NORTH SOUTH UNIVERSITY

Department of Electrical and Computer Engineering

LAB TASK 5-8 USING MICROCONTROLLER (STM32F103C8T6)

Amiur Rahman 2131466642

Instructor:Syed Mahmud Husain (smh2)
CSE 331.Section 4

1. Introduction

This project demonstrates a complete master-slave embedded communication system using two STM32F103C8T6 microcontrollers. The master board performs environmental monitoring using sensors (DHT11 and Ultrasonic), displays the data on an OLED, and sends a UART signal to a slave board. The slave board, upon receiving the UART signal, blinks an LED. Additionally, it independently uses an IR sensor to detect objects and rotates a servo motor by 90 degrees based on detection.

Hardware and tools used:

- STM32F103C8T6 (x2)
- DHT11 Temperature and Humidity Sensor
- Ultrasonic Sensor (HC-SR04)
- SSD1306 OLED Display (I2C)
- Servo Motor
- IR Sensor
- STM32CubeIDE + HAL Library

2. System Architecture

2.1 Overview

- **Master Board**: Handles sensing (DHT11, Ultrasonic), display (OLED), and UART transmission
- **Slave Board**: Receives UART signal to blink LED; also handles IR-triggered servo motor control independently

2.2 Communication

- **UART**: Unidirectional (Master TX -> Slave RX)
- **I2C**: OLED communication on Master

- **GPIO & TIM**: IR and Servo handled via GPIO input and PWM output

(Insert architecture diagram placeholder here)

3. Scenario-Based Functional Breakdown

Task5: Control the Door Entrance with an Automated System

.ioc Configuration (Slave):

- **GPIO**:
    - IR Sensor Input: GPIOA Pin 2
- **TIM2**:
    - PWM Output Channel: TIM2_CH2 (e.g., PA1)

Code Implementation:

```
// Check if the IR sensor detects an object (active-low output)
if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_2) == GPIO_PIN_RESET)
{
   set_servo_angle(90);  // Rotate servo to 90° to open the door
}
else
{
   set_servo_angle(0);   // Return servo to 0° to close the door
}

HAL_Delay(100); // Debounce delay to prevent rapid triggering

// Function to set servo angle by adjusting the PWM pulse width
void set_servo_angle(uint8_t angle)
{
  // Map angle (0°-180°) to PWM pulse width (50 to 250 ticks)
  uint16_t pulse = ((angle * (250 - 50)) / 180) + 50;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, pulse); // Set PWM duty cycle
}
```

This loop checks whether the IR sensor detects an object by checking the GPIO pin (PA2). If detection is triggered, the servo rotates to 90° using PWM on TIM2_CH2 (PA1). The set_servo_angle function maps the desired angle to the corresponding PWM duty cycle, calculated to produce the required pulse width (typically 1ms to 2ms for 0° to 180°). A short delay is added to avoid bouncing from IR fluctuations.

Task 6: Display Distance of Object in OLED Display

.ioc Configuration (Master):

- **I2C1**:
    - SDA: PB7
    - SCL: PB6
- **GPIO**:
    - Ultrasonic Trigger: PA11
    - Ultrasonic Echo: PA12

Code Implementation Highlights:

```
// Set cursor to row 3 of OLED
SSD1306_GotoXY(0, 30);
// Print label
SSD1306_Puts("Distance: ", &Font_7x10, 1);
// Format distance value as string
char distStr[10];
sprintf(distStr, "%dcm", distance);
// Display the distance string
SSD1306_Puts(distStr, &Font_7x10, 1);
// Refresh OLED to show the updated text
SSD1306_UpdateScreen();
```

This code snippet handles the visual display of the ultrasonic sensor's output. The distance value, retrieved and formatted as a string, is positioned on the third line of the OLED screen using the SSD1306 library. The SSD1306_UpdateScreen() function ensures the buffer is flushed to the actual screen.

Task 7: Display Temperature and Humidity in OLED Display

.ioc Configuration (Master):

- **I2C1**:
    - SDA: PB7
    - SCL: PB6
- **GPIO**:
    - DHT11 Data: PB5

Code Implementation Highlights:

```
// Set OLED cursor to top-left corner
SSD1306_GotoXY(0, 0);
// Print temperature label
SSD1306_Puts("Temp: ", &Font_7x10, 1);
char tempStr[10];
sprintf(tempStr, "%dC", temperature);
// Display temperature value
SSD1306_Puts(tempStr, &Font_7x10, 1);

// Move to next line and display humidity
SSD1306_GotoXY(0, 15);
SSD1306_Puts("Humidity: ", &Font_7x10, 1);
char humStr[10];
sprintf(humStr, "%d%%", humidity);
SSD1306_Puts(humStr, &Font_7x10, 1);

// Refresh the OLED screen
SSD1306_UpdateScreen();
```

This portion of the code shows how data from the DHT11 sensor is formatted and sent to the OLED display via I2C. It first places temperature data on the top line and then writes humidity information below it. The OLED is refreshed after writing to ensure the values are rendered correctly.

Task 8: Interfacing One Board with Another for LED Blinking

.ioc Configuration (Master & Slave):

- **Master USART2 TX**: PA2
- **Slave USART2 RX**: PA3
- **Slave GPIO**: PA5 (for external LED)

Master Code:

```
// Prepare UART data to send to slave
uint8_t txData = '1';
// Transmit the byte over USART2 with blocking mode
HAL_UART_Transmit(&huart2, &txData, 1, HAL_MAX_DELAY);
// Wait 2 seconds before next transmission
HAL_Delay(2000);
```

Slave Code:

```
// Create a buffer to store received byte
uint8_t rxData;
// Receive one byte with a timeout of 100ms
if (HAL_UART_Receive(&huart2, &rxData, 1, 100) == HAL_OK) {
   // If received byte is '1', trigger LED blink
   if (rxData == '1') {
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET); // Turn LED on
      HAL_Delay(200);                                     // Wait for 200 ms
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET); // Turn LED off
   }
}
```

This implementation shows how UART communication facilitates inter-board signaling. The master sends a predefined byte ('1') every 2 seconds. The slave listens using HAL_UART_Receive() and, upon receiving the correct signal, activates a GPIO pin (PA5) to blink an external LED for a brief duration. This confirms successful communication and response execution.

4. Challenges & Considerations

- UART timing and ensuring correct baud rate across both boards\

- Power distribution for Servo Motor
- Delays and blocking functions in main loop affecting real-time behavior
- Ensuring sensor reads do not interfere with UART transmission or servo motion
- Accurate PWM tuning for servo control

5. Conclusion

The project successfully implements a master-slave communication model where the master handles sensor data collection and transmission, while the slave responds with visual feedback and independent control logic. This architecture demonstrates modular embedded development with real-time data flow, peripheral integration, and reliable UART communication between STM32 microcontrollers.

6.FULL CODE:

main.c //MASTER BOARD

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

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

******************************************************************************
 * @attention
 *
 * Copyright (c) 2025 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 "fonts.h"
#include "ssd1306.h"
#include "stdio.h"
#include "string.h"
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
```

```c
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim1;
UART_HandleTypeDef huart2;
/* USER CODE BEGIN PV */
#define TRIG_PIN GPIO_PIN_9
#define TRIG_PORT GPIOA
#define ECHO_PIN GPIO_PIN_8
#define ECHO_PORT GPIOA
uint32_t pMillis;
uint32_t Value1 = 0;
uint32_t Value2 = 0;
uint16_t Distance  = 0;  // cm
char strCopy[32];
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM1_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
#define DHT11_PORT GPIOB
#define DHT11_PIN GPIO_PIN_9
uint8_t RHI, RHD, TCI, TCD, SUM;
uint32_t pMillis, cMillis;
float tCelsius = 0;
float tFahrenheit = 0;
float RH = 0;
uint8_t TFI = 0;
uint8_t TFD = 0;
char strCopy[32];
```

```
void microDelay (uint16_t delay)
{
 __HAL_TIM_SET_COUNTER(&htim1, 0);
 while (__HAL_TIM_GET_COUNTER(&htim1) < delay);
}
uint8_t DHT11_Start (void)
{
 uint8_t Response = 0;
 GPIO_InitTypeDef GPIO_InitStructPrivate = {0};
 GPIO_InitStructPrivate.Pin = DHT11_PIN;
 GPIO_InitStructPrivate.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStructPrivate.Speed = GPIO_SPEED_FREQ_LOW;
 GPIO_InitStructPrivate.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as output
 HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 0);   // pull the pin low
 HAL_Delay(20);   // wait for 20ms
 HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 1);   // pull the pin high
 microDelay (30);   // wait for 30us
 GPIO_InitStructPrivate.Mode = GPIO_MODE_INPUT;
 GPIO_InitStructPrivate.Pull = GPIO_PULLUP;
 HAL_GPIO_Init(DHT11_PORT, &GPIO_InitStructPrivate); // set the pin as input
 microDelay (40);
 if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
 {
  microDelay (80);
  if ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN))) Response = 1;
 }
 pMillis = HAL_GetTick();
 cMillis = HAL_GetTick();
 while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
 {
  cMillis = HAL_GetTick();
 }
 return Response;
}
uint8_t DHT11_Read (void)
{
 uint8_t a,b;
 for (a=0;a<8;a++)
 {
```

```
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  {  // wait for the pin to go high
    cMillis = HAL_GetTick();
  }
  microDelay (40);   // wait for 40 us
  if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))   // if the pin is low
    b&= ~(1<<(7-a));
  else
    b|= (1<<(7-a));
  pMillis = HAL_GetTick();
  cMillis = HAL_GetTick();
  while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)) && pMillis + 2 > cMillis)
  {  // wait for the pin to go low
    cMillis = HAL_GetTick();
  }
 }
 return b;
}
/* 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_I2C1_Init();
MX_TIM1_Init();
MX_USART2_UART_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start(&htim1);
HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET);  // pull the TRIG pin
low
SSD1306_Init();
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */
          // Trigger ultrasonic sensor
              HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_SET);
              __HAL_TIM_SET_COUNTER(&htim1, 0);
              while (__HAL_TIM_GET_COUNTER(&htim1) < 10);
              HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET);
              pMillis = HAL_GetTick();
              while (!(HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN)) && pMillis + 10 >
HAL_GetTick());
              Value1 = __HAL_TIM_GET_COUNTER(&htim1);
              pMillis = HAL_GetTick();
              while ((HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN)) && pMillis + 50 >
HAL_GetTick());
              Value2 = __HAL_TIM_GET_COUNTER(&htim1);
              Distance = (Value2 - Value1) * 0.034 / 2;
              int distanceInt = Distance;
              int distanceDec = (Distance - distanceInt) * 10;
              // Read DHT11
              if (DHT11_Start())
              {
                 RHI = DHT11_Read();
                 RHD = DHT11_Read();
                 TCI = DHT11_Read();
                 TCD = DHT11_Read();
                 SUM = DHT11_Read();
              }
              if (RHI + RHD + TCI + TCD == SUM)
```

```
            {
              tCelsius = (float)TCI + (float)(TCD/10.0);
              tFahrenheit = tCelsius * 9/5 + 32;
              RH = (float)RHI + (float)(RHD/10.0);
              // --- Trigger ultrasonic sensor ---
              HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_SET);
              __HAL_TIM_SET_COUNTER(&htim1, 0);
              while (__HAL_TIM_GET_COUNTER(&htim1) < 10);
              HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET);
              // --- Capture echo time ---
              pMillis = HAL_GetTick();
              while (!(HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN)) && pMillis + 10 >
HAL_GetTick());
              Value1 = __HAL_TIM_GET_COUNTER(&htim1);
              pMillis = HAL_GetTick();
              while ((HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN)) && pMillis + 50 >
HAL_GetTick());
              Value2 = __HAL_TIM_GET_COUNTER(&htim1);
              Distance = (Value2 - Value1) * 0.034 / 2;
              // --- Display on OLED ---
              SSD1306_Clear();
              // Line 1: Temperature (°C)
              sprintf(strCopy, "T: %d.%d C   ", TCI, TCD);
              SSD1306_GotoXY(0, 0);
              SSD1306_Puts(strCopy, &Font_11x18, 1);
              // Line 2: Humidity (%)
              sprintf(strCopy, "H: %d.%d %%  ", RHI, RHD);
              SSD1306_GotoXY(0, 20);
              SSD1306_Puts(strCopy, &Font_11x18, 1);
              // Line 3: Distance (cm)
              int distanceInt = Distance;
              int distanceDec = (Distance - distanceInt) * 10;
              sprintf(strCopy, "D: %d.%d cm", distanceInt, distanceDec);
              SSD1306_GotoXY(0, 40);
              SSD1306_Puts(strCopy, &Font_11x18, 1);
              SSD1306_UpdateScreen();
            }
             // Send every 1 second
            uint8_t txData = '1';
                  HAL_UART_Transmit(&huart2, &txData, 1, HAL_MAX_DELAY);
```

```
                    HAL_Delay(2000);
            }
  /* 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};
 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
 RCC_OscInitStruct.HSEState = RCC_HSE_ON;
 RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
 RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
 RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
 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();
 }
}
```

```c
/**
 * @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 = 400000;
 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 */
}
/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{
 /* USER CODE BEGIN TIM1_Init 0 */
 /* USER CODE END TIM1_Init 0 */
 TIM_ClockConfigTypeDef sClockSourceConfig = {0};
 TIM_MasterConfigTypeDef sMasterConfig = {0};
 /* USER CODE BEGIN TIM1_Init 1 */
 /* USER CODE END TIM1_Init 1 */
```

```
htim1.Instance = TIM1;
htim1.Init.Prescaler = 71;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 65535;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
  Error_Handler();
}
/* USER CODE BEGIN TIM1_Init 2 */
/* USER CODE END TIM1_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 = 9600;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
```

```
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX;
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};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOD_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_RESET);
/*Configure GPIO pin : PA8 */
GPIO_InitStruct.Pin = GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
/*Configure GPIO pin : PA9 */
GPIO_InitStruct.Pin = GPIO_PIN_9;
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 pin : PB9 */
GPIO_InitStruct.Pin = GPIO_PIN_9;
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);
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* 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 */
```

main.c //SLAVE BOARD

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2025 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 -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim2;
```

```c
UART_HandleTypeDef huart1;
/* USER CODE BEGIN PV */
uint8_t rxData;
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */
void set_servo_angle(uint8_t angle);
/* 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_USART1_UART_Init();
 MX_TIM2_Init();
 /* USER CODE BEGIN 2 */
 HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
 /* USER CODE END 2 */
 /* Infinite loop */
```

```c
/* USER CODE BEGIN WHILE */
while (1)
{
    // === IR Sensor Logic on PA2 ===
    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_2) == GPIO_PIN_RESET)
    {
        set_servo_angle(90);  // Rotate to 90° when object is detected
    }
    else
    {
        set_servo_angle(0);   // Return to 0° when no object
    }
    // === UART Receive Logic ===
    if (HAL_UART_Receive(&huart1, &rxData, 1, 100) == HAL_OK)
    {
        if (rxData >= '0' && rxData <= '9')  // Only allow valid digit input
        {
            HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
            if (rxData == '1')
            {
                HAL_Delay(200);
            }
        }
    }
    // === Debounce/Loop Delay ===
    HAL_Delay(100);  // Small delay to prevent over-polling
}
/* USER CODE END 3 */
}
/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 /** 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_NONE;
 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_HSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != 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 = 79;
 htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim2.Init.Period = 1999;
```

```
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
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_2) !=
HAL_OK)
{
  Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */
/* USER CODE END TIM2_Init 2 */
HAL_TIM_MspPostInit(&htim2);
}
/**
 * @brief USART1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART1_UART_Init(void)
```

```c
{
 /* USER CODE BEGIN USART1_Init 0 */
 /* USER CODE END USART1_Init 0 */
 /* USER CODE BEGIN USART1_Init 1 */
 /* USER CODE END USART1_Init 1 */
 huart1.Instance = USART1;
 huart1.Init.BaudRate = 9600;
 huart1.Init.WordLength = UART_WORDLENGTH_8B;
 huart1.Init.StopBits = UART_STOPBITS_1;
 huart1.Init.Parity = UART_PARITY_NONE;
 huart1.Init.Mode = UART_MODE_RX;
 huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
 huart1.Init.OverSampling = UART_OVERSAMPLING_16;
 if (HAL_UART_Init(&huart1) != HAL_OK)
 {
   Error_Handler();
 }
 /* USER CODE BEGIN USART1_Init 2 */
 /* USER CODE END USART1_Init 2 */
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
 GPIO_InitTypeDef GPIO_InitStruct = {0};
 /* USER CODE BEGIN MX_GPIO_Init_1 */
 /* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOA_CLK_ENABLE();
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);
 /*Configure GPIO pin : PA2 */
 GPIO_InitStruct.Pin = GPIO_PIN_2;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_PULLUP;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
 /*Configure GPIO pins : PA4 PA5 */
```

```
 GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5;
 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);
 /* USER CODE BEGIN MX_GPIO_Init_2 */
 /* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
void set_servo_angle(uint8_t angle)
{
 // 1ms to 2ms pulse = 100 to 200 ticks at 10 µs resolution
 uint16_t pulse = ((angle * (250 - 50)) / 180) + 50;
 __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, pulse);
}
/* 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 */
```