized by the fact that they move in discrete steps. Inside a stepper motor, there are groups of coils termed “phases.”. With each excitation sequence to the phases, the stepper motor rotates one step at a time.

Stepper motors produce high torque and have low vibration at low-speeds. This makes it ideal for applications where fine positioning and control over the motor is necessary. As we increase the speed of stepper motor, the torque decreases. Stepper motors are managed by a driver, which drives the motor by sending pulses into it.

There are three main types of stepper motors -

• Permanent Magnet Stepper: The magnets used in this type of motor are permanent magnets, which interact with the magnets of the stator to rotate.

• Variable Reluctance Stepper: An electromechanical energy conversion mechanism known as a variable reluctance stepper motor transforms electrical energy into mechanical energy. It follows the reluctance principle, which states that magnetic flux always travels through a minimal reluctance channel.

• Hybrid Synchronous Stepper: An electromechanical energy conversion mechanism known as a variable reluctance stepper motor transforms electrical energy into mechanical energy. It operates on the reluctance principle, which states that magnetic flux always travels through a minimal reluctance channel.

Electrical pulses are used to drive stepper motors. The "pulse rate" is the rate of a pulse, which is expressed in pulses per second (pps). Accurate placement is made possible by the stepper motor's spin, which is proportional to the input pulses. A stepper motor excels at producing holding torque at zero speed, among other things. In comparison servo motors require higher power to provide holding torque. The quirk of a stepper motor is to divide a complete revolution into many smaller, equal sized rotations. These may be used to command the stepper motor to rotate through certain angles or degrees for practical applications. Document printers, 3D printers, image scanners, camera lenses, CNC machines and routers etc use stepper motors for fine and accurate movement. For applications that need for fast positioning over a close range, stepper motors are the best choice.

Required Components:

Item	Price
Relay	
Motor Driver*	
ESP8266(BT and WiFi)	
USBASP Programmer	
4*4 Keypad*	
ADC	
Breadboard	
IR Sensor	
LED Lights*	
7-SEG Display	
Dot Matrix Display	
1602 LCD Display*	
Humidity Sensor	
AT89S52 Microcontroller Chip*	
Buzzer*	
RTC Module	**Total = 70000BDT**

*marked are used in the project

Circuit Diagram:

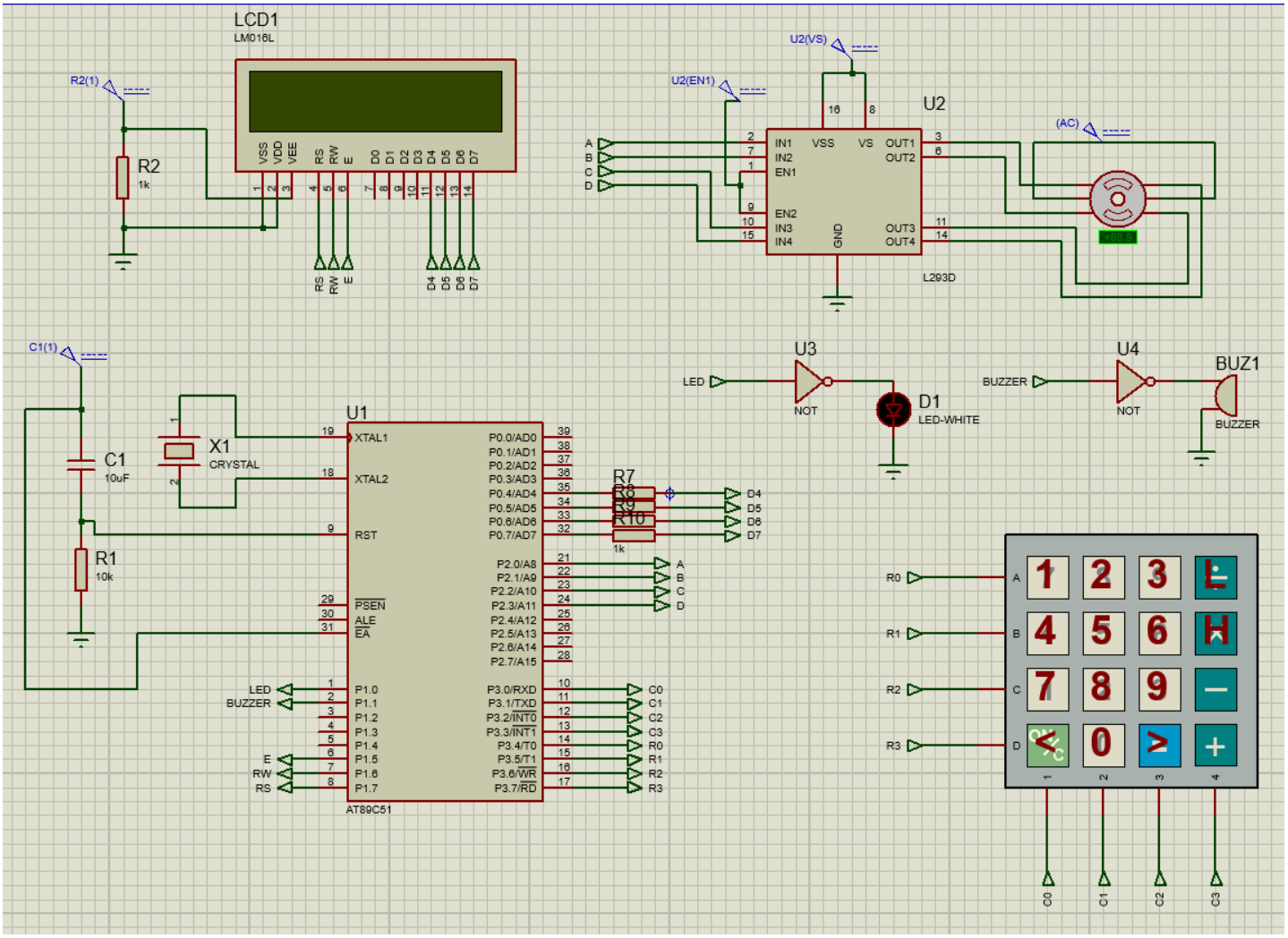


Fig: Proteus schematic diagram

Features:

Mandatory Feature:

- Proper use of keypad
  The keypad was connected to Port 3 and was used to take input from the user. When taking the angle input from the user, we limited the input to 3 digits. That is the input can be anything in the range of 000~999. These inputs were taking as ASCII characters and proper processing were needed to be done to this input to make them into BCD for further manipulation.

- Control over the use of stepper motor
  The stepper motor in this project has proper control over its operation

- Use of motor driver
  The L293D motor driver circuit included in the microcontroller project board was used to drive the stepper motor. The VCC and ground pins of the driver were used from the available power pins in the

microcontroller board. The excitation sequences of the stepper motor are provided to this motor driver circuit by the AT89S52 microcontroller.

Additional Features:

- Clockwise and couter-clockwise rotation
  After taking the angle input from the user, the program asks for the rotation direction from the user. Selecting '>' gives clockwise rotation and selecting '<' gives counter clockwise rotation.

- Dual speed mode
  Upon giving the rotation direction input, the program asks for the speed mode from the user. There exists two different speed of operation for the motor. Selecting 'H' runs the motor at high speed where the torque is lower, whereas selecting 'L' runs the motor at a lower speed but provides higher torque.

- 4 data line LCD driving
  Usually, the LCD used requires 8 data pins connected to the microcontroller for input. However, the LCD being used in the project (1602) can also operate in 4-bit mode. The inputs here are needed to be given in two separate 4 bits sequence. Modifications also need to be made to the LCD command write and data write subroutine.

- Emergency stop button
  In case of hardware issues for example if the microcontroller becomes faulty, we might need to turn off the microcontroller immediately. This provision has been made in our project where a stop button has been connected to the VCC of the motor, allowing it to be shutdown at any emergency situation.

Working Principle:

Algorithm:

1. Initiate the microcontroller and LCD
   This includes setting the appropriate resistor banks, shifting the stack pointer etc, also setting the LEDs and buzzer pins to their initial conditions. After that the LCD display was initialized with appropriate commands.

2. Show angle input message
   After initialization, the first message displayed is the enter angle prompt. This is stored inside the ROM and then printed to the LCD using appropriate commands.

3. Take angle input from the user and store the ASCII in memory
   The user is to give input from the keypad connected to port 3 and the microcontroller takes 3-digit input from the user. That is the input can be anything from (000-999). The inputs are stored in memory location 30H, 31H and 32.These inputs are given as ASCII characters and they need to be processed further for manipulated, which is done afterwards.

4. Ask for rotation direction and store it in memory
   After the angle information, the program asks the user for the rotation direction of the motor. The motor can rotate either clockwise or counter clockwise. If the user gives '>' input, the motor rotates clockwise and if the user gives '<' input, the motor rotates counterclockwise. These input is taken from

the user through keyboard and this is also stored in a memory location for further use. This information is stored in the memory location 33H

5. Ask for rotation speed and store it in memory
   Two different operating speeds were set for the motor. A high speed mode which gives lower torque and a low speed mode which gives higher torque. According to the needs of the user, the user can select 'H' from the keypad for high speed operation or 'L' from the keypad for low speed operation. These inputs are also stored in a memory location for further checking later. This information is stored in memory location 34H.

6. Convert the Angle input from ASCII digits to BCD
   The inputs given by the user are in ASCII character format. Firstly from each of them, 30H needs to be subtracted. After subtracting, the single numbers need to be converted to BCD for numerical manipulation. Say the input after subtracting 30H from them is, 3, 6, 0 and they are stored in 3 different memory location. To convert this to a single BCD number this needs to be performed –
   $$3 \times 100 + 6 \times 10 + 0 \times 1 = 360$$
   This is stored in a memory afterwards for numerical manipulation. Notice that the input can be larger than 8 bit so further processing were done accordingly.
   After processing, this BCD value is stored in memory locations 40H and 44H. We needed two memory locations because for our input, this step number can exceed 8-bits.

7. Calculating the step number
   The stepper motor being used performs a full revolution after 2048 steps. Which means that each degree of rotation requires 5.688 or approximately 5.7 steps. The BCD converted angle input that we got from previous step was first divided by 10 and then multiplied with 57. Notice that the BCD number 360 is not 8-bit and thus in the code, 16-bit division and multiplication provisions were made. Upon doing this division and multiplication operation, we get the number of steps required to move the motor to the instructed angle.
   This step number information is stored in the memory location 57H and 58H.

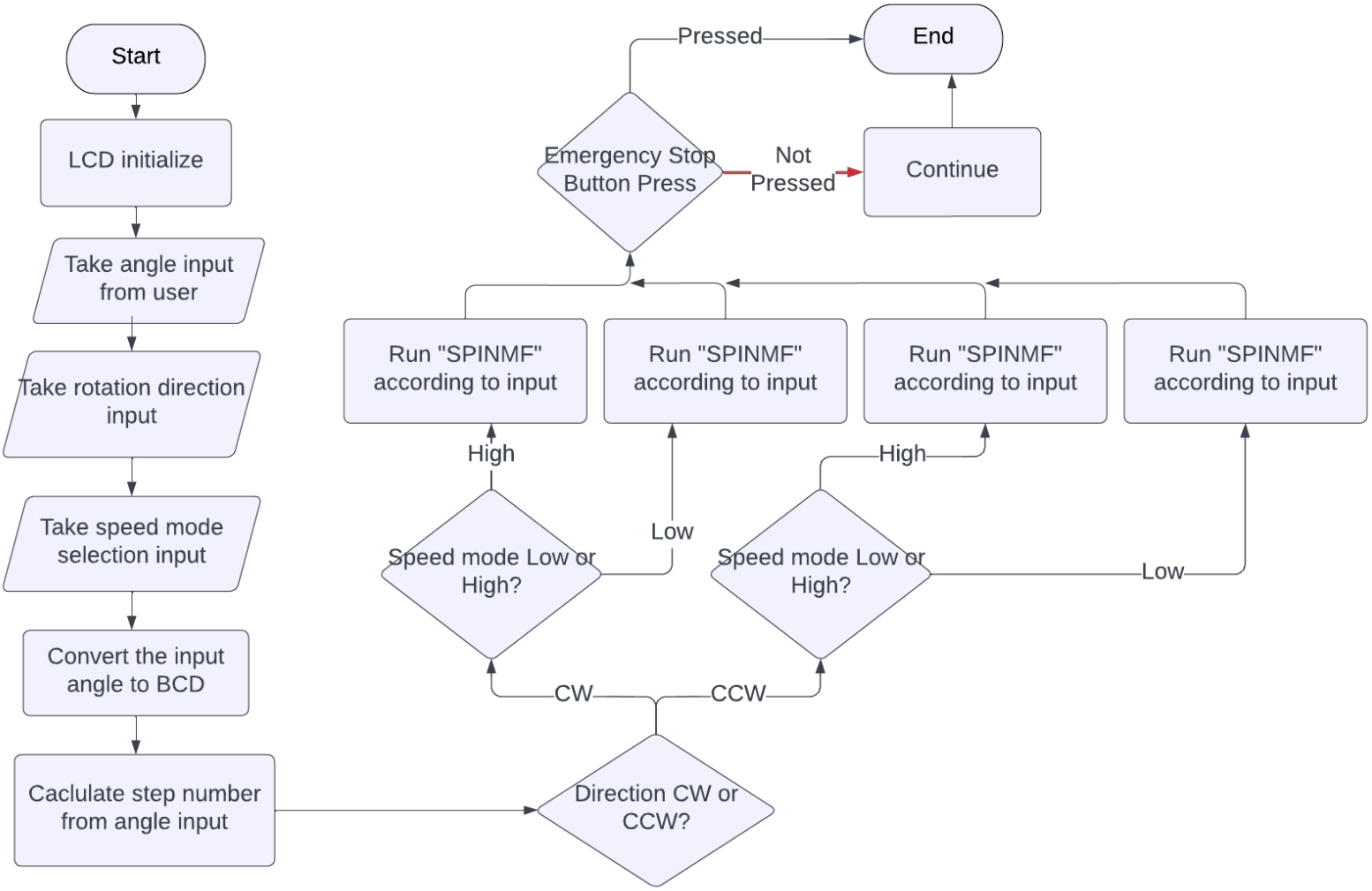
8. Spin the motor according to the step number calculated
   The step number calculated in the previous step is now used to rotate the motor. A subroutine called "SPINMF" was created which rotates the motor one step. This subroutine is looped according to the step number. In this "SPINMF" subroutine, the CW and CCW rotation and also the high and low speed operations are checked before giving excitation sequence. Thus the motor only rotates according to the input given by the user.

9. Turn on an LED during operation and beep the buzzer after finishing
   During the motor operation, an LED turns on and remains on until the motor finishes its operation. Upon finishing the operation a buzzer is turned on for a short amount of time. The LED was connected to pin 1.7 and pin 1.6, but inside the code 1.0 and 1.1 were used because of the inverted pins of port 1. This is discussed in more detail in the problems faced section.

10. Emergency stopping of motor
    In case of an emergency situation where the motor needs to be stopped, a hardware stop button is designed which can stop the motor during undesired operation. The stop button needs to be brought back to the initial state before using the motor again

11. Repeat
    Upon finishing the whole sequence once, the program again initiates from the beginning, allowing the user to do further movement of the motor.

Flowchart:


Start
LCD initialize
Take angle input from user
Take rotation direction input
Take speed mode selection input
Convert the input angle to BCD
Caclulate step number from angle input
Direction CW or CCW?
CW
CCW
Speed mode Low or High?
High
Low
Speed mode Low or High?
High
Low
Run "SPINMF" according to input
Run "SPINMF" according to input
Run "SPINMF" according to input
Run "SPINMF" according to input
Emergency Stop Button Press
Pressed
End
Not Pressed
Continue

Code:

```
      ORG 00H
      RS EQU P1.7
      RW EQU P1.6
      E EQU P1.5
      KEYPORT    EQU P3
      LCDPORT    EQU P0

      MOV SP,#70H
      MOV PSW,#00H
START:SETB P1.0             ; initially turns off the LED
      SETB P1.1             ; initially turns off the buzzer
      MOV R0,#30H           ; where the ASCII is to be stored
      MOV R5,#03            ; how many characters to take input

      ACALL LCD_INI         ; initializes the LCD

      MOV DPTR,#WELCOME1    ; moves DPTR to the location of WELCOME1
      ACALL STRPRNT         ; prints out the string
      MOV A,#0C0H
      ACALL COMNWRT         ; moves cursor to next line
      ACALL DELAY_1
      MOV DPTR,#WELCOME2    ; moves DPTR to the location of WELCOME2
      ACALL STRPRNT
      ACALL KEYPAD          ; taking input from the keypad
      ACALL BCD_C           ; converts that to BCD
      ACALL LCD_INI         ; clears the LCD
      MOV DPTR,#ROL         ; moves DPTR to the location of ROL
      CLR A
      ACALL STRPRNT         ; prints out the string
      MOV R5,#1             ; only 1 character to take input
      MOV R0,#33H           ; ASCII to be stored here
      ACALL KEYPAD          ; taking input from the keypad
      ACALL LCD_INI         ; clearing the LCD
      MOV DPTR,#SPEED       ; moves DPTR to the location of SPEED
      CLR A
      ACALL STRPRNT         ; prints out the string
      MOV A,#0C0H
      ACALL COMNWRT         ; moves cursor to next line
      ACALL DELAY_1
      MOV DPTR,#SPEED_C     ; moves DPTR to the location of SPEED_C
      CLR A
      ACALL STRPRNT         ; moves cursor to next line
      MOV R5,#1             ; only 1 input character
      MOV R0,#34H           ; ASCII storing location
      ACALL KEYPAD          ; taking input from keypad
      ACALL LCD_INI         ; clearing the LCD

      CLR A
      MOV DPTR,#ROTATING    ; moves DPTR to the location of ROTATING
      ACALL STRPRNT         ; prints out the string
      CLR P1.0              ;turns on the LED
      ACALL STEP_C          ; calculates the number of steps required to move the
                              specified angle
      ACALL SPINMF          ; spins according to the number of steps
      CLR P1.1              ; turns on buzzer
      ACALL DELAY_B         ; delay to control the duration of buzzer beep
      SETB P1.1             ; turns off buzzer
      SETB P1.0             ; turns off the LED
      ACALL LCD_INI         ; clears the LCD

      LJMP START            ; then jumps to start
```

```
KEYPAD:                                  ; subroutine to take input from keypad
      E1:   MOV   A, #0FH
            MOV   KEYPORT, A
      K1:   MOV   KEYPORT, #00001111B
            MOV   A, KEYPORT
            ANL   A, #00001111B
            CJNE  A, #00001111B, K1

      K2:   ACALL       DELAY_1
            MOV   A, KEYPORT
            ANL   A, #00001111B
            CJNE  A, #00001111B, OVER
            SJMP  K2

      OVER:       ACALL       DELAY_1
            MOV   A, KEYPORT
            ANL   A, #00001111B
            CJNE  A, #00001111B, OVER1
            SJMP  K2

      OVER1:      MOV   KEYPORT, #11101111B
            MOV   A, KEYPORT
            ANL   A, #00001111B
            CJNE  A, #00001111B, ROW_0

            MOV   KEYPORT, #11011111B
            MOV   A, KEYPORT
            ANL   A, #00001111B
            CJNE  A, #00001111B, ROW_1

            MOV   KEYPORT, #10111111B
            MOV   A, KEYPORT
            ANL   A, #00001111B
            CJNE  A, #00001111B, ROW_2

            MOV   KEYPORT, #01111111B
            MOV   A, KEYPORT
            ANL   A, #00001111B
            CJNE  A, #00001111B, ROW_3
            LJMP  K2

      ROW_0: MOV  DPTR, #KCODE0
            SJMP  FIND
      ROW_1: MOV  DPTR, #KCODE1
            SJMP  FIND
      ROW_2: MOV  DPTR, #KCODE2
            SJMP  FIND
      ROW_3: MOV  DPTR, #KCODE3

      FIND: RRC   A
            JNC   MATCH
            INC   DPTR
            SJMP  FIND

      MATCH:CLR A
            MOVC A,@A+DPTR
            ACALL DATAWRT
            ACALL DATASV
            ACALL DELAY_1
            DJNZ R5,E1

      RET
```

```
BCD_C:MOV A,30H                    ; subroutine to convert the number to BCD
      SUBB A,#30H
      MOV B,#100D
      MUL AB
      MOV 40H,A
      MOV 41H,B
      MOV A,31H
      SUBB A,#30H
      MOV B,#10D
      MUL AB
      MOV 42H,A
      MOV A,32H
      SUBB A,#30H
      MOV 43H,A
      ACALL ADDER
      RET

STEP_C:MOV R0,45H                  ; subroutine to calculate the number of steps
                                     required
      MOV R1,41H
      MOV R2,#0AH
      MOV R3,#0
      LCALL T_DIV
      MOV 55H,R4
      MOV 56H,R5
      LCALL T_MUL
      MOV 57H,R4
      MOV 58H,R5
      RET


SPINMF:MOV R1,57H                  ; subroutine to spin motor according to step
                                     number
      MOV R2,58H
      MOV R3,#0
      MOV R4,#0
      MOV 70H,#66H
LOOP_S:CLR C
      MOV A,70H
      MOV R7,33H
      CJNE R7,#'>',LEFT
      RR A
      SJMP CON
LEFT: RL A
CON:  MOV 70H,A
      MOV P2,A
      MOV A,34H
      CJNE A,#'L',D_2
      ACALL DELAY_3
      SJMP CT
D_2:  ACALL DELAY_2
CT:   MOV A,R3
      ADD A,#01
      MOV R3,A
      JNC CONT
      INC R4
      CLR C
CONT: MOV A,R3
      MOV B,R1
      CJNE A,B,LOOP_S
NEXT: MOV A,R4
      MOV B,R2
      CJNE A,B,LOOP_S
      RET
```

```
ADDER:CLR C                          ; adder subroutine used for STEP_C subroutine
      MOV A,42H
      MOV B,43H
      ADD A,B
      MOV 44H,A
      ACALL T_SUM

      RET


STRPRNT:CLR A                              ; subroutine used to print out string
      MOVC A,@A+DPTR
      JZ BR
      ACALL DATAWRT
      ACALL DELAY_1
      INC DPTR
      SJMP STRPRNT
BR:   RET

COMNWRT:                             ; LCD command subroutine
      MOV   B, A
      ANL   A, #0F0H
      MOV   LCDPORT, A
      CLR   RS
      CLR   RW
      SETB  E
      ACALL       DELAY_1
      CLR   E

      MOV   A, B
      SWAP  A
      ANL   A, #0F0H
      MOV   LCDPORT, A
      SETB  E
      ACALL       DELAY_1
      CLR   E

      MOV   A, B
      ACALL       DELAY_1
      RET


DATAWRT:                             ; LCD write subroutine
      MOV   B, A
      ANL   A, #0F0H
      MOV   LCDPORT, A
      SETB  RS
      CLR   RW
      SETB  E
      ACALL       DELAY_1
      CLR   E

      MOV   A, B
      SWAP  A
      ANL   A, #0F0H
      MOV   LCDPORT, A
      SETB  E
      ACALL DELAY_1
      CLR   E

      MOV   A, B
      ACALL       DELAY_1
      RET
```

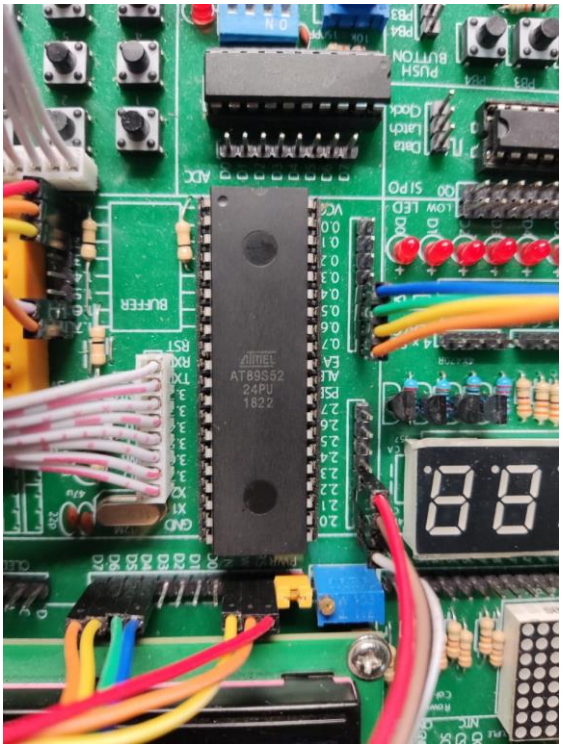
```
DATASV:MOV @R0,A                    ; subroutine to save data to RAM location
       INC R0
       RET

                                    ;LCD initialization subroutine
LCD_INI:MOV A, #02H           ; 4 bit mode
      ACALL COMNWRT

      MOV A,#28H                    ; 4 bit mode initialization
      ACALL COMNWRT
      ACALL DELAY_1
      MOV A,#0CH
      ACALL COMNWRT                 ; display on, cursor blinking
      ACALL DELAY_1
      MOV A,#06H                    ; move cursor to the right
      ACALL COMNWRT
      ACALL DELAY_1
      MOV A,#01H                    ; clear screen
      ACALL COMNWRT
      ACALL DELAY_1
      MOV A,#80H                    ; force cursor to the begining of 1st line
      ACALL COMNWRT
      ACALL DELAY_1
      RET

T_DIV:                              ; 16 bit division subroutine
      CLR C
      MOV R4,#00h
      MOV R5,#00h
      MOV B,#00h
div1:
      INC B
      MOV A,R2
      RLC A
      MOV R2,A
      MOV A,R3
      RLC A
      MOV R3,A
      JNC div1
div2:
      MOV A,R3
      RRC A
      MOV R3,A
      MOV A,R2
      RRC A
      MOV R2,A
      CLR C
      MOV 07h,R1
      MOV 06h,R0
      MOV A,R0
      SUBB A,R2
      MOV R0,A
      MOV A,R1
      SUBB A,R3
      MOV R1,A
      JNC div3
      MOV R1,07h
      MOV R0,06h
div3:
      CPL C
      MOV A,R4
      RLC A
      MOV R4,A
      MOV A,R5
      RLC A
      MOV R5,A
```

```
      DJNZ B,div2
      MOV R3,05H
      MOV R2,04H
      RET

T_MUL:MOV R0,56H       ;MSB1       ; 16 bit multiplication subroutine
      MOV R1,#0        ;MSB2
      MOV R2,55H       ;LSB1
      MOV R3,#57       ;LSB2
      MOV A,R2
      MOV B,R3
      MUL AB
      MOV R4,A
      MOV R5,B
      MOV A,R2
      MOV B,R1
      MUL AB
      MOV R6,B
      ADDC A,R5
      MOV R5,A
      MOV A,R1
      MOV B,R4
      MUL AB
      ADDC A,R5
      MOV R5,A
      MOV A,B
      ADDC A,R6
      MOV R6,A
      MOV A,R1
      MOV B,R2
      MUL AB
      ADDC A,R6
      MOV R6,A
      MOV A,B
      ADDC A,#00H
      MOV R7,A
      RET

T_SUM:CLR C                  ; subroutine used along with BCD_C subroutine
      MOV A,40H
      ADD A,44H
      MOV 45H,A
      JC INCRE
      RET
INCRE:INC 41H
      RET

DELAY_1:SETB PSW.3
      MOV   R3, #50
Y0:   MOV   R1, #255
Y1:   DJNZ  R1, Y1
      DJNZ  R3, Y0
      CLR   PSW.3
      RET

DELAY_2:MOV R6, #010H
L0:   MOV   R5, #0FFH
L1:   DJNZ  R5, L1
      DJNZ  R6, L0
      RET

DELAY_3:MOV R6, #028H
N0:   MOV   R5, #0FFH
N1:   DJNZ  R5, N1
      DJNZ  R6, N0
      RET
```

```
DELAY_B:MOV R7, #4H
pp00: MOV   R6, #0FFH
pp0:  MOV   R5, #0FFH
pp1:  DJNZ  R5, pp1
      DJNZ  R6, pp0
      DJNZ  R7, pp00
      RET

CLR_LCD:MOV A,#01H                    ; subroutine to clear LCD
      ACALL COMNWRT
      ACALL DELAY_1
      RET

KCODE0:     DB    '1','2','3','L'                ;ROW 0
KCODE1:     DB    '4','5','6','H'                ;ROW 1
KCODE2:     DB    '7','8','9','Y'                ;ROW 2
KCODE3:     DB    '<','0','>','N'                ;ROW 3
WELCOME1: DB      '3 DIGIT INPUT',0              ;first line of 1st MSG to print
WELCOME2: DB      '(000-999):',0                 ;second line of 1st MSG to print
ROL: DB           'ROTATE CW/CCW? ',0            ;second MSG to print
ROTATING: DB      'ROTATING...',0                ;third MSG to print
SPEED: DB   'SELECT SPEED:',0                    ;1st line of forth MSG to print
SPEED_C: DB 'L:LOW H:HIGH ~ ',0                  ;2nd line of forth MSG to print

END
```

Hardware Implementation:

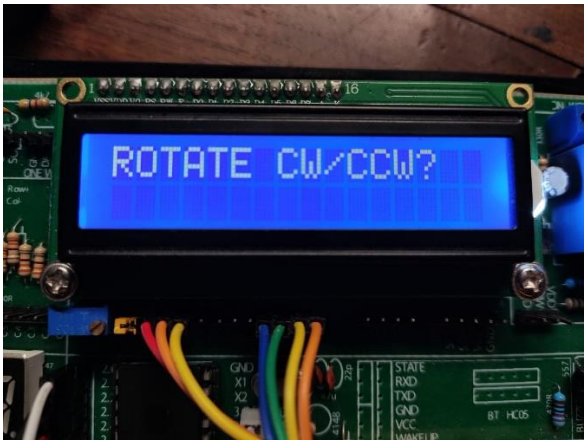
Hardware Setup:



Implementation:

1. **Angle input:**

2. **Rotation direction:**



3. **Speed control:**

Problems Faced:

1. Firstly, we found troubles limiting our calculations to 8-bit operation. Say our input was 360°, which corresponds to the binary –

   $$\underline{1}\ \underline{0110}\ \underline{1000}$$

   We cannot store this value using a single register or memory location. Hence for storing, multiple memory locations were used. Here in our project, input from range 000~999 works.

2. For our project, the stepper motor used was 28BYJ-48. For a full revolution, that is 360°, this particular motor requires 2048 steps. So, for every degree, we need to give (2048/360) = 5.688 steps or approximately 5.7 steps. As we cannot to multiply numbers with radix points in the 8052 microcontroller being used, we can either first multiply with 57 and then divide by 10, or we can first divide by 10 and then multiply with 57. Any one of them can be used because our program has both 16-bit multiplication and division provision included. For our project, we used the later mentioned approach.

3. The factor used for finding out the number of steps required was (2048/360) = 5.688. For our ease of calculation, we used 5.7 to find the number of steps. This will obviously introduce some round-off error, that is the movement of the stepper motor won't be fully accurate. But to solve this we would need to use larger multiplication factor like 5.68 which would mean dividing by 100 and multiplying with 568. This would easily go over 16-bit operation for our particular inputs. Thus, for simplicity we limited ourselves to 16-bit calculations and accepted the round off error. This is not significant enough to be observable in our hardware setup.

4. For our LED and buzzer connection, the connections in the hardware setup were all active low. Whereas, in the software simulation of proteus, the inputs were all active high. So, in proteus we use used a logical NOT gate to make the same program run both in software and hardware setup.
5. In our hardware setup, port 1 had a buffer connected to it. Because of this that port could only be used as an output and not input. So proper steps were taken to only connect devices that need output for operation in port 1.

6. In our hardware setup, the connection of port 1 was reversed. For instance, the pin 1.7 was actually pin 1.0. Because of this, devices connected to that port had to be also programmed according to the reversed pin connection notation.

Conclusion:

In this project, we operated a stepper motor using AT89S52 microcontroller along with this we incorporated additional features to the project. We interfaced with the LCD screen and Keypad for input output of the device. The stepper motor has full freedom to rotate from 0 degree to 999 degrees. Also the rotation of the motor can be predefined by the user also the rotation speed is also controllable. In this project we tried to give the user as freedom and controllability as possible and make the stepper motor operational for any kind of use.