Necessary Codes:

Code of blinking LED with 2s on and 2s off using Timer2 channel1 (in toggle mode):

```
#include "stm32f446xx.h"
#define SPEAKER_PORT GPIOA
#define SPEAKER_PIN  0

#define LED_PORT GPIOA
#define LED_PIN    5

#define BUTTON_PIN 13

#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                              This value must be a multiple of 0x200. */

static void LED_Pin_Init(){
        RCC->AHB1ENR          |= RCC_AHB1ENR_GPIOAEN;          // Enable GPIOA clock

        // Set mode as Alternative Function 1
                LED_PORT->MODER  &= ~(0x03 << (2*LED_PIN));                  // Clear bits
                LED_PORT->MODER  |=   0x02 << (2*LED_PIN);                   // Input(00), Output(01),
//AlterFunc(10), Analog(11)

                LED_PORT->AFR[0] &= ~(0xF << (4*LED_PIN));     //        AF 1 = TIM2_CH1
                LED_PORT->AFR[0] |=   0x1 << (4*LED_PIN);      //        AF 1 = TIM2_CH1

                //Set I/O output speed value as very high speed
                LED_PORT->OSPEEDR  &= ~(0x03<<(2*LED_PIN));
                LED_PORT->OSPEEDR  |=   0x03<<(2*LED_PIN); // Very high speed(11)
                //Set I/O as no pull-up pull-down
                LED_PORT->PUPDR    &= ~(0x03<<(2*LED_PIN));  // No PUPD(00, reset),      //Pullup(01),
//Pulldown(10), Reserved (11)
                //Set I/O as push pull
}

static void TIM2_CH1_Init(){

   RCC->APB1ENR         |= RCC_APB1ENR_TIM2EN;              // Enable TIMER clock

                // Counting direction: 0 = up-counting, 1 = down-counting
   TIM2->CR1 &= ~TIM_CR1_DIR;// 0 is set for upcounting

   TIM2->PSC = 4000-1;
   TIM2->ARR = 7999-1;
TIM2->CCMR1 &= ~TIM_CCMR1_OC1M;  // Clear ouput compare mode bits for channel 1
   TIM2->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1; // OC1M = 0011
                                                        //0011 means toggle mode
   TIM2->CCMR1 |= TIM_CCMR1_OC1PE;              // Output 1 preload enable

                // Select output polarity: 0 = active high, 1 = active low
                TIM2->CCER &= ~TIM_CCER_CC1NP; // select active high
```

```
    // Enable output for ch1
                    TIM2->CCER |= TIM_CCER_CC1E;

    // Main output enable (MOE): 0 = Disable, 1 = Enable
                    TIM2->BDTR |= TIM_BDTR_MOE;

                    //TIM2->CCR1  = 500;        // Output Compare Register for channel 1
                    TIM2->CR1  |= TIM_CR1_CEN; // Enable counter
}

int main(void){

// Default system clock 16 MHz

          TIM2_CH1_Init();
          LED_Pin_Init();
          while(1);
}
```

Code for controlling brightness of LED (In PWM mode1):

```
#include "stm32f446xx.h"
#define SPEAKER_PORT GPIOA
#define SPEAKER_PIN  0

#define LED_PORT GPIOA
#define LED_PIN    5

#define BUTTON_PIN 13

#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                              This value must be a multiple of 0x200. */


static void LED_Pin_Init(){
          RCC->AHB1ENR          |= RCC_AHB1ENR_GPIOAEN;             // Enable GPIOA clock

          // Set mode as Alternative Function 1
                    LED_PORT->MODER  &= ~(0x03 << (2*LED_PIN));                    // Clear bits
                    LED_PORT->MODER  |=   0x02 << (2*LED_PIN);      // Input(00), //Output(01), AlterFunc(10),
//Analog(11)

                    LED_PORT->AFR[0] &= ~(0xF << (4*LED_PIN));    // AF 1 = TIM2_CH1
                    LED_PORT->AFR[0] |=   0x1 << (4*LED_PIN);        //          AF 1 = TIM2_CH1

                    //Set I/O output speed value as very high speed
                    LED_PORT->OSPEEDR  &= ~(0x03<<(2*LED_PIN));                              // Speed mask
                    LED_PORT->OSPEEDR  |=   0x03<<(2*LED_PIN);
          // Very high speed
                    //Set I/O as no pull-up pull-down
```

```
                LED_PORT->PUPDR     &= ~(0x03<<(2*LED_PIN));                    // No //PUPD(00, reset),
//Pullup(01), Pulldown(10), Reserved (11)
                //Set I/O as push pull
          //LED_PORT->OTYPER   &=  ~(1<<LED_PIN) ; // Push-Pull(0, reset), Open-Drain(1)
}


static void TIM2_CH1_Init(){

   RCC->APB1ENR         |= RCC_APB1ENR_TIM2EN;       // Enable TIMER clock

                // Counting direction: 0 = up-counting, 1 = down-counting
                TIM2->CR1 &= ~TIM_CR1_DIR;

   TIM2->PSC =4000-1;
   TIM2->ARR = 8000-1; //make sure to set the PWM freq<100 Hz
   TIM2->CCMR1 &= ~TIM_CCMR1_OC1M;  // Clear ouput compare mode bits for channel 1
   TIM2->CCMR1 |= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2; // OC1M = 110 for
                                                                    //110 for PWM mode 1
     TIM2->CCMR1 |= TIM_CCMR1_OC1PE;   // Output 1 preload enable

                // Select output polarity: 0 = active high, 1 = active low
                TIM2->CCER &=  ~TIM_CCER_CC1NP; // select active high

   // Enable output for ch1
                TIM2->CCER |= TIM_CCER_CC1E;

   // Main output enable (MOE): 0 = Disable, 1 = Enable
                TIM2->BDTR |= TIM_BDTR_MOE;
                TIM2->CCR1  = 500;        // Output Compare Register for channel 1
                TIM2->CR1  |= TIM_CR1_CEN; // Enable counter
}

static int brightness = 1;

int main(void){
                int i;
                int n = 1;


// Default system clock 16 MHz

          LED_Pin_Init();
          TIM2_CH1_Init(); // Timer to control LED

          while(1){
                    if (brightness<1000)
                              {
                                        if (brightness>0)
                                                  goto label;
                                        else
                                                  n=0-n;
                              }

                              else
                                        n=0-n;
```

```
                        label:
                        brightness+=n;
                        TIM2->CCR1 = brightness;        // set brightness for channel 1
                        for(i=0;i<10000;i++);           // delay*/

        }
}
```

Codes for generating music with Timer:

```
#include "stm32f446xx.h"
#define SPEAKER_PORT GPIOA
#define SPEAKER_PIN  0

#define LED_PORT GPIOA
#define LED_PIN    5
#define BUTTON_PIN 13

#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                              This value must be a multiple of 0x200. */
static uint16_t mask;

static void enable_HSI(){
        // Enable High Speed Internal Clock (HSI = 16 MHz)
        RCC->CR |= ((uint32_t)RCC_CR_HSION);
        while ((RCC->CR & RCC_CR_HSIRDY) == 0); // Wait until HSI ready

        // Store calibration value
        PWR->CR |= (uint32_t)(16 << 3);

        // Reset CFGR register
        RCC->CFGR = 0x00000000;

        // Reset HSEON, CSSON and PLLON bits
        RCC->CR &= ~(RCC_CR_HSEON | RCC_CR_CSSON | RCC_CR_PLLON);
        while ((RCC->CR & RCC_CR_PLLRDY) != 0); // Wait until PLL disabled

        // Programming PLLCFGR register
        // RCC->PLLCFGR = 0x24003010; // This is the default value

        // Tip:
        // Recommended to set VOC Input f(PLL clock input) / PLLM to 1-2MHz
        // Set VCO output between 192 and 432 MHz,
        // f(VCO clock) = f(PLL clock input) × (PLLN / PLLM)
        // f(PLL general clock output) = f(VCO clock) / PLLP
        // f(USB OTG FS, SDIO, RNG clock output) = f(VCO clock) / PLLQ

        RCC->PLLCFGR = 0;
        RCC->PLLCFGR &= ~(RCC_PLLCFGR_PLLSRC); // PLLSRC = 0 (HSI 16 Mhz clock //selected as
//clock source)
        RCC->PLLCFGR |= 16 << RCC_PLLCFGR_PLLN_Pos;        // PLLM = 16, VCO input //clock = 16 MHz
/ /PLLM = 1 MHz
```

```
	RCC->PLLCFGR |= 336 << RCC_PLLCFGR_PLLN_Pos; // PLLN = 336, VCO output //clock = 1 MHz *
//336 = 336 MHz
	RCC->PLLCFGR |= 4 << RCC_PLLCFGR_PLLP_Pos; // PLLP = 4, PLLCLK = 336 //Mhz / PLLP = 84
//MHz
	RCC->PLLCFGR |= 7 << RCC_PLLCFGR_PLLQ_Pos; // PLLQ = 7, USB Clock = 336 //MHz / PLLQ =
//48 MHz

	// Enable Main PLL Clock
	RCC->CR |= RCC_CR_PLLON;
	while ((RCC->CR & RCC_CR_PLLRDY) == 0);  // Wait until PLL ready
	FLASH->ACR |= FLASH_ACR_ICEN | FLASH_ACR_PRFTEN | FLASH_ACR_LATENCY_2WS;

	RCC->CFGR &= ~RCC_CFGR_HPRE; // 84 MHz, not divided
	// PPRE1: APB Low speed prescaler (APB1)
	RCC->CFGR &= ~RCC_CFGR_PPRE1;
	RCC->CFGR |= RCC_CFGR_PPRE1_DIV2; // 42 MHz, divided by 2
	// PPRE2: APB high-speed prescaler (APB2)
	RCC->CFGR &= ~RCC_CFGR_PPRE2; // 84 MHz, not divided

	RCC->CFGR &= ~RCC_CFGR_SW;
	RCC->CFGR |= RCC_CFGR_SW_1;
	// while ((RCC->CFGR & RCC_CFGR_SWS_PLL) != RCC_CFGR_SWS_PLL);

	// Configure the Vector Table location add offset address
//	VECT_TAB_OFFSET  = 0x00UL; // Vector Table base offset field.
// This value must be a multiple of 0x200.
	SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; // Vector Table Relocation in Internal FLASH

}


static void LED_Pin_Init(){
	 RCC->AHB1ENR		|= RCC_AHB1ENR_GPIOAEN;		// Enable GPIOA clock

	 // Set mode as Alternative Function 1
		LED_PORT->MODER  &= ~(0x03 << (2*LED_PIN));		// Clear bits
		LED_PORT->MODER  |=  0x02 << (2*LED_PIN);	// Input(00), Output(01), //AlterFunc(10),
//Analog(11)

		LED_PORT->AFR[0] &= ~(0xF << (4*LED_PIN));  // AF 1 = TIM2_CH1
		LED_PORT->AFR[0] |=  0x1 << (4*LED_PIN);    //AF 1 = TIM2_CH1

		//Set I/O output speed value as very high speed
		LED_PORT->OSPEEDR  &= ~(0x03<<(2*LED_PIN));			// Speed mask
		LED_PORT->OSPEEDR  |=  0x03<<(2*LED_PIN);
	// Very high speed
		//Set I/O as no pull-up pull-down
		LED_PORT->PUPDR    &= ~(0x03<<(2*LED_PIN));			// No //PUPD(00,
//reset), Pullup(01), Pulldown(10), Reserved (11)
		//Set I/O as push pull
	  //LED_PORT->OTYPER   &=  ~(1<<LED_PIN) ; // Push-Pull(0, reset), Open-Drain(1)
}

static void SPEAKER_Pin_Init(){
  // Enable the clock to GPIO Port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
```

```
        // Set mode as Alternative Function 1
                // GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
                SPEAKER_PORT->MODER  &= ~(0x03 << (2*SPEAKER_PIN));     // Clear bits
                SPEAKER_PORT->MODER  |=   0x02 << (2*SPEAKER_PIN);                  // Input(00),
//Output(01), AlterFunc(10), Analog(11)

                SPEAKER_PORT->AFR[0]          &= ~(0xF << (4*SPEAKER_PIN));     // Clear AF
                SPEAKER_PORT->AFR[0] |=   0x2 << (4*SPEAKER_PIN);      //   //AF 2 = //TIM5_CH1

        // GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
                //Set I/O output speed value as very high speed
        SPEAKER_PORT->OSPEEDR  &= ~(0x03<<(2*SPEAKER_PIN));   // Speed mask
                SPEAKER_PORT->OSPEEDR  |=   0x03<<(2*SPEAKER_PIN);

        // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
        //SPEAKER_PORT->OTYPER &= ~(1U<<SPEAKER_PIN);       // Push-pull

// GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
SPEAKER_PORT->PUPDR  &= ~(3U<<(2*SPEAKER_PIN));  // No pull-up, no pull-down
}

static void TIM5_CH1_Init(){
   RCC->APB1ENR |= RCC_APB1ENR_TIM5EN;        // Enable TIMER clock
                // Counting direction: 0 = up-counting, 1 = down-counting
                TIM5->CR1 &= ~TIM_CR1_DIR;

   TIM5->PSC = 1;       // Prescaler = 23
   TIM5->ARR = 7999-1;   // Auto-reload: Upcouting (0..ARR), Downcouting (ARR..0)
TIM5->CCMR1 &= ~TIM_CCMR1_OC1M;  // Clear ouput compare mode bits for channel 1
TIM5->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1; // OC1M = 0011
//0011 means toggle mode
   TIM5->CCMR1 |= TIM_CCMR1_OC1PE; // Output 1 preload enable

                // Select output polarity: 0 = active high, 1 = active low
                TIM5->CCER |= TIM_CCER_CC1NP; // select active high

   // Enable output for ch1
                TIM5->CCER |= TIM_CCER_CC1E;

   // Main output enable (MOE): 0 = Disable, 1 = Enable
                TIM5->BDTR |= TIM_BDTR_MOE;

                //TIM5->CCR1  = 1135;        // Output Compare Register for channel 1
                TIM5->CR1  |= TIM_CR1_CEN; // Enable counter
}


int main(void){
                int i;
                int n = 1;
            uint16_t current_note = 0;

   static uint32_t note_freq[12] = {349,440,392,329,293,130,196,220,233,466,174,164}; //Hz
                static uint16_t song_notes[46] = {0,0,0,0,1,2,0,0,0,0,
```

```
                                                    3,0,2,2,0,3,4,3,3,5,5,5,

                                                    6,7,8,8,4,4,0,0,9,9,9,9,

                                                    7,6,6,7,6,10,11,6,10,10,10,10};
        enable_HSI(); //16 MHz
        SPEAKER_Pin_Init();

        TIM5_CH1_Init(); // Timer to control Servo, signal period = 20ms
        TIM5->ARR = (16000000 / 4 /  note_freq[current_note] ) - 1;

        while(1){
                TIM5->ARR = (16000000UL/2/ note_freq[song_notes[current_note]] ) - 1UL;
                                current_note = current_note+1;
                                if (current_note > 45 ||  current_note < 0) current_note = 0;
                        for(i=0;i<1000000;i++);                 // delay
        }}
```

Problem 1

Initialize TIM5 so the ARR period happens at 50Hz (20ms)

Solution:

We know, $f_{CNT}=f_{sys}/(1+PSC)(1+ARR)$
f_{sys}=16MHz, f_{CNT}=50Hz
So, (1+PSC)(1+ARR)=320k

I set (1+PSC)=1k, so PSC=999
And (1+ARR)=320, so ARR=319

Code Snippet:

```
static void SPEAKER_Pin_Init(){
  // Enable the clock to GPIO Port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

        // Set mode as Alternative Function 1
                // GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
                SPEAKER_PORT->MODER  &= ~(0x03 << (2*SPEAKER_PIN));                     // Clear
bits
                SPEAKER_PORT->MODER  |=   0x02 << (2*SPEAKER_PIN);              // Input(00),
```

```
Output(01), AlterFunc(10), Analog(11)

                SPEAKER_PORT->AFR[0]          &= ~(0xF << (4*SPEAKER_PIN));      // Clear AF
                SPEAKER_PORT->AFR[0] |=   0x2 << (4*SPEAKER_PIN);       //   //AF 2 = //TIM5_CH1

        // GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
                //Set I/O output speed value as very high speed
        SPEAKER_PORT->OSPEEDR  &= ~(0x03<<(2*SPEAKER_PIN));   // Speed mask
                SPEAKER_PORT->OSPEEDR  |=   0x03<<(2*SPEAKER_PIN);




        // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
        //SPEAKER_PORT->OTYPER &= ~(1U<<SPEAKER_PIN);       // Push-pull

// GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
SPEAKER_PORT->PUPDR  &= ~(3U<<(2*SPEAKER_PIN));  // No pull-up, no pull-down
}


static void TIM5_CH1_Init(){
   RCC->APB1ENR |= RCC_APB1ENR_TIM5EN;        // Enable TIMER clock
                // Counting direction: 0 = up-counting, 1 = down-counting
                TIM5->CR1 &= ~TIM_CR1_DIR;

   TIM5->PSC = 999;
   TIM5->ARR = 319;
TIM5->CCMR1 &= ~TIM_CCMR1_OC1M;  // Clear ouput compare mode bits for channel 1
 TIM5->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1; // OC1M = 0011
//0011 means toggle mode
   TIM5->CCMR1 |= TIM_CCMR1_OC1PE; // Output 1 preload enable

                // Select output polarity: 0 = active high, 1 = active low
                TIM5->CCER |= TIM_CCER_CC1NP; // select active high

    // Enable output for ch1
                TIM5->CCER |= TIM_CCER_CC1E;

   // Main output enable (MOE): 0 = Disable, 1 = Enable
                TIM5->BDTR |= TIM_BDTR_MOE;

                //TIM5->CCR1  = 1135;        // Output Compare Register for channel 1
                TIM5->CR1  |= TIM_CR1_CEN; // Enable counter
}
```

Problem 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.

Solution:

In up-counting PWM mode1,
DC=CCR/(1+ARR)
From previous question, for 50ms period ARR is fixed to 319. So, CCR will be varied to change duty cycle (DC)

Servo Motor Position (in degree)	Pulse width (in ms)	Duty cycle	CCR values
-90	1	1/20	16
0	1.5	3/40	24
90	2	1/40	32

Code Snippet:

```
#include "stm32f446xx.h"
#define LED_PORT GPIOA
#define LED_PIN 5
#define BUTTON_PIN 13
static void LED_Pin_Init(){
 RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // Enable GPIOA clock
// Set mode as Alternative Function 1
LED_PORT->MODER &= ~(0x03 << (2*LED_PIN)); // Clear bits
LED_PORT->MODER |= 0x02 << (2*LED_PIN); // Input(00), Output(01),
//AlterFunc(10), Analog(11)
LED_PORT->AFR[0] &= ~(0xF << (4*LED_PIN)); // AF 1 = TIM2_CH1
LED_PORT->AFR[0] |= 0x1 << (4*LED_PIN); // AF 1 = TIM2_CH1

//Set I/O output speed value as very high speed
LED_PORT->OSPEEDR &= ~(0x03<<(2*LED_PIN)); // Speed mask
LED_PORT->OSPEEDR |= 0x03<<(2*LED_PIN); // Very high speed

//Set I/O as no pull-up pull-down
LED_PORT->PUPDR &= ~(0x03<<(2*LED_PIN)); // No PUPD(00, reset), Pullup(01),
//Pulldown(10), Reserved (11)

//LED_PORT->OTYPER &= ~(1<<LED_PIN) ; // Push-Pull(0, reset), Open-Drain(1)
}
```

```
static void TIM2_CH1_Init(){
RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // Enable TIMER clock
// Counting direction: 0 = up-counting, 1 = down-counting

TIM2->CR1 &= ~TIM_CR1_DIR;

TIM2->PSC = 999;
 TIM2->ARR = 319;

TIM2->CCMR1 &= ~TIM_CCMR1_OC1M; // Clear ouput compare mode bits for channel 1
 TIM2->CCMR1 |= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2;
//OC1M = 110 for PWM Mode 1 output on ch1

 TIM2->CCMR1 |= TIM_CCMR1_OC1PE; // Output 1 preload enable
// Select output polarity: 0 = active high, 1 = active low

TIM2->CCER |= TIM_CCER_CC1NP; // select active high
// Enable output for ch1

TIM2->CCER |= TIM_CCER_CC1E;
 // Main output enable (MOE): 0 = Disable, 1 = Enable
TIM2->BDTR |= TIM_BDTR_MOE;
// zero angle = 24
// -90 = 16
// +90 = 32
TIM2->CCR1 = 24; // Output Compare Register for channel 1
TIM2->CR1 |= TIM_CR1_CEN; // Enable counter
}


int main(void){
LED_Pin_Init();
TIM2_CH1_Init(); // Timer to control LED

int i;

while(1){
        TIM2->CCR1 = 32; // set angle 90
        for(i=0;i<1000000;i++); // delay of 1s

        TIM2->CCR1 = 24; // set angle -0
        for(i=0;i<1000000;i++); // delay of 1s

        TIM2->CCR1 = 16; // set angle -90
        for(i=0;i<1000000;i++); // delay of 1s
        }
}
```

Problem 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.

Solution:

Same as previous one. Only the main function is written here with minor changes

```
int main(void){
LED_Pin_Init();
TIM2_CH1_Init(); // Timer to control LED

int i;

while(1){
        TIM2->CCR1 = 24; // set angle 0
        for(i=0;i<1000000;i++); // delay of 1s

        TIM2->CCR1 = 32; // set angle 90
        for(i=0;i<1000000;i++); // delay of 1s

        TIM2->CCR1 = 16; // set angle -90
        for(i=0;i<1000000;i++); // delay of 1s
        }
}
```

Problem 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.

Solution:

The experimental values are almost similar to the theoretical values. That's why the previous table is added here:

In up-counting PWM mode1,
DC=CCR/(1+ARR)
From previous question, for 50ms period ARR is fixed to 319. So, CCR will be varied to change duty cycle (DC)

Servo Motor Position (in degree)	Pulse width (in ms)	Duty cycle	CCR values
-90	1	1/20	16
0	1.5	3/40	24
90	2	1/40	32

Problem 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.

Solution:

```
#include "stm32f446xx.h"

#define  LED_PINA  5//stepper
#define  LED_PINB  6//stepper
#define  LED_PINC  7//stepper
#define  LED_PIND  8//stepper
#define LED_PIN   9//LED pin
```

```
#define EXTI_PIN 13//push button as interrupt

#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                              This value must be a multiple of 0x200. */

static void enable_HSI(){

        RCC->CR |= ((uint32_t)RCC_CR_HSION);
        while ((RCC->CR & RCC_CR_HSIRDY) == 0); // Wait until HSI ready

        // Store calibration value
        PWR->CR |= (uint32_t)(16 << 3);

        // Reset CFGR register
        RCC->CFGR = 0x00000000;

        // Reset HSEON, CSSON and PLLON bits
        RCC->CR &= ~(RCC_CR_HSEON | RCC_CR_CSSON | RCC_CR_PLLON);
        while ((RCC->CR & RCC_CR_PLLRDY) != 0); // Wait until PLL disabled

        RCC->PLLCFGR = 0;
        RCC->PLLCFGR &= ~(RCC_PLLCFGR_PLLSRC);                  /* PLLSRC = 0 (HSI 16 Mhz clock
//selected as clock source)*/
        RCC->PLLCFGR |= 16 << RCC_PLLCFGR_PLLN_Pos;     /* PLLM = 16, VCO input clock = 16 MHz /
//PLLM = 1 MHz*/
        RCC->PLLCFGR |= 336 << RCC_PLLCFGR_PLLN_Pos;    /* PLLN = 336, VCO output clock = 1 MHz
//* 336 = 336 MHz*/
        RCC->PLLCFGR |= 4 << RCC_PLLCFGR_PLLP_Pos;      // PLLP = 4, PLLCLK = 336 Mhz / PLLP =
//84 MHz
        RCC->PLLCFGR |= 7 << RCC_PLLCFGR_PLLQ_Pos;      // PLLQ = 7, USB Clock = 336 MHz /
//PLLQ = 48 MHz

        // Enable Main PLL Clock
        RCC->CR |= RCC_CR_PLLON;
        while ((RCC->CR & RCC_CR_PLLRDY) == 0);  // Wait until PLL ready


        // FLASH configuration block
        // enable instruction cache, enable prefetch, set latency to 2WS (3 CPU cycles)

        FLASH->ACR |= FLASH_ACR_ICEN | FLASH_ACR_PRFTEN | FLASH_ACR_LATENCY_2WS;

        // Configure the HCLK, PCLK1 and PCLK2 clocks dividers
        // AHB clock division factor
        RCC->CFGR &= ~RCC_CFGR_HPRE; // 84 MHz, not divided
        // PPRE1: APB Low speed prescaler (APB1)
        RCC->CFGR &= ~RCC_CFGR_PPRE1;
        RCC->CFGR |= RCC_CFGR_PPRE1_DIV2; // 42 MHz, divided by 2
        // PPRE2: APB high-speed prescaler (APB2)
        RCC->CFGR &= ~RCC_CFGR_PPRE2; // 84 MHz, not divided

        RCC->CFGR &= ~RCC_CFGR_SW;
        RCC->CFGR |= RCC_CFGR_SW_1;
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; // Vector Table Relocation in Internal FLASH

}
```

```
static void configure_LED_PINA(){
  // Enable the clock to GPIO Port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

        // GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
        GPIOA->MODER &= ~(3UL<<(2*LED_PINA));
        GPIOA->MODER |=  1UL<<(2*LED_PINA);       // Output(01)

        // GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
        GPIOA->OSPEEDR &= ~(3U<<(2*LED_PINA));
        GPIOA->OSPEEDR |=  2U<<(2*LED_PINA);  // Fast speed

        // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
        GPIOA->OTYPER &= ~(1U<<LED_PINA);      // Push-pull

        // GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
        GPIOA->PUPDR  &= ~(3U<<(2*LED_PINA));  // No pull-up, no pull-down

}


static void configure_LED_PINB(){
  // Enable the clock to GPIO Port B
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

        // GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
        GPIOA->MODER &= ~(3UL<<(2*LED_PINB));
        GPIOA->MODER |=  1UL<<(2*LED_PINB);       // Output(01)

        // GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
        GPIOA->OSPEEDR &= ~(3U<<(2*LED_PINB));
        GPIOA->OSPEEDR |=  2U<<(2*LED_PINB);  // Fast speed

        // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
        GPIOA->OTYPER &= ~(1U<<LED_PINB);      // Push-pull

        // GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
        GPIOA->PUPDR  &= ~(3U<<(2*LED_PINB));  // No pull-up, no pull-down

}


static void configure_LED_PINC(){
  // Enable the clock to GPIO Port C
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

        // GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
        GPIOA->MODER &= ~(3UL<<(2*LED_PINC));
        GPIOA->MODER |=  1UL<<(2*LED_PINC);       // Output(01)

        // GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
        GPIOA->OSPEEDR &= ~(3U<<(2*LED_PINC));
        GPIOA->OSPEEDR |=  2U<<(2*LED_PINC);  // Fast speed
```

```
        // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
        GPIOA->OTYPER &= ~(1U<<LED_PINC);      // Push-pull

        // GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
        GPIOA->PUPDR  &= ~(3U<<(2*LED_PINC));  // No pull-up, no pull-down

}


static void configure_LED_PIND(){
  // Enable the clock to GPIO Port D
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

        // GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
        GPIOA->MODER &= ~(3UL<<(2*LED_PIND));
        GPIOA->MODER |=   1UL<<(2*LED_PIND);       // Output(01)

        // GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
        GPIOA->OSPEEDR &= ~(3U<<(2*LED_PIND));
        GPIOA->OSPEEDR |=   2U<<(2*LED_PIND);  // Fast speed

        // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
        GPIOA->OTYPER &= ~(1U<<LED_PIND);      // Push-pull

        // GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
        GPIOA->PUPDR  &= ~(3U<<(2*LED_PIND));  // No pull-up, no pull-down

}


static void configure_LED_PIN(){
  // Enable the clock to GPIO Port D
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

        // GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
        GPIOA->MODER &= ~(3UL<<(2*LED_PIN));
        GPIOA->MODER |=   1UL<<(2*LED_PIN);       // Output(01)

        // GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
        GPIOA->OSPEEDR &= ~(3U<<(2*LED_PIN));
        GPIOA->OSPEEDR |=   2U<<(2*LED_PIN);  // Fast speed


        // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
        GPIOA->OTYPER &= ~(1U<<LED_PIN);      // Push-pull

        // GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
        GPIOA->PUPDR  &= ~(3U<<(2*LED_PIN));  // No pull-up, no pull-down

}
```

```
static void turn_on_LEDA(){
	GPIOA->ODR |= 1U << LED_PINA;
}
static void turn_on_LEDB(){
	GPIOA->ODR |= 1U << LED_PINB;
}
static void turn_on_LEDC(){
	GPIOA->ODR |= 1U << LED_PINC;
}
static void turn_on_LEDD(){
	GPIOA->ODR |= 1U << LED_PIND;
}
static void turn_on_LED(){
	GPIOA->ODR |= 1U << LED_PIN;
}


static void turn_off_LEDA(){
	GPIOA->ODR &= ~(1U << LED_PINA);
}

static void turn_off_LEDB(){
	GPIOA->ODR &= ~(1U << LED_PINB);
}
static void turn_off_LEDC(){
	GPIOA->ODR &= ~(1U << LED_PINC);
}
static void turn_off_LEDD(){
	GPIOA->ODR &= ~(1U << LED_PIND);
}
static void turn_off_LED(){
	GPIOA->ODR &= ~(1U << LED_PIN);
}

static void toggle_LED(){
	GPIOA->ODR ^= (1 << LED_PIN);
}


void config_EXTI(void) {
                        // GPIO Configuration
                        RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;

                        // GPIO Mode: Input(00, reset), Output(01), AlterFunc(10), Analog(11, reset)
                        GPIOC->MODER &= ~(3UL<<(2*EXTI_PIN)); //input
                        // GPIO PUDD: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)

                        GPIOC->PUPDR &= ~(3UL<<(2*EXTI_PIN)); // no pull-up, no pull down
                        // Connect External Line to the GPIO
                        RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
                        SYSCFG->EXTICR[3] &= ~SYSCFG_EXTICR4_EXTI13; // SYSCFG externa

                        //registers
                        SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI13_PC; // port C
                        // Ralling trigger selection register (RTSR)
```

```
                        EXTI->RTSR |= EXTI_RTSR_TR13; // 0 = disabled, 1 = enabled

                        // Interrupt Mask Register (IMR)
                        EXTI->IMR |= EXTI_IMR_IM13; // 0 = marked, 1 = not masked (i.e., enabled)
//interrupt configuration // EXIT Interrupt Enable
                        NVIC_EnableIRQ(EXTI15_10_IRQn);
                        NVIC_SetPriority(EXTI15_10_IRQn, 0); //HIGHEST PRIORITY




void EXTI15_10_IRQHandler(void) {
                        //NVIC_ClearPendingIRQ(EXTI15_10_IRQn);
                        uint32_t j;

                        // PR: Pending register
                        if (EXTI->PR & EXTI_PR_PR13) {
                        // cleared by writing a 1 to this bit
                        EXTI->PR |= EXTI_PR_PR13;
                        toggle_LED();
                        for(j=0;j<300000;j++);
                        }
}
int main(void){

                int i ;
                enable_HSI(); // clk = 16MHz
                configure_LED_PINA();
                configure_LED_PINB();
                configure_LED_PINC();
                configure_LED_PIND();
                configure_LED_PIN();

                config_EXTI();
                turn_on_LED();
                while(1){
                //Full stepping in counter-clockwise
                        turn_on_LEDA();//pin 6
                        for(i=0; i<50000; i++); // simple delay
                        turn_off_LEDA();

                        turn_on_LEDB();//pin7
                        for(i=0; i<50000; i++); // simple delay
                        turn_off_LEDB();

                        turn_on_LEDC();//pin8
                        for(i=0; i<50000; i++); // simple delay
                        turn_off_LEDC();

                        turn_on_LEDD();//pin9

                        for(i=0; i<50000; i++); // simple delay
                        turn_off_LEDD();}}
```

Problem 6

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. You can use the DCD values as a static array in C. Write your code below

Solution:

```
#include "stm32f446xx.h"
#define SPEAKER_PORT GPIOA
#define SPEAKER_PIN  0

#define LED_PORT GPIOA
#define LED_PIN    5

#define BUTTON_PIN 13

#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                              This value must be a multiple of 0x200. */

static void enable_HSI(){

        RCC->CR |= ((uint32_t)RCC_CR_HSION);
        while ((RCC->CR & RCC_CR_HSIRDY) == 0); // Wait until HSI ready

        // Store calibration value
        PWR->CR |= (uint32_t)(16 << 3);

        // Reset CFGR register
        RCC->CFGR = 0x00000000;

        // Reset HSEON, CSSON and PLLON bits
        RCC->CR &= ~(RCC_CR_HSEON | RCC_CR_CSSON | RCC_CR_PLLON);
        while ((RCC->CR & RCC_CR_PLLRDY) != 0); // Wait until PLL disabled


        RCC->PLLCFGR = 0;
RCC->PLLCFGR &= ~(RCC_PLLCFGR_PLLSRC); // PLLSRC = 0 (HSI 16 Mhz clock selected as clock source)
RCC->PLLCFGR |= 16 << RCC_PLLCFGR_PLLN_Pos; // PLLM = 16, VCO input clock = 16 MHz / PLLM = 1
//MHz
        RCC->PLLCFGR |= 336 << RCC_PLLCFGR_PLLN_Pos;       // PLLN = 336, VCO output clock = 1 MHz
//* 336 = 336 MHz
        RCC->PLLCFGR |= 4 << RCC_PLLCFGR_PLLP_Pos; // PLLP = 4, PLLCLK = 336 Mhz / PLLP = //84
//MHz
        RCC->PLLCFGR |= 7 << RCC_PLLCFGR_PLLQ_Pos;          // PLLQ = 7, USB Clock = 336 MHz /
//PLLQ = 48 MHz

        // Enable Main PLL Clock
        RCC->CR |= RCC_CR_PLLON;
        while ((RCC->CR & RCC_CR_PLLRDY) == 0);  // Wait until PLL ready
```

```
	FLASH->ACR |= FLASH_ACR_ICEN | FLASH_ACR_PRFTEN | FLASH_ACR_LATENCY_2WS;

	// Configure the HCLK, PCLK1 and PCLK2 clocks dividers
	// AHB clock division factor
	RCC->CFGR &= ~RCC_CFGR_HPRE; // 84 MHz, not divided
	// PPRE1: APB Low speed prescaler (APB1)
	RCC->CFGR &= ~RCC_CFGR_PPRE1;
	RCC->CFGR |= RCC_CFGR_PPRE1_DIV2; // 42 MHz, divided by 2
	// PPRE2: APB high-speed prescaler (APB2)
	RCC->CFGR &= ~RCC_CFGR_PPRE2; // 84 MHz, not divided

	RCC->CFGR &= ~RCC_CFGR_SW;
	RCC->CFGR |= RCC_CFGR_SW_1;
	// while ((RCC->CFGR & RCC_CFGR_SWS_PLL) != RCC_CFGR_SWS_PLL);

	// Configure the Vector Table location add offset address
//	VECT_TAB_OFFSET  = 0x00UL; // Vector Table base offset field.
			// This value must be a multiple of 0x200.
	SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; // Vector Table Relocation in Internal FLASH

}


static void LED_Pin_Init(){
	RCC->AHB1ENR		|= RCC_AHB1ENR_GPIOAEN;		// Enable GPIOA clock

	// Set mode as Alternative Function 1
		LED_PORT->MODER	&= ~(0x03 << (2*LED_PIN));		// Clear bits
		LED_PORT->MODER	|=  0x02 << (2*LED_PIN);		// Input(00), Output(01),
//AlterFunc(10), Analog(11)

		LED_PORT->AFR[0]	&= ~(0xF << (4*LED_PIN));	//	AF 1 = TIM2_CH1
		LED_PORT->AFR[0]	|=  0x1 << (4*LED_PIN);	//  AF 1 = TIM2_CH1

		//Set I/O output speed value as very high speed
		LED_PORT->OSPEEDR  &= ~(0x03<<(2*LED_PIN));		// Speed mask
		LED_PORT->OSPEEDR  |=  0x03<<(2*LED_PIN);
	// Very high speed
		//Set I/O as no pull-up pull-down
		LED_PORT->PUPDR	&= ~(0x03<<(2*LED_PIN));	// No PUPD(00, reset), Pullup(01),
//Pulldown(10), Reserved (11)
		//Set I/O as push pull
	 //LED_PORT->OTYPER   &=  ~(1<<LED_PIN) ;		// Push-Pull(0, reset), Open-Drain(1)
}

static void SPEAKER_Pin_Init(){
  // Enable the clock to GPIO Port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

	// Set mode as Alternative Function 1
		// GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
		SPEAKER_PORT->MODER		&= ~(0x03 << (2*SPEAKER_PIN));
	// Clear bits
		SPEAKER_PORT->MODER		|=  0x02 << (2*SPEAKER_PIN);		//
//Input(00), Output(01), AlterFunc(10), Analog(11)
```

```
                SPEAKER_PORT->AFR[0]            &= ~(0xF << (4*SPEAKER_PIN));     //       Clear AF
                SPEAKER_PORT->AFR[0] |=   0x2 << (4*SPEAKER_PIN);       //       AF 2 = TIM5_CH1

        // GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
                //Set I/O output speed value as very high speed
                SPEAKER_PORT->OSPEEDR  &= ~(0x03<<(2*SPEAKER_PIN));       // Speed mask
                SPEAKER_PORT->OSPEEDR  |=   0x03<<(2*SPEAKER_PIN); // Very high speed

        // GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
        //SPEAKER_PORT->OTYPER &= ~(1U<<SPEAKER_PIN);       // Push-pull

        // GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
        SPEAKER_PORT->PUPDR  &= ~(3U<<(2*SPEAKER_PIN));  // No pull-up, no pull-down

}

static void TIM5_CH1_Init(){

   RCC->APB1ENR |= RCC_APB1ENR_TIM5EN;                  // Enable TIMER clock

                // Counting direction: 0 = up-counting, 1 = down-counting
TIM5->CR1 &= ~TIM_CR1_DIR;

   TIM5->PSC = 1;
   TIM5->ARR = 7999-1;
TIM5->CCMR1 &= ~TIM_CCMR1_OC1M;  // Clear ouput compare mode bits for channel 1

 TIM5->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1; // OC1M = 0011
   TIM5->CCMR1 |= TIM_CCMR1_OC1PE;                  // Output 1 preload enable

                // Select output polarity: 0 = active high, 1 = active low
TIM5->CCER |= TIM_CCER_CC1NP; // select active high

   // Enable output for ch1
                TIM5->CCER |= TIM_CCER_CC1E;

   // Main output enable (MOE): 0 = Disable, 1 = Enable
                TIM5->BDTR |= TIM_BDTR_MOE;

                //TIM5->CCR1  = 1135;       // Output Compare Register for channel 1
                TIM5->CR1  |= TIM_CR1_CEN; // Enable counter
}



int main(void){
                int i,delay1,delay2;
                uint16_t current_note1 = 0;
                uint16_t current_note2 = 0;

   static uint32_t note_freq_twinkle[42] = {262, 262, 392, 392, 440, 440, 392, //; Twinkle twinkle Little star
                                        349, 349, 330, 330, 294, 294, 262,     //; How I wonder what you are
                                        392, 392, 349, 349, 330, 330, 294, //, Up above the world so high
                                        392, 392, 349, 349, 330, 330, 294, //, Like a diamond in the sky
                                        262, 262, 392, 392, 440, 440, 392, //; Twinkle twinkle Little star
                                        349, 349, 330, 330, 294, 294, 262}; //; How I wonder what you are!  //Hz
```

```
static uint16_t song_notes_twinkle[42] =  { 1, 1, 1, 1, 1, 1, 2, //setting BPM as 120
                                            1, 1, 1, 1, 1, 1, 2,
                                            1, 1, 1, 1, 1, 1, 2,
                                            1, 1, 1, 1, 1, 1, 2,
                                            1, 1, 1, 1, 1, 1, 2,
                                            1, 1, 1, 1, 1, 1, 2};


        static uint32_t note_freq_happy[25] ={392, 392, 440, 392, 523, 494, //; Happy Birthday to You
                                              392, 392, 440, 392, 523, 494, //; Happy Birthday to You
                                              392, 392, 784, 659, 523, 494, 440, //; Happy Birthday to Dear(name)
                                              349, 349, 330, 262, 294, 262}; //Happy Birthday to You

                static uint16_t song_notes_happy[25] =  {1, 1, 2, 2, 2, 4,//setting BPM as 120
                                                         1, 1, 2, 2, 2, 4,
                                                         1, 1, 2, 2, 2, 2, 6,
                                                         2, 2, 2, 2, 2, 4};


        SPEAKER_Pin_Init();

        TIM5_CH1_Init(); // Timer to control Servo, signal period = 20ms

        while(1){
                        TIM5->ARR = (16000000UL/2/ note_freq_twinkle[current_note1] ) - 1UL;
                                delay1=1000000*song_notes_twinkle[current_note1];
                         for(i=0;i<delay1;i++);//plaing twinkle twinkle litttle star

                                current_note1 = current_note1+1;
                                if (current_note1 > 41 ||  current_note1 < 0) current_note1 = 0;


                                TIM5->ARR = (16000000UL/2/ note_freq_happy[current_note2] ) - 1UL;
                                delay2=1000000*song_notes_happy[current_note2];
                         for(i=0;i<delay2;i++);//happy birthday to you

                                current_note2 = current_note2+1;
                                if (current_note2 > 24 ||  current_note2 < 0) current_note2 = 0;
        }
}
```