ECE 153B – Winter 2023
Digital Motor control

Lab 6 – Stepper and Servo Motors

Deadline: **Mar 7, 2023 @ 7:00 PM**

Objectives

1. Understand full and half stepping to control the speed and position of a stepper motor.
   a) Use GPIO pin in output mode
   b) Control the stepper motor through the GPIO pin
2. Use Pulse Width Modulation (PWM) to control a servo motor.
   a) Understand the concept of PWM
   b) Learn how to configure and program a timer

Grading

Part	Weight
Part A	50 %
Part B	50 %

Submit your code and answers to the questions to the submission link on GradeScope by the specified deadline. In the week following the submission on GradeScope, you will demo/checkoff your lab to the TA.

Please note that we take the Honor Code very seriously, do not copy the code from others.

Necessary Supplies

- STM32L4 Nucleo Board
- Type A Male to Mini B USB Cable
- Breadboard & Jumper Wires
- SG90 Micro Servo Motor
- 28BYJ-48 5v stepper motor + ULN2003 driver board

1 Lab Overview

A. Rotate the stepper motor shaft exactly 360 degrees clockwise or counter-clockwise via both half and full stepping, and adjust the speed of rotation.

B. Rotate the servo motor 180 degrees by adjusting the duty cycle of a PWM GPIO pin.

2 Part A - Stepper Motor

In this part of the lab, you will perform the 4-step sequence of full-stepping or the 8-step sequence of half-stepping to rotate the 28BYJ-48 5v stepper motor. This is accomplished by using output GPIO pins to control the current through the two coils of the motor, through the ULN2003 driver board. The motor and driver board are shown below.

Figure 1: Stepper Motor and Control Board (Model: 28BYJ-48).

The following is the hardware information.

Motor Model	**28BYJ-48**	Number of phases	2
Rated voltage	5V DC	Gear reduction ratio	1/64
DC resistance per phase	$50\Omega \pm 7\%(25°C)$	Pull in torque	>300gf.cm / 5VDC 100pp

2.1 Introduction

A stepper motor rotates by moving a rotor to evenly-spaced positions in a fixed sequence of steps that form a complete rotation. The rotor is moved to different positions by adjusting the current in two coils in fixed locations around the rotor. The two most common and preferred sequences (also known as driving methods) are called *Full-Stepping* and *Half-Stepping*, which split the rotation into 4 and 8 steps respectively. These sequences are shown in Figures 2 and 3 below.

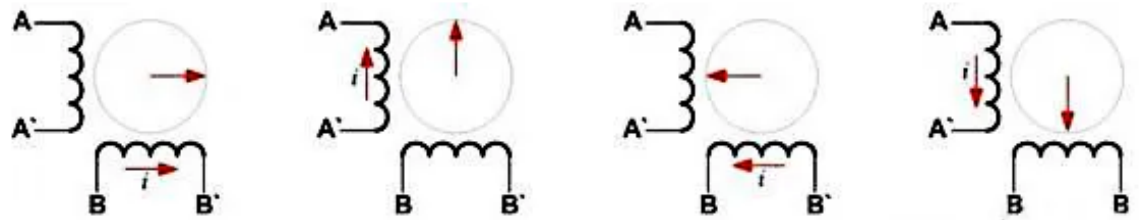


Figure 2: Full-Stepping Sequence

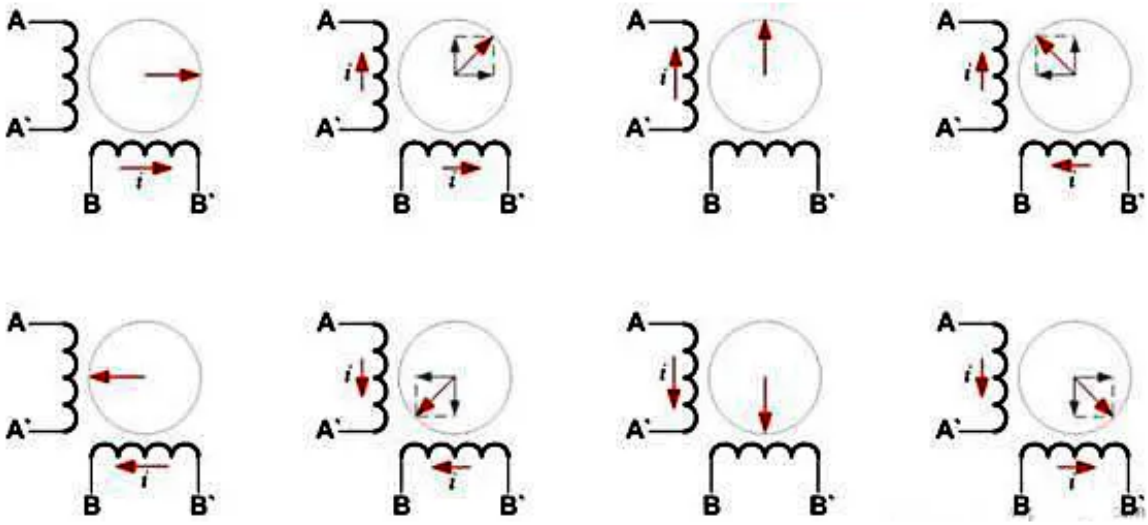


Figure 3: Half-Stepping Sequence

Note that it doesn't matter where you actually start each sequence, you just have to complete a full rotation. The sequences can also be different combinations of rotor angles, so long as they are evenly spaced (based on the number of steps) and form a complete rotation.

Splitting the rotation into just 4 or 8 steps means the rotations are not very smooth, and the motor can only be in specific positions. To fix both of these problems, stepper motors have significant gear reduction to decrease the number of rotations of the output shaft for each rotation of the input shaft. This means the differences between steps of the sequences are very small at the output, leading to smooth rotation and precise control of the position of the output shaft.

We can find the gear reduction ratio by counting the number of teeth in each of the gears between the input and output in the stepper motor. The gears of the 28BYJ-48 5v stepper motor are shown in Figure 4 below (note that some gears are out of place for better visual clarity).

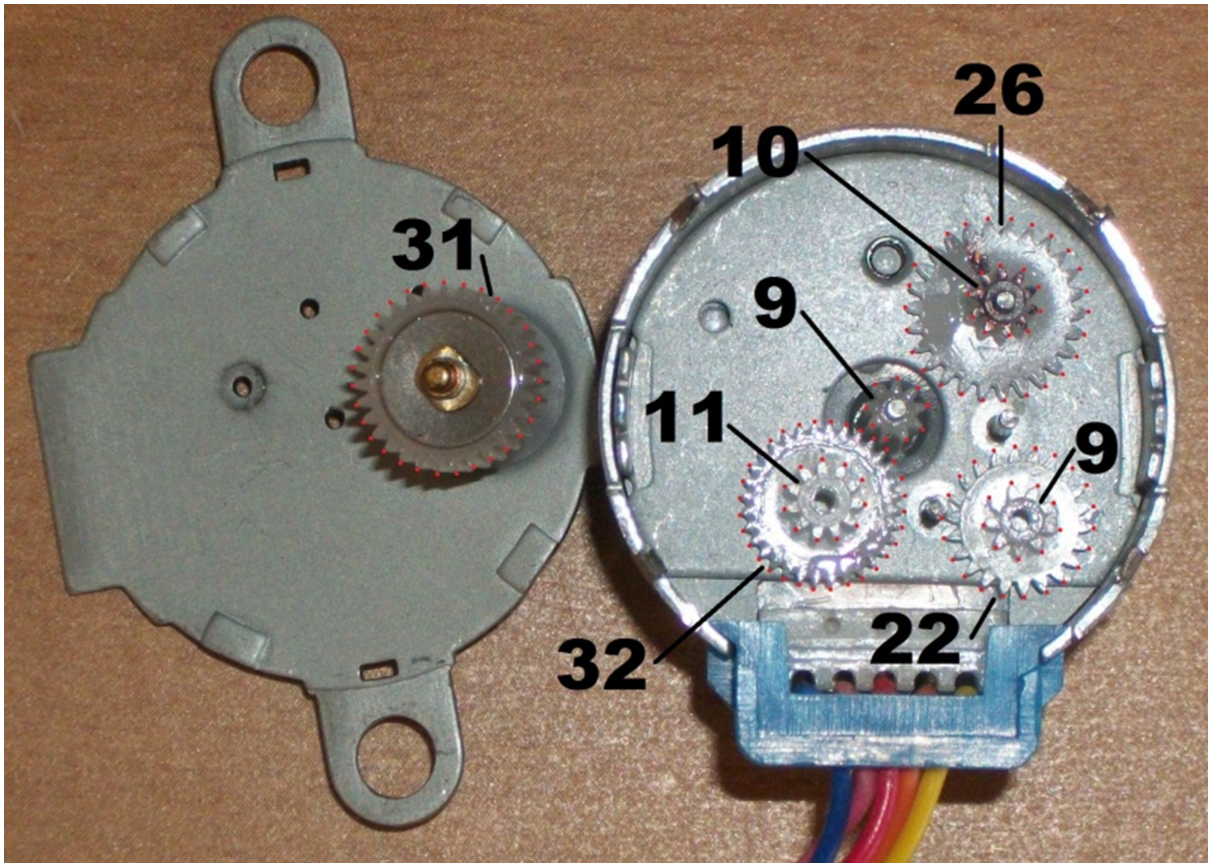


Figure 4: Internal gears of the stepper motor (image from forum.arduino.cc)

In this case, the gear ratio is:

$$\frac{31 * 32 * 26 * 22}{11 * 10 * 9 * 9} = 63.68395$$

If the output shaft rotates 1 resolution (gear with 31 teeth in the figure), the internal shaft (gear with 9 teeth in the middle) must rotate approximately 64 resolutions.

Full-stepping

- Internal motor: 32 steps per revolution
- Gear reduction ratio: $1/63.68395 \approx 1/64$
- Thus, it takes $32 * 64 = 2048$ steps, or 512 repeats of the 4-step pattern, for the output shaft to complete a rotation

Half-stepping

- Internal motor: 64 steps per revolution
- Gear reduction ratio: $1/63.68395 \approx 1/64$
- Thus, it takes $64 * 64 = 4096$ steps, or 512 repeats of the 8-step pattern, for the output shaft to complete a rotation

To rotate a specific degree, your program should do the math to convert the value in degrees to a number of steps, and then do the rotation.

2.2 Lab Exercise

Interfacing the stepper motor requires four pins, excluding +5V and ground. We will be using GPIO **PC 5**, **PC 6**, **PC 8**, and **PC 9**.

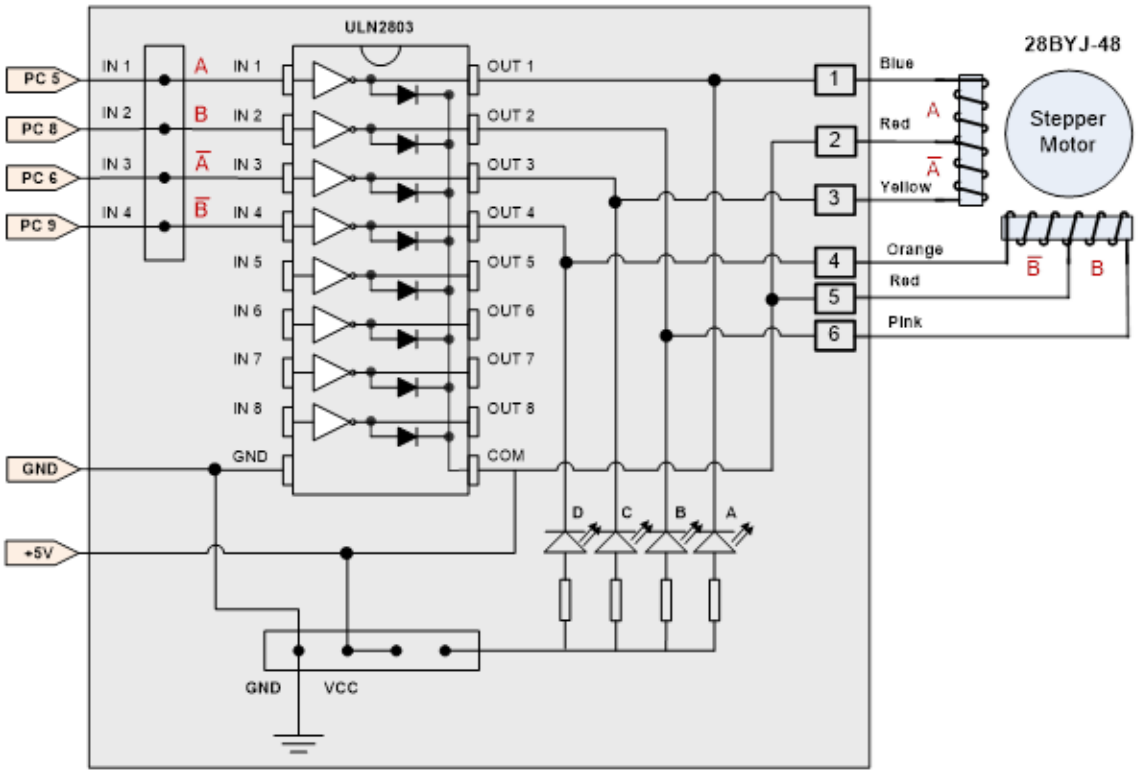
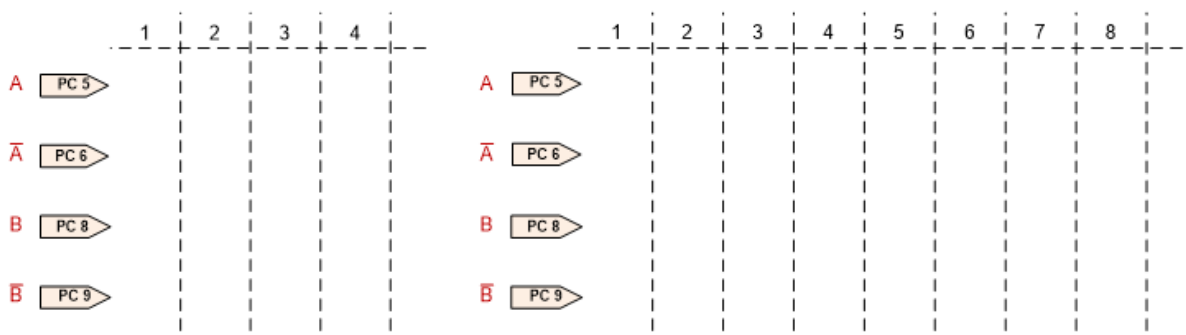


Figure 5: Connection diagram of the motor driver board (Model: 28BYJ-48).

Use Figure 5, as well as Figures 2 and 3, to sketch out below what the waveform should be on the pins for a full-stepping sequence and a half-stepping sequence.



Rotation Direction and Speed

a. How would you change the rotation speed of a stepper motor?

b. How would you reverse the rotation direction?

Warning: Motor Overheating

The motor constantly draws electrical currents and will be overheated if you leave the power on for an extended period. **Disconnect the 5V power (Vcc) to the Darlington array if you are not actively debugging/testing it.**

Use the following steps to initialize the GPIO pins **PC 5**, **PC 6**, **PC 8**, and **PC 9** on the Nucleo board. Refer to Section 9.4 in *STM32L4x6 Reference Manual* for details about registers and their bits.

1. Enable the clock for GPIO port C.
2. Configure pins 5, 6, 8, and 9 as output.
3. Set the output speed of the pins to fast.
4. Set the output type of the pins to push-pull.
5. Set the pins to no pull-up, no pull-down.

Requirements for this part of the lab
Tip: It can be hard to see the position of the motor while testing. A piece of tape stuck to the shaft can make it easier to see what direction it is pointing.

1. Rotate your stepper motor exactly 360 degrees clockwise or counter-clockwise using full-stepping.
2. Rotate your stepper motor exactly 360 degrees clockwise or counter-clockwise using half-stepping.
3. How fast can you rotate using full-stepping? (Reduce your delay loop until it no longer turns a full 360 degrees)
4. How fast can you rotate using half-stepping? (Reduce your delay loop until it no longer turns a full 360 degrees)
5. Is the highest update frequency of the half-stepping higher than that of full-stepping? Why?

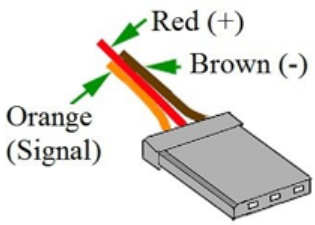
3 Part B - Servo Motor

In this part of the lab, you will control the position of the SG90 micro servo motor through various PWM duty cycles, as you previously learned in lab 3.

3.1 Introduction

The key difference between a servo motor and a stepper motor is that the servo motor is controlled by a closed-loop system that uses position feedback to perform self-adjusting. A servo motor has an internal position sensor, which continuously provides position feedback. On the contrary, a stepper motor is typically an open-loop system with no built-in position sensor.

A servo motor is preferred in applications that require higher speed. A stepper motor is often used in applications that need higher holding torque.



In this lab, we will use the SG90 servo motor. It can rotate approximately 180 degrees, with 90 degrees in each direction. The operating speed is 0.12sec/60 degrees (4.8V, no load). The motor has three wires, red wire for +5V, brown for ground, and orange for the PWM control signal.

The position is controlled by the pulse width, as shown in Figure 6 below.

- A pulse of 1.5 ms makes the motor return to the middle (0 degree)
- A pulse of 1 ms makes it rotate all the way to the left (-90 degrees)
- A pulse of 2 ms makes it rotate all the way to the right (90 degree)

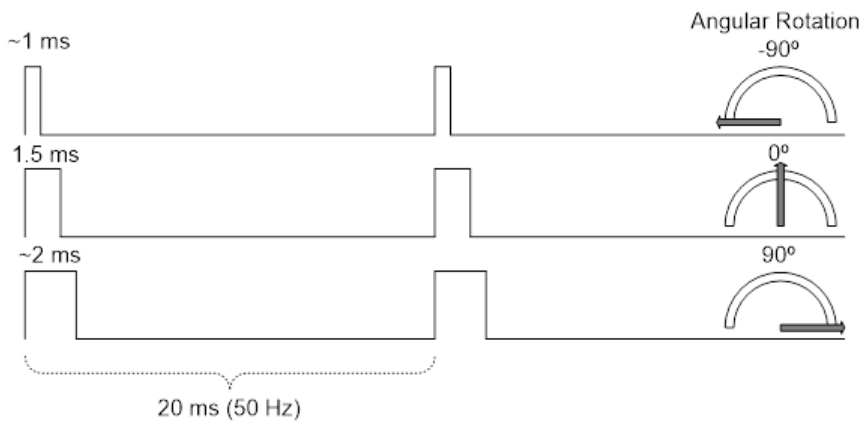


Figure 6: Position of servo motor for different duty cycles

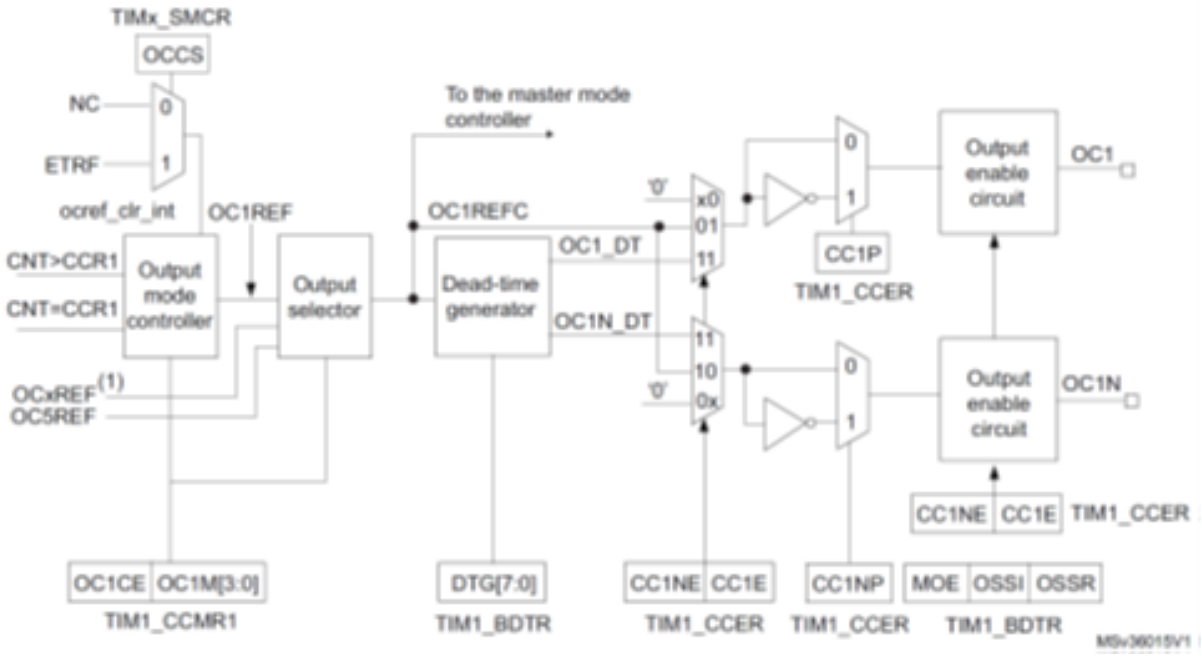


Figure 7: Timer x control for channel 1

A timer can generate an output or capture an input. The above diagram (Figure 7) shows the control of generating an output. Read the STM32L4 Reference Manual to find the details of these timer registers.

The stepper motor is connected to **PA 0**. The following table shows the Alternate functions available for this pin. In this lab, `TIM5_CH1` is selected for **PA 0**.

Peripheral	Pin	Available Alternate Functions
Servo Motor	PA 0	TIM2_CH1 / **TIM5_CH1** / TIM8_ETR / USART2_CTS / UART4_TX / SAI1_EXTCLK / TIM2_ETR / EVENTOUT

Table 1: Alternate function options for PA 0

Refer to the following table to control the outputs OCx and OCxN.

Control bits					Output states[1]	
MOE bit	**OSSI bit**	**OSSR bit**	**CCxE bit**	**CCxNE bit**	**OCx output state**	**OCxN output state**
1	X	X	0	0	Output disabled (not driven by the timer: Hi-Z) OCx=0, OCxN=0	
		0	0	1	Output disabled (not driven by the timer: Hi-Z) OCx=0	OCxREF + Polarity OCxN = OCxREF xor CCxNP
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Output Disabled (not driven by the timer: Hi-Z) OCxN=0
		X	1	1	OCREF + Polarity + dead-time	Complementary to OCREF (not OCREF) + Polarity + dead-time
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP	OCxREF + Polarity OCxN = OCxREF x or CCxNP
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Off-State (output enabled with inactive state) OCxN=CCxNP
0	0	X	X	X	Output disabled (not driven by the timer anymore). The output state is defined by the GPIO controller and can be High, Low or Hi-Z.	
	1		0	0		
			0	1	Off-State (output enabled with inactive state) Asynchronously: OCx=CCxP, OCxN=CCxNP (if BRK or BRK2 is triggered). Then (this is valid only if BRK is triggered), if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state (may cause a short circuit when driving switches in half-bridge configuration). **Note**: BRK2 can only be used if OSSI = OSSR = 1.	
			1	0		
			1	1		

1. When both outputs of a channel are not used (control taken over by GPIO), the OISx, OISxN, CCxP and CCxNP bits must be kept cleared.

3.2 Lab Exercise

In this lab we will set up a timer and use it to generate Pulse-width modulated (PWM) signals for the servo motor, similar to the Green LED in lab 3. The servo motor is connected to **PA 0**, which can function as Channel 1 of Timer 5 (`TIM5_CH1`). The requirements for this part of the lab are as follows.

1. Initialize TIM5 so the ARR period happens at 50Hz (20ms)
2. Calculate the values you need in the CCR1 register to end up with a duty cycle of 1ms, 1.5ms, and 2ms. Create a function you can call that sets the CCR1 register in this way so you can rotate to 90, 0, and -90 degrees.
3. Demo to the TA the pattern of 0 degrees, wait 1 second, move to 90 degrees, wait 1 second, move to -90 degrees, wait 1 second, then move to 0 degrees.

Note: You may find that for your servo 1ms / 2ms might not be the exact right values to rotate 90 degrees. Adjust the values until you get a good 90-degree rotation and document the values you used in the Lab demo section.

Use the following steps to configure the alternate function on PA 0

1. **Configure `RCC_AHB2ENR` to enable the clock of GPIO Port A**

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`AHB2ENR`														RNGEN		AESEN			ADCEN	OTGFSEN					GPIOPHEN	GPIOPGEN	GPIOPFEN	GPIOPEEN	GPIOPDEN	GPIOPCEN	GPIOPBEN	GPIOPAEN
Mask																																
Value																																

2. **Configure PA 0 (Servo Motor) as Alternate Function Mode**

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`MODER`	`MODE15[1:0]`		`MODE14[1:0]`		`MODE13[1:0]`		`MODE12[1:0]`		`MODE11[1:0]`		`MODE10[1:0]`		`MODE9[1:0]`		`MODE8[1:0]`		`MODE7[1:0]`		`MODE6[1:0]`		`MODE5[1:0]`		`MODE4[1:0]`		`MODE3[1:0]`		`MODE2[1:0]`		`MODE1[1:0]`		`MODE0[1:0]`	
Mask																																
Value																																

Note: Digital Input (00), Digital Output (01), Alternate Function (10), Analog (11)

3. **Configure and Select the Alternative Function for PA0**

   Set **GPIOE->AFR[0]** and **GPIOE->AFR[1]** for alternate function of Pin 0 to **TIM5_CH1**.

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`AFR[0]`	`AFRL7[3:0]`				`AFRL6[3:0]`				`AFRL5[3:0]`				`AFRL4[3:0]`				`AFRL3[3:0]`				`AFRL2[3:0]`				`AFRL1[3:0]`				**`AFRL0[3:0]`**			
Mask																																
Value																																
`AFR[1]`	`AFRH15[3:0]`				`AFRH14[3:0]`				`AFRH13[3:0]`				`AFRH12[3:0]`				`AFRH11[3:0]`				`AFRH10[3:0]`				`AFRH9[3:0]`				`AFRH8[3:0]`			
Mask																																
Value																																

4. **Configure PA 0 to No Pull-Up, No Pull-Down**
   GPIO PUPD: No Pull-Up, Pull-Down (`00`), Pull-Up (`01`), Pull-Down (`10`), Reserved (`11`)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`PUPDR`	`PUPD15[1:0]`		`PUPD14[1:0]`		`PUPD13[1:0]`		`PUPD12[1:0]`		`PUPD11[1:0]`		`PUPD10[1:0]`		`PUPD9[1:0]`		`PUPD8[1:0]`		`PUPD7[1:0]`		`PUPD6[1:0]`		`PUPD5[1:0]`		`PUPD4[1:0]`		`PUPD3[1:0]`		`PUPD2[1:0]`		`PUPD1[1:0]`		`PUPD0[1:0]`	
Mask																																
Value																																

5. **Configure the PWM output for Channel 1 of Timer 5**

- You do not have to work on the table directly.
- Sometimes, the output is `CH1N`, not `CN1`. For example, when the output PA7 can be `TIM1_CH1N`. If so, you need to set the `CC1NE` bit in the `TIMx_CCER` register. Refer to the table given on Page 2 of the lab handout for controlling `OC1N`.

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	CR1	Reserved																				UIFREMAP		CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
	Value																																
0x04	CR2	Reserved								MMS2[3:0]					OIS6		OIS5		OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	TI1S	MMS[2:0]			CCDS	CCUS		CCPC
	Value																																
0x08	SCMR	Reserved															SMS[3]	ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			OCCS	SMS[2:0]		
	Value																																
0x0C	DIER	Reserved																	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	Value																																
0x10	SR	Reserved														C6IF	C5IF			SBIF	CC4OF	CC3OF	CC2OF	CC1OF	B2IF	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
	Value																																
0x14	EGR	Reserved																							B2G	BG	TG	COM	CC4G	CC3G	CC2G	CC1G	UG
	Value																																
0x18	CCMR1 Output Compare Mode	Reserved							OC2M[3]	Reserved							OC1M[3]	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
	Value																																
	CCMR1 Input Capture Mode	Reserved																IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
	Value																																

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	CCMR2 Output Compare Mode	Reserved							OC4M[3]	Reserved							OC3M[3]	OC2CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]		OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
	Value																																
	CCMR2 Input Capture Mode	Reserved																IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]		IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
	Value																																
0x20	CCER	Reserved										CC6P	CC6E			CC5P	CC5E		CC4P	CC4E		CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
	Value																																
0x24	CNT	UIFCPY	Reserved															CNT[15:0]															
	Value																																
0x28	PSC	Reserved																PSC[15:0]															
	Value																																
0x2C	ARR	Reserved																ARR[15:0]															
	Value																																
0x34	CCR1	Reserved																CCR1[15:0]															
	Value																																
0x38	CCR2	Reserved																CCR2[15:0]															
	Value																																
0x3C	CCR3	Reserved																CCR3[15:0]															
	Value																																
0x40	CCR4	Reserved																CCR4[15:0]															
	Value																																
0x44	BDTR	Reserved						BK2P	BK2E	BK2F[3:0]				BKF[3:0]				MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[2:0]		DT[7:0]							
	Value																																
0x48	DCR	Reserved																			DBL[4:0]								DBA[4:0]				
	Value																																
0x4C	DMAR	Reserved																DMAB[15:0]															
	Value																																
0x50	OR1	Reserved																												TI1_RMP		ETR13_RMP	ITR1_RMP
	Value																																
0x60	OR2	Reserved															ETRSEL[2:0]			Reserved													
	Value																																

Setting up the TIM5 timer has a lot more options than the SysTick timer. You can find the settings in Chapter 30 of the STM manual.

You will need to set the following fields. Write the values to mask/set. If no mask is needed you can leave that section blank. You can use pre-defined names for the bits rather than raw hex values.

- Enable clock for **TIM5**
- Set **TIM5->CR1** for the counting direction to be up.
- We will use a 4MHz MSI clock for this lab. Set the prescaler **TIM5->PSC** to count at 100kHz.
- Set the **TIM5->ARR** register to a value that will give a period of 0.02s (50Hz)
- Clear the **OC1M** field in the **TIM5->CCMR1** register and select PWM Mode 1 (**OC1M = 110**):
- Enable the Output 1 preload enable in **TIM5->CCMR1**:
- Enable output of Channel 1 (**CH1**) by setting the **CC1E** bit in **TIM5->CCER**:
- Set the output compare register for channel 1 **TIM5->CCR1** to have a duty cycle of 50%:
- Enable counter by setting **CEN** bit in **TIM5->CR1**.

Note that we just set the duty cycle to 50% during initialization. To change the position of the servo motor, we change the duty cycle according to Figure 6. Calculate the values of CCR1 for each position to complete the table. As stated before, these may not be the values that rotate your servo motor 90° exactly, but they provide a good starting point.

Servo Motor Position	Pulse Width	Value of **TIM5->CCR1**
−90°	1 ms	
0°	1.5 ms	
90°	2 ms	

4 References

[1] STM32L4x6 Advanced ARM-based 32-bit MCUs Reference Manual

[2] "What is the difference between full-stepping, the half-stepping, and the micro-drive?," Automate. [Online]. Available: https://www.automate.org/case-studies/what-is-the-difference-between-full-stepping-the-half-stepping-and-the-micro-drive.

[3] Yifeng Zhu, "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C", ISBN: 0982692633