LAB: Stepper Motor

Date: 2023-11-10

Author: Gyeonheal An

Github: https://github.com/AnGyeonheal/Embedded_Control_GH

Demo Video: https://www.youtube.com/watch?v=A9xt91HmfZY

I. Introduction

In this lab, we will learn how to drive a stepper motor with digital output of GPIOs of MCU. You will use a FSM to design the algorithm for stepper motor control.

i. Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others:
  - 3Stepper Motor 28BYJ-48
  - Motor Driver ULN2003
  - breadboard

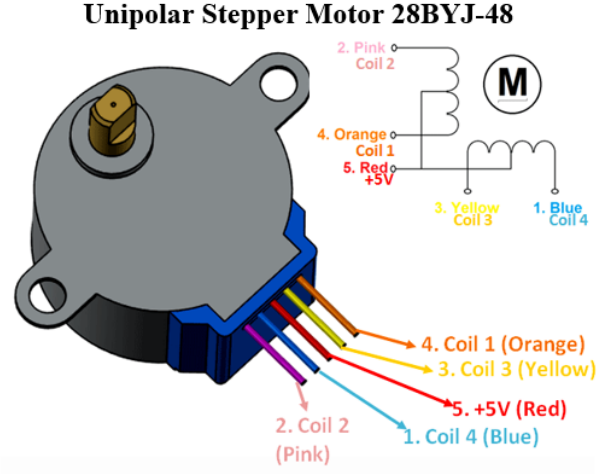
Software

- Clion(with PlatformIO core plugin)
  Library: STM32Cube library package(Official), EC_HAL library
  Compiler: GNU Arm Embedded Toolchain
  Debugger: ST-Link

II. Problem 1: Stepper Motor

i. Hardware Connection

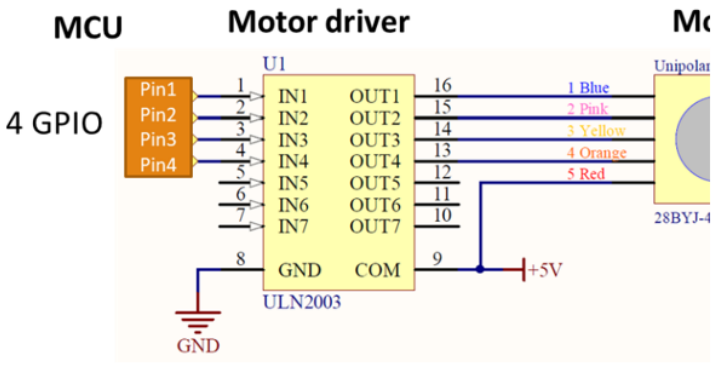
Read specification sheet of the motor and the motor driver for wiring and min/max input voltage/current.



- Rated Voltage: 5V DC
- Number of Phases: 4
- Stride Angle: 5.625°/64
- Gear ratio: 1/32
- Pull in torque: 300 gf.cm
- Coil: Unipolar 5 lead coil

MCU connection wiring

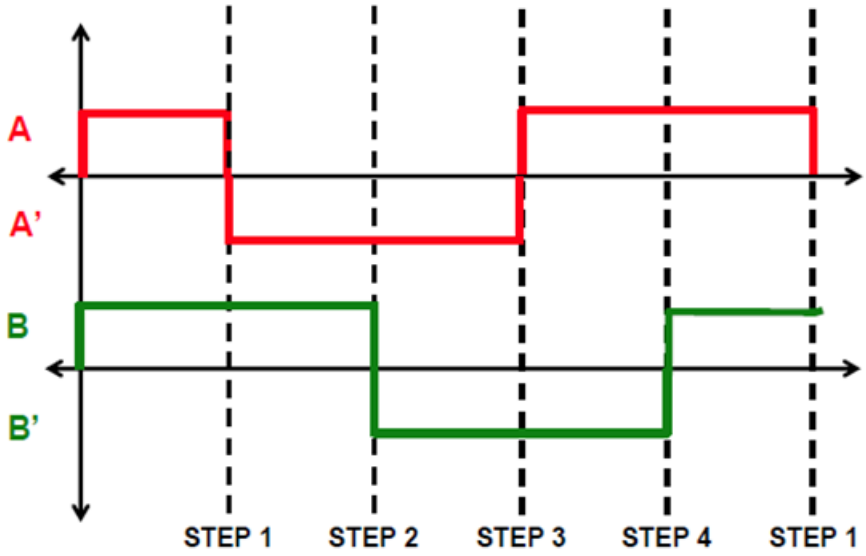
Motor driver (ULN2003)



ii. Stepper Motor Sequence

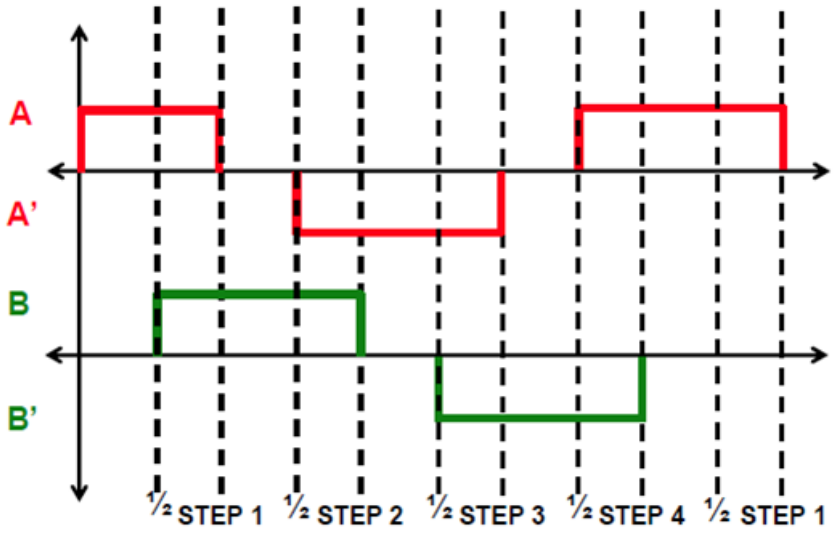
We will use unipolar stepper motor for this lab

Full-stepping sequence



Phase	Port Pin	Sequence			
		1	2	3	4
A	PB_10	H	L	L	H
B	PB_4	H	H	L	L
A'	PB_5	L	H	H	L
B'	PB_3	L	L	H	H

Half-stepping sequence



Phase	Port Pin	Sequence 1	2	3	4	5	6	7	8
A	PB_10	H	H	L	L	L	L	L	H
B	PB_4	L	H	H	H	L	L	L	L
A'	PB_5	L	L	L	H	H	H	L	L
B'	PB_3	L	L	L	L	L	H	H	H

iii. Finite State Machine

- Full-Stepping Sequence

State	Next State		Output
	DIR = 0	DIR = 1	(A B A' B')
S0	S1	S3	(1 1 0 0)
S1	S2	S0	(0 1 1 0)
S2	S3	S1	(0 0 1 1)
S3	S0	S2	(1 0 0 1)

- Half-Stepping Sequence

State	Next State		Output
	DIR = 0	DIR = 1	(A B A' B')
S0	S1	S7	(1 0 0 0)
S1	S2	S0	(1 1 0 0)
S2	S3	S1	(0 1 0 0)
S3	S4	S2	(0 1 1 0)
S4	S5	S3	(0 0 1 0)
S5	S6	S4	(0 0 1 1)
S6	S7	S5	(0 0 0 1)
S7	S0	S6	(1 0 0 1)

III. Problem 2: Firmware Programming

i. Create HAL library

ecStepper.h

```
// Initialize with 4 pins
// ( A, B,  AN,  BN)
void Stepper_init(GPIO_TypeDef* port1, int pin1, GPIO_TypeDef* port2, int pin2,
GPIO_TypeDef* port3, int pin3, GPIO_TypeDef* port4, int pin4);
//or   using ecPinNames.h
void Stepper_init(PinName_t A, PinName_t B,  PinName_t AN, PinName_t BN);
// whatSpeed [rev/min]
void Stepper_setSpeed(long whatSpeed);
// Run for n Steps
void Stepper_step(uint32_t steps, uint32_t direction, uint32_t mode);
// Immediate Stop.
void Stepper_stop(void);
```

> Note that these are blocking stepper controllers. While the stepper is running, the MCU cannot process other polling commands. If you can, modify it to be the non-blocking controller.

In the III.v.Code, when it's executed, the MCU board initially operates within the main function. Unless there is an EXTI or Timer Interrupt event, the code inside the while loop doesn't execute. Therefore, the Stepper_Step function runs within the while loop, and with each occurrence of a SysTick interrupt, the msTicks variable used in the delay_ms() function is brought into the main code. It is then used to count, which triggers the next state.

> You can also create your own functions different from the given instructions.

ii. Procedure

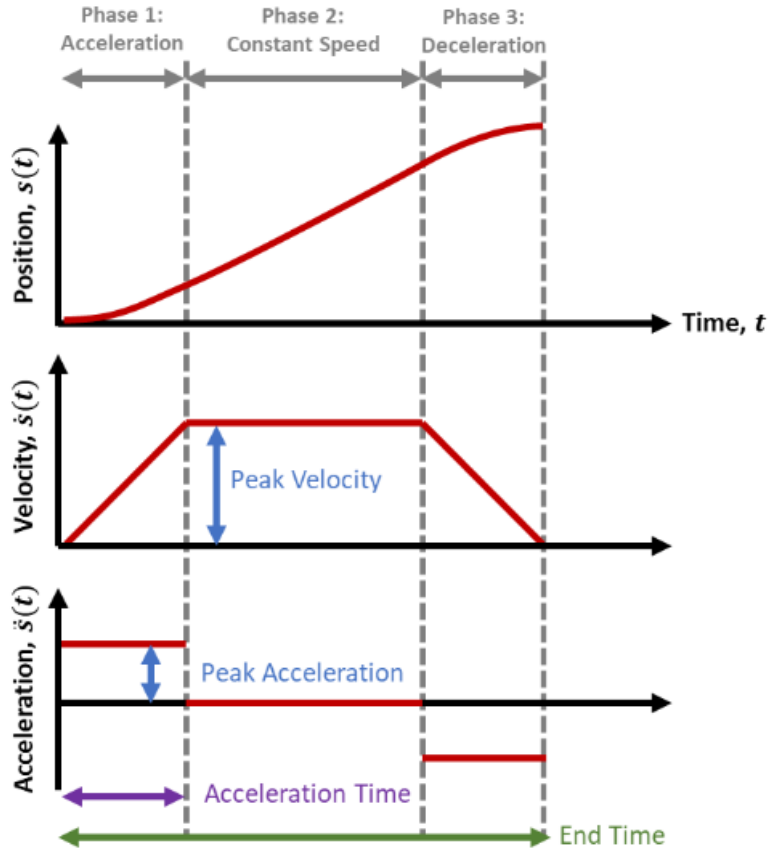
1. Find out the number of steps required to rotate 1 revolution using Full-steppping.
2. Then, rotate the stepper motor 10 revolutions with 2 rpm. Measure if the motor rotates one revolution per second.
3. Repeat the above process in the opposite direction.
4. Increase and decrease the speed of the motor as fast as it can rotate to find the maximum and minimum speed of the motor.
5. Apply the half-stepping and repeat the above.

iii. Configuration

Digital Out	SysTick
PB10, PB4, PB5, PB3 NO Pull-up Pull-down Push-Pull Fast	delay()

iv. Discussion

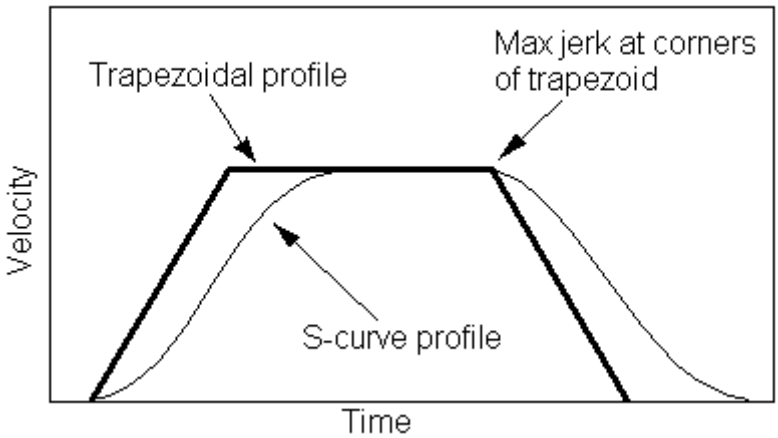
1. **Find out the trapezoid-shape velocity profile for a stepper motor. When is this profile necessary?**



A trapezoidal velocity profile is a common velocity profile used in motion control systems, including stepper motors. It consists of three distinct phases:

1. Acceleration: In this phase, the motor accelerates from standstill to a constant velocity. The acceleration rate is typically constant, resulting in a linear increase in velocity over time.
2. Constant Velocity: Once the desired velocity is reached, the motor maintains a constant speed, moving at a consistent rate.
3. Deceleration: In this phase, the motor decelerates to come to a stop. The deceleration rate is typically the same as the acceleration rate, resulting in a linear decrease in velocity.

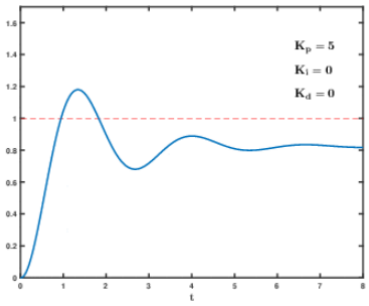
This trapezoidal profile is necessary in situations where you need precise control of the motor's movement, especially when it's essential to minimize overshoot and ensure accurate positioning. Stepper motors, which move in discrete steps, can benefit from this profile to achieve smoother motion and minimize vibrations.

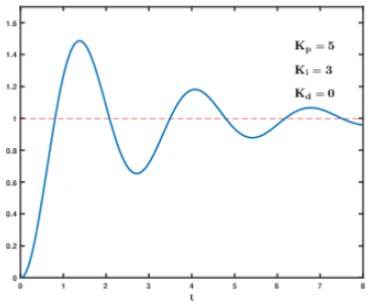


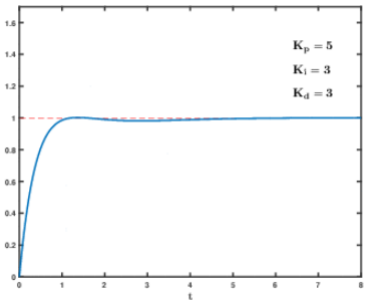
It's worth noting that for more advanced motion control applications, you might use other velocity profiles like S-curve or jerk-limited profiles to further improve motion quality and reduce wear and tear on the motor and mechanical components.

2. **How would you change the code more efficiently for micro-stepping control? You don't have to code this but need to explain your strategy.**

- Fine Step Subdivision:
  - By subdividing the 4 sequences of the Full step model into finer steps, the motor's movement becomes smoother and more precise.
  - Use microstepping drivers to divide full steps into finer steps, allowing the motor to move at smaller angles.
- PID Control:
  - PID control allows for error correction and precise position control.
  - The Proportional term (P) measures the current error and adjusts the motor's speed proportionally to the error's magnitude, helping to quickly reduce the error and converge to the desired position.
  - The Integral term (I) eliminates steady-state error by compensating for long-term errors, enabling precise position control.
  - The Derivative term (D) reduces overshoot and improves stability by detecting the rate of change of previous errors, maintaining smooth motion.







v. Code

```
#include "ecSTM32F411.h"

#define A 10
#define B 4
#define NA 5
#define NB 3
```

```
int RPM = 2;

void setup(void);

int main(){
    setup();
    Stepper_step(2048*10, 0, FULL);
    while(1){
    }
}
```

This is the main function and it contains a function that operates the stepper motor. Factors of this function include number of steps, direction, and mode. By multiplying the number of steps by the number of revolutions, you can enter how many turns you want to rotate.

```
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN)) {
        Stepper_stop();
        clear_pending_EXTI(BUTTON_PIN); // cleared by writing '1'
    }
}
```

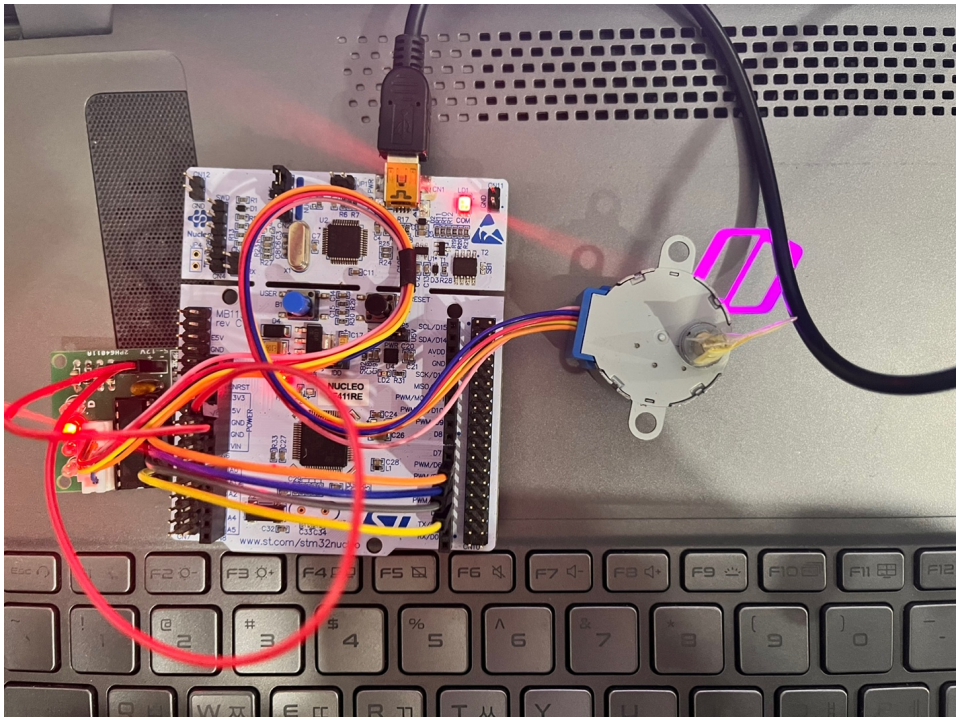
This function can make stepper motor stop using external interrupt (Button).

```
void setup(void){
    RCC_PLL_init();
    SysTick_init();

    GPIO_init(GPIOC, BUTTON_PIN, INPUT);           // GPIOC pin13 initialization
    EXTI_init(GPIOC, BUTTON_PIN, FALL,15);           // External Interrupt
Setting

    Stepper_init(GPIOB, A, GPIOB, B, GPIOB, NA, GPIOB, NB);
    Stepper_setSpeed(RPM);
}
```

This is initializing system clock, SysTick, Button Pin, Stepper motor.

It was initialized according to the configuration table above.

vi. Results



1. **Find out the number of steps required to rotate 1 revolution using Full-steppping.**
    - We need 2048 steps because the stepper motor's stride angle and gear ratio are 64 and 32 which means we need 64 x 32 steps.
    - FULL : 2048 steps
    - HALF : 2048 * 2 steps
2. **Measure if the motor rotates one revolution per second.**
    - FULL : about 28 sec
    - HALF : about 56 sec
3. **Increase and decrease the speed of the motor as fast as it can rotate to find the maximum and minimum speed of the motor.**
    - FULL : Maximum : 14 rpm, Minimum : 1 rpm
    - HALF : Maximum : 29 rpm, Minimum : 1 rpm

Add demo video link

IV. Reference

https://github.com/AnGyeonheal/Embedded_Control_GH

V. Troubleshooting