BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 416 – Microprocessor and Embedded Systems Laboratory
January 2023

Experiment 03

Arm Cortex M: Timers and Interrupts I: PWM, Musical Note Generation, EXTI

Evaluation Form:

IMPORTANT! You must complete this experiment during your scheduled lab period. All work for this experiment must be demonstrated to and verified by your lab instructor *before the end* of your scheduled lab period.

STEP	DESCRIPTION	MAX	SCORE
1	Blinking an LED with timer (Section 3)	10	
2	Fading LED with PWM (Section 4)	10	
3	Music Generation with timer Output compare (Section 5)	30	
4	LED Control with Push Button using External Interrupt (Section 6-7)	30	
5	Answer all questions	20	
	TOTAL	100	

Signature of Evaluator: ______________________________

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. *You will not receive credit for this lab experiment unless this statement is signed in the presence of your lab instructor.*

"In signing this statement, I hereby certify that the work on this experiment is my own and that I have not copied the work of any other student (past or present) while completing this experiment. I understand that if I fail to honor this agreement, I will receive a score of ZERO for this experiment and be subject to possible disciplinary action."

Last Name (Printed): ____________________ **Lab Group:** ____ **Date:** __________

E-mail: ________________@eee.buet.ac.bd **Signature:** ____________________

Contents

1 Introduction

Lab overview

At the end of this lab, you will be able to:

- Use Timers of STM32 NUCLEO-L446RE
- Blink LED using Timer Output Compare
- Fade LED using PWM
- Create music with speaker and Timer Output Compare
- Control Servo Motor with PWM

2 Pre-lab Study

Before attempting this lab, please do the following:

1. Make sure you have completed tasks for Experiment 06
2. Study:
    - Read [Zhu] Textbook **Chapter 15 General Purpose Timers (Section 15.1-15.3).**
    - Read [Zhu] Textbook **Chapter 11 Interrupts (Section 11.1-11.5)**
    - Watch YouTube Tutorials (http://web.eece.maine.edu/~zhu/book/tutorials.php)
        - Lecture 13. Timer PM output https://youtu.be/zkrVHIcLGww (17 minutes)
        - Lecture 12: System Timers https://www.youtube.com/watch?v=aLCUDv_fgoU(8 minutes)
        - Watch YouTube Tutorials on Interrupts (https://youtu.be/uFBNf7F3l60)
        - Watch YouTube Tutorials on External interrupts (https://youtu.be/uKwD3JuRWeA)

f_cnt=fsys/(1+psc)

t_period of counter =2 sec= (1+ARR)* T_clk

3 LED Blink with Timer Output Compare

The green LED is connected to the GPIO pin PA 5 The following table shows the Alternate functions available for the pin. PA 5 can work as TIM2_CH1, TIM2_ETR, TIM8_CH1N, or LPTIM2_ETR. In this lab, TIM2_CH1 is selected for PA 5.

Peripheral	Pin	Available Alternate Functions
Green LED	PA 5	TIM2_CH1/TIM2_ETR/TIM8_CH1N/SPI1_SCK/LPTIM2_ETR/EVENTOUT

For our example code, by default, system runs at 4MHz clock speed. What is the prescalar value if a timer needs to driven by a clock of 4kHz (Hint: $f_{CLK\_CNT} = f_{CLK\_PSC} / (PSC+1)$? Write below

```
TIM2->PSC =
```

What will be the value of ARR to turn an LED on for 2 seconds and off for 2 seconds?

```
TIM2->ARR =
```

Modify code for 7.3 to blink an LED with 2 seconds on and 2 seconds off: do not initialize system clock (let it run at default 4MHz) Use Timer 2 Channel 1.

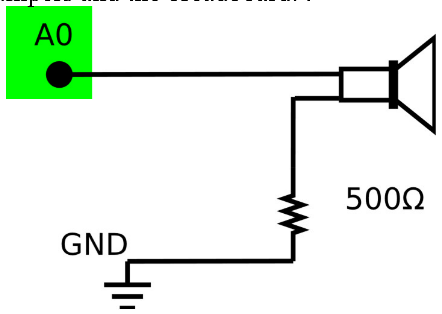
4 LED Fading with PWM

The code given in 7.4 gradually increases brightness of the LED and then abruptly changes brightness to zero. Modify the code, so that it gradually increases brightness, and decreases brightness. turns it off Modify code for 7.4, so that the LED to blink an LED with 2 seconds on and 2 seconds off: do not initialize system clock (let it run at default 4MHz) Use Timer 2 Channel 1.

5 Music Generation with Timer

In this lab, we will generate simple music with a timer output compare function, similar to the previous section. We will use only square waves to generate the music. Later using DAC, we will learn to make more accurate notes.

Please connect the speaker with the Nucleo board using the following circuit diagram. Use male-to-male jumpers and the breadboard. :



Warning: **DO NOT connect the speaker directly to the board without a series resistor**. Doing so will damage the speaker. Always keep atleast 330Ω resistance in series.

A bit of music theory:

When a piano string vibrates, it vibrates at a very specific frequency related to its length and tension, and by changing the tension we can choose exactly what frequency each string vibrates at – the same principle applies to all musical instruments, including drums. So, every note on the piano keyboard has a corresponding frequency. For example, 110 Hz is A2 on the musical scale. If we double the frequency of a note, we get 1 octave above its note, for example, A3 note's frequency is 220Hz, which is 1 octave above A2, and A4 is 440Hz, 1 octave above A3. Between any octave, the notes are divided into 12 notes spaced in frequency logarithmically. Starting from C the notes are written as:

C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab, A, A#/Bb, B,

C# (pronounced "C sharp") and Db (pronounced D flat) represents the same note.

Starting at any note the frequency to other notes may be calculated from its frequency by:

Freq = note x $2^{N/12}$,

where N is the number of notes away from the starting note. N may be positive, negative or zero.

For example, D and F is separated by 3 frequency intervals. Starting at D (146.84 Hz), the frequency to the next higher F is:

146.84 x $2^{3/12}$ = 174.62

(to see how frequencies of individual notes are calculated, please watch this video Frequencies of Musical Notes https://www.youtube.com/watch?v=oi1tY8HsKps

The chart below shows musical frequency at different octaves. Seven notes without the sharp (♯) or flat (♭), eg C D E F G A B make up a major scale, and can be used to construct many recognizable tunes.

130.81*2^(8/12)

E	OCTAVE 2	OCTAVE 3	OCTAVE 4	OCTAVE 5
C	65.41 Hz	130.81 Hz	261.63 Hz	523.25 Hz
C#/Db	69.30 Hz	138.59 Hz	277.18 Hz	554.37 Hz
D	73.42 Hz	146.83 Hz	293.66 Hz	587.33 Hz
D#/Eb	77.78 Hz	155.56 Hz	311.13 Hz	622.25 Hz
E	82.41 Hz	164.81 Hz	329.63 Hz	659.25 Hz
F	87.31 Hz	174.61 Hz	349.23 Hz	698.46 Hz
F#/Gb	92.50 Hz	185.00 Hz	369.99 Hz	739.99 Hz
G	98.00 Hz	196.00 Hz	392.00 Hz	783.99 Hz
G#/Ab	103.83 Hz	207.65 Hz	415.30 Hz	830.61 Hz
A	110.00 Hz	220.00 Hz	440.00 Hz	880.00 Hz
A#/Bb	116.54 Hz	233.08 Hz	466.16 Hz	932.33 Hz
B	123.47 Hz	246.94 Hz	493.88 Hz	987.77 Hz

We can declare an array of integers with frequency of each note in Hz.

```
static uint32_t note_freq[8] = {261, 294, 329, 349,392, 440, 494, 522};
```

Here, note_freq[0] represents the C4 note, and the note_freq[7] represents the C5 note. We can easily calculate the frequency of an octave above or below, by simply multiplying or dividing the frequency by 2.

The auto-reload value of the timer 5 can be calculated for any particular **note_freq[n]** where, **n** is a predeclared integer value between 0 and 7, **tfreq** is the frequency of the timer, **psc** is the prescale value stored in **TIM5->PSC**

```
TIM5->ARR = (tfreq/psc / note_freq[song_notes[current_note]] ) - 1UL;
```

Here, note_freq[0] represents the C4 note, and the note_freq[7] represents the C5 note. We can easily

Problem: Compile and load the project given for Exp 3.5, and run it in the Nucleo board pressing reset. Construct the circuit diagram and connect the board. Which song does the code represent? (Write the name below in Bangla)

The following notes are for the Bangla song একটি বাংলাদেশ, তুমি যাগ্রত জনতার. Each note is played for equal time duration. Note that the notes are spread into 3 octaves and have flat/sharp notes. Modify the given code to play the song using the Nucleo board.

```
F4  F4  F4  F4  A4  G4  F4  F4  F4  F4
E4  F4  G4  G4  F4  E4  D4  E4  E4  C3  C3  C3..
G3  A3  B♭3 B♭3 D4  D4  F4  F4  B♭4 B♭4 B♭4 B♭4
A3  G3  G3  A3  G3  F3  E3  G3  F3  F3  F3  F3
```

6 Introduction to Interrupt

An interrupt is a signal sent by the hardware or software processes calling for the immediate attention of the CPU. Once the CPU receives this signal, it stops whatever it is doing and takes care of the request. When the interrupt signal is activated, the microcontroller branches to a program called interrupt service routine. The microcontroller then executes the subroutine. After the completion of ISR, the microcontroller returns back to the main program it was executing earlier.

Interrupts can be **Internal** or **External.** Internal interrupts can be come from Timers and External interrupts can come from external devices for example a push button or a keypad. In the first part of this lab, we'll look into external interrupts and how to configure them.

4 ta port er total 16 ta kore je pin sobar jonoi interrupt generate kora jabe

7 External Interrupt

Each STM32F4 device has 23 external interrupt or event sources and are split into 2 sections. First interrupt section is for external pins (P0 to P15) on each port and other section is for other events like RTC interrupt, Ethernet interrupt.
Each pin of ports are connected to an external interrupt configuration register (SYSCFG_EXTICR). When a particular pin is configured as an interrupt, the SYSCFG_EXTICR will generate an interrupt request at the rising or falling edge of the voltage signal. This interrupt request will be sent to the NVIC (Nested-Vectored Interrupt Controller). The external interrupt controller supports 16 external interrupts, associated with GPIO pins and are named as external interrupt 0 to external interrupt 15. All pins with the same number are connected to line with the same number. They are multiplexed to one line. For example, the source of external interrupt 3 can be PA3, PB3, PC3 etc. SYSCFG_EXTICR specifies which pin will be selected as an external interrupt source. Multiple pins from a group cannot be used as an external interrupt source simultaneously. For example, PA0, PB0 and PC0 cannot be used simultaneously as the source for external interrupt 0 but PA0 and PA5 can be used at the same time as they are connected to different pins.

7.1 Interrupt Vector Table:

After receiving an interrupt, the microcontroller will go to the interrupt vector table to fetch the ISR. There are 7 interrupt handlers in STM32F4. They are given in the following table:

Interrupt Number	Interrupt Handler	Description
EXTI0_IRQn	EXTI0_IRQHandler	Handlers for pins connected to External Interrupt 0
EXTI1_IRQn	EXTI1_IRQHandler	Handlers for pins connected to External Interrupt 1
EXTI2_IRQn	EXTI2_IRQHandler	Handlers for pins connected to External Interrupt 2
EXTI3_IRQn	EXTI3_IRQHandler	Handlers for pins connected to External Interrupt 3
EXTI4_IRQn	EXTI4_IRQHandler	Handlers for pins connected to External Interrupt 4
EXTI9_5_IRQn	EXTI9_5_IRQHandler	Handlers for pins connected to External Interrupt 5 to 9
EXTI15_10_IRQn	EXTI15_10_IRQHandler	Handlers for pins connected to External Interrupt 10 to 15.

This table shows which IRQ we need to set for NVIC (1st column) and function names to handle the interrupts (2nd column). And external interrupt 0 to 4 have their own interrupt handler. External interrupts from number 5 to 9 share the same interrupt handler, EXTI9_5 and External interrupts from number 10 to 15 share the same interrupt handler, EXTI_15_10, IRQ handler.

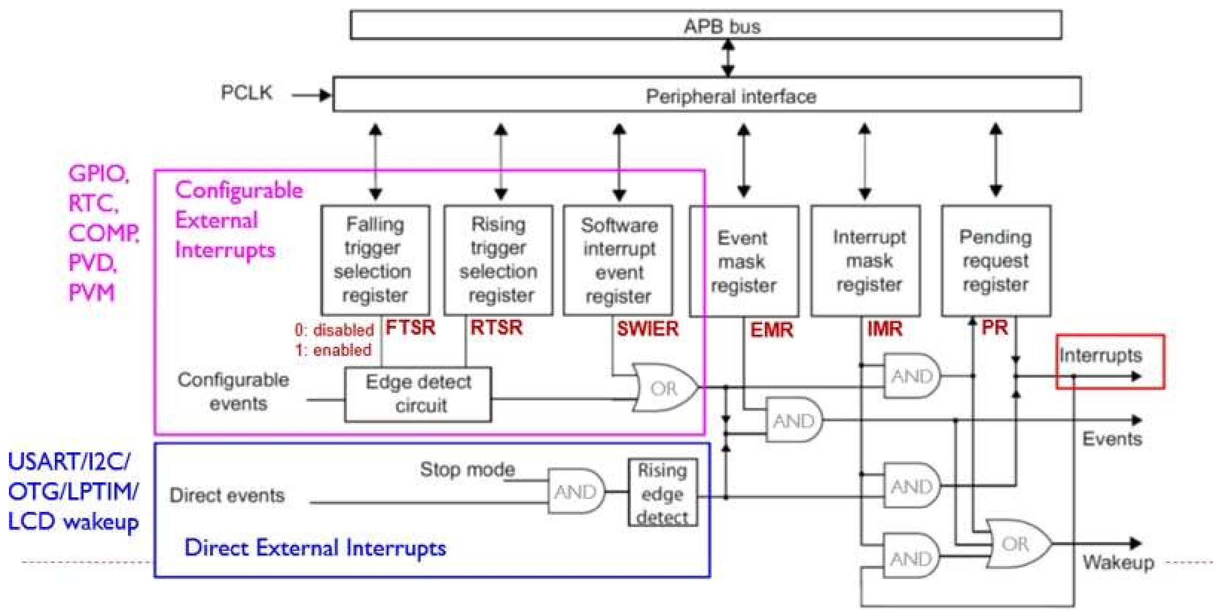


Fig 1: External Interrupt (EXTI) Controller

7.2 Registers associated with the External Interrupts:

The controller supports two types of EXTI: configurable EXTI and direct EXTI. Configurable EXTI include those associated with GPIO, RTC, comparators, power voltage detector and peripheral voltage monitoring. Direct EXTI are mainly used for communication peripherals, lower power time and LCD. Configurable EXTI can be triggered by rising, falling or both edges. Additionally, software can generate an interrupt request by writing 1 to the SWIER. But Direct EXTI can be triggered by rising edge alone. For generating an interrupt, if the corresponding bit in the interrupt mask register is 1. Now we'll look into how to configure external interrupt using software.

Problem:

In this problem we'll use the built-in Push Button as an external interrupt. First we'll try a simple exercise. The LED will be always on. Whenever the push-button is pressed the LED will be turned off. The LED Pin configuration is the same as before. We'll see how to configure the Push Button as an external interrupt.

- Write a function named **EXTI_INIT**(void) to configure the push button as external interrupt.

```
RCC -> AHB1ENR |=______________________// Enable the clock for Push

Button. GPOIC -> MODER &=______________// GPIO Mode configuration as

inputs GPIOC -> PUPDR &=_________________// No pull-up, no pull down
```

```
// External Interrupt Enable

NVIC_EnableIRQ(???) // Interrupt Number associated with Push button Pin.

NVIC_SetPriority( Number of External Interrupt Pin, 0) // 0 for highest priority

// Connect External Line to the GPIO

RCC -> APB2ENR |= RCC_APB2ENR_SYSCFGEN // To enable the clock of
SYSCFGEN.
```

To configure the external interrupt, we need External Interrupt Configuration Register

(EXTICR). These EXTICR are divided into 4 groups.

- SYSCFG_EXTICR[0] (EXTI0 to EXTI3)
- SYSCFG_EXTICR[1] (EXTI4 to EXTI7)
- SYSCFG_EXTICR[2] (EXTI8 to EXTI1)
- SYSCFG_EXTICR[3] (EXTI12 to EXTI15)

```
SYSCFG->EXTICR[??] &=
~SYSCFG_EXTICR4_EXTI13; SYSCFG->EXTICR[??]
```

Now we need to configure the interrupt mask register to make the interrupt not-masked.

```
EXTI->IMR |= EXTI_IMR_IM13; // 0 = masked 1 = not masked (enabled)
```

We could also use software to write 1 in that pin by using SWIER1 pin. As mentioned earlier, we can use rising trigger or falling trigger interrupt mode. In this case we'll use falling trigger.

```
EXTI->FTSR |= EXTI_FTSR_TR13; // 0 = disabled, 1 = enabled
```

For rising trigger,

```
EXTI->RTSR1 |= EXTI_RTSR1_RT13; // 0 = disabled, 1 = enabled.
```

We have completed the configuration of Push Button as external interrupt. Now we need to write the Interrupt Service Routine or Interrupt Handler. This is basically a function that will tell the microcontroller what to do if the interrupt is received.

```
static void EXTI15_10_IRQHandler (void) {
      NVIC_ClearPendingIRQ(???) // Number of the
      interrupt
      // PR: Pending Register
      if ((EXTI->PR & EXTI_PR_PR13) == EXTI_PR_PR13) {

           // cleared by writing a 1 to this bit
          EXTI->PR |= EXTI_PR_PR13; // This ensures that the interrupt has been served

          while(1)
         {
               turn_off_LED();
          }
}

}
```

Lab Exercise

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. You will demo to the class instructor 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.
4. 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.
5. Write a program that will do the following task:
   a. When a push button is not pressed, the Stepper motor will run counter-clockwise in full-step.
   b. When a push-button is pressed, the microcontroller will receive an interrupt signal and the LED will start blinking.

8 Question and Answer

Read the relevant chapter of the textbook and answer the following questions:

8.1 PWM

Suppose the 8-MHz MSI is selected as the input clock of timer.

1. To generate 1 Hz square wave with a duty cycle of 50%, how should we set up the timer? Indicate your counting mode and show the value of **ARR**, **CRR**, and **PSC** registers.

2. What is the smallest PWM frequency that can be generated?

8.2 Music

Read section 21.8.2 in your book. For our example in lab, we assumed each note is played for same duration. In reality, a note duration can be variable. Modify the C code so that it can play Twinkle Twinkle Little Stars and Happy Birthday to You as defined in Example 21-9. You can use the DCD values as a static array in C. Write your code below

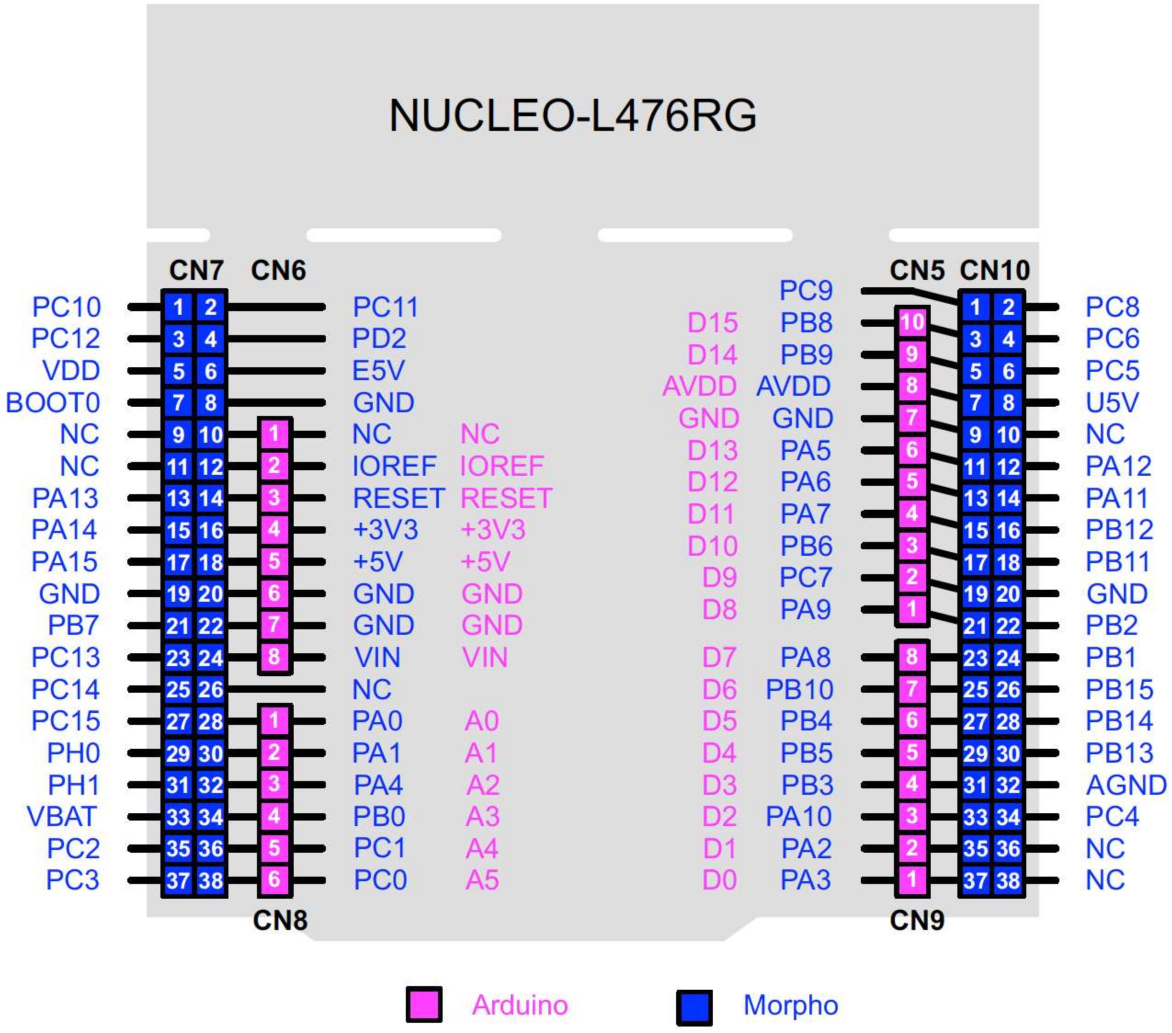
9 Code Printout

After this page, please attach printout of the main codes for each section. Use monospace fonts (Such as `Noto Mono`) for your codes. Use font size 8. Paste them in Microsoft Word. Give appropriate heading for each problem (Such as *Lab Exercise of Section 3*, etc).

Appendix A: Pin Connections on Nucleo-64 board

This diagram is common for both Nucleo-L476RG and Nucleo-L446RE

Board Component	**Microcontroller Pin**	**Comment**
Green LED	`PA 5`	SB42 closed and SB29 open by default
Blue user button	`PC 13`	Pulled up externally
Black reset button	`NRST`	Connect to ground to reset
ST-Link UART TX	`PA 2`	STLK_TX
ST-Link UART RX	`PA 3`	STLK_RX
ST-Link SWO/TDO	`PB 3`	Trace output pin/Test Data Out pin
ST-Link SWDIO/TMS	`PA 13`	Data I/O pin/Test Mode State pin
ST-Link SWDCLK/TCK	`PA 14`	Clock pin/Test Clock pin

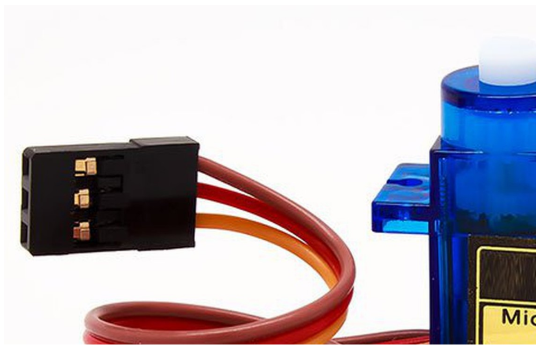


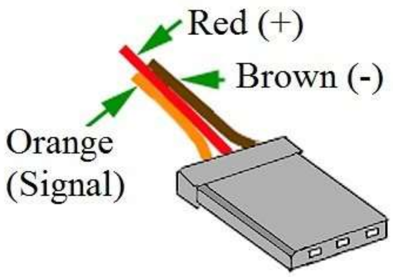
Appendix B: Servo Motor

The green LED is connected to the GPIO pin PA 5 The following table shows the Alternate functions available for the pin. PA 5 can work as TIM2_CH1, TIM2_ETR, TIM8_CH1N, or LPTIM2_ETR. In

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. However, a stepper motor typically is an open-loop system which 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.

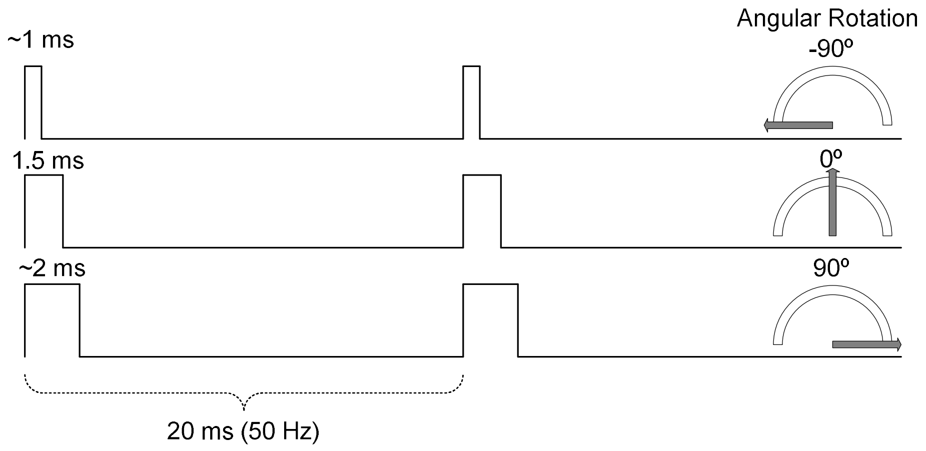




In this lab, we will use SG90 servo motor. It can rotate approximately 180 degrees, with 90 degrees in each direction. The operating speed is 0.12sec/60 degrees (4.8V, no load). The motor has three wires, red wire for +5V, brown for ground, and orange for the PWM control signal.

The position is controlled by the pulse width, as shown 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)



For system running at 4MHz clock, suppose the PWM signal period is 20 *ms* (50 Hz).

Fill up the following values:

`TIM5->PSC` = ____________________________

`TIM5->ARR` = ______________________________

Calculate the value of **`CCR1`** and complete the following table:

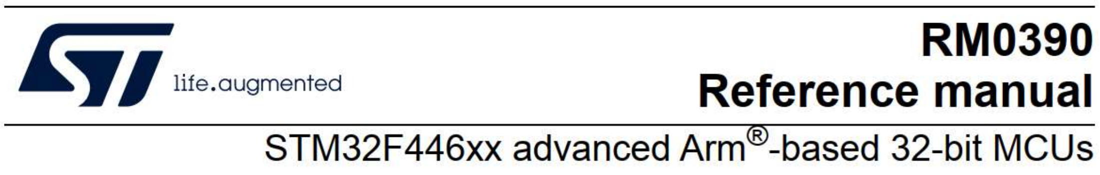
Servo Motor Position	Pulse Width	Value of **`TIM1->CCR1`**
-90˚	1 ms	
0˚	1.5 ms	
90˚	2 ms	

Appendix C: Acknowledgement and References

The labsheet is prepared by **Dr. Sajid Muhaimin Choudhury**, Dept of EEE, BUET, on 19/06/2023

The labsheet is based on Lab Materials provided as Instuctor Suppliment of "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, Third Edition." By Dr. Yifeng Zhu,. The Labsheets are modified from STM32L4 architecture to STM32F4 architecture. Significant portions are added for music generation including custom circuit sourced from local Bangladeshi market.

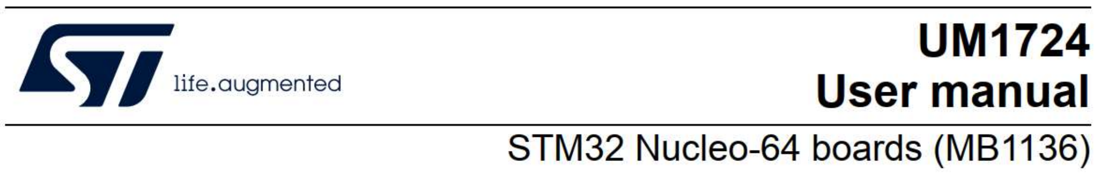
The following documents are strongly recommended as reference:


life.augmented

RM0390
Reference manual

STM32F446xx advanced Arm®-based 32-bit MCUs

https://www.st.com/resource/en/reference_manual/rm0390-stm32f446xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

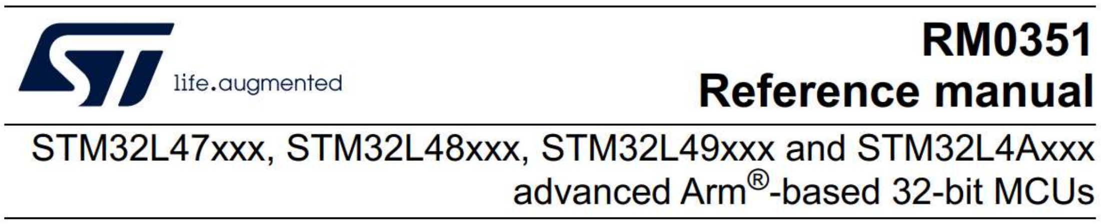

life.augmented

UM1724
User manual

STM32 Nucleo-64 boards (MB1136)

https://www.st.com/resource/en/user_manual/dm00105823-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf

The following Document is recommended for Yifeng Zhu's book literature


life.augmented

RM0351
Reference manual

STM32L47xxx, STM32L48xxx, STM32L49xxx and STM32L4Axxx advanced Arm®-based 32-bit MCUs

https://www.st.com/resource/en/reference_manual/rm0351-stm32l47xxx-stm32l48xxx-stm32l49xxx-and-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf