LAB: Stepper Motor

Date: 2022-11-01

Author/Partner: DongMin Kim / SeongJun Park

Github: https://github.com/DongminKim21800064/EC_dmkim-064

Demo Video: https://youtu.be/L4a_5KlcQZk

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.

Requirement

Hardware

- MCU
    - NUCLEO-F411RE
- Actuator
    - 3Stepper Motor 28BYJ-48
    - Motor Driver ULN2003

Software

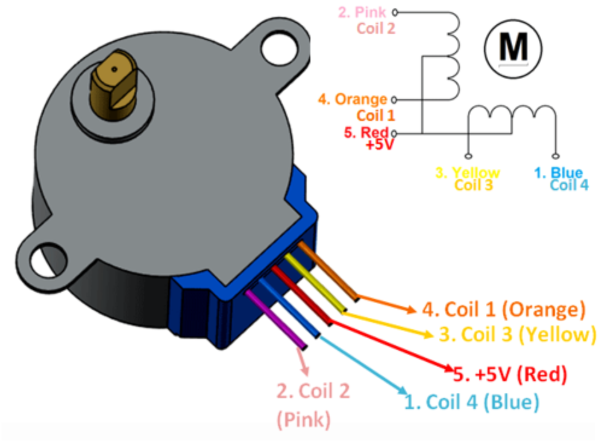
- Keil uVision, CMSIS, EC_HAL library

Problem 1: Stepper Motor

Hardware Connection

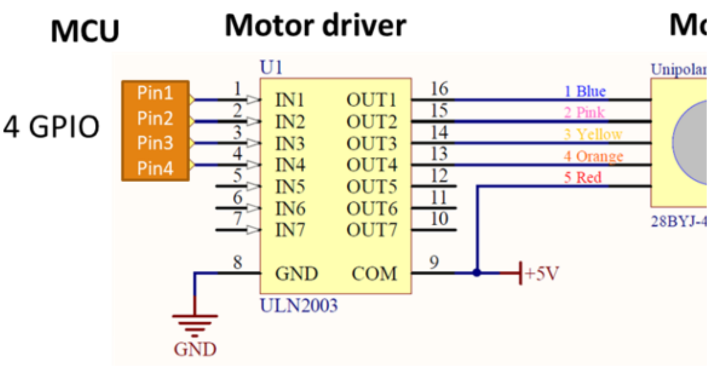
Read specification sheet of the motor and the motor driver for wiring and min/max input voltage/current.

Unipolar Stepper Motor 28BYJ-48



- 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



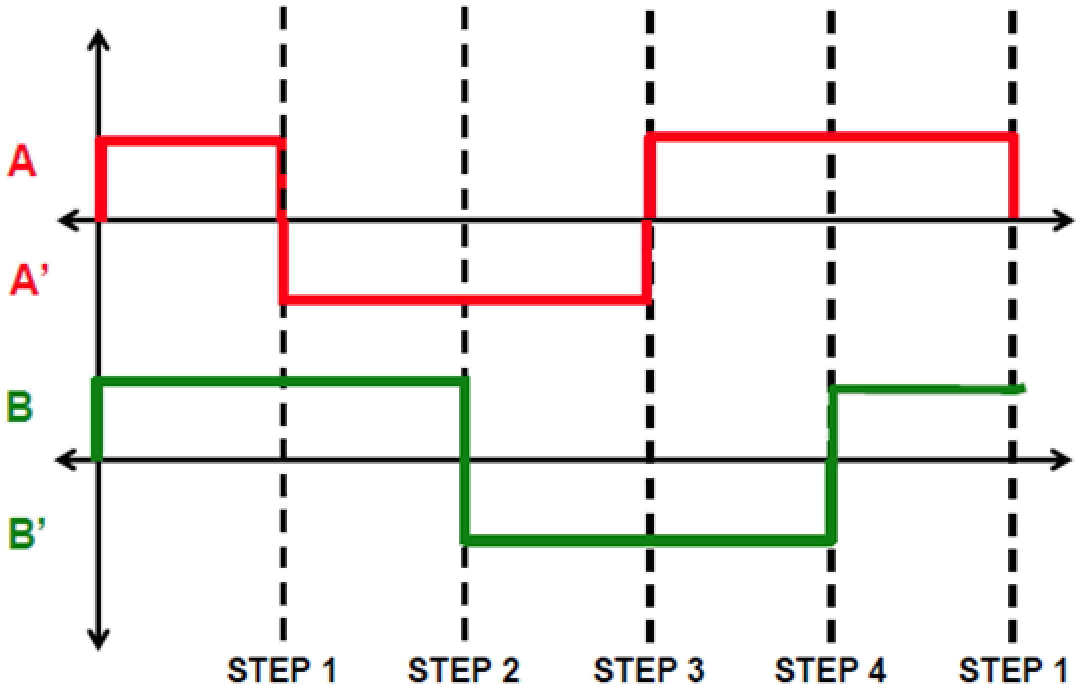
Motor driver (ULN2003)

Stepper Motor Sequence

We will use unipolar stepper motor for this lab

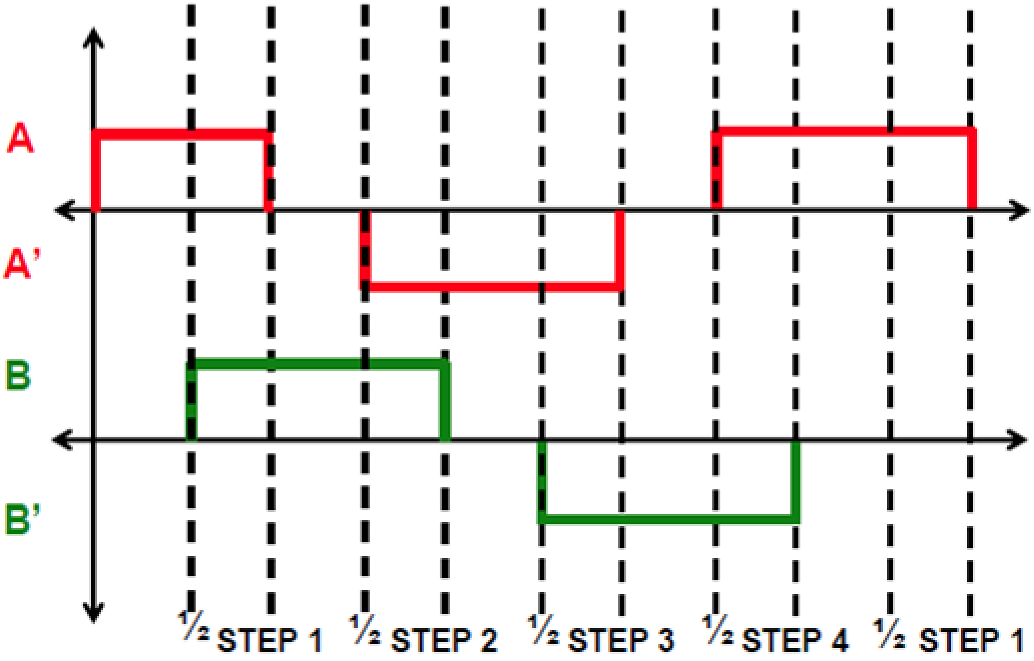
Fill in the blanks of each output data depending on the below sequence.

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

Finite State Machine

Draw a State Table for Full-Step Sequence. Use Moore FSM for this case. See *'Programming FSM'* for hints.

- Full-Stepping Sequence

State	Next State		Output
	DIR=0	DIR=1	(A B A' B')
S0	S3	S1	1100
S1	S0	S2	0110
S2	S1	S3	0011
S3	S2	S0	1001

- Half-Stepping Sequence

State	Next State		Output
	DIR=0	DIR=1	(A B A' B')
S0	S7	S1	1000
S1	S0	S2	1100
S2	S1	S3	0100
S3	S2	S4	0110
S4	S3	S5	0010
S5	S4	S6	0011
S6	S5	S7	0001
S7	S6	S0	1001

Problem 2: Firmware Programming

Create HAL library

Declare and Define the following functions in your library. You must

update your header files located in the directory `EC \lib\`.

ecStepper.h

```
// Initialize with 4 pins

void Stepper_init(GPIO_TypeDef* port1, int pin1, GPIO_TypeDef* port2, int pin2,
GPIO_TypeDef* port3, int pin3, GPIO_TypeDef* port4, int pin4);



// whatSpeed [rev/min]

void Stepper_setSpeed(long whatSpeed,int mode );



// Run for n Steps

void Stepper_step(int steps, int direction, int mode);



// Immediate Stop.

void Stepper_stop(void);
```

Procedure

1. Create a new project under the directory `\repos\EC\LAB\LAB_Stepper_Motor`
   - The project name is "**LAB_Stepper_Motor".**
   - Create a new source file named as "**LAB_Stepper_Motor.c"**

     > You MUST write your name on the source file inside the comment section.

2. Include your updated library in `\repos\EC\lib\` to your project.
   - **ecGPIO.h, ecGPIO.c**
   - **ecRCC.h, ecRCC.c**
   - **ecEXTI.h, ecEXTI.c**
   - **ecSysTick.h**, **ecSysTick.c**
   - **ecStepper.h ecStepper.h**
3. Connect the MCU to the motor driver and the stepper motor.

4. Find out the number of steps required to rotate 1 revolution using Full-steppping.

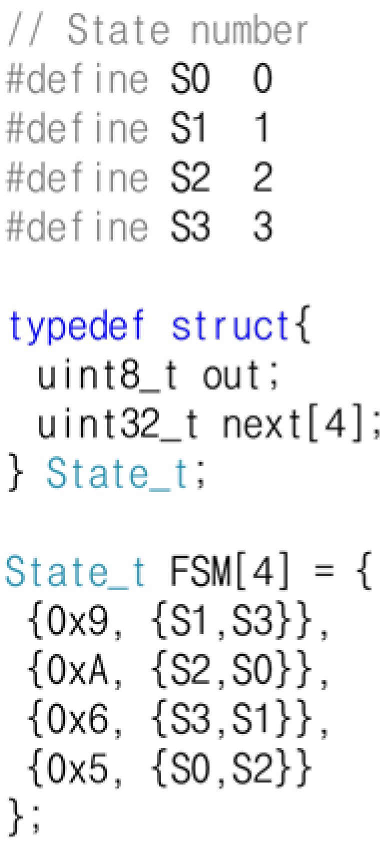
5. Then, rotate the stepper motor 10 revolutions with 2 rpm. Measure if the motor rotates one revolution per second.
6. Repeat the above process with the opposite direction.
7. Increase and decrease the speed of the motor as fast as it can rotate to find the max speed of the motor.
8. Apply the half-stepping and repeat the above.

Configuration

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

Requirement

You have to program the stepping sequence using the state table. You can define the states using structures. Refer to *'Programming FSM'* for hints.

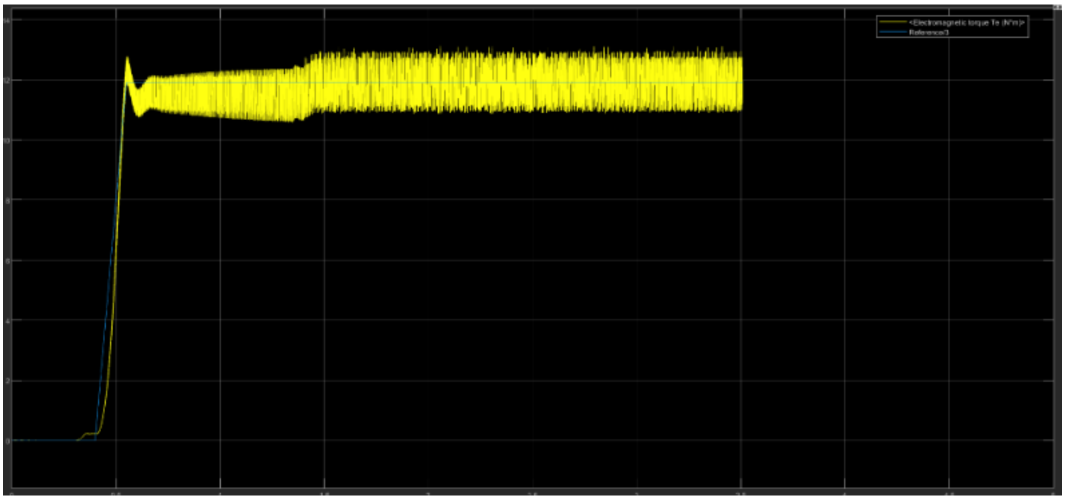
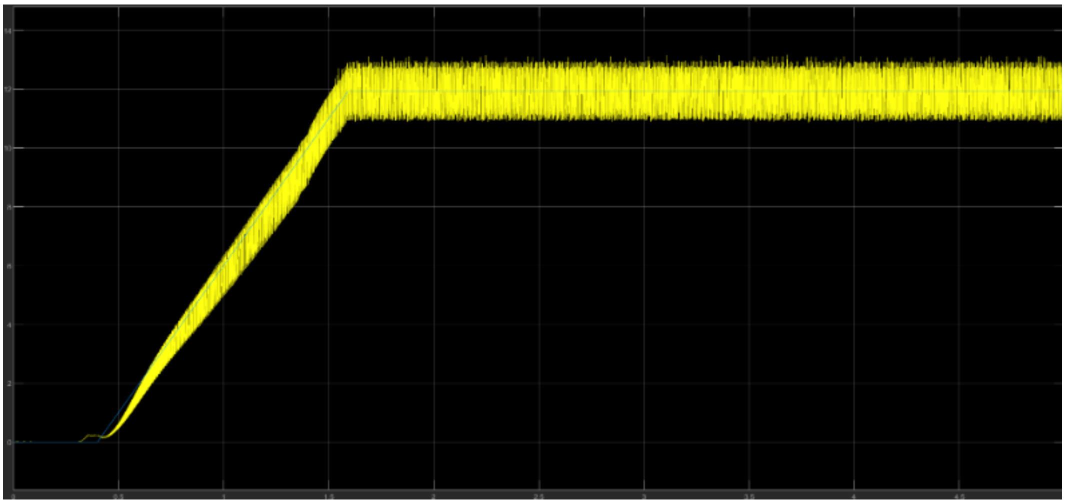


```c
// State number
#define S0  0
#define S1  1
#define S2  2
#define S3  3

typedef struct{
  uint8_t out;
  uint32_t next[4];
} State_t;

State_t FSM[4] = {
 {0x9, {S1,S3}},
 {0xA, {S2,S0}},
 {0x6, {S3,S1}},
 {0x5, {S0,S2}}
};
```

image

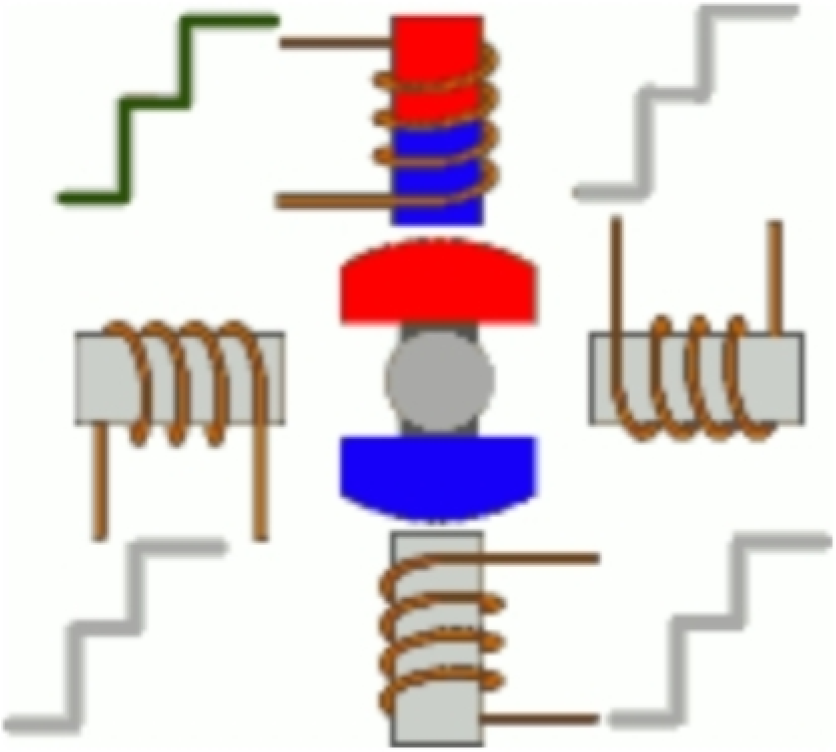
Discussion

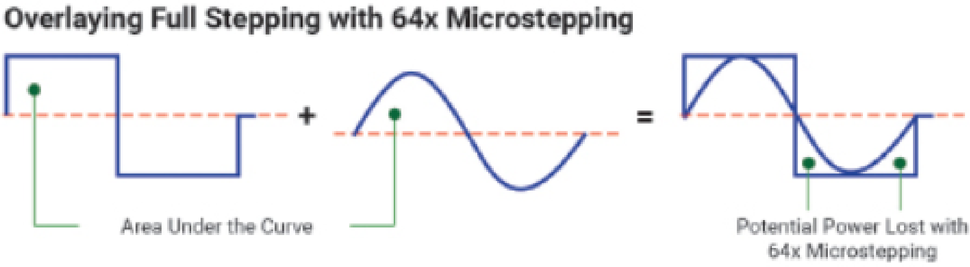
1. Find out the trapezoid-shape velocity profile for stepper motor. When is this profile necessary?

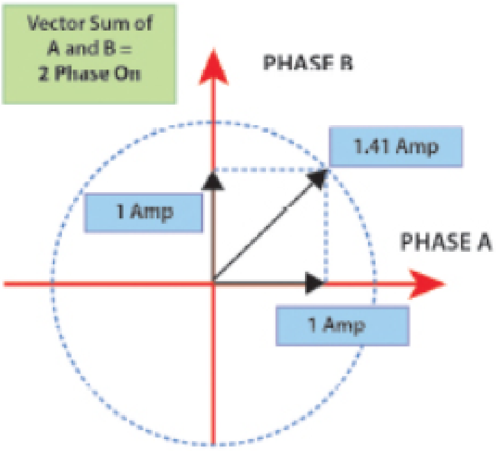
> **Rectangular**
>
> 
>
>
> **Trapezoid**
>
> 
>
> If the output power of the motor suddenly changes from LOW to HIGH, overshoot is generated at the point where it becomes HIGH, as in the Rectangular waveform, resulting in vibration of the motor.
>
> This is related to the law of inertia. Increasing the power of the motor, such as the trapezoid, low generates inertia, reducing the generation of overshoot and reducing the vibration of the motor.
>
> Therefore, the trapezoid can be more stable for motor control.

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.

MicroStepping




Overlaying Full Stepping with 64x Microstepping
+
=
Area Under the Curve
Potential Power Lost with 64x Microstepping



Micro-stepping, which is used as a way to control the operation of the step motor more precisely, is controlled by dividing the rotation angle corresponding to the pull-step into multiple sub-steps. When the full step angle is 90 degrees, it is decomposed into multiple steps such as 1/4, 1/8, 1/16, 1/32 to control the rotation at a more precise step angle.

> In my code, decomposed into multipel steps like
>
> S0-> (S0_1,S0_2,S0_3,S0_4), ... S0-> (S3_1,S3_2,S3_3,S3_4) and
>
> S0_1 = (0.25 0.25 0 0), S0_2 = (0.5 0.5 0 0), S0_3 = (0.75 0.75 0 0), S0_4 = (1 1 0 0)...

Code

Your code goes here: https://github.com/DongminKim21800064/EC_dmkim-064

Explain your source code with necessary comments.

LAB_Stepper_Motor.c

```c
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecEXTI.h"
#include "ecSysTick.h"
#include "ecStepper.h"

void setup(void);
volatile uint32_t rev = 10;
volatile uint32_t rpm = 20;
volatile uint32_t mode = HALF;
int main (void){
    // Initialization------------------------------------------------
    setup();

    Stepper_step(2048*rev, 0, mode); // (Step : 2048, Direction : 0 or 1, Mode : FULL or HALF)

    // Infinite Loop -------------------------------------------------
    while(1){;}
}

// Initialization
void setup(void){
    RCC_PLL_init();
// System Clock = 84Mz
    SysTick_init(1);
// Systick init

    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);      // External Interrupt Setting
    GPIO_init(GPIOC, BUTTON_PIN, INPUT);        // GPIOC pin 13 Initialization
    // Stepper GPIO pin Initializaion
    // No pull-up Pull-down , Push-Pull, Fast
    Stepper_init(GPIOB, 10, GPIOB, 4 , GPIOB,5, GPIOB,3);
    Stepper_setSpeed(rpm,mode);                 // set stepper motor speed
}

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

main

> Initialize set up, and declare "Stepper_step" function.
>
> "Stepper_step" set total step, Direction(CW/CCW), and MODE(FULL/HALF).

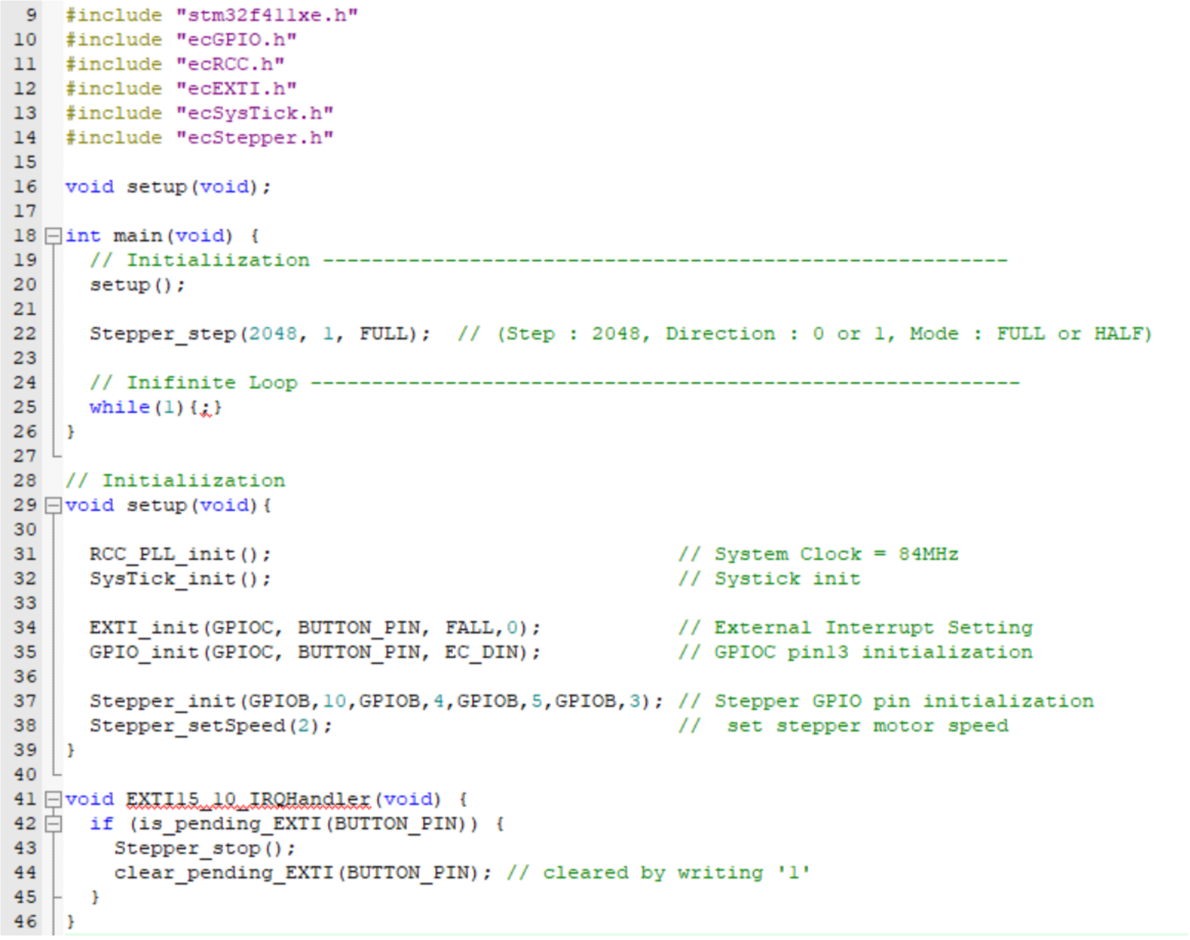
setup

> Set up RCC_PLL, and SysTick(1ms or us).
>
> Set up external interrupt setting and GPIO(BUTTON PIN).
>
> Set up Stepper initialization and speed.

EXTI15_10_IRQHandler

> EXTI Interrupt.
>
> If button pin pushed, stepper motor go to stop mode.

Sample Code : Stepper Motor

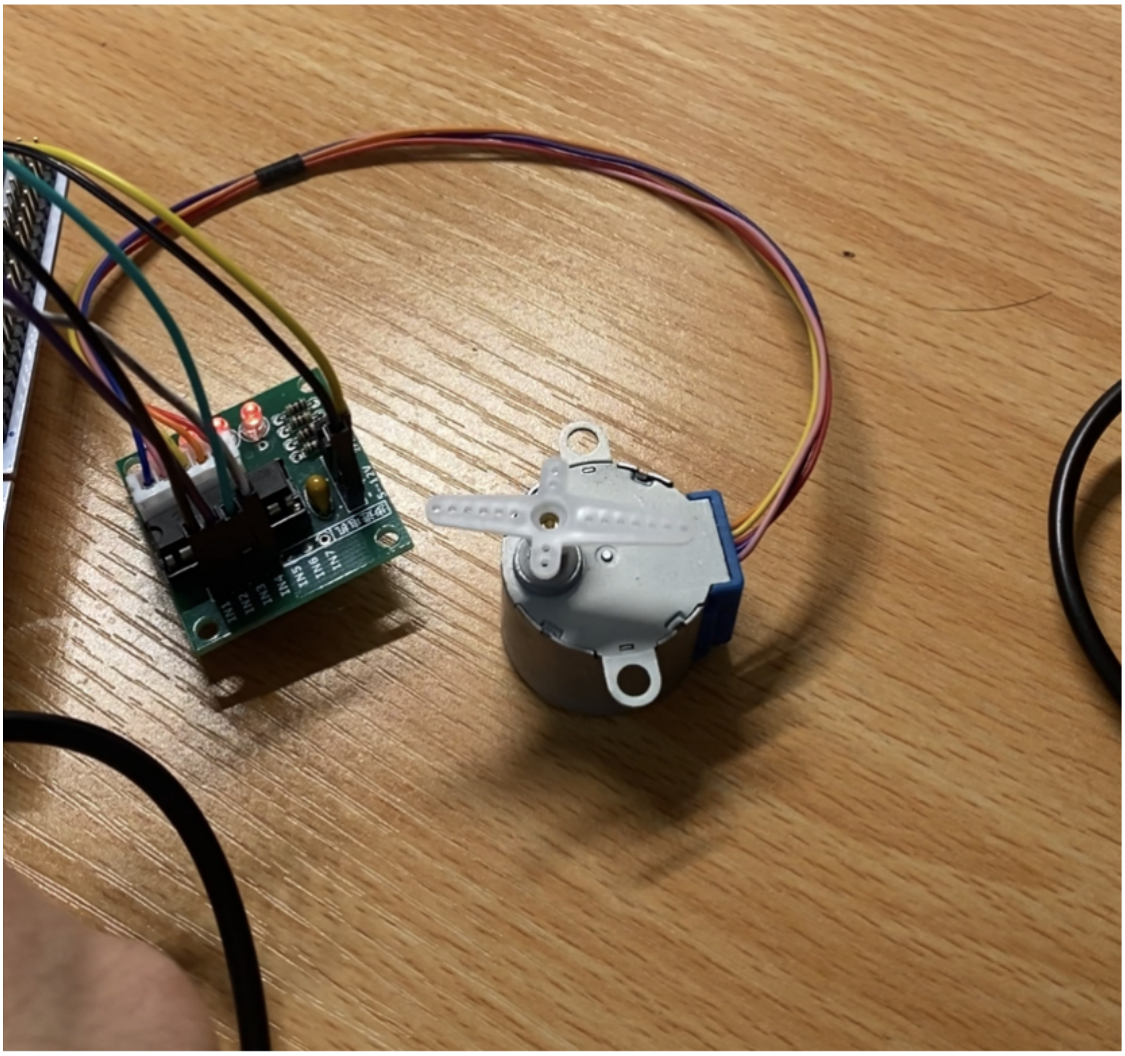


```c
#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecEXTI.h"
#include "ecSysTick.h"
#include "ecStepper.h"

void setup(void);

int main(void) {
  // Initialiization --------------------------------------------------------
  setup();

  Stepper_step(2048, 1, FULL);  // (Step : 2048, Direction : 0 or 1, Mode : FULL or HALF)

  // Inifinite Loop ----------------------------------------------------------
  while(1){;}
}

// Initialiization
void setup(void){

  RCC_PLL_init();                                 // System Clock = 84MHz
  SysTick_init();                                 // Systick init

  EXTI_init(GPIOC, BUTTON_PIN, FALL,0);           // External Interrupt Setting
  GPIO_init(GPIOC, BUTTON_PIN, EC_DIN);           // GPIOC pin13 initialization

  Stepper_init(GPIOB,10,GPIOB,4,GPIOB,5,GPIOB,3); // Stepper GPIO pin initialization
  Stepper_setSpeed(2);                            //  set stepper motor speed
}

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

image

Results

Experiment images and results



Demo Video Link : https://youtu.be/L4a_5KlcQZk

Analysis

In this experiment, an experiment was conducted to control the stepper motor. The four coils of the stepper motor were controlled according to the finite state with four pins outputs. The rotation direction and speed of the motor could be controlled through the code, and it could be made into a pause state through the EXTI interrupt. In addition, I learned the difference between Full stepping and Half stepping.

Reference

Given reference

https://ykkim.gitbook.io/ec/course/lab/lab-stepper-motor

Troubleshooting

Delay mode issue

> In the delay_ms function, an integer type must be input, but a real number less than 1 was input, causing a problem. To avoid this, there is a method of allowing a function input to be received in a real number type or creating a delay_us.

> In the case of the first method, a type problem may occur in ecSysTick.h, so it was solved using the second method. If the second method is used, it can have a value of 1 or more since the denominator is smaller than the molecule at
>
> (60000*1000)/(step_per_rev *whatSpeed).