STRATEC Hardware Challenge

Vlad Cristescu

Instead of simply drawing a schematic which may or may not be functional, I decided to use an online simulator so you can see how the circuit I came up with works. The project can be accessed using the following link:

https://wokwi.com/projects/426531553420450817

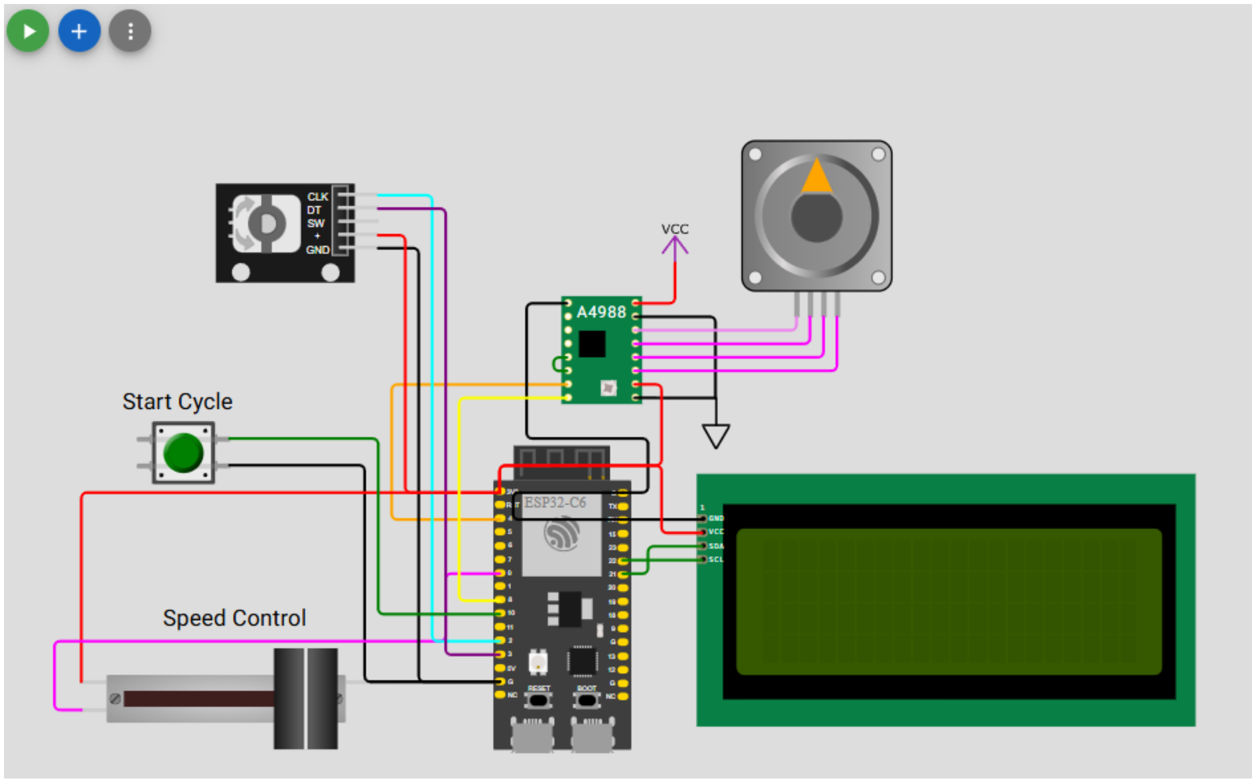

Figure 1. Motor control circuit

I decided to use an ESP32-C6, a bipolar stepper motor with the A4988 stepper motor driver, a pushbutton to start the cycle, a slide potentiometer for adjusting the speed, a KY-040 rotary encoder for reading the position of the shaft and an LCD to display the angular position, velocity and acceleration of the shaft.

The program is written so that the system waits for the user to push the start button, then turns 5 times clockwise, followed by 1 counterclockwise rotation to reach the predetermined home position.

Unfortunately, the simulator is bound by some constraints that prevent the motor to rotate smoothly due to the floating point arithmetic necessary for the velocity and acceleration and other factors. After compiling the project in its initial state to check the functionality of the system as a whole, you can comment lines 87 through 108 to compare the smooth, continuous movement of the shaft when it isn't constrained by time intensive operations such as clearing the LCD and performing modulo operations on a float variable which prevent the motor position to be refreshed as quickly inside the simulator display. This effect can also be observed on the LCD.

For further discussions about the project, I look forward to an interview.

```
#include <Arduino.h>
#include <AccelStepper.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Peripheral Pins
#define ENCODER_CLK_PIN 2
#define ENCODER_DT_PIN 3
#define START_BUTTON 10
#define POT_PIN A0  // Potentiometer speed control

// Stepper motor setup (A4988 Driver Mode)
#define STEP_PIN 4
#define DIR_PIN 8
#define STEPS_PER_REV 200
AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);

// LCD Display setup
#define I2C_ADDR 0x27
#define SDA_PIN 21
#define SCL_PIN 22
LiquidCrystal_I2C lcd(0x27, 20, 4);

// State tracking
bool motorRunning = false;
bool returningHome = false;
int targetTurns = 5;
volatile long encoderPosition = 0;

// Variables for calculations
unsigned long lastUpdateTime = 0;     // Timer for velocity/acceleration
unsigned long updateInterval = 500;   // Update every 500ms
long previousPosition = 0;
float velocity = 0;
float previousSpeed = 0;
float acceleration = 0;

void setup() {
    Serial.begin(115200);
    stepper.setAcceleration(300);

    // Pin setup
    pinMode(START_BUTTON, INPUT_PULLUP);
    pinMode(POT_PIN, INPUT);
    pinMode(ENCODER_CLK_PIN, INPUT);
    pinMode(ENCODER_DT_PIN, INPUT);

    // LCD initialization
    Wire.begin(SDA_PIN, SCL_PIN);
    lcd.begin(20, 4);
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Hardware Challenge");
    lcd.setCursor(0, 2);
    lcd.print("Vlad Cristescu");
    delay(2000);
}

void loop() {
    // Set motor speed according to potentiometer position
    int potValue = analogRead(POT_PIN);
    int motorSpeed = map(potValue, 0, 1023, 10, 300);
    stepper.setMaxSpeed(motorSpeed);

    // Start motor cycle
    if (digitalRead(START_BUTTON) == LOW && !motorRunning) {
        motorRunning = true;
        returningHome = false;
        stepper.move(targetTurns * STEPS_PER_REV);
    }

    if (motorRunning) {
        stepper.run();

        // Return to home position
        if (stepper.distanceToGo() == 0 && !returningHome) {
            stepper.move(-STEPS_PER_REV);
            returningHome = true;
        }
        // Cycle complete
        if (stepper.distanceToGo() == 0 && returningHome) {
            motorRunning = false;
            stepper.setCurrentPosition(0);
        }
    }

    unsigned long currentMillis = millis();
    if (currentMillis - lastUpdateTime >= updateInterval) {     // If 500ms have
passed since last reading
        lastUpdateTime = currentMillis;                         // to reduce
computational load

        encoderPosition = stepper.currentPosition();                  // Simulate
the encoder being turned mechanically by the motor
        float angularPosition = fmod((encoderPosition * 1.8), 360);   // Number
of steps * 1.8 deg/step % 360 => [deg]
        if (angularPosition < 0) angularPosition += 360;              // Wrap
around 360 during last cycle when direction is reversed

        velocity = ((encoderPosition - previousPosition) * 1.8) / (updateInterval
/ 1000.0);  // Number of steps * 1.8 deg/step / s => [deg/s]
        acceleration = ((motorSpeed - previousSpeed) * 1.8) / (updateInterval /
1000.0);      // Steps/s * 1.8 deg/step / s => [deg/s^2]

        previousPosition = encoderPosition;
        previousSpeed = motorSpeed;

        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Ang pos: "); lcd.print(angularPosition); lcd.print("deg");
        lcd.setCursor(0, 1);
        lcd.print("Speed: "); lcd.print(velocity); lcd.print("deg/s");
        lcd.setCursor(0, 2);
        lcd.print("Accel: "); lcd.print(acceleration); lcd.print("deg/s^2");
    }
}
```