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

EEE 416 (January 2022) **A1**
Microprocessor and Embedded Systems Laboratory

Final Project Report

µP Trainer with STM32f446 microprocessor

Evaluation Form:

STEP	DESCRIPTION	MAX	SCORE
1	Report (Format, Reference)	10	
2	Design Method and Complete Design (Hardware Implementation)	15	
3	Video Demonstration	10	
4	Novelty of Design	15	
5	Project Management and Cost Analysis	10	
6	Considerations to Public Health and Safety, Environment and Cultural and Societal Needs	10	
7	Assessment of Societal, Health, Safety, Legal and Cultural issues relevant to the solution	10	
8	Evaluation of the sustainability and impact of designed solution in societal and environmental contexts	10	
9	Individual Contribution (Viva)	20	
10	Team work and Diversity	10	
	TOTAL	120	

Signature of Evaluator: _____

Academic Honesty Statement:

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

"In signing this statement, We hereby certify that the work on this project is our own and that we have not copied the work of any other students (past or present), and cited all relevant sources while completing this project. We understand that if we fail to honor this agreement, We will each receive a score of ZERO for this project and be subject to failure of this course."

Full Name: Mrinmoy Kun **Student ID:** 1706001	**Full Name:** **Student ID:** 1706009
Full Name: **Student ID:** 1706007	**Full Name:** **Student ID:** 1706022

Table of Contents

1 Abstract

The main purpose of this project is to implement a microprocessor trainer board using STM32F446RE micro-processor. STM32F446 is an Advanced RISC Machines(ARM) based microprocessor prepared by ST-Electronics mounted on the Nucleo-stm32f446re board. In this project,a goal was set to take advantage of all the peripherals and available customizable pins of the whole nucleo board. Compactness, smoothness and simplicity were the key factors for designing the whole project. STM32F series micro- processors offer a variety of operations such as signal processing ,serial communication, wireless communicating modules interfacing, display interfacing,pwm applications and many other things. This trainer board features almost all the common features stm32f series offers as well as customizable operations such as interfacing LCD-display,LED Matrix,7-segment display,bluetooth module,speaker,rgb,keypad,sonar-sensor, stepper motor, analog to digital and digital to analog converter.

2 Introduction

A trainer board is a general purpose experiment board equipped with different built-in circuits as well as a standard breadboard.Previously in our EEE 416 laboratory,8086 microprocessor trainer kit was used.8086 microprocessor trainer kit is proposed to smooth the progress of learning and developing designs of 8086 microprocessor from Intel. It has facility to connect PC's 101/104 Keyboard making this kit as standalone ,User can enter user programs in assembly languages. User verifies the programs through LCD or PC. User friendly firmware confirms facilitating the beginners learns operations of a microprocessor quickly.

As we know,Arm has become the cornerstone of mobile devices, the world's largest computer ecosystem, and many experts consider Arm the future of cloud computing. Reasons for its growing popularity include its low power consumption, flexible licensing, and low costs.STMicroelectronics widens access to high-performance STM32 microcontroller line with new devices in small memory sizes.The latest STM32F446 microcontrollers from STMicroelectronics introduce new choices for designers by combining enhanced ARM® Cortex®-M4 processing performance, compact 256Kbyte or 512KByte on-chip flash options all with 128Kbyte RAM, efficient memory-extension interfaces, and extended connectivity and communication capabilities.

In our EEE 416 laboratory,we performed different experiments using stm32f446 microcontroller board.For those tasks,we had to construct different circuits and then test our code.To perform different experiment as well as different tasks of the same experiment,the corresponding circuits had to reconstruct again.But,in a trainer board where all the necessary circuits are built,students can test their codes very easily because there is no need to worry about the hardware circuit connections/constructions.Therefore,students can easily focus on the embedded system coding or logic enhancement sector.

3 Design

3.1 Design Method:

The design for this microprocessor trainer board was optimized for smoother and simpler use. The whole design took a lot of tuning to reach the final destination. Basically, main portions of the trainer board are mounted on the pcb board except the keyboard, an additional bread-board, the speaker and a three pin switch. The additional breadboard was used for any general purpose use in the future research and lab works using the Nucleo stm32f446re board. Here, some of the main points of a general design procedure.

3.1.1 Simplicity:

The overall design was kept simple so the overall interface doesn't complicate the user and all the operational tasks become easier to implement. The upper portion contains the LED-Matrix, 7-segment display, LCD display and the switches whereas the lower portion contains the keypad,stepper motor,push buttons for easier on hand experiences.

3.1.2 Compactness:

One of the main objectives of overall design management was to keep proper and balanced compactness,so the pcb board can be configured to a minimal form factor. For balanced compactness,the nucleo board was mounted on the bottom layer of the pcb board and all the other components except the nucleo board were mounted on the upper layer of the pcb board. Minimum distances among the components were maintained so that no components interrupt or disrupt during operating with other components.

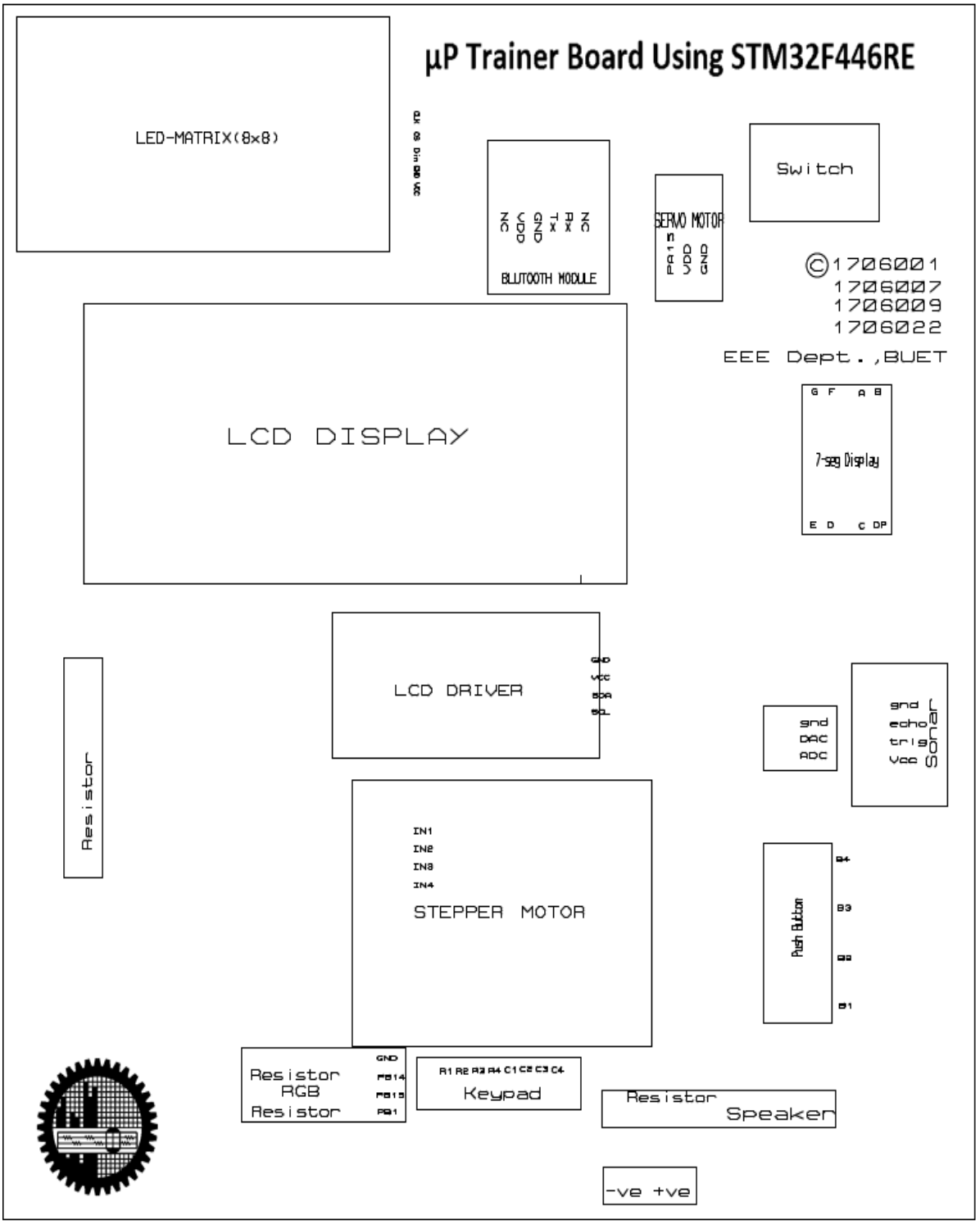
3.1.3 Generalization:

A generalized trainer board was aimed to be implemented from the very start of the overall journey. To implement the idea, an 8-pin dip switch was used so that every pin of the stm32f446re microprocessor can be re-used for several operations. As we know, the stm32f446re can be used for a variety of purposes like signal processing,sampling and discretization,filter analysis and other numerous simulations. An additional breadboard is provided for any circuit connections which might be needed to do the operations. A separate switch is added so the system reset pin can be used easily for the outer interface.

Things we kept in consideration during design implementation:

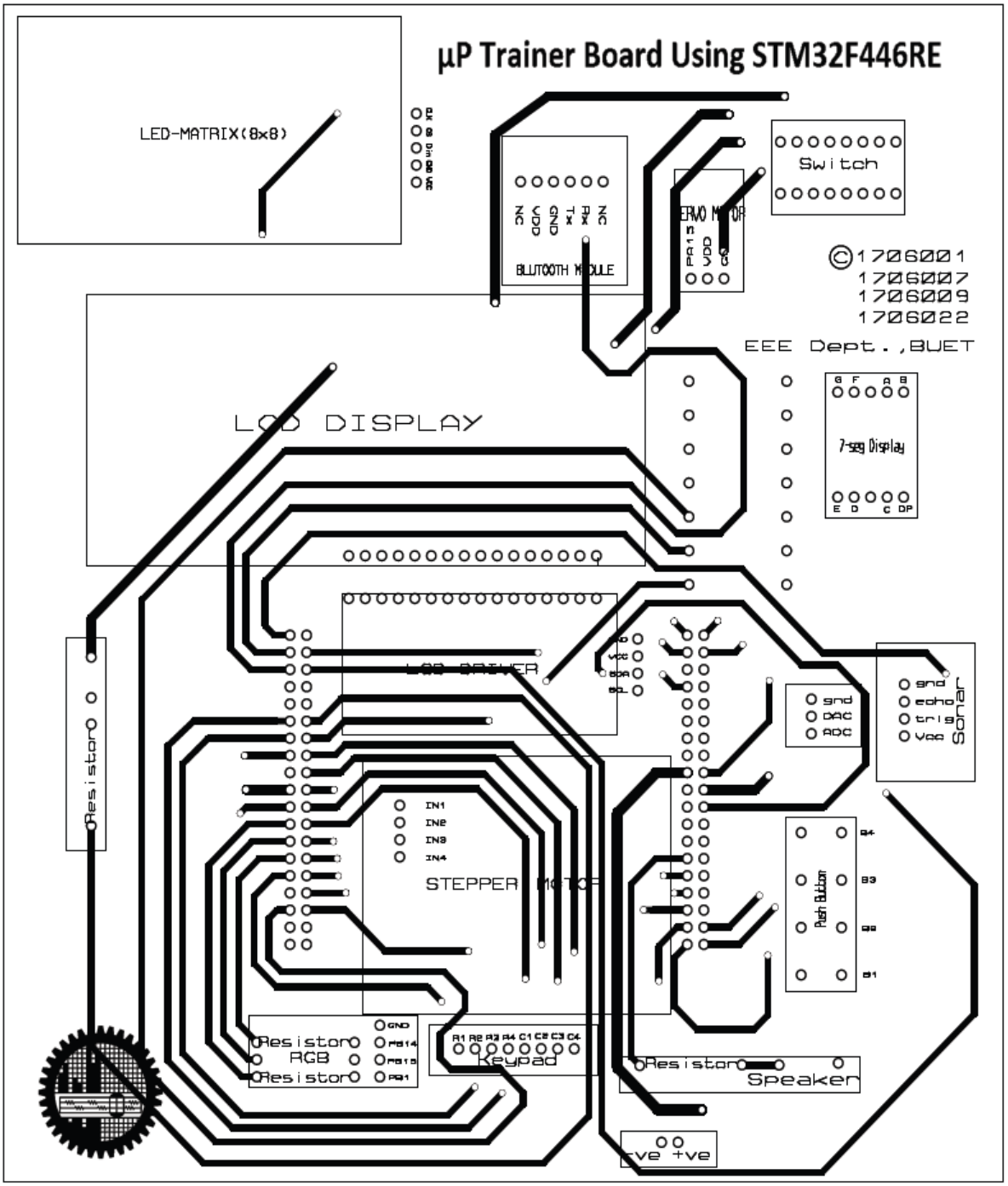
1. We tried to use all the available pins of the nucleo stm32f446re board.
2. During pin allocation to a certain component,we tried to choose nearby pins to draw the traces easily on the pcb board.
3. To generate interrupt properly if needed, we avoided choosing the same pin number from different ports.
4. Convenient timer pins were chosen so that configurations don't become complicated.

Here's a conceptual look of the design :

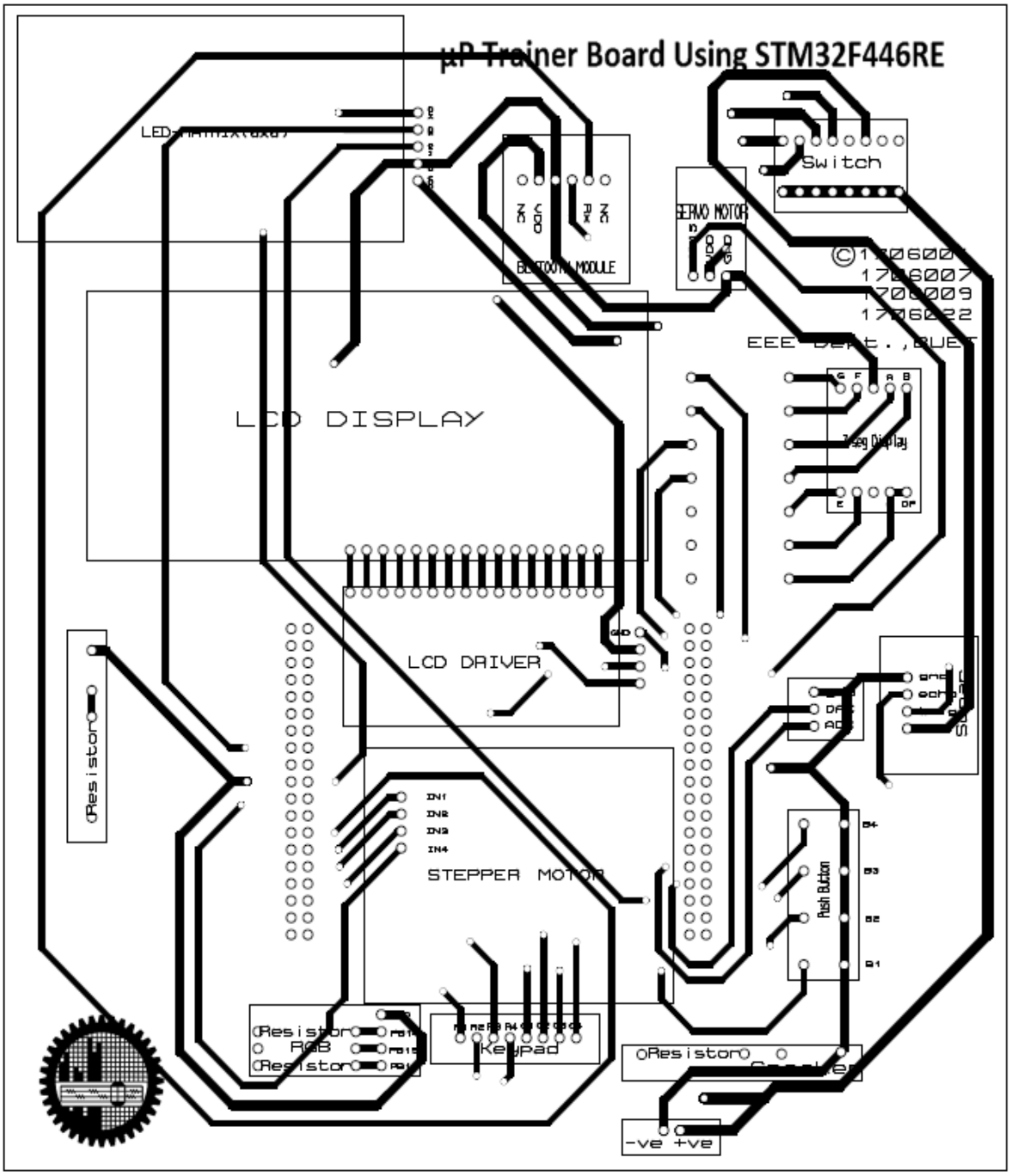


3.2 Circuit Diagram:

3.2.1. Upper Layer Connections



3.2.2: Bottom Layer Connections



3.3 Full Source Code of Firmware

3.3.1 Led blinky(PA5)(keil µ-Vision):

```
#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

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

        /* Enable Power Control clock */
        /* RCC->APB1ENR |= RCC_APB1LPENR_PWRLPEN; */

        // Regulator voltage scaling output selection: Scale 2
        // PWR->CR |= PWR_CR_VOS_1;

        // 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


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

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

static void toggle_LED(){
        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

        // Select PLL as system clock source
        // 00: HSI oscillator selected as system clock
        // 01: HSE oscillator selected as system clock
        // 10: PLL selected as system clock
        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 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

}/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */
```

```
                GPIOA->ODR ^= (1 << LED_PIN);
}

int main(void){
                uint32_t i;

                enable_HSI();
                configure_LED_pin();
                turn_off_LED();
                turn_on_LED();

 // Dead loop & program hangs here
                while(1){
                                                toggle_LED();
                                                for(i=0; i<300000; i++); // simple
delay
                }
}
```

3.3.2 Led I2c(STMcubeIDE):

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE
file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

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

  while (1)
  {
   /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
                lcd_put_cur(0, 0);
                lcd_send_string ("HELLO World");
                HAL_Delay(1000);
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int row=0;
int col=0;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* 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();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */
  lcd_init ();

  lcd_send_string ("HELLO WORLD");

  HAL_Delay(1000);

  lcd_put_cur(1, 0);

  lcd_send_string("from CTECH");

  HAL_Delay(2000);

  lcd_clear ();
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */


 /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */
```

```
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VO
LTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */


  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
```

```
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
 UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */
```

```
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

3.3.3 LED-MATRIX(STMCubeIDE):

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE
file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "max_matrix_stm32.h"
/* USER CODE END Includes */


  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
                  scroll_string((uint8_t *) "Rafi cdi!", 250, left);
                  //write_char ('A', 1);
                  HAL_Delay (1000);
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VO
LTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

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

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* 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();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  max_init (0x03);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
    HAL_RCC_GPIOA_CLK_ENABLE();
```

```
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

3.3.4 Keypad(STMCubeIDE):

```
/* USER CODE BEGIN Header */
/**

************************************************************
******************
```

```
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, Din_Pin|CS_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(Clk_GPIO_Port, Clk_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : Din_Pin CS_Pin */
  GPIO_InitStruct.Pin = Din_Pin|CS_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : Clk_Pin */
  GPIO_InitStruct.Pin = Clk_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(Clk_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
  *

************************************************************
******************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
//#include "i2c-lcd.h"
#include "i2c-lcd.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define R1_PORT GPIOB
#define R1_PIN GPIO_PIN_13

#define R2_PORT GPIOA
#define R2_PIN GPIO_PIN_11
```

```
  * @file           : main.c
  * @brief          : Main program body
******************************************************************************
  ** This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * COPYRIGHT(c) 2019 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without
modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright
notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright
notice,
  *      this list of conditions and the following disclaimer in the
documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its
contributors
  *      may be used to endorse or promote products derived from this
software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
OR CONTRIBUTORS BE LIABLE


/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t key;

char read_keypad (void);


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* 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();

  /* USER CODE BEGIN SysInit */


#define R3_PORT GPIOA
#define R3_PIN GPIO_PIN_10

#define R4_PORT GPIOA
#define R4_PIN GPIO_PIN_12

#define C1_PORT GPIOA
#define C1_PIN GPIO_PIN_8

#define C2_PORT GPIOA
#define C2_PIN GPIO_PIN_9

#define C3_PORT GPIOB
#define C3_PIN GPIO_PIN_6

#define C4_PORT GPIOA
#define C4_PIN GPIO_PIN_7


/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VO
LTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */

                lcd_init();
                lcd_send_cmd(0x80);
                lcd_send_string("KEY: ");
  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
                                key = read_keypad ();

                                if (key!=0x01)
                                {
                                lcd_send_cmd(0x85);
                                lcd_send_data(key);
                                }
  }
  /* USER CODE END 3 */
}

/**

  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
```

```
/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  GPIO_InitStruct.Pin = R1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(R1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : C3_Pin */
  GPIO_InitStruct.Pin = C3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(C3_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
char read_keypad (void)
{
                /* Make ROW 1 LOW and all other ROWs HIGH */
                HAL_GPIO_WritePin (R1_PORT, R1_PIN,
GPIO_PIN_RESET);  //Pull the R1 low
                HAL_GPIO_WritePin (R2_PORT, R2_PIN,
GPIO_PIN_SET);  // Pull the R2 High
                HAL_GPIO_WritePin (R3_PORT, R3_PIN,
GPIO_PIN_SET);  // Pull the R3 High
                HAL_GPIO_WritePin (R4_PORT, R4_PIN,
GPIO_PIN_SET);  // Pull the R4 High

                if (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)))   // if the
Col 1 is low
                {
                                while (!(HAL_GPIO_ReadPin (C1_PORT,
C1_PIN)));   // wait till the button is pressed
                                return '1';
                }

                if (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)))   // if the
Col 2 is low
                {
                                while (!(HAL_GPIO_ReadPin (C2_PORT,
C2_PIN)));   // wait till the button is pressed
                                return '2';
                }

                if (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)))   // if the
Col 3 is low
                {
                                while (!(HAL_GPIO_ReadPin (C3_PORT,
C3_PIN)));   // wait till the button is pressed
                                return '3';
                }

                if (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)))   // if the
Col 4 is low
                {
                                while (!(HAL_GPIO_ReadPin (C4_PORT,
C4_PIN)));   // wait till the button is pressed
```

```
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LD2_Pin|R3_Pin|R2_Pin|R4_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(R1_GPIO_Port, R1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin R3_Pin R2_Pin R4_Pin */
  GPIO_InitStruct.Pin = LD2_Pin|R3_Pin|R2_Pin|R4_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : C4_Pin C1_Pin C2_Pin */
  GPIO_InitStruct.Pin = C4_Pin|C1_Pin|C2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : R1_Pin */


                if (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)))   // if the
Col 3 is low
                {
                        while (!(HAL_GPIO_ReadPin (C3_PORT,
C3_PIN)));   // wait till the button is pressed
                        return '6';
                }

                if (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)))   // if the
Col 4 is low
                {
                        while (!(HAL_GPIO_ReadPin (C4_PORT,
C4_PIN)));   // wait till the button is pressed
                        return 'B';
                }


                /* Make ROW 3 LOW and all other ROWs HIGH */
                HAL_GPIO_WritePin (R1_PORT, R1_PIN,
GPIO_PIN_SET);  //Pull the R1 low
                HAL_GPIO_WritePin (R2_PORT, R2_PIN,
GPIO_PIN_SET);  // Pull the R2 High
                HAL_GPIO_WritePin (R3_PORT, R3_PIN,
GPIO_PIN_RESET);  // Pull the R3 High
                HAL_GPIO_WritePin (R4_PORT, R4_PIN,
GPIO_PIN_SET);  // Pull the R4 High

                if (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)))   // if the
Col 1 is low
                {
                        while (!(HAL_GPIO_ReadPin (C1_PORT,
C1_PIN)));   // wait till the button is pressed
                        return '7';
                }

                if (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)))   // if the
Col 2 is low
                {
                        while (!(HAL_GPIO_ReadPin (C2_PORT,
C2_PIN)));   // wait till the button is pressed
                        return '8';
                }

                if (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)))   // if the
Col 3 is low
                {
                        while (!(HAL_GPIO_ReadPin (C3_PORT,
C3_PIN)));   // wait till the button is pressed
                        return '9';
                }

                if (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)))   // if the
Col 4 is low
                {
                        return 'A';
                }

                /* Make ROW 2 LOW and all other ROWs HIGH */
                HAL_GPIO_WritePin (R1_PORT, R1_PIN,
GPIO_PIN_SET);  //Pull the R1 low
                HAL_GPIO_WritePin (R2_PORT, R2_PIN,
GPIO_PIN_RESET);  // Pull the R2 High
                HAL_GPIO_WritePin (R3_PORT, R3_PIN,
GPIO_PIN_SET);  // Pull the R3 High
                HAL_GPIO_WritePin (R4_PORT, R4_PIN,
GPIO_PIN_SET);  // Pull the R4 High

                if (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)))   // if the
Col 1 is low
                {
                        while (!(HAL_GPIO_ReadPin (C1_PORT,
C1_PIN)));   // wait till the button is pressed
                        return '4';
                }

                if (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)))   // if the
Col 2 is low
                {
                        while (!(HAL_GPIO_ReadPin (C2_PORT,
C2_PIN)));   // wait till the button is pressed
                        return '5';
                }
                        while (!(HAL_GPIO_ReadPin (C2_PORT,
C2_PIN)));   // wait till the button is pressed
                        return '0';
                }

                if (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)))   // if the
Col 3 is low
                {
                        while (!(HAL_GPIO_ReadPin (C3_PORT,
C3_PIN)));   // wait till the button is pressed
                        return '#';
                }

                if (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)))   // if the
Col 4 is low
                {
                        while (!(HAL_GPIO_ReadPin (C4_PORT,
C4_PIN)));   // wait till the button is pressed
                        return 'D';
                }

}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

3.3.5 RGB(STMCubeIDE):

```
                        while (!(HAL_GPIO_ReadPin (C4_PORT,
C4_PIN)));   // wait till the button is pressed
                        return 'C';
                }


                /* Make ROW 4 LOW and all other ROWs HIGH */
                HAL_GPIO_WritePin (R1_PORT, R1_PIN,
GPIO_PIN_SET);  //Pull the R1 low
                HAL_GPIO_WritePin (R2_PORT, R2_PIN,
GPIO_PIN_SET);  // Pull the R2 High
                HAL_GPIO_WritePin (R3_PORT, R3_PIN,
GPIO_PIN_SET);  // Pull the R3 High
                HAL_GPIO_WritePin (R4_PORT, R4_PIN,
GPIO_PIN_RESET);  // Pull the R4 High

                if (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)))   // if the
Col 1 is low
                {
                        while (!(HAL_GPIO_ReadPin (C1_PORT,
C1_PIN)));   // wait till the button is pressed
                        return '*';
                }

                if (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)))   // if the
Col 2 is low
                {

/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim12;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM12_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

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

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
```

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE
file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
  */


  /* USER CODE END Init */

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

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM3_Init();
  MX_TIM12_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);  //B
  HAL_TIM_PWM_Start(&htim12, TIM_CHANNEL_1); //R
  HAL_TIM_PWM_Start(&htim12, TIM_CHANNEL_2); //G
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
//  TIM12->CCR1 = 500;
//  TIM12->CCR2 = 500;
//  TIM3->CCR4 = 500;
  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
                int32_t CH1_DC = 0;
                int32_t R = 0;
                int32_t G = 0;
                int32_t B = 0;
//      while(CH1_DC < 65535)
//      {
//          TIM2->CCR1 = CH1_DC;
//          TIM12->CCR1 = R;
//          TIM12->CCR2 = G;
//          TIM3->CCR4 = B;
//          HAL_Delay(1);
//          CH1_DC += 70;
//      }
//      while(CH1_DC > 0)
//      {
//          TIM2->CCR1 = CH1_DC;
//          TIM12->CCR1 = R;
//          TIM12->CCR2 = G;
//          TIM3->CCR4 = B;
//          HAL_Delay(1);
//          CH1_DC -= 70;
//      }
                int step = 10000;
                for(B = 0; B < 65535;  B = B + 1000){
                        for(G = 0; G < 65535;  G = G + step){
                                for(R = 0; R < 65535;  R = R +
step){
```

```
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

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

  /* USER CODE BEGIN Init */
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VO
LTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 65535;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_ENABLE;

//TIM2->CCR1 = CH1_DC;

TIM12->CCR1 = R;

TIM12->CCR2 = G;

TIM3->CCR4 = B;

HAL_Delay(1);
                                                }
                                        }
                                }
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)

  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 0;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65535;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
```

```
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
 sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */
  HAL_TIM_MspPostInit(&htim3);

}

/**
  * @brief TIM12 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM12_Init(void)
{

  /* USER CODE BEGIN TIM12_Init 0 */

  /* USER CODE END TIM12_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM12_Init 1 */

  /* USER CODE END TIM12_Init 1 */
  htim12.Instance = TIM12;
  htim12.Init.Prescaler = 0;
  htim12.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim12.Init.Period = 65535;
  htim12.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim12.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim12) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim12, &sClockSourceConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim12) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim12, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim12, &sConfigOC,
TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM12_Init 2 */

  /* USER CODE END TIM12_Init 2 */
  HAL_TIM_MspPostInit(&htim12);

}

/**
  * @brief USART2 Initialization Function
  * @param None
  }
  sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */
  __disable_irq();
  while (1)
```

```
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

3.3.6 Servo Motor(keil µ-Vision):

```
#include "stm32f446xx.h"

/* Board name: NUCLEO-F446RE

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

 Base file for LED PWM

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

 Based on Instructor Companion of Yifeng Zhu
*/

#define SPEAKER_PORT GPIOA
#define SPEAKER_PIN  0


#define LED_PORT GPIOA
#define LED_PIN    15

#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[1]     &= ~(0xF <<
(4*(LED_PIN-8)));       //              AF 1 = TIM2_CH1
                                LED_PORT->AFR[1]     |=   0x1 <<
(4*(LED_PIN-8));        //              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(){
                                // tim uptade frequency =
TIM_CLK/(TIM_PSC+1)/(TIM_ARR + 1)
                  // 4000000 / 40 / 1000 = 100Hz
                                // Enable the timer clock
   RCC->APB1ENR                 |= RCC_APB1ENR_TIM2EN;
// Enable TIMER clock

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

   TIM2->PSC = 79;//39;      // Prescaler = 23
 {
 }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
   TIM2->ARR = 1000-1;   // Auto-reload: Upcouting (0..ARR),
Downcouting (ARR..0)
                                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 = 300
                                // -90 = 100
                                // +90 = 500
                                TIM2->CCR1  = 300;        // Output Compare
Register for channel 1
                                TIM2->CR1  |= TIM_CR1_CEN; // Enable
counter
}

//static int brightness = 0;

int main(void){

// Default system clock 4 MHz

                LED_Pin_Init();



                TIM2_CH1_Init(); // Timer to control LED
                int i;

                while(1){

                                                TIM2->CCR1 = 100;      // set angle
-90

                                                for(i=0;i<1000000;i++);
                // delay
                                                TIM2->CCR1 = 300;      // set angle
-0
                                                for(i=0;i<1000000;i++);
                // delay


                                                TIM2->CCR1 = 500;      // set angle
90
                                                for(i=0;i<1000000;i++);
                // delay
                                        }
}
```

3.3.7 Seven Segment Display(STMCubeIDE):

```
/* USER CODE BEGIN Header */
/**
******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
uint8_t buttonState = 0;
uint8_t number = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/*
==========================================================================
If
		the USER BUTTON is pressed then number on 7-segment will increment by 1
else
		Number on the 7-segment will decrement by 1
==========================================================================
*/
// Implement button functionality
void HAL_GPIO_EXTI_Falling_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function should not be modified, when the callback is needed,
         the HAL_GPIO_EXTI_Falling_Callback could be implemented in the user file
   */
  if (GPIO_Pin == USER_BUTTON_Pin) {
		buttonState = 1;
  }
}

void SevenSegment_Update(uint8_t number){
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, ((number>>0)&0x01)); // a
		HAL_GPIO_WritePin(GPIOC, GPIO_PIN_11, ((number>>1)&0x01)); // b

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

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, ((number>>2)&0x01)); // c
		HAL_GPIO_WritePin(GPIOC, GPIO_PIN_6, ((number>>3)&0x01)); // d
		HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, ((number>>5)&0x01)); // e
		HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, ((number>>4)&0x01)); // f
		HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, ((number>>6)&0x01)); // g
}
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
uint8_t segmentNumber[16] = {

		0x3f,  // 0

		0x06,  // 1

		0x5b,  // 2

		0x4f,  // 3

		0x66,  // 4

		0x6d,  // 5

		0x7d,  // 6

		0x07,  // 7

		0x7f,  // 8

		0x6f,  // 9

		0x77,  // A

		0x7c,  // B

		0x39,  // C

		0x5e,  // D

		0x79,  // E

		0x71   // F

		};
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
```

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* 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();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_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 */
                if ( buttonState == 1) {
                                if (number == 15) {
                                                number = 0;
                                } else {
                                                number++;
                                }
                                buttonState = 0;
                } else if (number == 0) {
                                number = 15;
                } else {
                                number--;
                }

                 SevenSegment_Update(segmentNumber[number]);
                 HAL_Delay(1000);
    }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VO
LTAGE_SCALE3);

  /*Configure GPIO pins : PC4 PC6 PC10 PC11
                           PC12 */
  GPIO_InitStruct.Pin =
GPIO_PIN_4|GPIO_PIN_6|GPIO_PIN_10|GPIO_PIN_11
                        |GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PD2 */
  GPIO_InitStruct.Pin = GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LD2_Pin|GPIO_PIN_6,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC,
GPIO_PIN_4|GPIO_PIN_6|GPIO_PIN_10|GPIO_PIN_11
                        |GPIO_PIN_12, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);

  /*Configure GPIO pin : USER_BUTTON_Pin */
  GPIO_InitStruct.Pin = USER_BUTTON_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(USER_BUTTON_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin PA6 */
  GPIO_InitStruct.Pin = LD2_Pin|GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */









////////////////////////////////////////////////////////////////////////////////


/*
 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
```

3.3.8 Sonar(keil µ-Vision):

```
#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
}

                GPIOC->PUPDR &= ~(3UL << (2*8));

                // GPIO Output Type: Output push-pull (0, reset), Output open
drain (1)
                GPIOC->OTYPER &= ~(1UL<<8);      // Push-pull

                        // Set TIM1 Channel 2 as PWM output
```

```
 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_TIM3_CH3(){
//////////////////////////////////////////////////////////////////////
//
//		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_GPIOCEN;
		// GPIO Mode: Input(00), Output(01), AlterFunc(10),
Analog(11, reset)
		GPIOC->MODER &= ~(3UL<<(2*8));
		GPIOC->MODER |=   2UL<<(2*8);     // AF(10)

		GPIOC->AFR[1] &= ~(15UL<<4*(8-8));		// Clear pin
9 for alternate function
		GPIOC->AFR[1] |= (2UL<<(4*(8-8))); // Set pin 9 to alternate
function 1 (enables TIM4)

		// Configure PullUp/PullDown to No Pull-Up, No Pull-Down
		GPIOB->OTYPER &= ~(1UL<<9);
					// PUSH PULL

		GPIOB->AFR[1] &= ~(15UL<<(4*1));
		// Clear pin 6 for alternate function
		GPIOB->AFR[1] |= (1UL<<(4*1));
		// Set pin 6 to alternate function 2 (enables TIM4)

		RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // 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
		TIM2->CCMR1 &= ~TIM_CCMR1_CC2S;
		TIM2->CCMR1 |= TIM_CCMR1_CC2S_0; // 01 = input,
CC1 is mapped on timer Input 1

		TIM2->PSC = 16-1;			// 16M/16=1M
		TIM2->ARR = 0xFFFF; // can count to 65536 us

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

		RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;  // Enable
the clock of TIM1

		TIM3->PSC = 160-1;
		// Set Prescaler

		TIM3->ARR = 0xFFFF;			// Set auto-reload
register to 65535

		// Counting direction: 0 = up-counting, 1 = down-counting
		TIM3->CR1 &= ~TIM_CR1_DIR;

		TIM3->CCMR2 &= ~(TIM_CCMR2_OC3M); // Clear
OC2M (Channel 2)
		TIM3->CCMR2 |=
(TIM_CCMR2_OC3M_1|TIM_CCMR2_OC3M_2);
				// Enable PWM Mode 1, on Channel 2 = 110
		TIM3->CCMR2 |= (TIM_CCMR2_OC3PE); // Enable output
preload bit for channel 2

		TIM3->CR1  |= (TIM_CR1_ARPE);  // Set Auto-Reload
Preload Enable
		TIM3->CCER |= TIM_CCER_CC3E;
		// Set CC2E Bit
		TIM3->CCER |= TIM_CCER_CC3NE;		// 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
		TIM3->BDTR |= TIM_BDTR_MOE | TIM_BDTR_OSSR |
TIM_BDTR_OSSI;
		//TIM1->BDTR |= TIM_BDTR_MOE;

		//TIM1->CCR2 &= ~(TIM_CCR2_CCR2);		// Clear
CCR2 (Channel 2)
		TIM3->CCR3 = 1;
							// Load the register

		TIM3->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_TIM2_CH2() {

		// 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*9));
 GPIOB->MODER |= (2UL<<(2*9));				// Set to
Alternate Function Mode

		GPIOB->OSPEEDR |= (3UL<<(2*9));
		// Set output speed of the pin to 40MHz (Highspeed = 0b11)

		GPIOB->PUPDR &= ~(3UL << (2*9));		// No
PULL UP, NO PULL DOWN
		//GPIOB->PUPDR &= 2 << (2*6);	// PULL DOWN



void TIM2_IRQHandler(void) {
		if((TIM2->SR & TIM_SR_UIF) != 0) {		// Check if
overflow has taken place
			overflow++;

							// If overflow occurred,
increment counter
			TIM2->SR &= ~TIM_SR_UIF;
							// Clear the UIF Flag
			//printf("Hi\r\n");
		}

		// Captures events with consideration of overflows
		if((TIM2->SR & TIM_SR_CC1IF) != 0) {
```

```
	// Disable digital filtering by clearing IC1F[3:0] bits
	// because we want to capture every event
	TIM2->CCMR1 &= ~TIM_CCMR1_IC2F;

	// 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.
	TIM2->CCER |= (TIM_CCER_CC2NP|TIM_CCER_CC2P); // 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)
	TIM2->CCMR1 &= ~(TIM_CCMR1_IC2PSC); // Clear filtering because we need to capture every event

	// Enable Capture/compare output enable for channel 1
	TIM2->CCER |= TIM_CCER_CC2E;

	// Enable related interrupts
	TIM2->DIER |= TIM_DIER_CC2IE;			// Enable Capture/Compare interrupts for channel 1
	TIM2->DIER |= TIM_DIER_UIE;
	// Enable update interrupts

	TIM2->CR1 |= TIM_CR1_CEN;
			// Enable the counter

	NVIC_SetPriority(TIM2_IRQn, 1); // Set priority to 1

	NVIC_EnableIRQ(TIM2_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
			This value must be a multiple of 0x200. */
/*
 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 uint16_t mask;

static void enable_HSI(){

	/* Enable Power Control clock */
	/* RCC->APB1ENR |= RCC_APB1LPENR_PWRLPEN; */

	// Regulator voltage scaling output selection: Scale 2
	// PWR->CR |= PWR_CR_VOS_1;

	// 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
```

```
			current = TIM2->CCR2;  // 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_TIM3_CH3();
		config_TIM2_CH2();

		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++);
		}

}
```

3.3.9 Speaker(keil µ-Vision):

```
#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

 Music Generation 7.5 base file

 Based on Instructor Companion of Yifeng Zhu
*/

#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.
		// 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

		// Select PLL as system clock source
		// 00: HSI oscillator selected as system clock
		// 01: HSE oscillator selected as system clock
		// 10: PLL selected as system clock
		RCC->CFGR &= ~RCC_CFGR_SW;
		RCC->CFGR |= RCC_CFGR_SW_1;
		// while ((RCC->CFGR & RCC_CFGR_SWS_PLL) != RCC_CFGR_SWS_PLL);
```

```
    // 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 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;


                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(){
    //function for musical frequency in timer5
    //tim uptade frequency =
TIM_CLK/(TIM_PSC+1)/(TIM_ARR + 1)
    // 16000000 / 2 / 2000 = 50Hz
                // Enable the timer clock
  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


    // 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 B
  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)




                                                        2, 2, 2, 2,
1, 1, 1, 1,



                                                        0, 0, 0, 0,
0, 0, 0, 0};  // 0=C, 1=D ....


// Default system clock 4 MHz

    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 > 32 ||
current_note < 0) current_note = 0;
```

```
    TIM5->CCMR1 |= TIM_CCMR1_OC1PE;                  // Output 1
preload enable

                        // Select output polarity: 0 = active high, 1 =
active low
                        TIM5->CCMR1 |= 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[8] = {261,                294, 329, 349, 392, 440,
494, 522}; //Hz
                        static uint16_t song_notes[32] = {2, 2, 3, 2, 4, 4,
4, 4,


                                                                1, 1, 2, 1,
3, 3, 3, 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 disabSTEPPER_PIN_

                // 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 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;
                                        for(i=0;i<1000000;i++);
                                // delay
                }
}
```

3.3.10 Stepper Motor(keil µ-Vision):

```
#include "stm32f446xx.h"
/* Board name: NUCLEO-F446RE
 PA.5  <--> Green STEPPER_PIN_ (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 STEPPER_PIN_1    4
#define STEPPER_PIN_2    5
#define STEPPER_PIN_3    3
#define STEPPER_PIN_4    2
#define STEPPER_PORT GPIOB

#define BUTTON_PIN 1
#define BUTTON_PORT GPIOC

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

                /* Enable Power Control clock */
                /* RCC->APB1ENR |= RCC_APB1LPENR_PWRLPEN; */

                // Regulator voltage scaling output selection: Scale 2
                // PWR->CR |= PWR_CR_VOS_1;

                // 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);

 // Enable the clock to GPIO Port B
 RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;

                // GPIO Mode: Input(00), Output(01), AlterFunc(10),
Analog(11, reset)
                STEPPER_PORT->MODER &=
~(3UL<<(2*STEPPER_PIN_1));
                STEPPER_PORT->MODER |=
1UL<<(2*STEPPER_PIN_1);     // Output(01)
                STEPPER_PORT->MODER &=
~(3UL<<(2*STEPPER_PIN_2));
                STEPPER_PORT->MODER |=
1UL<<(2*STEPPER_PIN_2);     // Output(01)
                STEPPER_PORT->MODER &=
~(3UL<<(2*STEPPER_PIN_3));
                STEPPER_PORT->MODER |=
1UL<<(2*STEPPER_PIN_3);     // Output(01)
                STEPPER_PORT->MODER &=
~(3UL<<(2*STEPPER_PIN_4));
                STEPPER_PORT->MODER |=
1UL<<(2*STEPPER_PIN_4);     // Output(01)

                // GPIO Speed: Low speed (00), Medium speed (01), Fast
speed (10), High speed (11)
                STEPPER_PORT->OSPEEDR &=
~(3U<<(2*STEPPER_PIN_1));
                STEPPER_PORT->OSPEEDR |=
2U<<(2*STEPPER_PIN_1);  // Fast speed
                STEPPER_PORT->OSPEEDR &=
~(3U<<(2*STEPPER_PIN_2));
                STEPPER_PORT->OSPEEDR |=
2U<<(2*STEPPER_PIN_2);  // Fast speed
```

```
	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

	// Select PLL as system clock source
	// 00: HSI oscillator selected as system clock
	// 01: HSE oscillator selected as system clock
	// 10: PLL selected as system clock
	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 configure_STEPPER_PIN__pin(){
	BUTTON_PORT->OTYPER &= ~(1U<<BUTTON_PIN); // Push-pull

	// GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
	BUTTON_PORT->PUPDR  &= ~(3U<<(2*BUTTON_PIN)); // No pull-up, no pull-down
	BUTTON_PORT->PUPDR  |= 1U<<(2*BUTTON_PIN);

}
static void turn_on_STEPPER_PIN_(){
	STEPPER_PORT->ODR |= 1U << STEPPER_PIN_1;
}
static void turn_off_STEPPER_PIN_(){
	STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_1);
}
static void toggle_STEPPER_PIN_(){
	STEPPER_PORT->ODR ^= (1 << STEPPER_PIN_1);
}
int main(void){
	uint32_t i;

	enable_HSI();
	configure_STEPPER_PIN__pin();
	configure_PUSH_button();


	int delay = 10000;
	while(1){
			STEPPER_PORT->ODR |= 1U << STEPPER_PIN_1;
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_2);
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_3);
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_4);
			for(i=0; i<delay; i++);

			STEPPER_PORT->ODR |= 1U << STEPPER_PIN_2;
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_1);
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_3);
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_4);
			for(i=0; i<delay; i++);

			STEPPER_PORT->ODR |= 1U << STEPPER_PIN_3;
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_1);
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_2);
			STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_4);
			for(i=0; i<delay; i++);

			STEPPER_PORT->ODR |= 1U << STEPPER_PIN_4;
```

```
	STEPPER_PORT->OSPEEDR &= ~(3U<<(2*STEPPER_PIN_3));
	STEPPER_PORT->OSPEEDR |= 2U<<(2*STEPPER_PIN_3);  // Fast speed
	STEPPER_PORT->OSPEEDR &= ~(3U<<(2*STEPPER_PIN_4));
	STEPPER_PORT->OSPEEDR |= 2U<<(2*STEPPER_PIN_4);  // Fast speed

	// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
	STEPPER_PORT->OTYPER &= ~(1U<<STEPPER_PIN_1); // Push-pull
	STEPPER_PORT->OTYPER &= ~(1U<<STEPPER_PIN_2); // Push-pull
	STEPPER_PORT->OTYPER &= ~(1U<<STEPPER_PIN_3); // Push-pull
	STEPPER_PORT->OTYPER &= ~(1U<<STEPPER_PIN_4); // Push-pull

	// GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
	STEPPER_PORT->PUPDR  &= ~(3U<<(2*STEPPER_PIN_1));  // No pull-up, no pull-down
	STEPPER_PORT->PUPDR  &= ~(3U<<(2*STEPPER_PIN_2));  // No pull-up, no pull-down
	STEPPER_PORT->PUPDR  &= ~(3U<<(2*STEPPER_PIN_3));  // No pull-up, no pull-down
	STEPPER_PORT->PUPDR  &= ~(3U<<(2*STEPPER_PIN_4));  // No pull-up, no pull-down

}
static void configure_PUSH_button(){
  // Enable the clock to GPIO Port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;

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

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

	// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
```

```
                                STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_2);
                                STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_3);
                                for(i=0; i<delay; i++);

                                STEPPER_PORT->ODR |= 1U << STEPPER_PIN_3;
                                STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_1);
                                STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_2);
                                STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_4);
                                for(i=0; i<delay; i++);

                                STEPPER_PORT->ODR |= 1U << STEPPER_PIN_2;
                                STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_1);
                                STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_3);
                                STEPPER_PORT->ODR &= ~(1U << STEPPER_PIN_4);
                                for(i=0; i<delay; i++);
        }
        }
}
```

3.3.11 Bluetooth Module(STMCubeIDE):

```
/* USER CODE BEGIN Header */
```

```
                        STEPPER_PORT->ODR &= ~(1U <<
STEPPER_PIN_1);
                        STEPPER_PORT->ODR &= ~(1U <<
STEPPER_PIN_2);
                        STEPPER_PORT->ODR &= ~(1U <<
STEPPER_PIN_3);
                        for(i=0; i<delay; i++);

                        while(GPIOC->IDR & GPIO_IDR_IDR_1){
                                STEPPER_PORT->ODR |= 1U <<
STEPPER_PIN_1;
                                STEPPER_PORT->ODR &= ~(1U
<< STEPPER_PIN_2);
                                STEPPER_PORT->ODR &= ~(1U
<< STEPPER_PIN_3);
                                STEPPER_PORT->ODR &= ~(1U
<< STEPPER_PIN_4);
                                for(i=0; i<delay; i++);

                                STEPPER_PORT->ODR |= 1U <<
STEPPER_PIN_4;
                                STEPPER_PORT->ODR &= ~(1U
<< STEPPER_PIN_1);
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim3;

UART_HandleTypeDef huart3;

/* USER CODE BEGIN PV */

int flag,j,a;
uint8_t rx;
char str1[60]={0};

int len;
char buffer[100];

int i;


/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM3_Init(void);
static void MX_USART3_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

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

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
if(huart==&huart3)
{
if(rx == 0x0A)
        flag=1;
        str1[a]=rx;
 a++;
}
HAL_UART_Receive_IT(&huart3,&rx,1);
}


void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
if(htim==&htim3)
{
        sprintf(buffer,"%i Rafi Tushar Mrinmoy Voktho\r\n",i);
        i++;
                len=strlen(buffer);
                HAL_UART_Transmit(&huart3,buffer,len,100);
}
}

/* USER CODE END 0 */

/**
```

```
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE
file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "string.h"
#include "stdio.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */



  /* 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();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */

  HAL_UART_Receive_IT(&huart3,&rx,1);
  HAL_TIM_Base_Start_IT(&htim3);

  /* USER CODE END 2 */

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

                        if(flag==1)
                        {
                                a=0;
        HAL_Delay(50);
        for(j=0;j<60;j++){

                        if((str1[j]=='l') && (str1[j+1]=='e') &&
(str1[j+2]=='d') && (str1[j+3]=='o') && (str1[j+4]=='n') ){
                        //HAL_Delay(1000);
                        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_13,0);
```

```
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VO
LTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 7999;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 1999;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) !=
HAL_OK)
                        for(j=0;j<60;j++){str1[j]=0;}

                }}

                for(j=0;j<60;j++){

                        if((str1[j]=='l') && (str1[j+1]=='e') &&
(str1[j+2]=='d') && (str1[j+3]=='o') && (str1[j+4]=='f') &&
(str1[j+5]=='f') ){
                        //HAL_Delay(1000);
                        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_13,1);
                        for(j=0;j<60;j++){str1[j]=0;}

                        }}
                flag=0;
                        }

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
 }
 /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

  if (HAL_TIMEx_MasterConfigSynchronization(&htim3,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */

}

/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */

  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 9600;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */

  /* USER CODE END USART3_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
```

```
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;


/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

```
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
```

Table: Source Code for the main program

4 Implementation

4.1 Description

Using Nucleo-64 STM32F446RE, goal of our project is to implement a compact trainer board which will host the following functionalities

- LCD
- 7 Segment Display
- 8x8 LED matrix
- Generation of PWM signal
    - LED dimmer
    - RGB LED
- ADC and DAC Port
- Speaker
- Stepper motor
- Servo motor
- Sonar
- Serial communication.
    - I2C (With LCD)
    - Synchronous Serial Interface (SSI) (With 8x8 LED matrix)
    - USART (With Bluetooth module)

- Input device
    - Key pad
    - Push button (4)

We incorporate a standard dual in-line package (DIP) switch on to the trainer board which will control the 5V volt power connection to following modules

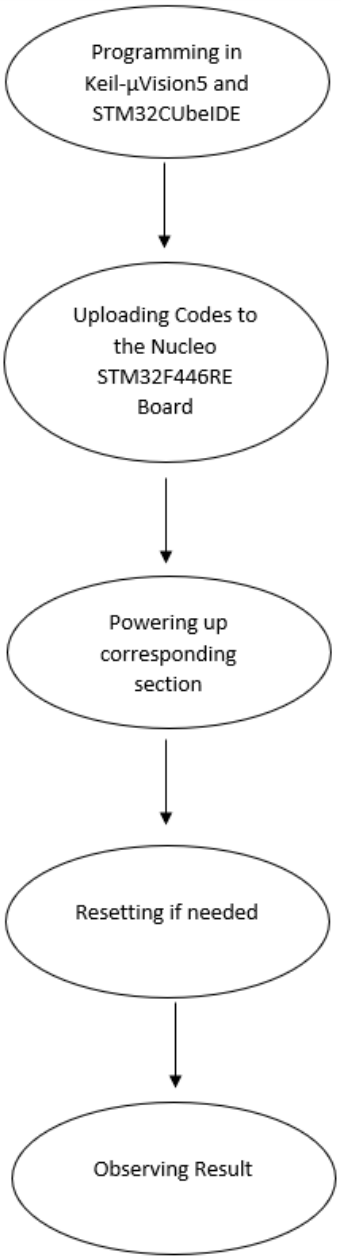
Switch Number	Connected Module
1	Bluetooth
2	Servo Motor
3	8x8 LED Matrix
4	LCD
5	Stepper Motor
6	Sonar

Since, all of these modules on the trainer board will draw their power from the STM32 Nucleo board, it is a good strategy to isolate the power when they are not being used to reduce excessive power stress on the STM32.

For the stepper motor, we provide a dedicated (5V,GND) port to which we can connect using a jumper wire connection.

An extra ‘Reset Button’ was attached to the side of the trainer board to give hardware reset flexibility to the user.

Process flow:



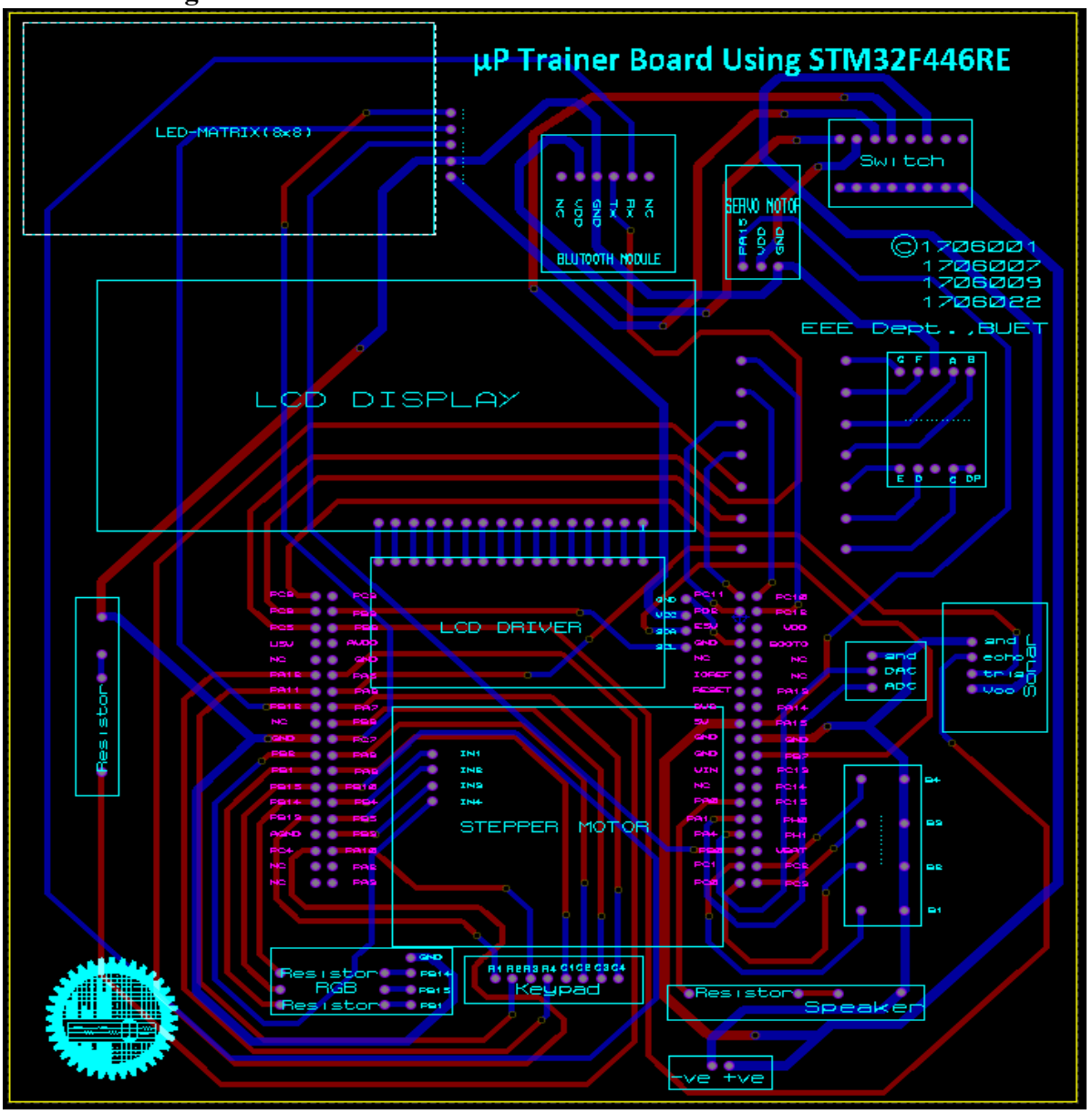
4.1.1 Pin Allocation of the Nucleo-64 STM32F446RE

Functionality	Pin No.	Remarks	Alternate Function
Speaker	PA0		AF1 (TIM2_CH1) AF2(TIM5_CH1)
ADC	PA1		
DAC	PA4		
NT	PA2		

	PA3		
	PA13		
	PA14		
	PC14		
	PC15		
Keypad	PA8	C1	
	PA9	C2	
	PB6	C3	
	PA7	C4	
	PB13	R1	
	PA11	R2	
	PA10	R3	
	PA12	R4	
Servo Motor	PA15		AF1(TIM2_CH1)
RGB	PB1	Blue	AF2 (TIM3_CH4)
	PB15	Green	AF9 (TIM12_CH2)
	PB14	Red	AF9(TIM12_CH1)
Stepper Motor	PB4	A1	AF2(TIM3_CH1)
	PB5	A2	AF2(TIM3_CH2)
	PB3	A3	AF1(TIM2_CH2)
	PB2	A4	AF1(TIM2_CH4)
LCD with I2C Driver	PB8	SCL	AF4(I2C1_SCL)
	PB7	SDA	AF4(I2C1_SDA)
7 Segment	PC10	f	
	PC12	g	
	PC11	b	
	PD2	a	
	PA6	c	
	PC6	d	
	PC4	e	
Bluetooth	PB10	Tx	AF7(USART3_TX)

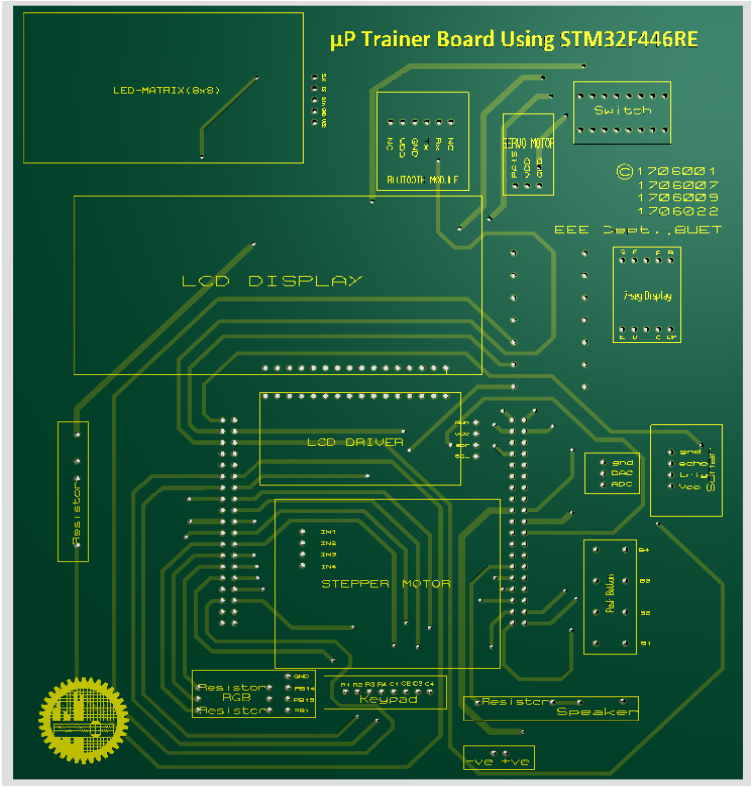
	PC5	Rx	AF8(USART6_RX)
LED Matrix with MAX7219 Driver	PB12	CS	
	PB0	Din	
	PC7	Clk	
Push Button	PC1	B1	
	PC0	B2	
	PC3	B3	
	PC2	B4	
Built-in LED	PA5		AF1(TIM2_CH1)
Sonar	PB9	echo	AF1(TIM2_CH2) AF2(TIM4_CH4)
	PC8	trig	AF2(TIM3_CH3) AF3(TIM8_CH3)

4.1.2 PCB Design in Proteus 8.13 Pro

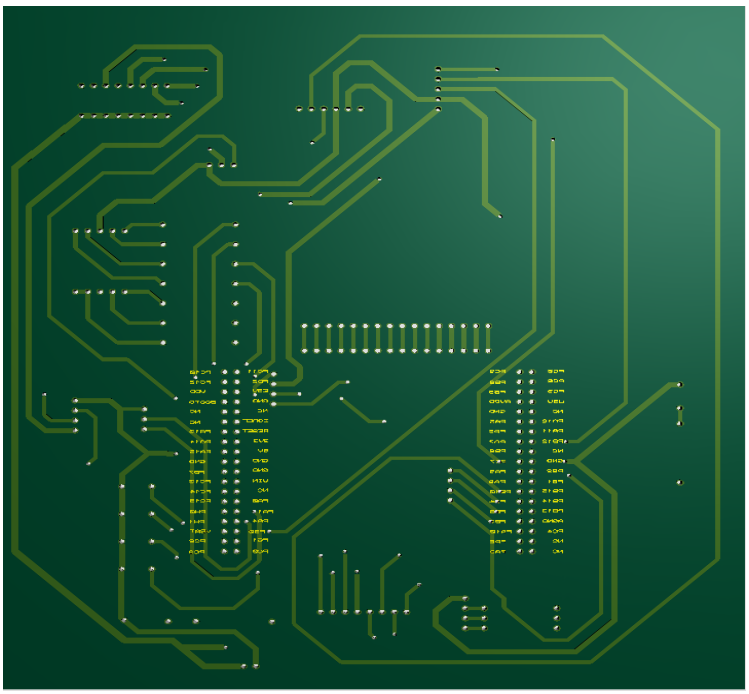


4.1.3 3D visualizer

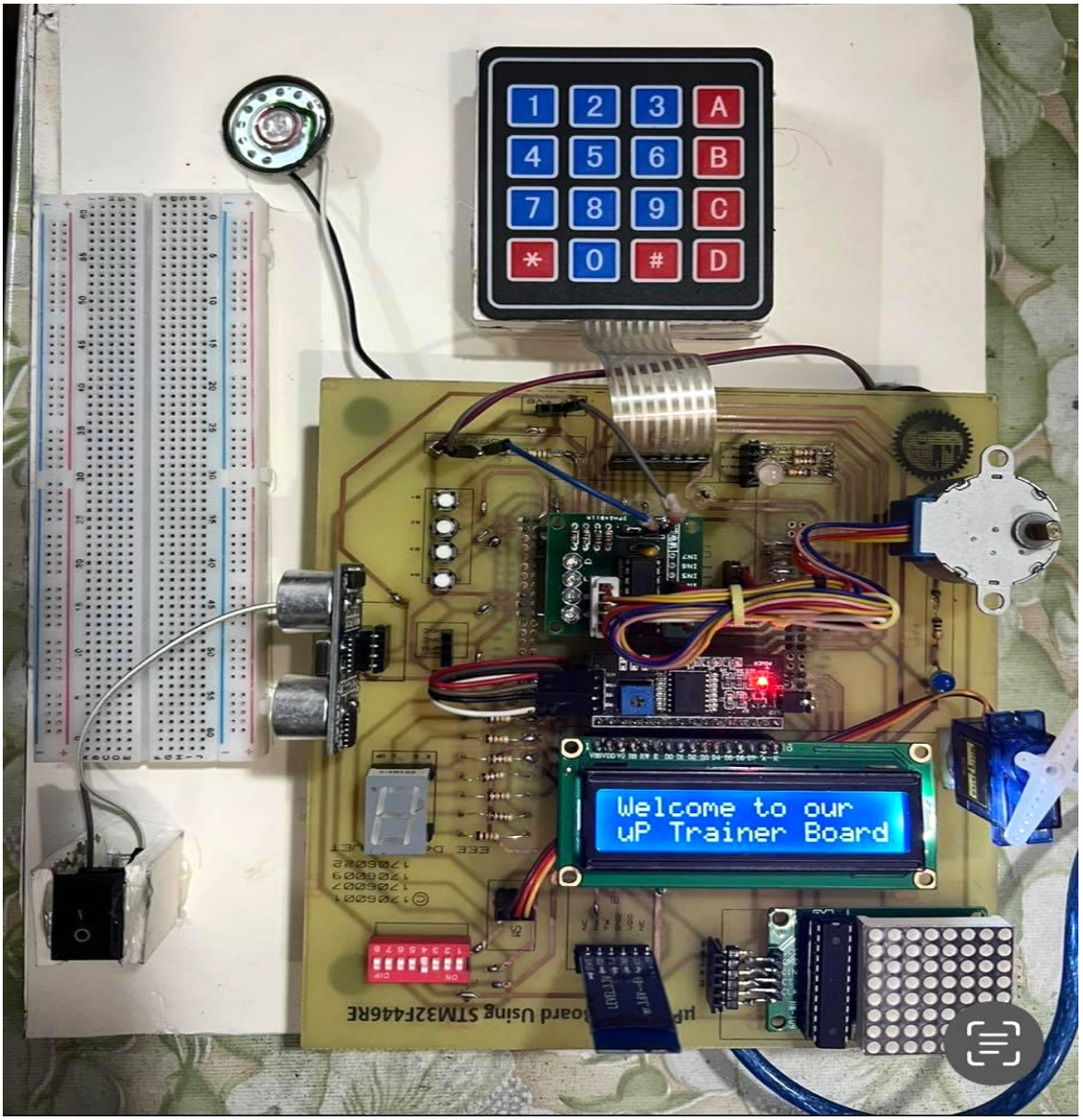
Top view



Bottom view

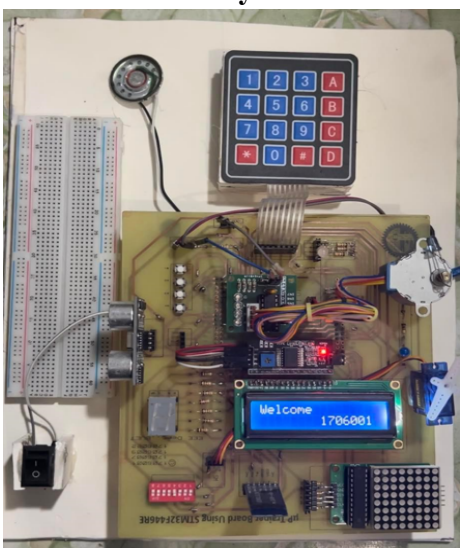


4.1.4 Actual Implementation

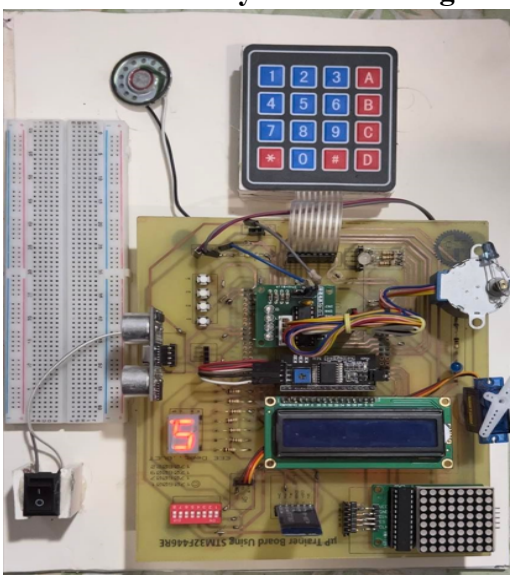


4.2 Result

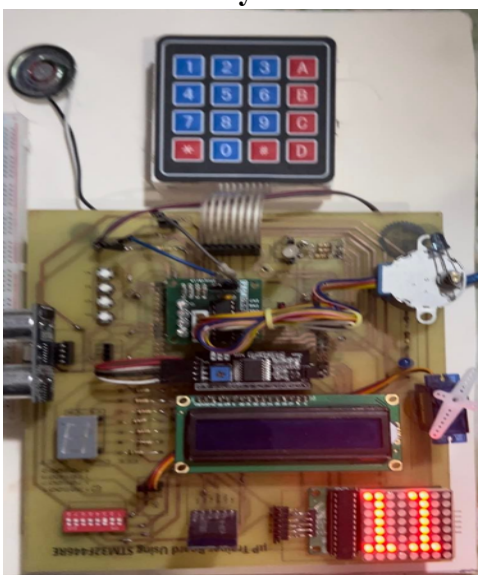
4.2.1 Functionality check of LCD with Keypad



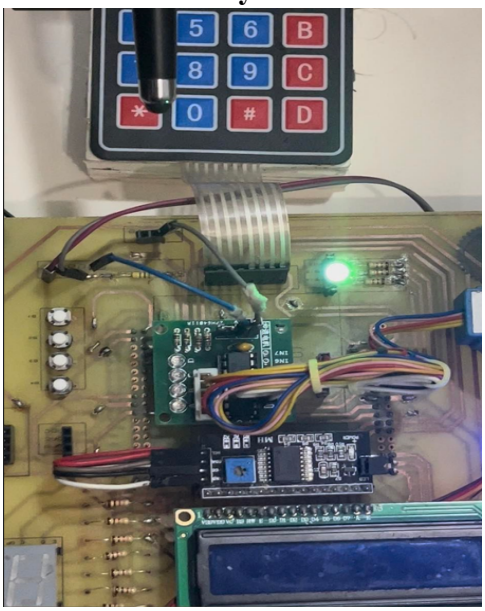
4.2.2 Functionality check of 7 Segment Display



4.2.3 Functionality check of 8x8 LED Matrix



4.2.4 Functionality check of RGB LED



4.2.5 Functionality check of Speaker

Demonstrated in the youtube video.

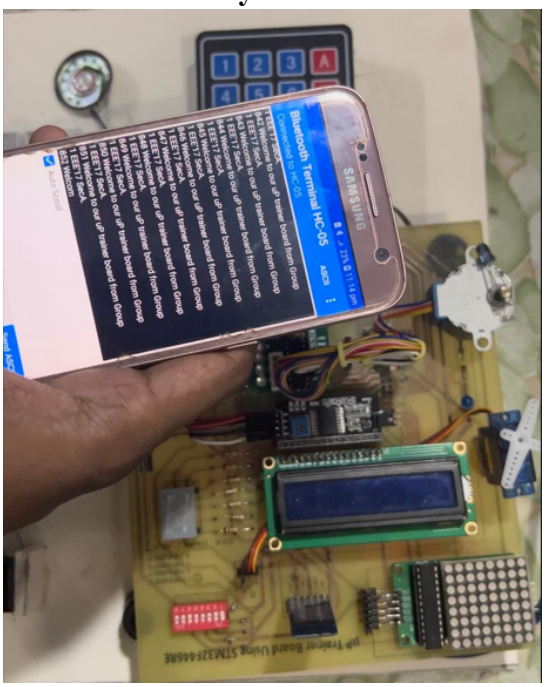
4.2.6 Functionality check of Stepper Motor with Push button

Demonstrated in the youtube video.

4.2.7 Functionality check of Servo Motor

Demonstrated in the youtube video.

4.2.8 Functionality check of Bluetooth module



4.2.9 Functionality check of Sonar

Demonstrated in the youtube video.

4.3 Github Link

https://github.com/mrinmoykundu/uP-Trainer-with-STM32f446-microprocessor.git

4.4 YouTube Link

https://www.youtube.com/watch?v=9OJPnnVvqg4

5 Design Analysis and Evaluation

5.1 Novelty

In our project, we design an original circuit for the PCB from scratch, and incorporated various functionality incorporating STM32F466RE and peripherals which can first of its kind

- Removable modules on the board
- Provision of remote control of the trainer board
- Isolated power control using DIP switch
- External master reset pin

5.2 Project Management and Cost Analysis

5.2.1 Bill of Materials

- Nucleo stm32f446re

- Various Peripherals
    - Display Modules
        - LCD display
        - 7-segment display
        - LED matrix
    - Driver
        - LCD I2C driver
        - MAX7219 driver
        - ULN2003 driver
    - Sonar
    - Motors
        - Servo motor
        - Stepper motor
    - Speaker
    - Input Modules
        - Push button
        - DIP switch
        - 3-pin switch
        - Keypad
    - Light/Indicator
        - Led
        - RGB light
    - Bluetooth Module
- Miscellaneous circuit elements
    - Resistors
    - Jumper wire (male-male, male-female)
    - Header pin (male-male, male-female)
    - L-shape male to male header
    - Breadboard
- Packaging
    - Cardboard
    - Glue Gun
    - Soldering gun

5.2.2 Calculation of Per Unit Cost of Prototype

Our final product with the assembled PCB sums up to a dimension of 15.5 cm x 17.3cm.

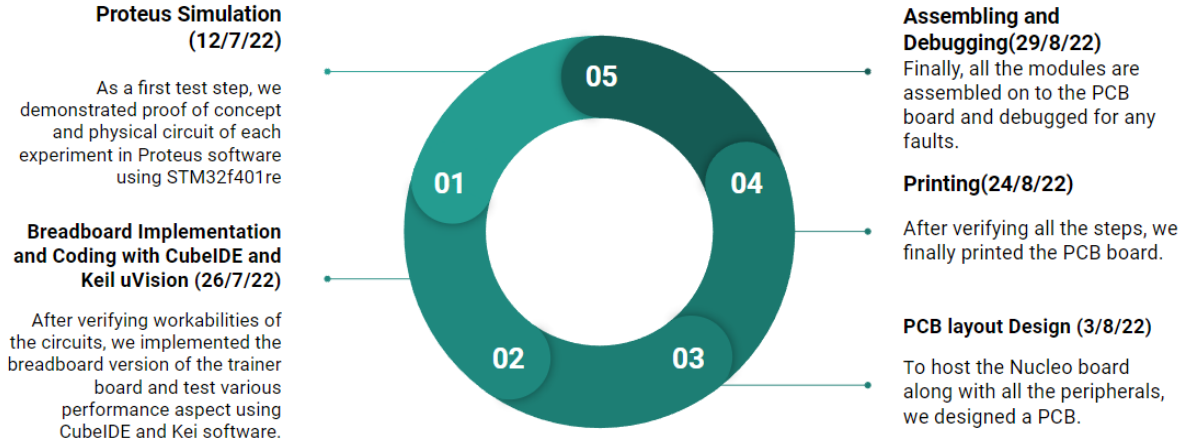
Component name	**Quantity**	**per unit price(tk)**	**Total Price(tk)**
8x8 LED matrix 7219	1	200	200
7 segment	2	10	20
8 ohm speaker	1	20	20
push buttons	4	3	12
Keypad	1	60	60

Female-Female wire	2 bunch	50	100
STM32F446 Nucleo Board	1	1	4000
PCB v1	1	728	728
PCB v2	1	800	800
DIP 8 pin switch	1	30	30
male rail	4 pieces	15	60
female rail	5 pieces	25	125
led wire	4	3	12
lifting knob	1	60	60
47 ohm resistor	10 pieces	1	10
1k ohm resistor	10 pieces	2	20
		Total	6257tk

5.2.3 Calculation of Per Unit Cost of Mass-Produced Unit

Roughly, excluding the contingency cost for reprinting the PCB, we can estimate that a mass-produced final product will cost 5000tk, where the major portion will be consumed by the STM32F446 Nucleo Board.

5.2.4 Timeline of Project Implementation



5.3 Practical Considerations of the Design to Address Public Health and Safety, Environment, Cultural, and Societal Needs

5.3.1 Considerations to public health and safety

We have no element in the design to cause public safety issue

5.3.2 Considerations to environment

We try to compact our design as far as possible to reduce plastic and carbon footprint

5.3.3 Considerations to cultural and societal needs

Our trainer board can be used as an experimental platform for the upcoming EEE 416 course, thus more focus can be put on the theory of ISA and embedded system instead of the hardware demonstration.

5.4 Assessment of the Impact of the Project on Societal, Health, Safety, Legal and Cultural Issues r

5.4.1 Assessment of Legal Issues

We have no legal conflict regarding intellectual property

5.5 Evaluation of the Sustainability the and Impact of the Designed Solution in the Societal and Environmental Contexts

5.5.1 Evaluation of Sustainability

- Nucleo stm32f446re board is a pretty solid board which can even sustain under unwanted short-circuit conditions.
- All the components on the pcb board were mounted through female/male headers.So, the components are easily removable and can be easily changed/swapped/removed easily if needed.

6 Reflection on Individual and Team work

6.1 Individual Contribution of Each Member

We didn't find a solid line to differentiate individual contributions to the completion of the project. We all struggled on the same platform whenever we faced a problem. From the background study to the final Assembly of the whole trainer board, we all participated equally.

6.2 Mode of TeamWork

Boardly, we can divide the works which were put into completion of the project by the following serial

- Background study
- Hardware prototype testing
- PCB Designing
- Printing, Soldering and Circuit testing
- Purchasing of various components
- Integration of each modules
- Coding
- Debugging and troubleshooting

- Final Demonstration

As a team, we had our ups and down throughout the project. At some point, we need to replace the whole design of the pcb, only one week before submission. But, in the end, we came out successful. We cannot stress much that our efficient organization, support and good understanding of each other played a vital role in our overall team work.

6.3 Diversity Statement of Team

Though we were from different major groups, we corporated efficiently to handle our academic works and projects. Also, different regional and college backgrounds also help us put diverse experiences. Different skill sets of each member in the field of theory, hardware, logistics and marketing also helped us to integrate into a dynamic team.

6.4 Log Book of Project Implementation

Date	Milestone achieved	Individual Role	Team Role	Comments
12.7.22	Simulation of STM32F401 chip using Proteus	Equal	United as a group	The total number of tasks were divided and distributed to all the group members.
26.7.22	Breadboard implementation of each segments			
3.8.22	Primary PCB Design			
5.8.22	Second Revision of PCB Design			
13.8.22	1st print of the PCB			
24.8.22	Revision of the design and 2nd print of PCB			
26.8.22	Soldering			
28.8.22	Revision of Soldering			
29.8.22	Integration of each modules			
30.8.22	Final Testing and Report			

7 References

1. Details intuition of STM32F446 Microcontroller
2. Youtube videos containing basic theoretical knowledge of STM32F446 Microcontroller
3. Details view & intuition of GPIO pins
4. Proteus Tutorial : Getting Started with Proteus PCB Design (Version 8.6)