LAB: Stepper Motor

Date: 2023-11-09

Author: EunChan Kim 21801017

Demo Video: Youtube link

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/Sensor/Others:
  - 3Stepper Motor 28BYJ-48
  - Motor Driver ULN2003
  - breadboard

Software

- Keil uVision, CMSIS, EC_HAL library

Problem 1: Stepper Motor

Hardware Connection

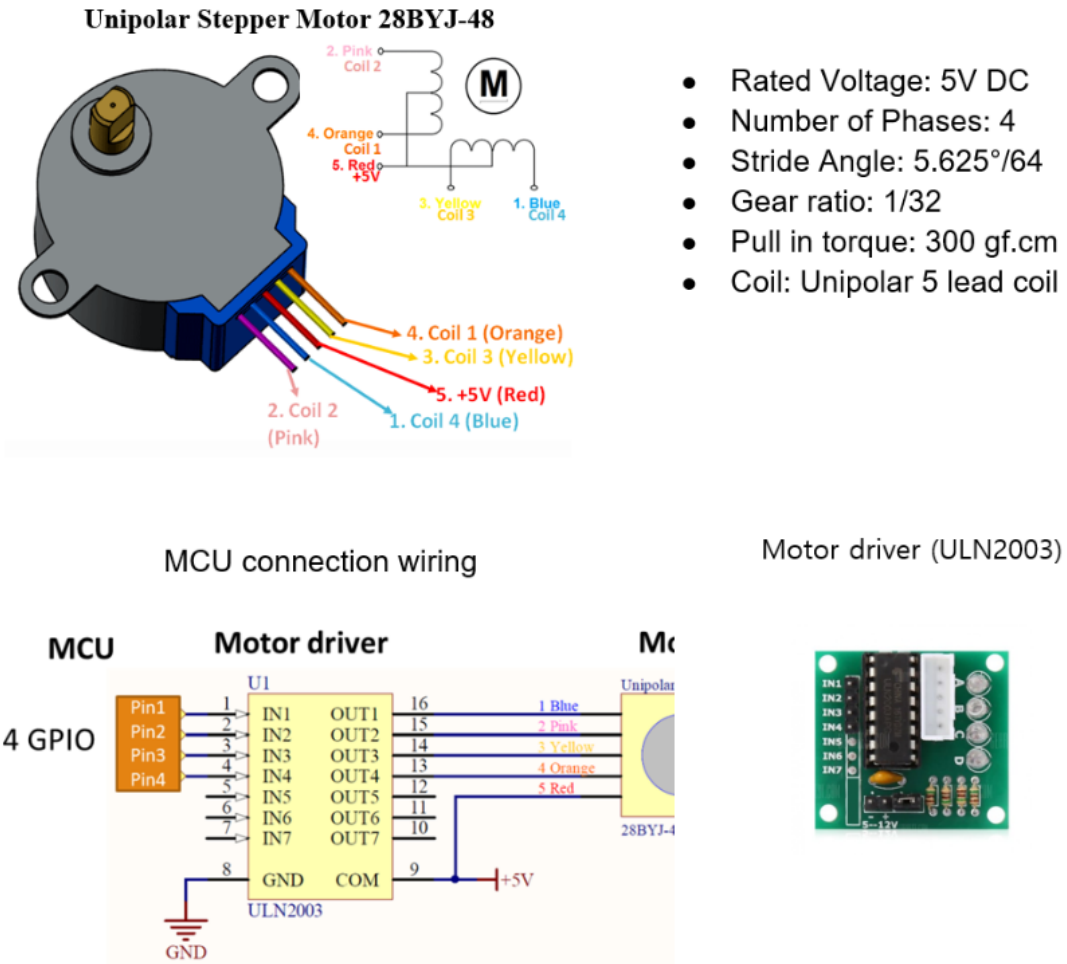


Figure1. Stepper Motor Specification

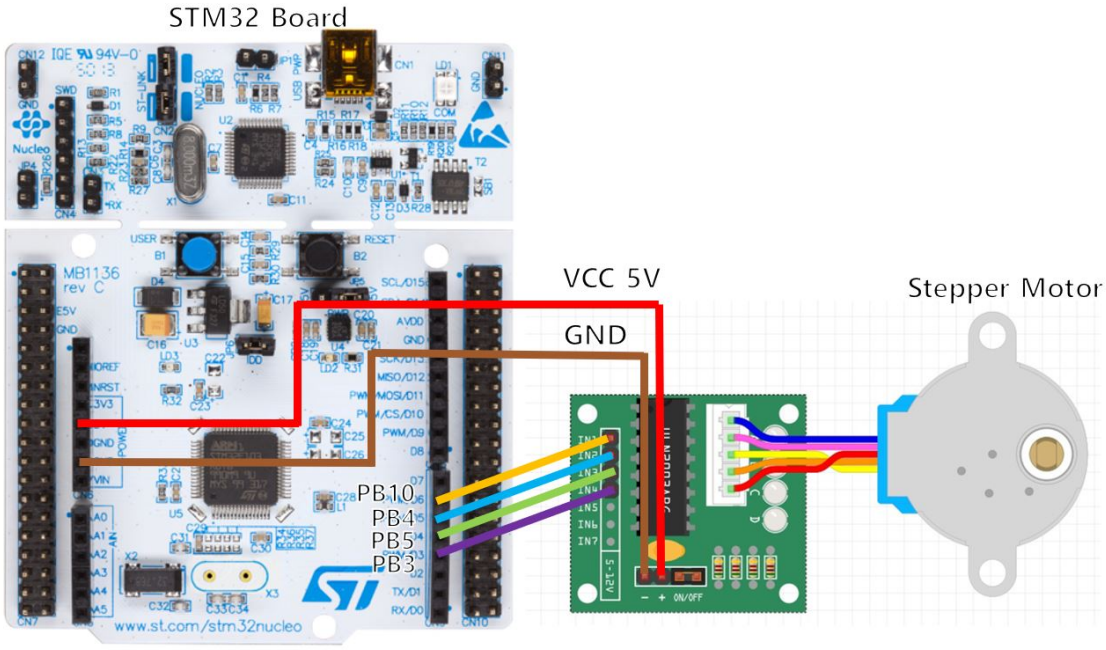
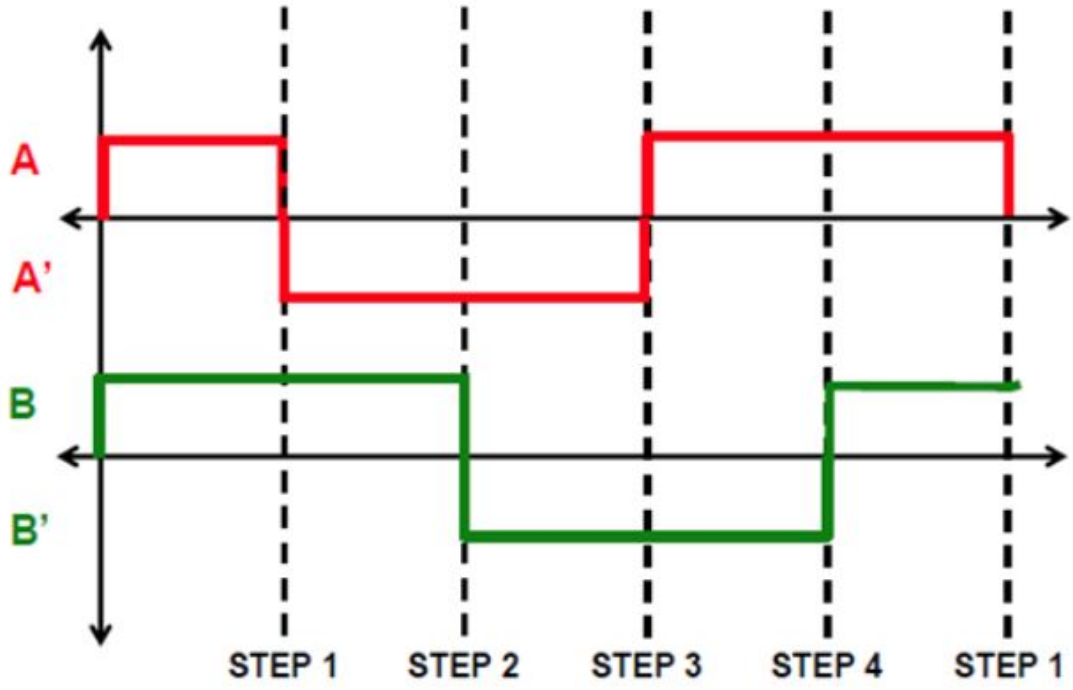


Figure2. Stepper Motor Circuit

Stepper Motor Sequence

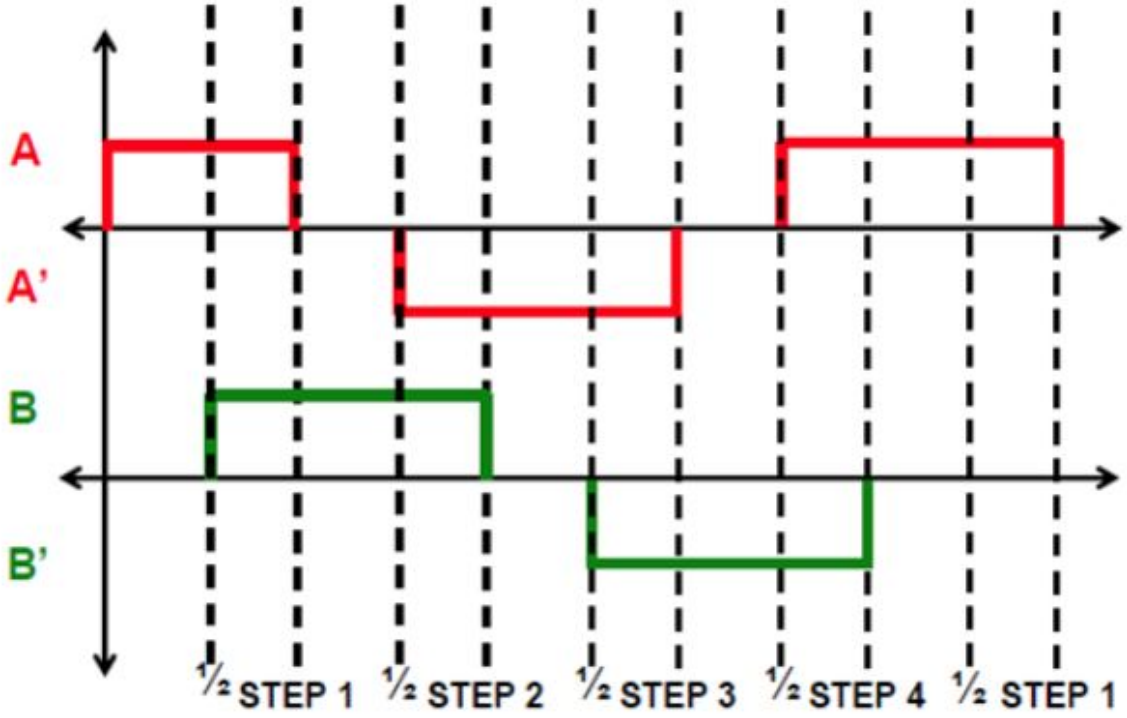
Full-stepping sequence



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

Figure3. Full-stepping sequence and Table

Half-stepping sequence



Phase	Port_Pin	Sequence							
		1	2	3	4	5	6	7	8
A	PB_10	H	H	0	L	L	L	0	H
B	PB_4	0	H	H	H	0	L	L	L
A'	PB_5	L	L	0	H	H	H	0	L
B'	PB_3	0	L	L	L	0	H	H	H

Figure4. Half-stepping sequence and Table

Finite State Machine

Full-Stepping Sequence

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

Figure5. FSM of Full-stepping sequence table

Half-Stepping Sequence

State	Next State		Output
	DIR = 0	DIR = 1	(A B A' B')
S0	S7	S1	H0L0
S1	S0	S2	HHLL
S2	S1	S3	0H0L
S3	S2	S4	LHHL
S4	S3	S5	L0H0
S5	S4	S6	LLHH
S6	S5	S7	0L0H
S7	S6	S0	HLLH

Figure6. FSM of Half-stepping sequence table

Problem 2: Firmware Programming

Create HAL library

ecStepper_student.c

The library was configured according to the above Full or Half stepping sequence.

```
#include "stm32f4xx.h"
#include "ecStepper.h"

//State number
#define S0 0
#define S1 1
#define S2 2
#define S3 3
#define S4 4
#define S5 5
#define S6 6
#define S7 7


// Stepper Motor function
uint32_t direction = 1;
uint32_t step_delay = 100;
uint32_t step_per_rev = 64*32;


// Stepper Motor variable
volatile Stepper_t myStepper;


//FULL stepping sequence  - FSM
typedef struct {
    uint8_t out;
    uint32_t next[2];
} State_full_t;
```

```
State_full_t FSM_full[4] = {
    {0b1100,{S1,S3}},
    {0b0110,{S2,S0}},
    {0b0011,{S3,S1}},
    {0b1001,{S0,S2}}
};

//HALF stepping sequence
typedef struct {
    uint8_t out;
    uint32_t next[2];
} State_half_t;

State_half_t FSM_half[8] = {
    {0b1000,{S1,S7}},
    {0b1100,{S2,S0}},
    {0b0100,{S3,S1}},
    {0b0110,{S4,S2}},
    {0b0010,{S5,S3}},
    {0b0011,{S6,S4}},
    {0b0001,{S7,S5}},
    {0b1001,{S0,S6}}

};



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

    //  GPIO Digital Out Initiation
    myStepper.port1 = port1;
    myStepper.pin1  = pin1;
    // Repeat for port2,pin3,pin4
    myStepper.port2 = port2;
    myStepper.pin2  = pin2;
    myStepper.port3 = port3;
    myStepper.pin3  = pin4;
    myStepper.port4 = port4;
    myStepper.pin4  = pin4;

    GPIO_init(port1, pin1, OUTPUT);
    GPIO_init(port2, pin2, OUTPUT);
    GPIO_init(port3, pin3, OUTPUT);
    GPIO_init(port4, pin4, OUTPUT);

    //  GPIO Digital Out Initiation
    // No pull-up Pull-down , Push-Pull, Fast
    // Pin1 ~ Port4
    GPIO_pupd(port1, pin1, EC_NONE);
    GPIO_otype(port1, pin1,EC_PUSH_PULL);
    GPIO_ospeed(port1, pin1, EC_FAST);

    GPIO_pupd(port2, pin2, EC_NONE);
    GPIO_otype(port2, pin2,EC_PUSH_PULL);
    GPIO_ospeed(port2, pin2, EC_FAST);

    GPIO_pupd(port3, pin3, EC_NONE);
```

```
    GPIO_otype(port3, pin3,EC_PUSH_PULL);
    GPIO_ospeed(port3, pin3, EC_FAST);

    GPIO_pupd(port4, pin4, EC_NONE);
    GPIO_otype(port4, pin4,EC_PUSH_PULL);
    GPIO_ospeed(port4, pin4, EC_FAST);
}


void Stepper_pinOut (uint32_t state, uint32_t mode){
    if (mode == FULL){          // FULL mode
            GPIO_write(myStepper.port1, myStepper.pin1, FSM_full[state].out >> 3
& 1);
            GPIO_write(myStepper.port2, myStepper.pin2, FSM_full[state].out >> 2
& 1);
            GPIO_write(myStepper.port3, myStepper.pin3, FSM_full[state].out >> 1
& 1);
            GPIO_write(myStepper.port4, myStepper.pin4, FSM_full[state].out >> 0
& 1);

    }
    else if (mode == HALF){     // HALF mode

            GPIO_write(myStepper.port1, myStepper.pin1, FSM_half[state].out >> 3
& 1);
            GPIO_write(myStepper.port2, myStepper.pin2, FSM_half[state].out >> 2
& 1);
            GPIO_write(myStepper.port3, myStepper.pin3, FSM_half[state].out >> 1
& 1);
            GPIO_write(myStepper.port4, myStepper.pin4, FSM_half[state].out >> 0
& 1);
    }
}


void Stepper_setSpeed (long whatSpeed){      // rpm [rev/min]
        step_delay = 60000 / (step_per_rev*whatSpeed); //YOUR CODE   // Convert
rpm to  [msec] delay
}


void Stepper_step(uint32_t steps, uint32_t direction, uint32_t mode){
     uint32_t state = 0;
     myStepper._step_num = steps;

     for(; myStepper._step_num > 0; myStepper._step_num--){ // run for step size
        // YOUR CODE                          // delay (step_delay);
        if (mode == FULL)
                state = FSM_full[state].next[direction];// YOUR CODE       //
state = next state
        else if (mode == HALF)
                state = FSM_half[state].next[direction];// YOUR CODE       //
state = next state

        Stepper_pinOut(state, mode);
        delay_ms(step_delay);
    }
}
```

```
void Stepper_stop (void){
        myStepper._step_num = 0;
    // All pins(A,AN,B,BN) set as DigitalOut '0'
            GPIO_write(myStepper.port1, myStepper.pin1, myStepper._step_num);
            GPIO_write(myStepper.port2, myStepper.pin2, myStepper._step_num);
            GPIO_write(myStepper.port3, myStepper.pin3, myStepper._step_num);
            GPIO_write(myStepper.port4, myStepper.pin4, myStepper._step_num);
}
```

Procedure

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

The contents of the written FSM graph above were implemented into code by creating a structure to implement the Full & Half stepping sequence.

```
//FULL stepping sequence  - FSM
typedef struct {
    uint8_t out;
    uint32_t next[2];
} State_full_t;

State_full_t FSM_full[4] = {
    {0b1100,{S1,S3}},
    {0b0110,{S2,S0}},
    {0b0011,{S3,S1}},
    {0b1001,{S0,S2}},
};
```

```
//HALF stepping sequence
typedef struct {
    uint8_t out;
    uint32_t next[2];
} State_half_t;

State_half_t FSM_half[8] = {
    {0b1000,{S1,S7}},
    {0b1100,{S2,S0}},
    {0b0100,{S3,S1}},
    {0b0110,{S4,S2}},
    {0b0010,{S5,S3}},
    {0b0011,{S6,S4}},
    {0b0001,{S7,S5}},
    {0b1001,{S0,S6}},
};
```

Discussion

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

> The trapezoidal-shaped velocity profile of a stepper motor is primarily used when the speed increases or decreases while the acceleration and deceleration are maintained constant. This aids in smoothly accelerating and decelerating the motor, hence it is necessary in applications that require high precision and smooth motion.
>
> The trapezoidal speed profile can be divided into three sections:
>
> Acceleration section: The stepper motor increases its speed at a constant acceleration. This section continues until the speed reaches its maximum.
>
> Constant speed section: The motor operates steadily at maximum speed. This section continues until the motor begins to decelerate.
>
> Deceleration section: The motor reduces its speed at a constant deceleration. This section ends when the speed reaches 0.
>
> This type of speed profile is necessary in high-speed, high-precision applications, and also when you want to avoid actions that start or stop abruptly. If the acceleration or deceleration is too high, the motor may unnecessarily overheat, and the lifespan of the parts may be shortened.

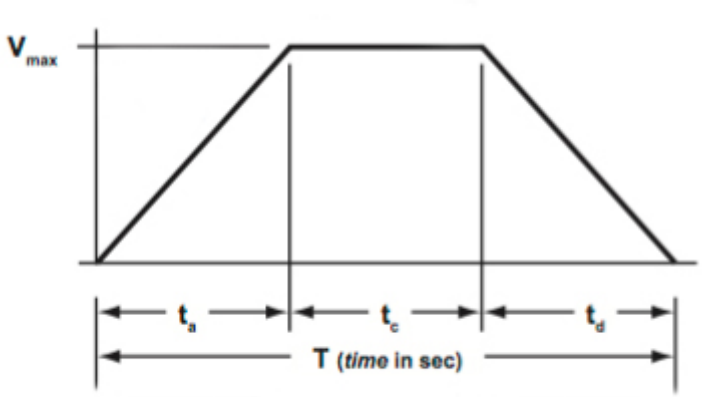


Figure7. The trapezoidal-shaped velocity profile

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.

> Selection of interpolation algorithm: An interpolation algorithm can be selected to improve microstep control. For example, smoother motor operation can be achieved using linear interpolation, sine curve interpolation, or other advanced algorithms.
>
> Adjustment of step size: In microstep control, the accuracy of the motor can be increased by adjusting the size of the step. By adjusting the size of the step smaller, the control of the motor position can be made more sophisticated.
>
> Change of step angle: Generally, the motor steps in units of 1.8 degrees. However, the precision of the motor can be improved by setting the step to a smaller angle than this. For example, more sophisticated control can be achieved by setting the step in units of 0.9 degrees.

Code

LAB_Stepper Motor.c

The execution of the program is initiated by the `main()` function. The `setup()` function is called to perform initial settings, followed by the call to the `Stepper_step()` function that rotates the stepper motor by a certain number of steps. Subsequently, it enters an infinite loop with `while(1)`, where the operation of the stepper motor is controlled depending on the value of the flag variable.

If the flag is 1, the Stepper_stop() function is called to stop the motor, otherwise, the `Stepper_step()` function is called to rotate the motor. The setup() function performs the initial settings of the system. In this function, clock settings, SysTick timer initialization, GPIO and external interrupt settings, stepper motor initialization, stepper motor speed settings, etc., are carried out.

The `EXTI15_10_IRQHandler()` function is an external interrupt handler. This function is called when a falling edge is detected at the GPIO pin, toggles the value of the flag variable, and calls the `Stepper_stop()` function to stop the motor. Afterwards, the interrupt is cleared to prepare for the next interrupt.

```
#include "stm32f411xe.h"
#include "math.h"
#include "ecGPIO.h"
#include "ecEXTI.h"
#include "ecRCC.h"
#include "ecTIM.h"
#include "ecSysTIck.h"
#include "ecStepper.h"


#define BUTTON_PIN 13


void setup(void);
```

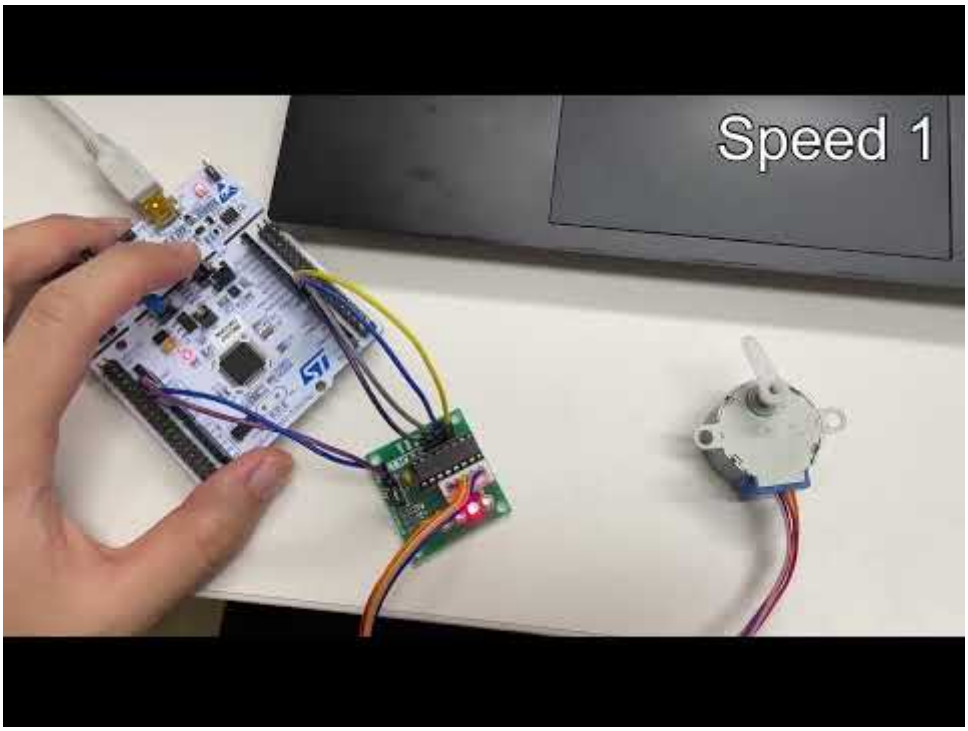
```
void EXTI15_10_IRQHandler(void);

uint32_t flag = 0;

int main(void){

    setup();

    Stepper_step(2048, 1, HALF); //steps/rev = 64x32

    while(1){
        if(flag == 1){
            Stepper_stop();
        }
        else{
            Stepper_step(2048, 1, HALF);
        }
    }
}


void setup(){

    RCC_PLL_init();
    SysTick_init();

    GPIO_init(GPIOC, BUTTON_PIN, INPUT);
    EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);

    Stepper_init(GPIOB, 10, GPIOB, 4, GPIOB, 5, GPIOB, 3);
    Stepper_setSpeed(2);
}

void EXTI15_10_IRQHandler(void) {
   if (is_pending_EXTI(BUTTON_PIN)) {
            flag ^= 1;
            Stepper_stop();
            clear_pending_EXTI(BUTTON_PIN);
   }
}
```

Results

As can be seen in the video, the motor operates normally at speeds of 1 to 3. It was also confirmed that when the button is pressed, an interrupt occurs, and the motor stops.

However, a problem occurs at speeds of 4 and above, where the motor's vibrations are transmitted, but it does not move. Given that the same logic algorithm applies up to speed 3 but not beyond, it seems to be a hardware limitation. Additionally, it was observed that when the motor is supposed to rotate in the opposite direction of the clock, only the motor's vibrations are transmitted and it does not rotate. Since the sequence in the original FSM is simply reversed, it is speculated that there is no issue with the code, and it is suspected that this could also be a hardware issue.

Demo video↓



Reference

- STM32 Cortex®-M4 MCUs and MPUs programming manual Download Link
- STM32 Cortex®-M4 MCUs and MPUs reference manual Download Link

Troubleshooting

When the initial code was completed, an issue arose where the motor did not stop even when a button interrupt occurred. The `Stepper_step` function operates in a blocking manner, which hinders the execution of other codes while the motor is rotating for the given number of steps. This was why the motor did not stop even when a button interrupt occurred.

To resolve this, the `Stepper_step` function was modified to operate in a non-blocking manner. By doing this, the `Stepper_step_non_blocking` function performs the given step when it's time to do so and immediately returns if not. This does not interfere with the execution of other codes.

Two static variables, `state` and `last_step_time`, are used. `State` records the current state of the stepper motor, and `last_step_time` records the time when the last step was performed. `Current_time` represents the current time, which is obtained using the `SysTick_val` function. The function only performs a step when the difference between `current_time` and `last_step_time` is equal to or greater than `step_delay`. This maintains a regular interval between each step. When performing a step, `steps` is first assigned to `myStepper._step_num`, and then a step is performed as long as `myStepper._step_num` is greater than 0. `MyStepper._step_num` is decremented by 1 each time a step is performed.

```
void Stepper_step(uint32_t steps, uint32_t direction, uint32_t mode){
    static uint32_t state = 0;
    static uint32_t last_step_time = 0;
```

```
    uint32_t current_time = SysTick_val();

    if ((current_time - last_step_time) >= step_delay){
        last_step_time = current_time;
        myStepper._step_num = steps;

        if(myStepper._step_num > 0){ // run for step size
            if (mode == FULL)
                state = FSM_full[state].next[direction]; // state = next state
            else if (mode == HALF)
                state = FSM_half[state].next[direction]; // state = next state


            Stepper_pinOut(state, mode);
                  delay_ms(step_delay);
            myStepper._step_num--;
        }
    }
}
```