Lab Exercise:

```
#include "stm32f446xx.h"


/* Board name: NUCLEO-F446RE


 PA.5  <--> Green LED (LD2)
 PC.13 <--> Blue user button (B1)


 Base Header Code by Dr. Sajid Muhaimin Choudhury, Department of EEE, BUET 22/06/2022


 Based on Instructor Companion of Yifeng Zhu
*/
#define LED_PIN    5


#define BUTTON_PIN 13


volatile uint32_t TimeDelay=0;


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






/////////////////// ENABLE 16MHz CLOCK BY SADMAN SAKIB AHBAB////////////////////////////


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 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;



        // Select HSI as system clock source
        // 00: HSI oscillator selected as system clock
        // 01: HSE oscillator selected as system clock
        // 10: PLL_P selected as system clock
        // 10: PLL_R selected as system clock
        RCC->CFGR &= ~RCC_CFGR_SW;

        // Configure the HCLK, PCLK1 and PCLK2 clocks dividers
        // AHB clock division factor
        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

        // 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
}
////////////////////////////////////////////////////////////////////////////////////////////



/*
 User HSI (high-speed internal) as the processor clock
 See Page 94 on Reference Manual to see the clock tree
 HSI Clock: 16 Mhz, 1% accuracy at 25 oC
 Max Freq of AHB: 84 MHz
 Max Freq of APB2: 84 MHZ
 Max Freq of APB1: 42 MHZ
 SysTick Clock = AHB Clock / 8
*/




static void configure_LED_pin(){
  // 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_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_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
        TimeDelay = 2000;
  uint32_t kk;
        for (kk = 0; kk < TimeDelay; kk++)
        {
                                toggle_LED();
        }
}


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.
        // systick clock is chosen as system clock/8 = 16M/8 = 2M in the
        // [SysTick->CTRL &= ~SysTick_CTRL_CLKSOURCE_Msk;] line.
        // so interrupt freq = 2M/2K = 1KHz, so every 1ms an interrupt is generated
        // systick handler is reducing global volatile variable TimeDealy
        // in the Delay funtion, we set the TimeDealy var to rquired ms,
        // then wait in a while loop for systick handler to reduce it to zero.
        // check with stop watch



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

}
```

Lab Exercise:

```
#include "stm32f446xx.h"

/* Board name: NUCLEO-F446RE

 PA.5  <--> Green LED (LD2)
 PC.13 <--> Blue user button (B1)

 Base Header Code by Dr. Sajid Muhaimin Choudhury, Department of EEE, BUET 22/06/2022

 Based on Instructor Companion of Yifeng Zhu
*/
#define LED_PIN    5

#define BUTTON_PIN 13

volatile uint32_t TimeDelay=0;

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




///////////////// ENABLE 16MHz CLOCK BY SADMAN SAKIB AHBAB/////////////////////////

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 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;



// Select HSI as system clock source
// 00: HSI oscillator selected as system clock
// 01: HSE oscillator selected as system clock
// 10: PLL_P selected as system clock
// 10: PLL_R selected as system clock
RCC->CFGR &= ~RCC_CFGR_SW;

// Configure the HCLK, PCLK1 and PCLK2 clocks dividers
// AHB clock division factor
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

// 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
```

```
}
////////////////////////////////////////////////////////////////////////////////////////////



/*
 User HSI (high-speed internal) as the processor clock
 See Page 94 on Reference Manual to see the clock tree
 HSI Clock: 16 Mhz, 1% accuracy at 25 oC
 Max Freq of AHB: 84 MHz
 Max Freq of APB2: 84 MHZ
 Max Freq of APB1: 42 MHZ
 SysTick Clock = AHB Clock / 8
*/





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




void config_TIM1_CH2(){
////////////////////////////////////////////////////////////////////
//
//      TIM1_CH2 CONNECTED TO PA9 AS AF1
//      WILL BE USED IN OUTPUT PWM MPDE 1 FOR TRIGGER
//      Timer 1: input clock 16 Mhz
//   * prescaler = 159
//   * counter clock frequency = 16 MHz / (159 + 1) = 0.1 MHz
//   * CCR = 1
//   * pulse width = CCR * 1/0.1MHz = 10 us
//   * ARR = 0xFFFF (max: 65535)
//   * period = (ARR + 1) * 1/0.1MHz = 0.6 s
//
////////////////////////////////////////////////////////////////////
        // 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*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 TIM4)
        // 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

        // Counting direction: 0 = up-counting, 1 = down-counting
        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

        // Set Main Output Enable (MOE) bit
        // Set Off-State Selection for Run mode (OSSR) bit
        // Set Off-State Selection for Idle mode (OSSI) 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
}



////////////////////////////////////////////////////////////////////////////////////////////////
// PB.6 (TIM4_CH1): Input capture for the sensor echo
// Timer 4:
//    * input clock = 16 MHz
//    * prescaler = 16-1, input clk frq = 1MHz
//    so period = 1us
void config_TIM4_CH1() {


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


        // MODE: 00: Input mode,            01: General purpose output mode
 //      10: Alternate function mode, 11: Analog mode (reset state)
        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;


// Disable digital filtering by clearing IC1F[3:0] bits

// because we want to capture every event

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 edges.


// Program the input prescaler

// To capture each valid transition, set the input prescaler to zero;

// IC1PSC[1:0] bits (input capture 1 prescaler)
```

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

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

    // Enable related interrupts
    TIM4->DIER |= TIM_DIER_CC1IE;           // Enable Capture/Compare interrupts for channel 1
    TIM4->DIER |= TIM_DIER_UIE;             // 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
        //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 falling edge
```

```
			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(); // clk = 16MHz

		configure_LED_pin();


		config_TIM1_CH2();

		config_TIM4_CH1();


		while(1){

			toggle_LED();

			dist = ((float)time)/58; // in cm

			if (time>38000) printf("No obj\r\n"); // greater than 38ms

			else printf("dist: %f cm\r\n", dist);

			//printf("%d\r\n",i);

			//MYDelay(5000);

			for(i=0;i<300000;i++);

		}


}
```