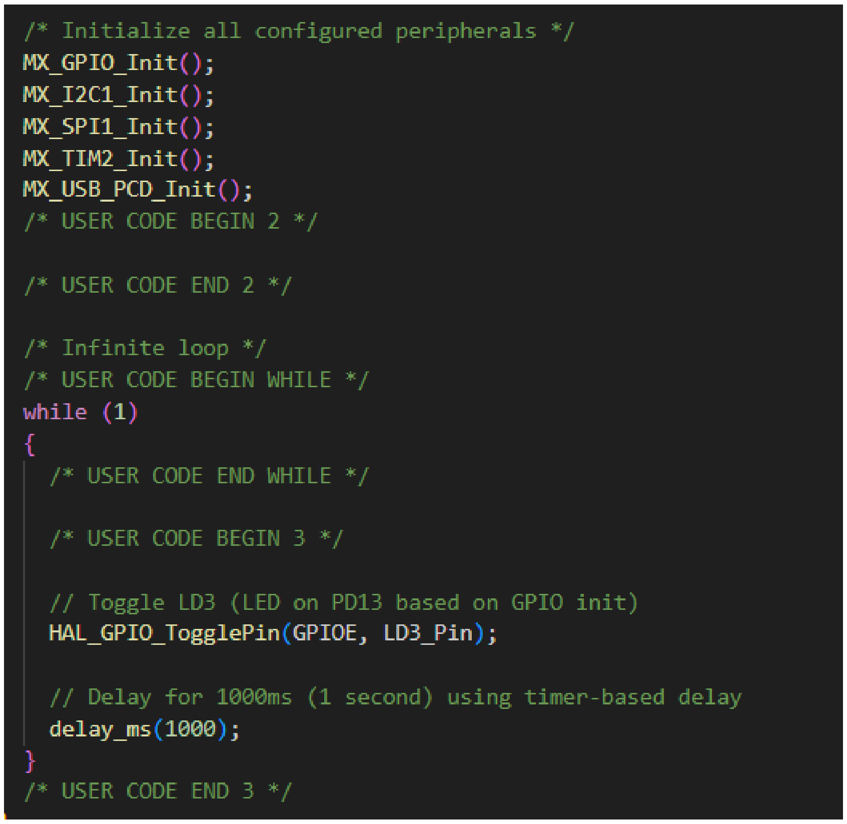
Ali Hussain Zaidi (sz10384), Haider Zaidi (mz10270) lab4

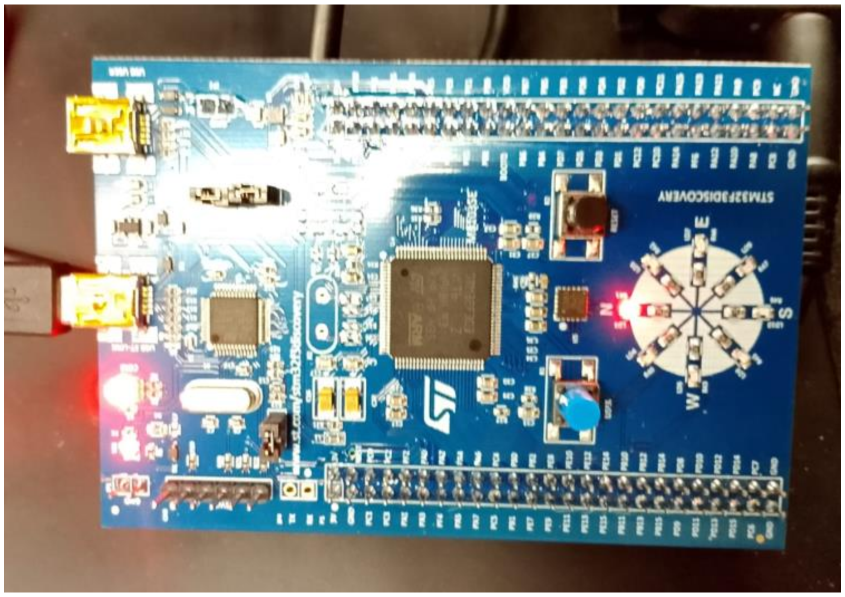
<u>Task # 1</u>

Code snippet



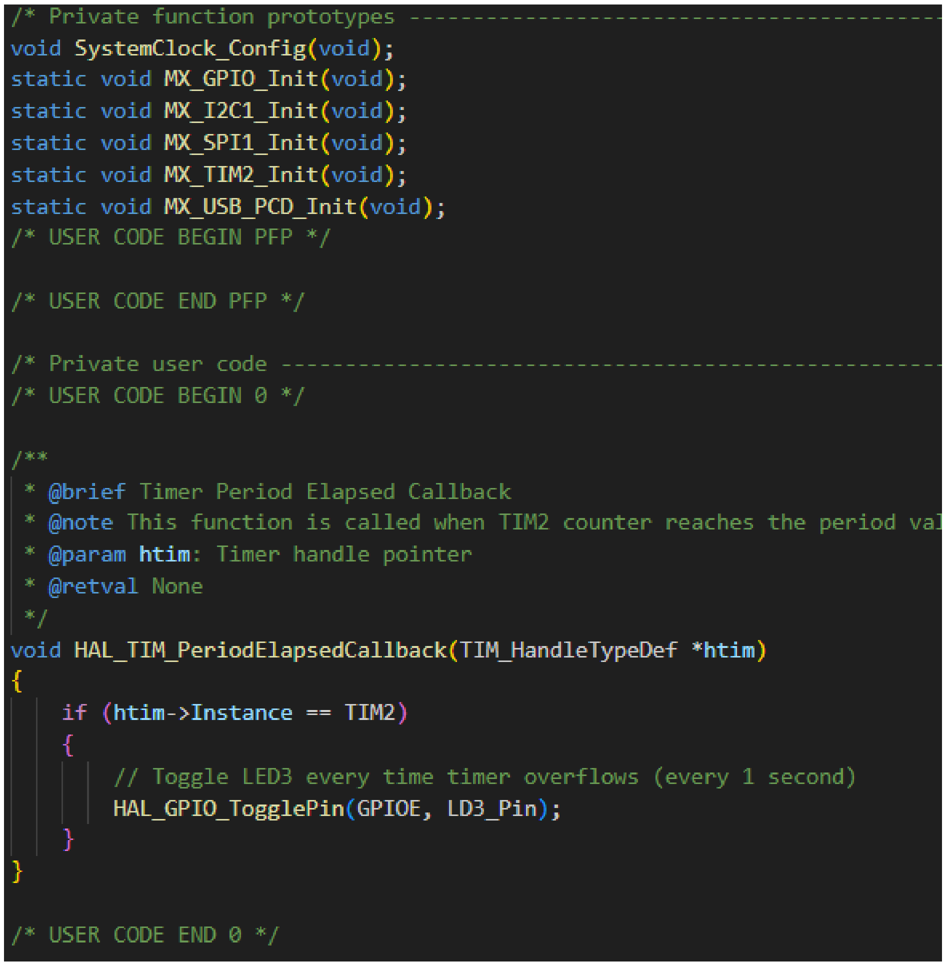
```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_I2C1_Init();
MX_SPI1_Init();
MX_TIM2_Init();
MX_USB_PCD_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  // Toggle LD3 (LED on PD13 based on GPIO init)
  HAL_GPIO_TogglePin(GPIOE, LD3_Pin);

  // Delay for 1000ms (1 second) using timer-based delay
  delay_ms(1000);
}
/* USER CODE END 3 */
```

Output:



<u>Task # 2</u>

Code Snippet:

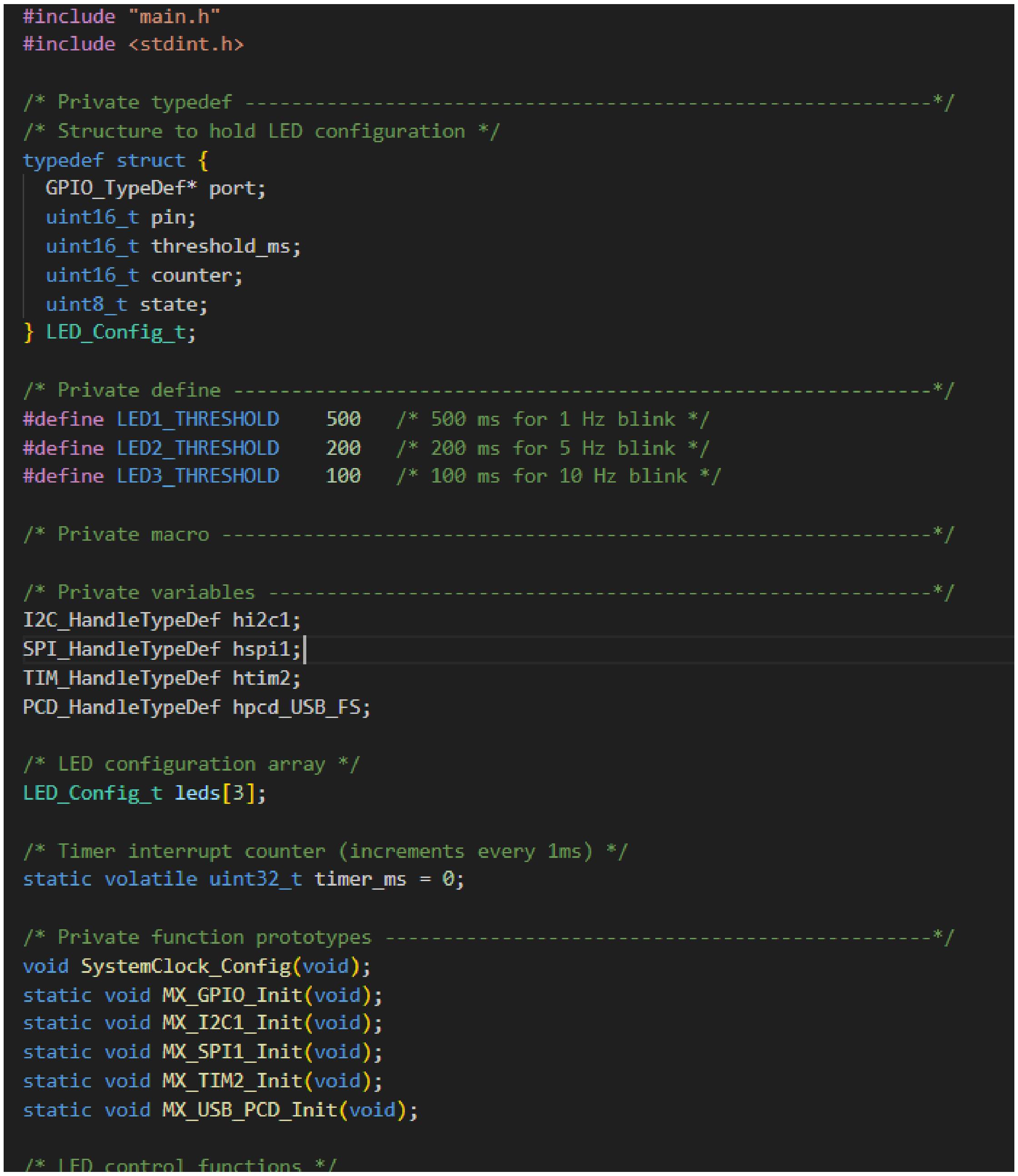


```
/* Private function prototypes -----------------------------------------
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI1_Init(void);
static void MX_TIM2_Init(void);
static void MX_USB_PCD_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code --------------------------------------------------
/* USER CODE BEGIN 0 */

/**
 * @brief Timer Period Elapsed Callback
 * @note This function is called when TIM2 counter reaches the period val
 * @param htim: Timer handle pointer
 * @retval None
 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM2)
    {
        // Toggle LED3 every time timer overflows (every 1 second)
        HAL_GPIO_TogglePin(GPIOE, LD3_Pin);
    }
}

/* USER CODE END 0 */
```

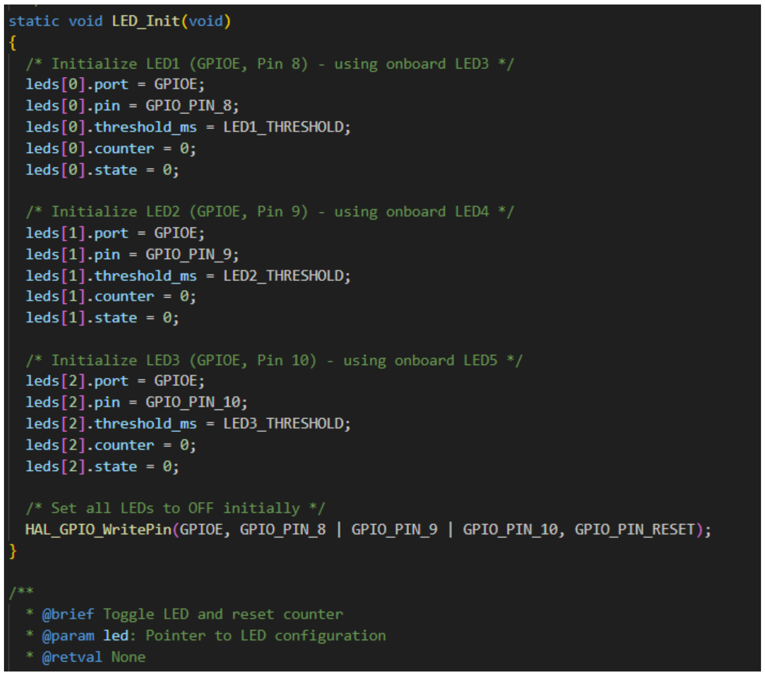
Cannot attach the video

Task # 3:

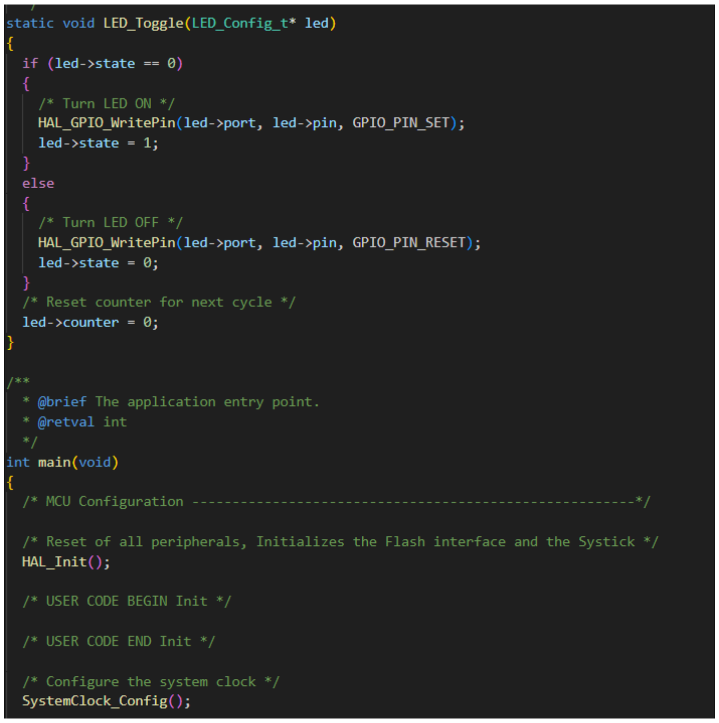
Code Snippet:



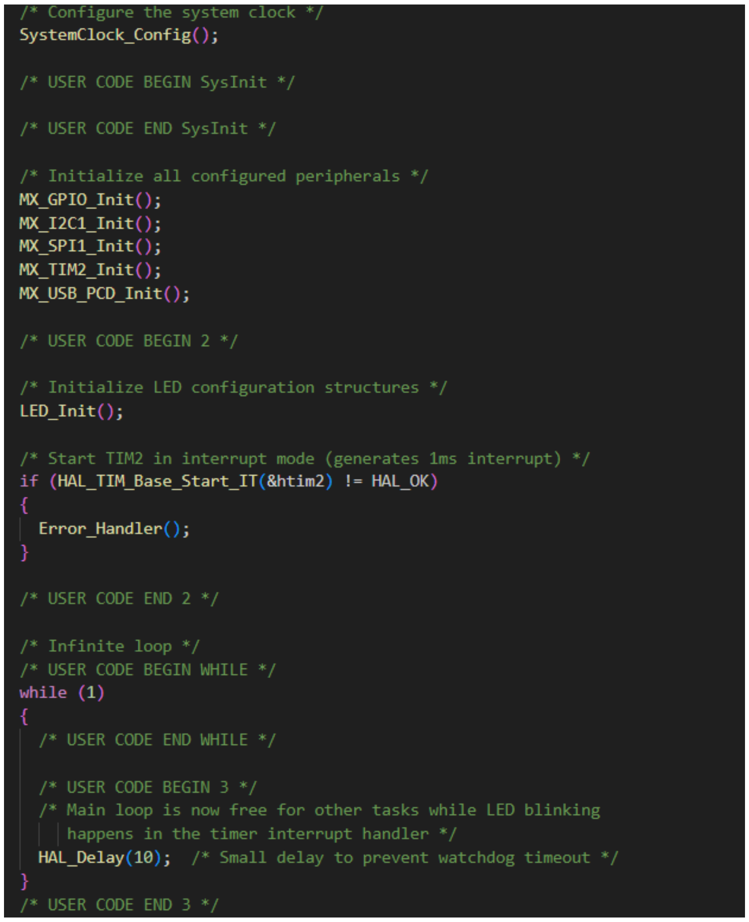
```
#include "main.h"
#include <stdint.h>

/* Private typedef -----------------------------------------------------------*/
/* Structure to hold LED configuration */
typedef struct {
  GPIO_TypeDef* port;
  uint16_t pin;
  uint16_t threshold_ms;
  uint16_t counter;
  uint8_t state;
} LED_Config_t;

/* Private define ------------------------------------------------------------*/
#define LED1_THRESHOLD    500   /* 500 ms for 1 Hz blink */
#define LED2_THRESHOLD    200   /* 200 ms for 5 Hz blink */
#define LED3_THRESHOLD    100   /* 100 ms for 10 Hz blink */

/* Private macro -------------------------------------------------------------*/

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
SPI_HandleTypeDef hspi1;
TIM_HandleTypeDef htim2;
PCD_HandleTypeDef hpcd_USB_FS;

/* LED configuration array */
LED_Config_t leds[3];

/* Timer interrupt counter (increments every 1ms) */
static volatile uint32_t timer_ms = 0;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI1_Init(void);
static void MX_TIM2_Init(void);
static void MX_USB_PCD_Init(void);

/* LED control functions */
```



```c
static void LED_Init(void)
{
  /* Initialize LED1 (GPIOE, Pin 8) - using onboard LED3 */
  leds[0].port = GPIOE;
  leds[0].pin = GPIO_PIN_8;
  leds[0].threshold_ms = LED1_THRESHOLD;
  leds[0].counter = 0;
  leds[0].state = 0;

  /* Initialize LED2 (GPIOE, Pin 9) - using onboard LED4 */
  leds[1].port = GPIOE;
  leds[1].pin = GPIO_PIN_9;
  leds[1].threshold_ms = LED2_THRESHOLD;
  leds[1].counter = 0;
  leds[1].state = 0;

  /* Initialize LED3 (GPIOE, Pin 10) - using onboard LED5 */
  leds[2].port = GPIOE;
  leds[2].pin = GPIO_PIN_10;
  leds[2].threshold_ms = LED3_THRESHOLD;
  leds[2].counter = 0;
  leds[2].state = 0;

  /* Set all LEDs to OFF initially */
  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10, GPIO_PIN_RESET);
}

/**
  * @brief Toggle LED and reset counter
  * @param led: Pointer to LED configuration
  * @retval None
```

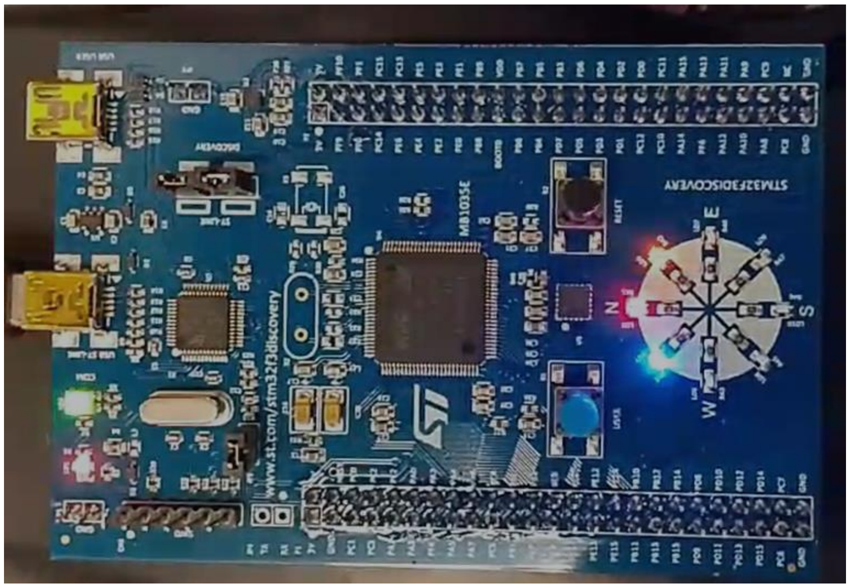


```c
static void LED_Toggle(LED_Config_t* led)
{
  if (led->state == 0)
  {
    /* Turn LED ON */
    HAL_GPIO_WritePin(led->port, led->pin, GPIO_PIN_SET);
    led->state = 1;
  }
  else
  {
    /* Turn LED OFF */
    HAL_GPIO_WritePin(led->port, led->pin, GPIO_PIN_RESET);
    led->state = 0;
  }
  /* Reset counter for next cycle */
  led->counter = 0;
}

/**
  * @brief The application entry point.
  * @retval int
  */
int main(void)
{
  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();
```



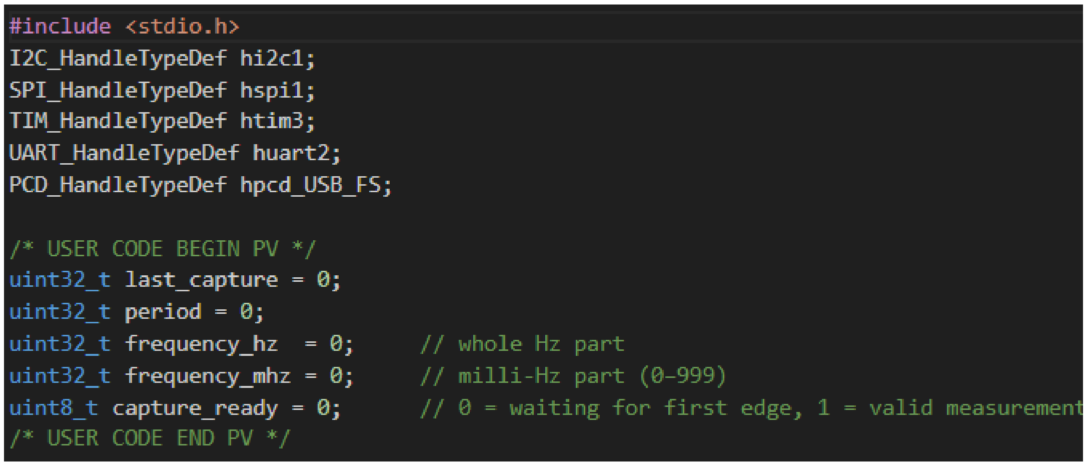
```
/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_I2C1_Init();
MX_SPI1_Init();
MX_TIM2_Init();
MX_USB_PCD_Init();

/* USER CODE BEGIN 2 */

/* Initialize LED configuration structures */
LED_Init();

/* Start TIM2 in interrupt mode (generates 1ms interrupt) */
if (HAL_TIM_Base_Start_IT(&htim2) != HAL_OK)
{
  Error_Handler();
}

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  /* Main loop is now free for other tasks while LED blinking
     happens in the timer interrupt handler */
  HAL_Delay(10);  /* Small delay to prevent watchdog timeout */
}
/* USER CODE END 3 */
```

Output:

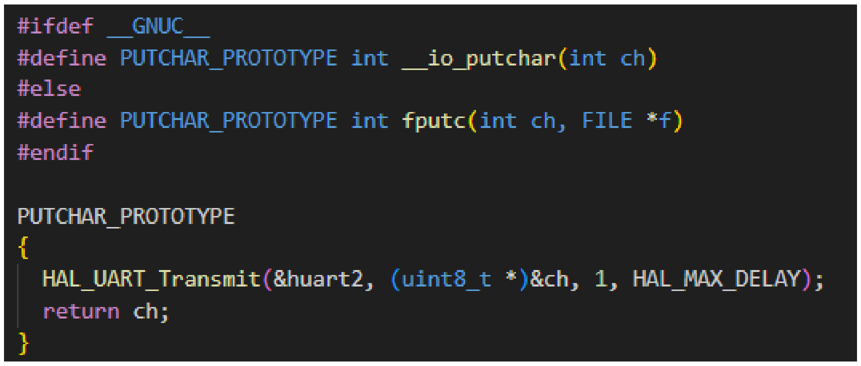


Task # 4

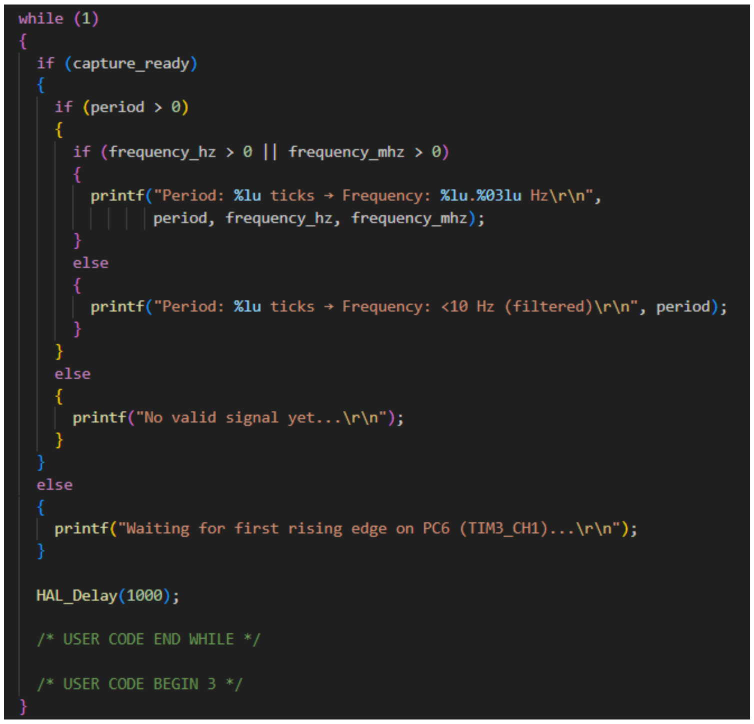
Code Snippet:



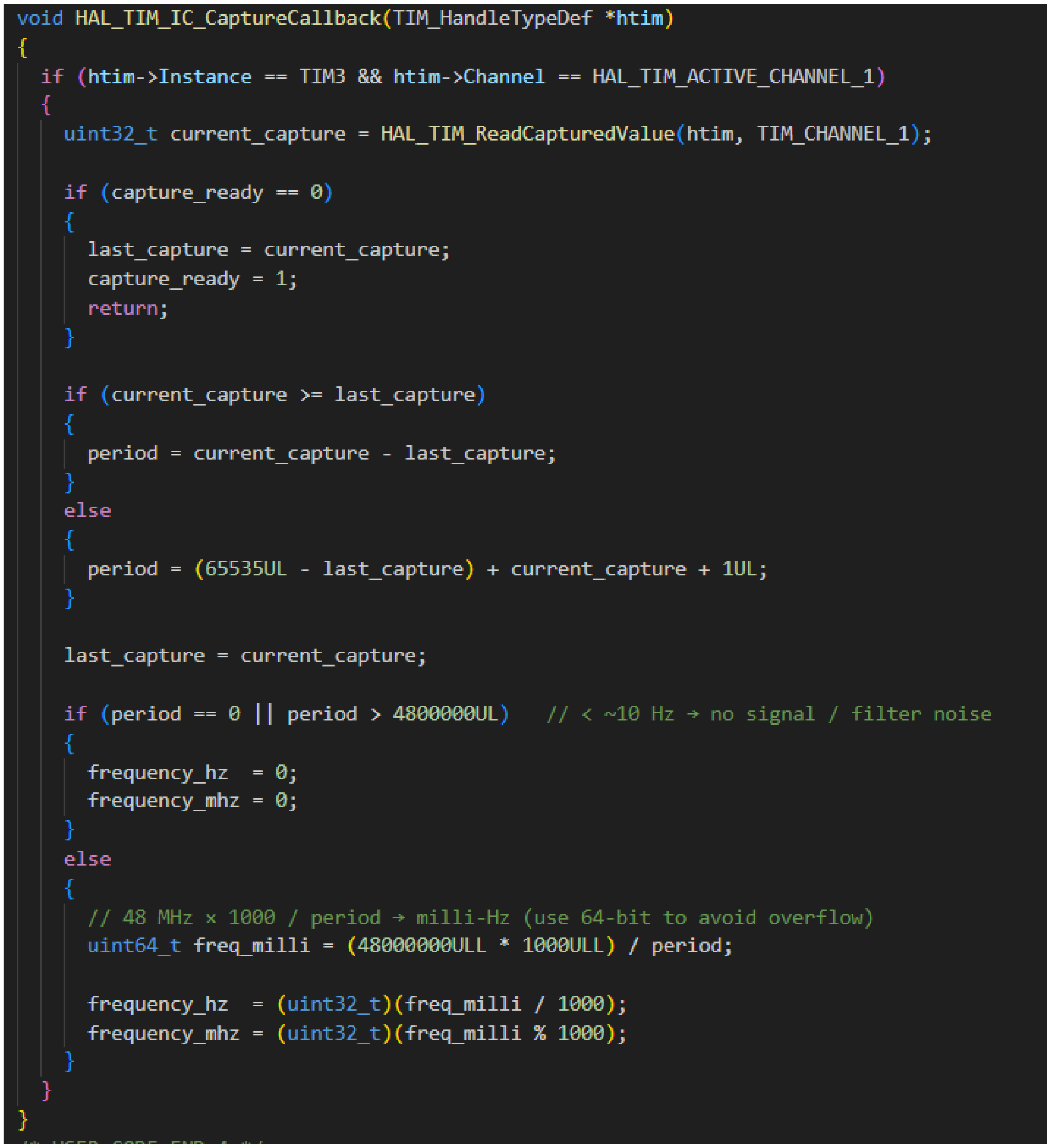
```c
#include <stdio.h>
I2C_HandleTypeDef hi2c1;
SPI_HandleTypeDef hspi1;
TIM_HandleTypeDef htim3;
UART_HandleTypeDef huart2;
PCD_HandleTypeDef hpcd_USB_FS;

/* USER CODE BEGIN PV */
uint32_t last_capture = 0;
uint32_t period = 0;
uint32_t frequency_hz  = 0;     // whole Hz part
uint32_t frequency_mhz = 0;     // milli-Hz part (0-999)
uint8_t capture_ready = 0;      // 0 = waiting for first edge, 1 = valid measurement
/* USER CODE END PV */
```



```c
#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif

PUTCHAR_PROTOTYPE
{
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
  return ch;
}
```

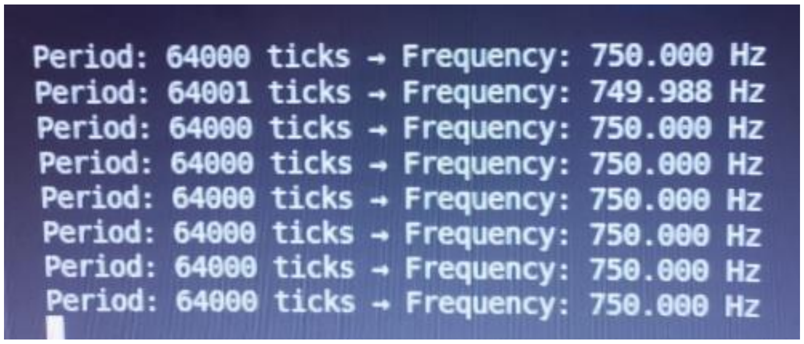


```c
while (1)
{
  if (capture_ready)
  {
    if (period > 0)
    {
      if (frequency_hz > 0 || frequency_mhz > 0)
      {
        printf("Period: %lu ticks → Frequency: %lu.%03lu Hz\r\n",
               period, frequency_hz, frequency_mhz);
      }
      else
      {
        printf("Period: %lu ticks → Frequency: <10 Hz (filtered)\r\n", period);
      }
    }
    else
    {
      printf("No valid signal yet...\r\n");
    }
  }
  else
  {
    printf("Waiting for first rising edge on PC6 (TIM3_CH1)...\r\n");
  }

  HAL_Delay(1000);

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```



```c
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM3 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
  {
    uint32_t current_capture = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);

    if (capture_ready == 0)
    {
      last_capture = current_capture;
      capture_ready = 1;
      return;
    }

    if (current_capture >= last_capture)
    {
      period = current_capture - last_capture;
    }
    else
    {
      period = (65535UL - last_capture) + current_capture + 1UL;
    }

    last_capture = current_capture;

    if (period == 0 || period > 4800000UL)   // < ~10 Hz → no signal / filter noise
    {
      frequency_hz  = 0;
      frequency_mhz = 0;
    }
    else
    {
      // 48 MHz × 1000 / period → milli-Hz (use 64-bit to avoid overflow)
      uint64_t freq_milli = (48000000ULL * 1000ULL) / period;

      frequency_hz  = (uint32_t)(freq_milli / 1000);
      frequency_mhz = (uint32_t)(freq_milli % 1000);
    }
  }
}
```

Output:



```
Period: 64000 ticks → Frequency: 750.000 Hz
Period: 64001 ticks → Frequency: 749.988 Hz
Period: 64000 ticks → Frequency: 750.000 Hz
Period: 64000 ticks → Frequency: 750.000 Hz
Period: 64000 ticks → Frequency: 750.000 Hz
Period: 64000 ticks → Frequency: 750.000 Hz
Period: 64000 ticks → Frequency: 750.000 Hz
Period: 64000 ticks → Frequency: 750.000 Hz
```

What is the advantage of using a hardware timer for delays instead of a for or while loop?

Using a hardware timer set up in STM32CubeMX gives much more accurate and reliable delays because it runs independently of the CPU. Unlike for or while loops that block the processor and depend on compiler behavior, a timer allows the CPU to do other work or sleep, making the code cleaner, more efficient, and closer to real-world STM32F3 design practice.