Problem 1

Toggle an LED every 1s using SysTick

Solution:

```
#include "stm32f446xx.h"
#define LED_PIN    5
#define BUTTON_PIN 13

#define VECT_TAB_OFFSET  0x00

static void sys_clk_config(){
        RCC->CR |= RCC_CR_HSION;
        while ((RCC->CR & RCC_CR_HSIRDY) == 0);
        RCC->CFGR = 0x00000000;

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

        RCC->CFGR &= ~RCC_CFGR_SW;

        RCC->CFGR &= ~RCC_CFGR_HPRE; // 16 MHz, not divided
        // PPRE1: APB Low speed prescaler (APB1)
        RCC->CFGR &= ~RCC_CFGR_PPRE1; // 16 MHz, not divided
        // PPRE2: APB high-speed prescaler (APB2)
        RCC->CFGR &= ~RCC_CFGR_PPRE2; // 16 MHz, not divided

        SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; // Vector Table Relocation in
        //Internal FLASH
}

static void configure_LED_pin(){
  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

        GPIOA->OTYPER &= ~(1U<<LED_PIN);      // Push-pull

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

```
static void turn_on_LED(){
	GPIOA->ODR |= 1U << LED_PIN;
}

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

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



static void configure_SysTick(uint32_t ticks){
   SysTick->CTRL = 0;           // Disable SysTick
   SysTick->LOAD = ticks - 1;    // Set reload register.

   // Set interrupt priority of SysTick to least urgency (i.e., largest priority value)
   NVIC_SetPriority (SysTick_IRQn, (1<<__NVIC_PRIO_BITS) - 1);

   SysTick->VAL = 0;            // Reset the SysTick counter value

   // Select processor clock/8 : 1 = processor clock; 0 = external clock = processor clock/8
   SysTick->CTRL &= ~SysTick_CTRL_CLKSOURCE_Msk;

		// Enables SysTick exception request
		// 1 = counting down to zero asserts the SysTick exception request
		// 0 = counting down to zero does not assert the SysTick exception request
		SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;
		// Enable SysTick
		SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
}

void SysTick_Handler (void) { // SysTick interrupt service routine
		toggle_LED();
}

int main(void){
	uint32_t i;

	sys_clk_config(); // clk = 16MHz
	configure_LED_pin();
	configure_SysTick(2000000); // ARR of SysTick = 2M.
					//and systick will generate interrupt after every 1s
	while(1);
}
```

Problem 2

Implement a function called mydelay() that takes time in ms as input and creates that delay.

Solution:

```
#include "stm32f446xx.h"
#define LED_PIN    5
#define BUTTON_PIN 13

volatile uint32_t TimeDelay=0;

#define VECT_TAB_OFFSET  0x00

static void sys_clk_config(){
        RCC->CR |= RCC_CR_HSION;
        while ((RCC->CR & RCC_CR_HSIRDY) == 0);
        RCC->CFGR = 0x00000000;

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

        RCC->CFGR &= ~RCC_CFGR_SW;

        RCC->CFGR &= ~RCC_CFGR_HPRE; // 16 MHz, not divided
        // PPRE1: APB Low speed prescaler (APB1)
        RCC->CFGR &= ~RCC_CFGR_PPRE1; // 16 MHz, not divided
        // PPRE2: APB high-speed prescaler (APB2)
        RCC->CFGR &= ~RCC_CFGR_PPRE2; // 16 MHz, not divided

        SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; // Vector Table Relocation in
        //Internal FLASH
}


static void configure_LED_pin(){
  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

        GPIOA->OTYPER &= ~(1U<<LED_PIN);      // Push-pull
```

```
	// GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved
	GPIOA->PUPDR  &= ~(3U<<(2*LED_PIN));  // No pull-up, no pull-
}
static void turn_on_LED(){
	GPIOA->ODR |= 1U << LED_PIN;
}

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

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



static void configure_SysTick(uint32_t ticks){
   SysTick->CTRL = 0;           // Disable SysTick
   SysTick->LOAD = ticks - 1;    // Set reload register.

   // Set interrupt priority of SysTick to least urgency (i.e., largest priority value)
   NVIC_SetPriority (SysTick_IRQn, (1<<__NVIC_PRIO_BITS) - 1);

   SysTick->VAL = 0;            // Reset the SysTick counter value

   // Select processor clock/8 : 1 = processor clock; 0 = external clock = processor clock/8
   SysTick->CTRL &= ~SysTick_CTRL_CLKSOURCE_Msk;

                // Enables SysTick exception request
                // 1 = counting down to zero asserts the SysTick exception request
                // 0 = counting down to zero does not assert the SysTick exception request
                SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;

                // Enable SysTick
                SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;

}

void SysTick_Handler (void) { // SysTick interrupt service routine
	if(TimeDelay>0)
		TimeDelay=TimeDelay-1;
}
```

```
void MYDelay (uint32_t nTime) {
  // nTime: specifies the delay time length
  TimeDelay = nTime;      // TimeDelay must be declared as volatile
  while(TimeDelay != 0);  // Busy wait
}
int main(void){
        uint32_t i;

        sys_clk_config(); // clk = 16MHz
        configure_LED_pin();
        configure_SysTick(2000); // ARR of SysTick = 2K.

        while(1){
                toggle_LED();
                MYDelay(1000);
        }
}
```

Problem 3

1. Build the circuit and write code for distance measurement with ultrasonic sensor. Connect the Gnd pin of the sensor before connecting the Vcc pin.
2. Upload your code to STM32 board.
3. Setup Debug(printf) window.
4. Show results.

Solution:

//Already performed in Lab
//Codes are as same as in the "SONAR" folder

```
#include "stm32f446xx.h"

#define VECT_TAB_OFFSET  0x00

static void sys_clk_config(){
	RCC->CR |= RCC_CR_HSION;
	while ((RCC->CR & RCC_CR_HSIRDY) == 0); // Wait until HSI ready

	RCC->CFGR = 0x00000000;

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

	RCC->CFGR &= ~RCC_CFGR_SW;

	RCC->CFGR &= ~RCC_CFGR_HPRE; // 16 MHz, not divided
	// PPRE1: APB Low speed prescaler (APB1)
	RCC->CFGR &= ~RCC_CFGR_PPRE1; // 16 MHz, not divided
	// PPRE2: APB high-speed prescaler (APB2)
	RCC->CFGR &= ~RCC_CFGR_PPRE2; // 16 MHz, not divided

	SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; // Vector Table Relocation in
}

void config_TIM1_CH2(){
	// Enable the clock to GPIO Port A//PA9
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
```

```
	// GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
	GPIOA->MODER &= ~(3UL<<(2*9));
	GPIOA->MODER |=   2UL<<(2*9);      // AF(10)

	GPIOA->AFR[1] &= ~(15UL<<4*(9-8));  // Clear pin 9 for alternate function
	GPIOA->AFR[1] |= (1UL<<(4*(9-8))); // Set pin 9 to alternate function 1
//(enablesTIM4)

	// Configure PullUp/PullDown to No Pull-Up, No Pull-Down
	GPIOA->PUPDR &= ~(3UL << (2*9));

	// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
	GPIOA->OTYPER &= ~(1UL<<9);      // Push-pull

		// Set TIM1 Channel 2 as PWM output
	RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;     // Enable the clock of TIM1

	TIM1->PSC = 160-1;                                  // Set Prescaler

	TIM1->ARR = 0xFFFF;               // Set auto-reload register to 65535

	TIM1->CR1 &= ~TIM_CR1_DIR;

	TIM1->CCMR1 &= ~(TIM_CCMR1_OC2M); // Clear OC2M (Channel 2)
	TIM1->CCMR1 |= (TIM_CCMR1_OC2M_1|TIM_CCMR1_OC2M_2);
	// Enable PWM Mode 1, on Channel 2 = 110
	TIM1->CCMR1 |= (TIM_CCMR1_OC2PE); // Enable output preload bit for channel 2

	TIM1->CR1  |= (TIM_CR1_ARPE); // Set Auto-Reload Preload Enable
	TIM1->CCER |= TIM_CCER_CC2E;                    // Set CC2E Bit
	TIM1->CCER |= TIM_CCER_CC2NE;       // Set CC2NE Bit

	TIM1->BDTR |= TIM_BDTR_MOE | TIM_BDTR_OSSR | TIM_BDTR_OSSI;
	//TIM1->BDTR |= TIM_BDTR_MOE;

	//TIM1->CCR2 &= ~(TIM_CCR2_CCR2);  // Clear CCR2 (Channel 2)
	TIM1->CCR2 = 1;
	TIM1->CR1 |= TIM_CR1_CEN;       // Enable the counter
}

void config_TIM4_CH1() {

	// Set PB.6 as alternate function 2
	RCC->AHB1ENR  |= RCC_AHB1ENR_GPIOBEN;
```

```
	GPIOB->MODER &= ~(3UL<<(2*6));
	GPIOB->MODER |= (2UL<<(2*6));                    // Set to Alternate Function Mode

	GPIOB->OSPEEDR |= (3UL<<(2*6));// Set output speed of the pin to 40MHz

	GPIOB->PUPDR &= ~(3UL << (2*6));       // No PULL UP, NO PULL DOWN

	GPIOB->OTYPER &= ~(1UL<<6);                                    // PUSH PULL

	GPIOB->AFR[0] &= ~(15UL<<(4*6));                 // Clear pin 6 for alternate function
	GPIOB->AFR[0] |= (2UL<<(4*6));   // Set pin 6 to alternate function 2 (enables TIM4)

	RCC->APB1ENR |= RCC_APB1ENR_TIM4EN; // Enable the clock of timer 4

	TIM4->CCMR1 &= ~TIM_CCMR1_CC1S;
	TIM4->CCMR1 |= TIM_CCMR1_CC1S_0; // 01 = inputCC1 is mapped on timer
//Input 1//Setting CCS to 1 will configure this in input capture mode

	TIM4->PSC = 16-1;            // 16M/16=1M
	TIM4->ARR = 0xFFFF; // can count to 65536 us

	// Counting direction: 0 = up-counting, 1 = down-counting
	TIM4->CR1 &= ~TIM_CR1_DIR;

	TIM4->CCMR1 &= ~TIM_CCMR1_IC1F;

	// Select the edge of the active transition
	// Detect only rising edges in this example
	// CC1NP:CC1P bits
	// 00 = rising edge,
	// 01 = falling edge,
	// 10 = reserved,
	// 11 = both edges
	//TIM4->CCER |= (1<<1 | 1<<3);            // Both rising and falling edges.
	TIM4->CCER |= (TIM_CCER_CC1NP|TIM_CCER_CC1P); // Both rising and
//falling

	// IC1PSC[1:0] bits (input capture 1 prescaler)
	TIM4->CCMR1 &= ~(TIM_CCMR1_IC1PSC); // Clear filtering because we need to

	// Enable Capture/compare output enable for channel 1
	TIM4->CCER |= TIM_CCER_CC1E;

	// Enable related interrupts
	TIM4->DIER |= TIM_DIER_CC1IE;       //interrupt for rising and falling edges
	// Enable Capture/Compare interrupts for channel 1
```

```
	TIM4->DIER |= TIM_DIER_UIE;//interrupt for update event// Enable update
//interrupts

	TIM4->CR1 |= TIM_CR1_CEN;	// Enable the counter

	NVIC_SetPriority(TIM4_IRQn, 1); // Set priority to 1

	NVIC_EnableIRQ(TIM4_IRQn);	// Enable TIM4 interrupt in NVIC
}

volatile int overflow = 0;
volatile int current = 0;
volatile int last = 0;
volatile int time = 0;
volatile uint32_t signal_edge= 0; // Assume input is Low initially

void TIM4_IRQHandler(void) {
	if((TIM4->SR & TIM_SR_UIF) != 0) {	// Check if overflow has taken place
		overflow++;
		// If overflow occurred, increment counter
		TIM4->SR &= ~TIM_SR_UIF; // Clear the UIF Flag
	}

	// Captures events with consideration of overflows
	if((TIM4->SR & TIM_SR_CC1IF) != 0) {
		current = TIM4->CCR1;  // Reading CCR1 clears CC1IF
		signal_edge = 1-signal_edge; // will become 1 at a rising edge, o at next falling
		if(signal_edge == 0) time = (current - last) + (overflow*65536);

		last = current;
		overflow = 0;
	}
}



int main(void){
	uint32_t i=0;
	float dist=0;

	sys_clk_config(); // clk = 16MHz

	config_TIM1_CH2();//PA9 for trigger pin
	config_TIM4_CH1();//PB6 for echo pin

	while(1){
```

```
            dist = ((float)time)/58; // in cm and time was in us
            if (time>38000) printf("No obj\r\n"); // greater than 38ms
            else printf("dist: %f cm\r\n", dist);
            for(i=0;i<300000;i++);
        }
}
```

Problem 4

Interface the audio speaker with your micro controller board as you did in previous experiment. Write code in such a way that when there is no object in front of ultrasonic sensor, the speaker makes no sound. When sonar sensor detects any object, the speaker makes a sound and its frequency gradually increases as the object approaches the sonar sensor.

Solution:

Codes of Additional Exercise:

```
#include "stm32f446xx.h"

#define SPEAKER_PORT GPIOA
#define SPEAKER_PIN  0
#define VECT_TAB_OFFSET  0x00

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

            SPEAKER_PORT->MODER  &= ~(0x03 << (2*SPEAKER_PIN));

            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

            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
        SPEAKER_PORT->PUPDR  &= ~(3U<<(2*SPEAKER_PIN));  // No pull-up, no pul
}

static void TIM5_CH1_Init(){
   RCC->APB1ENR |= RCC_APB1ENR_TIM5EN;                // Enable TIMER clock
   TIM5->CR1 &= ~TIM_CR1_DIR;

   TIM5->PSC = 1;       // Prescaler = 23
   TIM5->ARR = 7999;  // Auto-reload: Upcouting (0..ARR), Downcouting (ARR..0)
   TIM5->CCMR1 &= ~TIM_CCMR1_OC1M;  // Clear ouput compare mode bits

        TIM5->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1; // OC1M =
//0011 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;
                TIM5->BDTR |= TIM_BDTR_MOE;

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

static void sys_clk_config(){
        RCC->CR |= 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;
        FLASH->ACR |= FLASH_ACR_ICEN | FLASH_ACR_PRFTEN |
FLASH_ACR_LATENCY_2WS;

        RCC->CFGR &= ~RCC_CFGR_SW;
        RCC->CFGR &= ~RCC_CFGR_HPRE; // 16 MHz, not divided
        // PPRE1: APB Low speed prescaler (APB1)
        RCC->CFGR &= ~RCC_CFGR_PPRE1; // 16 MHz, not divided
        // PPRE2: APB high-speed prescaler (APB2)
        RCC->CFGR &= ~RCC_CFGR_PPRE2; // 16 MHz, not divided
```

```
	SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; // Vector Table Relocation in
//Internal FLASH
}

void config_TIM1_CH2(){

  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
	// GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
	GPIOA->MODER &= ~(3UL<<(2*9));
	GPIOA->MODER |=   2UL<<(2*9);      // AF(10)

	GPIOA->AFR[1] &= ~(15UL<<4*(9-8));  // Clear pin 9 for alternate function
	GPIOA->AFR[1] |= (1UL<<(4*(9-8))); // Set pin 9 to alternate function 1 (enables

	// Configure PullUp/PullDown to No Pull-Up, No Pull-Down
	GPIOA->PUPDR &= ~(3UL << (2*9));

	// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
	GPIOA->OTYPER &= ~(1UL<<9);      // Push-pull

// Set TIM1 Channel 2 as PWM output
	RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;    // Enable the clock of TIM1
	TIM1->PSC = 160-1;                              // Set Prescaler
	TIM1->ARR = 0xFFFF;                  // Set auto-reload register to 65535

	TIM1->CR1 &= ~TIM_CR1_DIR;

	TIM1->CCMR1 &= ~(TIM_CCMR1_OC2M); // Clear OC2M (Channel 2)
	TIM1->CCMR1 |= (TIM_CCMR1_OC2M_1|TIM_CCMR1_OC2M_2);
	// Enable PWM Mode 1, on Channel 2 = 110
	TIM1->CCMR1 |= (TIM_CCMR1_OC2PE); // Enable output preload bit for channel 2

	TIM1->CR1  |= (TIM_CR1_ARPE); // Set Auto-Reload Preload Enable
	TIM1->CCER |= TIM_CCER_CC2E;                    // Set CC2E Bit
	TIM1->CCER |= TIM_CCER_CC2NE;       // Set CC2NE Bit

	TIM1->BDTR |= TIM_BDTR_MOE | TIM_BDTR_OSSR | TIM_BDTR_OSSI;
	//TIM1->BDTR |= TIM_BDTR_MOE;

	//TIM1->CCR2 &= ~(TIM_CCR2_CCR2);  // Clear CCR2 (Channel 2)
	TIM1->CCR2 = 1; // Load the register

	TIM1->CR1 |= TIM_CR1_CEN;       // Enable the counter
}
```

```
void config_TIM4_CH1() {

	// Set PB.6 as alternate function 2
	RCC->AHB1ENR  |= RCC_AHB1ENR_GPIOBEN;


	GPIOB->MODER &= ~(3UL<<(2*6));
  GPIOB->MODER |= (2UL<<(2*6));		// Set to Alternate Function Mode
	GPIOB->OSPEEDR |= (3UL<<(2*6));		// Set output speed of the pin to
//40MHz (Highspeed = 0b11)

	GPIOB->PUPDR &= ~(3UL << (2*6));	// No PULL UP, NO PULL DOWN
	//GPIOB->PUPDR &= 2 << (2*6);	// PULL DOWN

	GPIOB->OTYPER &= ~(1UL<<6);		// PUSH PULL

	GPIOB->AFR[0] &= ~(15UL<<(4*6));	// Clear pin 6 for alternate function
	GPIOB->AFR[0] |= (2UL<<(4*6));		// Set pin 6 to alternate function 2
//(enables TIM4)

	RCC->APB1ENR |= RCC_APB1ENR_TIM4EN; // Enable the clock of timer 4

	////////// Set TIM4 Channel 1 as input capture /////////
	// Set the direction as input and select the active input
	// CC1S[1:0] for channel 1;
	// 00 = output
	// 01 = input, CC1 is mapped on timer Input 1
	// 10 = input, CC1 is mapped on timer Input 2
	// 11 = input, CC1 is mapped on slave timer
	TIM4->CCMR1 &= ~TIM_CCMR1_CC1S;
	TIM4->CCMR1 |= TIM_CCMR1_CC1S_0; // 01 = input, CC1 is mapped on timer
//Input 1

	TIM4->PSC = 16-1;		// 16M/16=1M
	TIM4->ARR = 0xFFFF; // can count to 65536 us

	// Counting direction: 0 = up-counting, 1 = down-counting
	TIM4->CR1 &= ~TIM_CR1_DIR;
	TIM4->CCMR1 &= ~TIM_CCMR1_IC1F;
```

```
        // Select the edge of the active transition
        // Detect only rising edges in this example
        // CC1NP:CC1P bits
        // 00 = rising edge,
        // 01 = falling edge,
        // 10 = reserved,
        // 11 = both edges
        //TIM4->CCER |= (1<<1 | 1<<3);              // Both rising and falling edges.
        TIM4->CCER |= (TIM_CCER_CC1NP|TIM_CCER_CC1P); // Both ri

        TIM4->CCMR1 &= ~(TIM_CCMR1_IC1PSC); // Clear filtering because we need to

        TIM4->CCER |= TIM_CCER_CC1E;

        TIM4->DIER |= TIM_DIER_CC1IE;           //interrupt for rising and falling edges
        // Enable Capture/Compare interrupts for channel 1
        TIM4->DIER |= TIM_DIER_UIE;//interrupt for update eve

        TIM4->CR1 |= TIM_CR1_CEN;       // Enable the counter
        NVIC_SetPriority(TIM4_IRQn, 1); // Set priority to 1
        NVIC_EnableIRQ(TIM4_IRQn);      // Enable TIM4 interrupt in NVIC
}

volatile int overflow = 0;
volatile int current = 0;
volatile int last = 0;
volatile int time = 0;
volatile uint32_t signal_edge= 0; // Assume input is Low initially

void TIM4_IRQHandler(void) {
        if((TIM4->SR & TIM_SR_UIF) != 0) {      // Check if overflow has taken place
                overflow++;
                // If overflow occurred, increment counter
                TIM4->SR &= ~TIM_SR_UIF; // Clear the UIF Flag
                //printf("Hi\r\n");
        }

        // Captures events with consideration of overflows
        if((TIM4->SR & TIM_SR_CC1IF) != 0) {
                current = TIM4->CCR1;  // Reading CCR1 clears CC1IF
                signal_edge = 1-signal_edge; // will become 1 at a rising edge, o at next fallin
                if(signal_edge == 0) time = (current - last) + (overflow*65536);

                last = current;
                overflow = 0;
                //printf("hello\r\n");
        }}
```

```
int main(void){
	uint32_t i=0;
	float dist=0;

	sys_clk_config(); //system clock enabled
	SPEAKER_Pin_Init(); //PA0 configured as speaker PIN

	TIM5_CH1_Init();// for playing music at the speaker PIN
	config_TIM1_CH2();//PA9 for trigger pin
	config_TIM4_CH1();//PB6 for echo pin

	while(1){

		dist = ((float)time)/58; // in cm and time was in us


		if (time>38000)// greater than 38ms
		{
				TIM5->CR1&= (~TIM_CR1_CEN);//TIM5 is disabled
		}
		else
		{
				TIM5->CR1 |= (TIM_CR1_CEN);//TIM5 is enabled
				TIM5->ARR= 7999+(4*dist);//
				//if dist increases then ARR increases and freq decreases
				//if dist decreases then ARR decreases and freq increases
		}
		for(i=0;i<300000;i++);
	}

}
```