Higher Technological Institute

Electrical Engineering Department

Stepper Motor Angle Control System using Keypad and LCD Interface

By:

Name	ID
Abdelrahman Ehab Salem	20200544
Abdelrahman Mohamed Gabr	20200552
Omar Saad Elsayed	20210474
Ahmed Sabry Soror	20210459
Mohamed Ayman Hosny	20180666
Ahmed Sherif Lotfy	20210594

Supervised by: **A.Prof. Mohamed Torad**

Department of Electrical Engineering, Higher Technological Institute 10th of Ramadan City. Egypt

December 2025

Abstract

This report details the design and implementation of a **Stepper Motor** Angle Control System utilizing a **Keypad** and a Liquid Crystal Display (**LCD**) interface. The primary objective was to create user-friendly system capable of achieving precise, user-defined angular positioning of a stepper motor.

The system architecture employs a microcontroller to interpret digital input from the Keypad, where the user enters the desired angle in degrees. The microcontroller translates this angle into the corresponding number of steps and pulse sequence required to drive the stepper motor via a motor circuit.

Concurrently, the LCD module provides real-time feedback to the user, displaying both the entered angle and the motor's current positional status. Experimental results confirm that the system successfully achieves accurate and repeatable rotational control, validating its effectiveness as a reliable solution for applications demanding finite and controlled angular movements, such as automated camera platforms

Acknowledgement

First and foremost, we express our deepest gratitude to our families, whose unwavering love, patience, and support have been the cornerstone of our journey. Your belief in us has provided the strength and determination needed to overcome challenges and achieve this milestone.

Additionally, we wish to express our profound gratitude to Dr. Mohamed Torad. His valuable information and guidance were pivotal in directing our efforts and ensuring the project's success. We are also deeply grateful to all our professors at Higher Technological Institute, whose guidance and mentorship were instrumental in the successful completion of this project.

Your inspiration and encouragement have been invaluable in our pursuit of greater knowledge and experience. We hold the work of the engineers in our department in the highest regard and extend our sincere thanks to all of them for their daily contributions. Your dedication and professionalism have been a constant source of motivation for us. To everyone who supported us, your exceptional supervisory efforts have significantly enriched our learning experience.

We deeply appreciate your commitment to ensuring our understanding of the significance of this project. Thank you for your unwavering support and encouragement.

Table of Contents

List of figures

Abbreviations

LCD	Liquid Crystal Display
KPD	Keypad
AVR	Alf and Vegard's RISC processor (The ATmega32 architecture)
ATmega32	Atmel Mega Microcontroller with 32KB Flash Memory
IC	Integrated Circuit (e.g., the ULN2003A)
ULN2003A	Darlington Transistor Array (High-current motor driver)
I/O	Input/Output
DDR	Data Direction Register (used in code to set pins as input/output)
PORT	Data Register (used in code to output values to pins)
PIN	Port Input Pins Register (used in code to read keypad inputs)
F_CPU	Frequency of the Central Processing Unit
ADC	Analog-to-Digital Converter (Pins PA0-PA7 on the ATmega32)
RS	Register Select (LCD control pin)
RW	Read/Write (LCD control pin)
EN	Enable (LCD control pin)
ASCII	American Standard Code for Information Interchange

1. Problem Statement

Manual and analog methods for controlling rotational devices often lack the necessary precision and repeatability required for modern automated systems. Specifically, achieving a precise, digitally specified angular position for a stepper motor requirement for many robotics and laboratory applications—usually involves complex or expensive proprietary control hardware. A gap exists for a cost-effective, easily programmable, and interactive system that allows a user to input a desired motor angle directly and receive clear status feedback, without relying on complex external programming tools for every adjustment.

1.2 Project Objectives

The primary objectives of this project were to design, construct, and validate a Stepper Motor Angle Control System that fulfills the following criteria:

1.2.1 User Input

Implement a Keypad interface to allow the user to input the target angle (in degrees) directly and easily.

1.2.2 Precision Control

Develop the necessary software and hardware interface to accurately translate the input angle into the correct step sequence for the stepper motor.

1.2.3 Visual Feedback

Integrate an LCD module to display the user's angle input and the real-time status of the motor’s movement.

1.2.4 Reliability

Ensure the system consistently and accurately moves the motor to the specified position with high repeatability.

2. Hardware Architecture and Proteus Analysis

The Proteus schematic visualizes an integrated system where the **ATmega32[1]** microcontroller serves as the central processing unit, coordinating inputs from a keypad and outputting data to an LCD and a stepper motor.

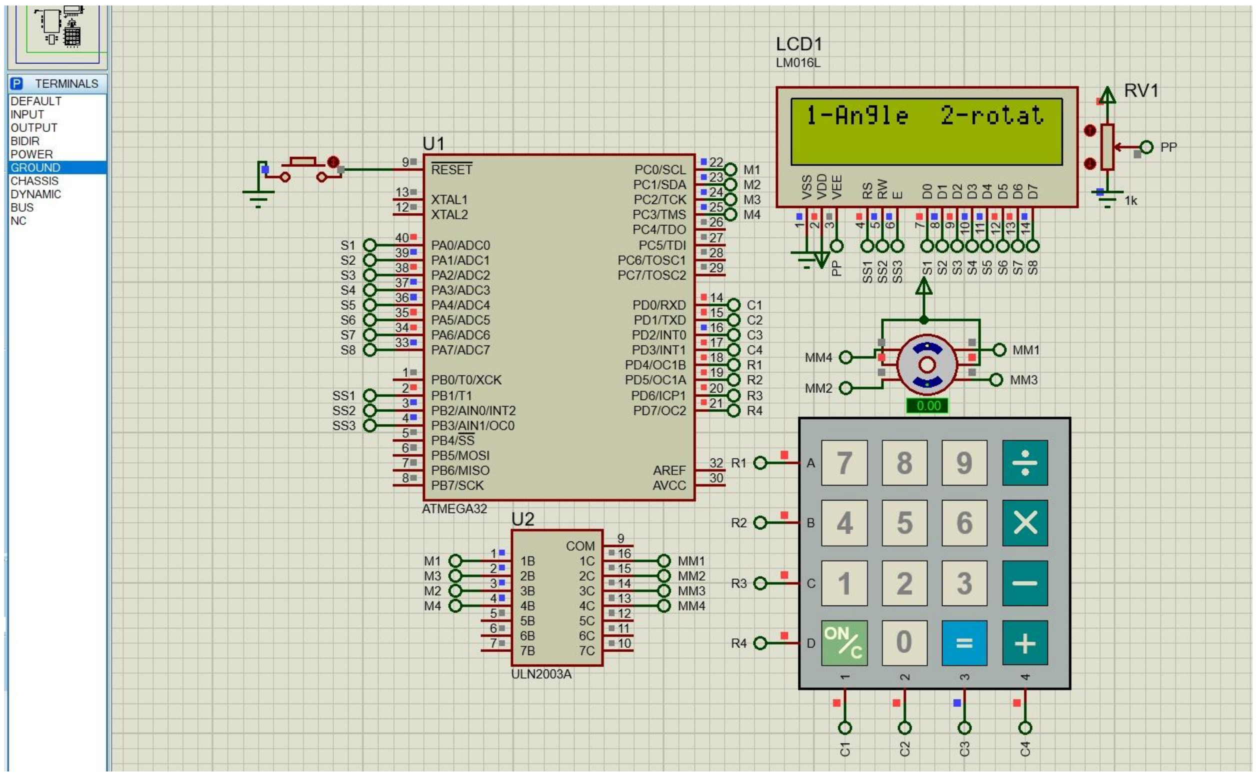

Figure 1: Interface Design of Stepper Motor with 4x4 Keypad and 16x2 LCD

2.1 The Microcontroller (U1: ATmega32)

The ATmega32 is the system's "brain."

- **Port A (PA0-PA7):** Configured as output (DDRA = 0xFF) to send 8-bit data to the LCD.
- **Port B (PB1-PB3):** Used for LCD control signals: Register Select (**RS**), Read/Write (**RW**), and Enable (**EN**).
- **Port C (PC0-PC3):** Configured as output (STEPPER_DIR |= 0x0F) to provide the step sequences to the motor driver.
- **Port D (PD0-PD7):** Interfaced with the **4x4 Keypad** using a row/column scanning method.

2.2 User Interface Components

- **LCD1 (LM016L):** A 16x2 character display that provides real-time feedback. In the image, generated by main() loop:
- 1-Ang1e 2-rotat.
- **4x4 Matrix Keypad:** This allows the user to input numerical values for angles or rotations. code uses a mapping matrix map [4] [4] to translate electrical contact at a specific row and column into a character like '7', '8', or 'C' (Clear).

2.3 Drive Circuitry (U2: ULN2003A)

The **ULN2003A** is a high-voltage, high-current Darlington transistor array. It is essential because the ATmega32 pins cannot provide enough current to drive the motor coils directly.

- **Input (Pins 1-4):** Receives the step sequence (M1–M4) from Port C.
- **Output (Pins 13-16):** Switches the motor coils to ground, allowing current to flow through the stepper motor windings (MM1–MM4).

---

3. Software Logic & Precision Control

The system achieves your project objectives through specific software functions:

3.1 Translating Angle to Steps

The core logic for precision is found in your Stepper_Go_To_Angle function. It uses the following mathematical relationship to convert the user's input into physical motion:

$$\text{total\_steps} = \frac{|\text{target\_angle}| \times \text{STEPS\_PER\_REV}}{360}$$

- **Steps Per Revolution**: this is defined as 64.
- **Sequence:** we used a full_step_sequence (0b1001, 0b0011, etc.) to ensure high torque during rotation.

3.2 System Workflow

3.2.1 Menu Selection

The user selects 1 for Angle or 2 for Rotation count using the keypad.

3.2.2 Input Processing

The Get_Number_Input function handles character-to-integer conversion, allowing for negative numbers (reverse rotation) and clearing mistakes with the 'C' key.

3.2.3 Actuation

The for loop in the stepper logic iterates through the calculated number of steps, updating the STEPPER_PORT every 20ms to provide a smooth, visible rotation in the simulation.

3.2.4 Feedback

The LCD updates status from "Moving..." to "Done!" to satisfy the **Visual Feedback** objective.

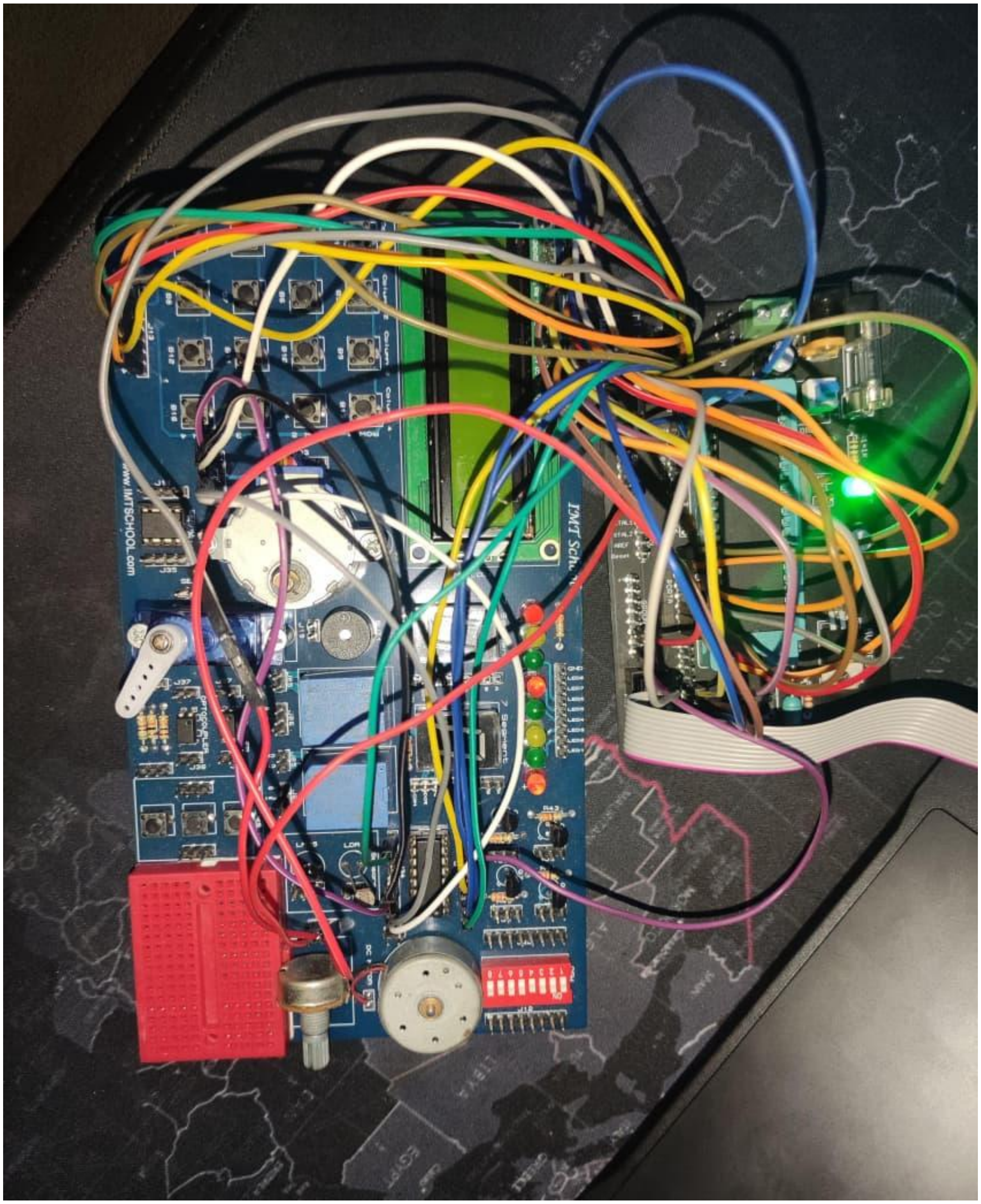

Figure 2: Physical Prototype of the Stepper Motor Control System

As shown in Figure2, here is a brief description of what is seen in the photo to prove it matches your simulation:

- **Microcontroller Board:** The central processing unit executing your C-code logic.
- **LCD Module:** Currently displaying the system status.
- **Stepper Motor:** The actuator used for precise angular displacement.
- **Matrix Keypad:** The push-button interface for digit entry.
- **ULN2003 Driver IC:** Visible on board to handle the motor's current requirements.

C Code

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <math.h>

typedef unsigned char u8;
typedef signed char s8;
typedef signed long s32;

// --- Macros ---
#define SET_BIT(VAR,BITNO) (VAR) |=  (1 << (BITNO))
#define CLR_BIT(VAR,BITNO) (VAR) &= ~(1 << (BITNO))
#define GET_BIT(VAR,BITNO) (((VAR) >> (BITNO)) & 0x01)

// --- CONFIG ---
#define LCD_DATA_PORT    PORTA
#define LCD_DATA_DIR     DDRA
#define LCD_CTRL_PORT    PORTB
#define LCD_CTRL_DIR     DDRB
#define LCD_RS 1
#define LCD_RW 2
#define LCD_EN 3

#define KPD_PORT        PORTD
#define KPD_PIN         PIND
#define KPD_DIR         DDRD
#define NOT_PRESSED     0xFF

#define STEPPER_PORT     PORTC
```

```c
#define STEPPER_DIR     DDRC
#define STEPS_PER_REV    64

/**************** 1. LCD & KEYPAD FUNCTIONS ****************/
// (دوال الـ LCD والـ Keypad الأساسية بدون تغيير لضمان الاستقرار)
void LCD_SendCommand(u8 cmd) {
    CLR_BIT(LCD_CTRL_PORT, LCD_RS); CLR_BIT(LCD_CTRL_PORT, LCD_RW);
    LCD_DATA_PORT = cmd; SET_BIT(LCD_CTRL_PORT, LCD_EN);
    _delay_ms(1); CLR_BIT(LCD_CTRL_PORT, LCD_EN); _delay_ms(2);
}
void LCD_SendData(u8 data) {
    SET_BIT(LCD_CTRL_PORT, LCD_RS); CLR_BIT(LCD_CTRL_PORT, LCD_RW);
    LCD_DATA_PORT = data; SET_BIT(LCD_CTRL_PORT, LCD_EN);
    _delay_ms(1); CLR_BIT(LCD_CTRL_PORT, LCD_EN); _delay_ms(2);
}
void LCD_Init(void) {
    LCD_DATA_DIR = 0xFF; LCD_CTRL_DIR |= (1<<LCD_RS)|(1<<LCD_RW)|(1<<LCD_EN);
    _delay_ms(20); LCD_SendCommand(0x38); LCD_SendCommand(0x0C); LCD_SendCommand(0x01);
}
void LCD_PrintString(char *str) { while(*str) LCD_SendData(*str++); }
void LCD_Clear(void) { LCD_SendCommand(0x01); _delay_ms(2); }
void LCD_GoTo(u8 row, u8 col) { LCD_SendCommand((row==0)? (0x80+col):(0xC0+col)); }
void LCD_WriteNum(s32 num) { char buffer[16]; itoa(num, buffer, 10); LCD_PrintString(buffer); }

u8 KPD_GetPressed(void) {
    u8 col, row; KPD_DIR = 0x0F; KPD_PORT = 0xFF;
    for(col=0; col<4; col++) {
        CLR_BIT(KPD_PORT, col);
        for(row=0; row<4; row++) {
            if(GET_BIT(KPD_PIN, row+4) == 0) {
                u8 map[4][4] = {{'7','8','9','/'},{'4','5','6','*'},{'1','2','3','-'},{'C','0','=','+'}};
                u8 key = map[row][col];
                while(GET_BIT(KPD_PIN, row+4) == 0); _delay_ms(20); return key;
            }
        }
        SET_BIT(KPD_PORT, col);
    }
    return NOT_PRESSED;
}

/**************** 2. PRECISE STEPPER LOGIC (لم يتم التعديل عليه) ****************/
const u8 full_step_sequence[4] = { 0b1001, 0b0011, 0b0110, 0b1100};

void Stepper_Go_To_Angle(s32 target_angle) {
    float steps_needed = (fabs(target_angle) * (float)STEPS_PER_REV) / 360.0;
    s32 total_steps = (s32)(steps_needed + 0.5);
    static s8 current_idx = 0;
    u8 direction = (target_angle > 0) ? 1 : 0;
```

```
  for (s32 i = 0; i < total_steps; i++) {
    if (direction == 1) current_idx++; else current_idx--;
    if (current_idx > 3) current_idx = 0;
    if (current_idx < 0) current_idx = 3;
    STEPPER_PORT = (STEPPER_PORT & 0xF0) | full_step_sequence[current_idx];
    _delay_ms(20);
  }
  STEPPER_PORT &= 0xF0;
}

/**************** 3. USER INTERFACE LOGIC ****************/
s32 Get_Number_Input(char* prompt) {
  s32 val = 0; u8 key; s8 sign = 1;
  LCD_Clear(); LCD_PrintString(prompt); LCD_GoTo(1,0);
  while(1) {
    key = KPD_GetPressed();
    if(key != NOT_PRESSED) {
      if(key == '=') break;
      if(key == '-') { sign = -1; LCD_SendData('-'); }
      else if(key >= '0' && key <= '9') { LCD_SendData(key); val = val * 10 + (key - '0'); }
      else if(key == 'C') { val = 0; sign = 1; LCD_Clear(); LCD_PrintString(prompt); LCD_GoTo(1,0); }
    }
  }
  return val * sign;
}

/**************** 4. MAIN PROGRAM ****************/
int main(void) {
  LCD_Init();
  STEPPER_DIR |= 0x0F;

  // --- Welcome Screen ---
  LCD_Clear();
  LCD_GoTo(0, 4); // توسيط الكلمة
  LCD_PrintString("Welcome");
  _delay_ms(100); // تظهر لمدة ثانيتين

  while(1) {
    LCD_Clear();
    LCD_PrintString("1-Angle  2-rotate");

    u8 choice = NOT_PRESSED;
    while(choice == NOT_PRESSED) { choice = KPD_GetPressed(); }

    if (choice == '1') {
      s32 angle = Get_Number_Input("Enter Angle:");
      LCD_Clear(); LCD_PrintString("Moving...");
```

```
         Stepper_Go_To_Angle(angle);
      }
      else if (choice == '2') {
         s32 revs = Get_Number_Input("Enter rot num:");
         s32 angle_from_revs = revs * 360; // تحويل اللفات لزوايا
         LCD_Clear(); LCD_PrintString("Moving...");
         Stepper_Go_To_Angle(angle_from_revs);
      }

      LCD_Clear();
      LCD_PrintString("Done!");
      _delay_ms(100);
   }
}
```

Assembly Code

```
.include "m32def.inc"

.equ F_CPU = 16000000

; --------- PINS ----------
.equ LCD_RS = 1          ; PB1
.equ LCD_RW = 2          ; PB2
.equ LCD_EN = 3          ; PB3

.equ NOT_PRESSED = 0xFF
.equ STEPS_PER_REV = 64

; --------- REGISTERS ----------
.def temp   = r16
.def temp2  = r17
.def key    = r18
.def col    = r19
.def row    = r20
.def stepIx = r21

; 16-bit work
.def aL     = r22       ; angle/value low
.def aH     = r23       ; angle/value high
.def sL     = r24       ; steps low
.def sH     = r25       ; steps high
```

```
; scratch for division
.def dL     = r26
.def dH     = r27
.def qL     = r28
.def qH     = r29
.def rL     = r30
.def rH     = r31

; ---------------------------------------------------------
.org 0x0000
    rjmp RESET

; ============================================================
;                    FLASH STRINGS
; ============================================================
; Strings stored in program memory (Z points to them)
S_WELCOME:     .db "Welcome",0
S_MENU:        .db "1-Angle  2-rot",0
S_ANG_PROMPT:  .db "Enter Angle:",0
S_ROT_PROMPT:  .db "Enter rot num:",0
S_MOVING:      .db "Moving...",0
S_DONE:        .db "Done!",0

; Keypad map [row][col]
; Rows: PD4..PD7, Cols: PD0..PD3
; 7 8 9 /
; 4 5 6 *
; 1 2 3 -
; C 0 = +
KMAP:
    .db '7','8','9','/'
    .db '4','5','6','*'
    .db '1','2','3','-'
    .db 'C','0','=','+'

; Stepper full-step sequence (lower nibble)
STEPSEQ:
    .db 0b1001, 0b0011, 0b0110, 0b1100

; ============================================================
;                      RESET / MAIN
; ============================================================
RESET:
    ; Stack init
    ldi temp, high(RAMEND)
    out SPH, temp
    ldi temp, low(RAMEND)
    out SPL, temp
```

```
   ; LCD data PORTA output
   ldi temp, 0xFF
   out DDRA, temp

   ; LCD control pins PB1,PB2,PB3 output
   ldi temp, (1<<DDB1)|(1<<DDB2)|(1<<DDB3)
   out DDRB, temp

   ; Stepper PORTC lower nibble output
   ldi temp, 0x0F
   out DDRC, temp
   ; clear stepper outputs
   in temp, PORTC
   andi temp, 0xF0
   out PORTC, temp

   ; Keypad: PD0..PD3 output, PD4..PD7 input + pull-up
   ldi temp, 0x0F
   out DDRD, temp
   ldi temp, 0xFF
   out PORTD, temp

   ; step index = 0
   clr stepIx

   rcall LCD_INIT

   ; Welcome
   rcall LCD_CLEAR
   ldi temp, 0          ; row0 col4
   ldi temp2, 4
   rcall LCD_GOTO
   ldi ZH, high(S_WELCOME<<1)
   ldi ZL, low(S_WELCOME<<1)
   rcall LCD_PRINT_P
   rcall DELAY_2S

MAIN_LOOP:
   rcall LCD_CLEAR
   ldi ZH, high(S_MENU<<1)
   ldi ZL, low(S_MENU<<1)
   rcall LCD_PRINT_P

   ; wait choice
WAIT_CH:
   rcall KPD_GET
   cpi key, NOT_PRESSED
```

```
    breq WAIT_CH

    cpi key, '1'
    breq DO_ANGLE
    cpi key, '2'
    breq DO_ROT
    rjmp MAIN_LOOP

DO_ANGLE:
    ldi ZH, high(S_ANG_PROMPT<<1)
    ldi ZL, low(S_ANG_PROMPT<<1)
    rcall GET_NUMBER_INPUT     ; returns value in aH:aL (signed)
    ; compute steps from angle -> sH:sL
    rcall ANGLE_TO_STEPS       ; uses aH:aL -> sH:sL, direction by sign
    rjmp EXEC_MOVE

DO_ROT:
    ldi ZH, high(S_ROT_PROMPT<<1)
    ldi ZL, low(S_ROT_PROMPT<<1)
    rcall GET_NUMBER_INPUT     ; revs in aH:aL (signed)
    ; steps = abs(revs) * 64
    rcall REVS_TO_STEPS
    rjmp EXEC_MOVE

EXEC_MOVE:
    rcall LCD_CLEAR
    ldi ZH, high(S_MOVING<<1)
    ldi ZL, low(S_MOVING<<1)
    rcall LCD_PRINT_P

    ; if steps == 0 -> skip stepping
    mov temp, sL
    or  temp, sH
    breq AFTER_MOVE

    ; direction flag in temp2: 1 = CW (positive), 0 = CCW (negative)
    rcall STEPPER_RUN

AFTER_MOVE:
    rcall LCD_CLEAR
    ldi ZH, high(S_DONE<<1)
    ldi ZL, low(S_DONE<<1)
    rcall LCD_PRINT_P
    rcall DELAY_100MS
    rjmp MAIN_LOOP


; ============================================================
;                    LCD ROUTINES (8-bit)
; ============================================================

LCD_INIT:
    rcall DELAY_20MS
    ldi temp, 0x38
    rcall LCD_CMD
    ldi temp, 0x0C
    rcall LCD_CMD
    ldi temp, 0x01
    rcall LCD_CMD
    rcall DELAY_2MS
    ret

LCD_CLEAR:
    ldi temp, 0x01
    rcall LCD_CMD
    rcall DELAY_2MS
    ret

; temp=row (0/1), temp2=col
LCD_GOTO:
    cpi temp, 0
    breq _ROW0
    ; row1 -> 0xC0 + col
    ldi rL, 0xC0
    add rL, temp2
    mov temp, rL
    rcall LCD_CMD
    ret
_ROW0:
    ldi rL, 0x80
    add rL, temp2
    mov temp, rL
    rcall LCD_CMD
    ret

; Send command in temp
LCD_CMD:
    cbi PORTB, LCD_RS
    cbi PORTB, LCD_RW
    out PORTA, temp
    sbi PORTB, LCD_EN
    rcall DELAY_1MS
    cbi PORTB, LCD_EN
    rcall DELAY_2MS
    ret
```

```
; Send data in temp
LCD_DATA:
    sbi PORTB, LCD_RS
    cbi PORTB, LCD_RW
    out PORTA, temp
    sbi PORTB, LCD_EN
    rcall DELAY_1MS
    cbi PORTB, LCD_EN
    rcall DELAY_2MS
    ret

; Print 0-terminated string from program memory at Z
LCD_PRINT_P:
    lpm temp, Z+
    cpi temp, 0
    breq _PR_DONE
    rcall LCD_DATA
    rjmp LCD_PRINT_P
_PR_DONE:
    ret


; ============================================================
;                    KEYPAD ROUTINES (4x4)
; ============================================================
; Returns:
;   key = ASCII key or NOT_PRESSED
KPD_GET:
    ; Ensure cols outputs, rows inputs, pull-ups enabled
    ldi temp, 0x0F
    out DDRD, temp
    ldi temp, 0xFF
    out PORTD, temp

    clr col

KPD_COL_LOOP:
    cpi col, 4
    brge KPD_NONE

    ; Drive one column low: PD(col)=0
    ; Start all high then clear one bit
    ldi temp, 0xFF
    out PORTD, temp
    ; clear bit col
    mov temp, col
    ldi temp2, 1
    ; temp2 = (1<<col)
    tst temp
    breq _MASK_OK
_MASK_SHIFT:
    lsl temp2
    dec temp
    brne _MASK_SHIFT
_MASK_OK:
    com temp2              ; ~mask
    in  temp, PORTD
    and temp, temp2
    out PORTD, temp

    ; Read rows PD4..PD7 (active low)
    in temp, PIND
    andi temp, 0xF0
    cpi temp, 0xF0
    breq KPD_NEXT_COL

    ; Determine row index (0..3)
    ; check PD4 then PD5 then PD6 then PD7
    clr row
    sbis PIND, 4
    rjmp KPD_FOUND
    inc row
    sbis PIND, 5
    rjmp KPD_FOUND
    inc row
    sbis PIND, 6
    rjmp KPD_FOUND
    inc row
    sbis PIND, 7
    rjmp KPD_FOUND
    rjmp KPD_NEXT_COL

KPD_FOUND:
    ; key = KMAP[row*4 + col]
    ldi ZH, high(KMAP<<1)
    ldi ZL, low(KMAP<<1)
    mov temp, row
    lsl temp
    lsl temp               ; temp = row*4
    add ZL, temp
    adc ZH, __zero_reg__
    add ZL, col
    adc ZH, __zero_reg__
    lpm key, Z

    ; Wait release (simple debounce)
```

```
KPD_WAIT_REL:
   in temp, PIND
   andi temp, 0xF0
   cpi temp, 0xF0
   brne KPD_WAIT_REL
   rcall DELAY_20MS
   ret

KPD_NEXT_COL:
   inc col
   rjmp KPD_COL_LOOP

KPD_NONE:
   ldi key, NOT_PRESSED
   ret


; ============================================================
;              GET NUMBER INPUT (signed 16-bit)
; ============================================================
; Input:
;   Z -> prompt string in flash
; Behavior:
;   shows prompt on row0, user types on row1
; Keys:
;   digits -> build value
;   '-' -> negative sign (only once)
;   'C' -> clear
;   '=' -> finish
; Output:
;   aH:aL = signed value
; Side:
;   temp2 used as sign flag: 1 positive, 0 negative (internal)
GET_NUMBER_INPUT:
   ; sign = +1 (temp2=1), value=0
   clr aL
   clr aH
   ldi temp2, 1

   rcall LCD_CLEAR
   rcall LCD_PRINT_P
   ldi temp, 1
   ldi temp, 1          ; row=1
   ldi temp2, 0         ; col=0
   rcall LCD_GOTO

GIN_LOOP:
   rcall KPD_GET
```

```
cpi key, NOT_PRESSED
breq GIN_LOOP

cpi key, '='
breq GIN_DONE

cpi key, 'C'
breq GIN_CLEAR

cpi key, '-'
breq GIN_NEG

; digit?
cpi key, '0'
brlo GIN_LOOP
cpi key, '9'
brhi GIN_LOOP

; print digit
mov temp, key
rcall LCD_DATA

; value = value*10 + (key-'0')
subi key, '0'
; multiply a by 10: a*10 = a*8 + a*2
mov dL, aL
mov dH, aH

; a*2 -> (aH:aL) <<=1
lsl aL
rol aH

; a*8 -> (dH:dL) <<=3
lsl dL
rol dH
lsl dL
rol dH
lsl dL
rol dH

; add (a*2) + (a*8) => now a = a*10
add aL, dL
adc aH, dH

; add digit
add aL, key
adc aH, __zero_reg__
rjmp GIN_LOOP
```

```
GIN_NEG:
    ; set negative sign once
    cpi temp2, 0
    breq GIN_LOOP
    ldi temp2, 0
    ldi temp, '-'
    rcall LCD_DATA
    rjmp GIN_LOOP

GIN_CLEAR:
    clr aL
    clr aH
    ldi temp2, 1
    rcall LCD_CLEAR
    rcall LCD_PRINT_P
    ldi temp, 1
    ldi temp2, 0
    rcall LCD_GOTO
    rjmp GIN_LOOP

GIN_DONE:
    ; apply sign if negative
    cpi temp2, 1
    breq GIN_RET
    ; two's complement: a = -a
    com aL
    com aH
    adiw aL, 1
GIN_RET:
    ret


; ============================================================
;              CONVERSIONS TO STEPS (unsigned steps)
; ============================================================
; ANGLE_TO_STEPS:
;   input aH:aL signed angle
;   output sH:sL unsigned steps
;   temp2 = direction (1 if angle>0 else 0)
ANGLE_TO_STEPS:
    ; temp2 = 1 if positive, 0 if negative or zero
    mov temp, aH
    tst temp
    brmi ANG_NEG
    ; positive or zero
    ldi temp2, 1
    rjmp ANG_ABS
```

```
ANG_NEG:
    ldi temp2, 0

ANG_ABS:
    ; abs(a) into dH:dL
    mov dL, aL
    mov dH, aH
    tst aH
    brpl ANG_ABS_OK
    ; negative -> abs
    com dL
    com dH
    adiw dL, 1
ANG_ABS_OK:
    ; steps = (abs*64 + 180)/360
    ; abs*64 = abs << 6
    ; put into rH:rL (use rH:rL = dH:dL shifted)
    mov rL, dL
    mov rH, dH
    lsl rL
    rol rH
    lsl rL
    rol rH
    lsl rL
    rol rH
    lsl rL
    rol rH
    lsl rL
    rol rH
    lsl rL
    rol rH          ; <<6 done

    ; add 180 for rounding
    ldi temp, 180
    add rL, temp
    adc rH, __zero_reg__

    ; divide (rH:rL) by 360 -> quotient in qH:qL
    mov dL, rL
    mov dH, rH
    ldi rL, low(360)
    ldi rH, high(360)
    rcall UDIV16        ; qH:qL = d / r
    mov sL, qL
    mov sH, qH
    ret
```

```
; REVS_TO_STEPS:
; input aH:aL signed revs
; output sH:sL = abs(revs)*64
; temp2 direction (1 if revs>0 else 0)
REVS_TO_STEPS:
    ; direction
    tst aH
    brmi REV_NEG
    ldi temp2, 1
    rjmp REV_ABS
REV_NEG:
    ldi temp2, 0
REV_ABS:
    mov dL, aL
    mov dH, aH
    tst aH
    brpl REV_ABS_OK
    com dL
    com dH
    adiw dL, 1
REV_ABS_OK:
    ; abs*64 = <<6
    mov sL, dL
    mov sH, dH
    lsl sL
    rol sH
    lsl sL
    rol sH
    lsl sL
    rol sH
    lsl sL
    rol sH
    lsl sL
    rol sH
    lsl sL
    rol sH
    ret


; ============================================================
;                  STEPPER RUN (sH:sL steps)
; ============================================================
; uses:
;   sH:sL number of steps
;   temp2 direction (1=forward,0=reverse)
STEPPER_RUN:
STEP_LOOP:
    ; if steps == 0 stop
```

```
   mov temp, sL
   or  temp, sH
   breq STEP_STOP

   ; update stepIx
   cpi temp2, 1
   breq STEP_FWD

STEP_REV:
   ; stepIx--
   tst stepIx
   brne _DEC_OK
   ldi stepIx, 4
_DEC_OK:
   dec stepIx
   rjmp STEP_OUT

STEP_FWD:
   inc stepIx
   cpi stepIx, 4
   brlo STEP_OUT
   clr stepIx

STEP_OUT:
   ; output pattern
   ldi ZH, high(STEPSEQ<<1)
   ldi ZL, low(STEPSEQ<<1)
   add ZL, stepIx
   adc ZH, __zero_reg__
   lpm temp, Z

   ; write to PORTC lower nibble (preserve upper)
   in temp2, PORTC
   andi temp2, 0xF0
   or  temp2, temp
   out PORTC, temp2

   rcall DELAY_20MS

   ; steps--
   sbiw sL, 1
   rjmp STEP_LOOP

STEP_STOP:
   ; de-energize coils: clear lower nibble
   in temp, PORTC
   andi temp, 0xF0
   out PORTC, temp
```

```
  ret


; ============================================================
;                 16-bit Unsigned Division
; ============================================================
; UDIV16:
;   dividend in dH:dL
;   divisor  in rH:rL
;   quotient -> qH:qL
;   remainder-> (ignored)
UDIV16:
  clr qL
  clr qH
  clr temp
  clr temp2
  clr aL
  clr aH          ; remainder in aH:aL

  ldi col, 16
UDIV_LOOP:
  ; shift left dividend into remainder
  lsl dL
  rol dH
  rol aL
  rol aH

  ; remainder -= divisor
  mov temp, aL
  mov temp2, aH
  sub aL, rL
  sbc aH, rH
  brcs UDIV_RESTORE

  ; set quotient bit = 1
  lsl qL
  rol qH
  ori qL, 1
  rjmp UDIV_NEXT

UDIV_RESTORE:
  ; restore remainder
  mov aL, temp
  mov aH, temp2
  ; quotient bit = 0
  lsl qL
  rol qH
```

```
UDIV_NEXT:
    dec col
    brne UDIV_LOOP
    ret


; ============================================================
;                       DELAYS (approx)
; ============================================================
DELAY_1MS:
    ; ~1ms @ 16MHz (rough)
    ldi temp, 250
D1:
    nop
    dec temp
    brne D1
    ret

DELAY_2MS:
    rcall DELAY_1MS
    rcall DELAY_1MS
    ret

DELAY_20MS:
    ldi temp, 20
D20:
    rcall DELAY_1MS
    dec temp
    brne D20
    ret

DELAY_100MS:
    ldi temp, 100
D100:
    rcall DELAY_1MS
    dec temp
    brne D100
    ret

DELAY_2S:
    ; 2000ms
    ldi temp, 20
D2S:
    rcall DELAY_100MS
    dec temp
    brne D2S
    ret
```

References

1. **Microchip Technology Inc.**
ATmega32 Datasheet – 8-bit AVR Microcontroller, Microchip (Atmel), Available online.

AVR Instruction Set Manual,Official AVR Assembly Language Reference.

AVR Libc User Manual, Documentation for C programming with AVR microcontrollers.

2. **Texas Instruments**

ULN2003A Darlington Transistor Array Datasheet, Texas Instruments, Used for stepper motor driving applications.

3. **GitHub**

Open Source AVR Projects*, Stepper Motor Control using AVR Microcontrollers (C & Assembly), Various open-source repositories and examples.*

4. **Proteus Design Suite Documentation**,
Microcontroller Simulation and Embedded System Design, Lab center Electronics.

5. OpenAI

ChatGPT: Large Language Model for Programming Assistance, used for code explanation, debugging, and documentation support.

6. Google

Gemini AI: Artificial Intelligence Model for Code Analysis and Generation, Used for assistance in software logic development and optimization.