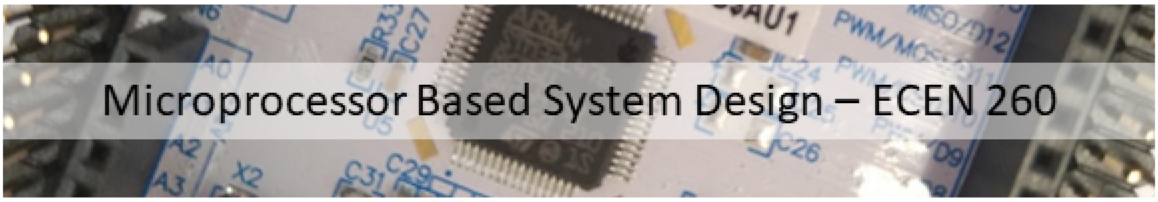

ECEN 260 - Final Project

Range Finder

Kelton Webb

Instructor: Brother Allred

July 18, 2023

Contents

List of Tables

List of Figures

1 Lab Overview

This project allows us to dive into the realm of ultrasonic sensors, understand how they function, and use them for something practical - measuring distances. Our aim is to turn invisible sound waves into visible numbers on an LCD screen. The numbers might seem simple, but behind them is a complex system of signals and calculations happening at the speed of sound.

1.1 Objectives

The objectives of this lab are:

- Understand the principle of ultrasonic sensing
- Interface with the HCSR04 Ultrasonic sensor with the STM32 Microcontroller
- Capture and process signals
- Display data on an LCD

1.2 Specifications

This project focuses on utilizing the HCSR04 ultrasonic sensor to measure distance, which are then displayed on a LCD screen in cm. It is interfaced with an STM32 microcontroller mounted on a Nucleo-L476RG development board. This setup requires knowledge of the following concepts:

- Interrupts
- Timers
- Display with I2C
- Digital communication with I/O

1.3 Use Cases

The primary use case for this project is to serve as a foundation for various applications that require proximity or distance measurements. Examples include, but are not limited to, obstacle detection or motion detection systems, level measurement, parking sensors, or as part of a larger robotics project.

1.4 Operating Instructions

- Connect the HCSR04 ultrasonic sensor to the STM32 microcontroller. Make sure VCC is connected to 5V and GND is connected. Have Echo and Trigger connected with the appropriate GPIO pins. The Trig pin should be GPIO Output and the Echo pin should be set as input capture direct mode.

- Connect the LCD display to the board using the on-board SCL/SDA pins.
- Upload the provided script onto the board using an IDE such as STM32CubeIDE.
- Apply power to the board. The sensor will start emitting ultrasound waves and receiving the echo.
- The microcontroller calculates the distance based on the time it takes for the echo to return to the sensor, and displays it on the LCD screen.

1.5 Operating Constraints

Here are the constraints to this setup:

- The HCSR04 sensor is limited to a range of 2cm-400cm under normal conditions.
- The sensor's accuracy can be affected by factors such as temperature, humidity, and the reflectivity of the target object.

1.6 Parts List

- HCSR04 ultrasonic sensor
- Nucleo-L476RG development board with STM32 microcontroller
- LCD Display
- Connecting wires
- Breadboard
- 5V Power supply (optional)

2 Schematics

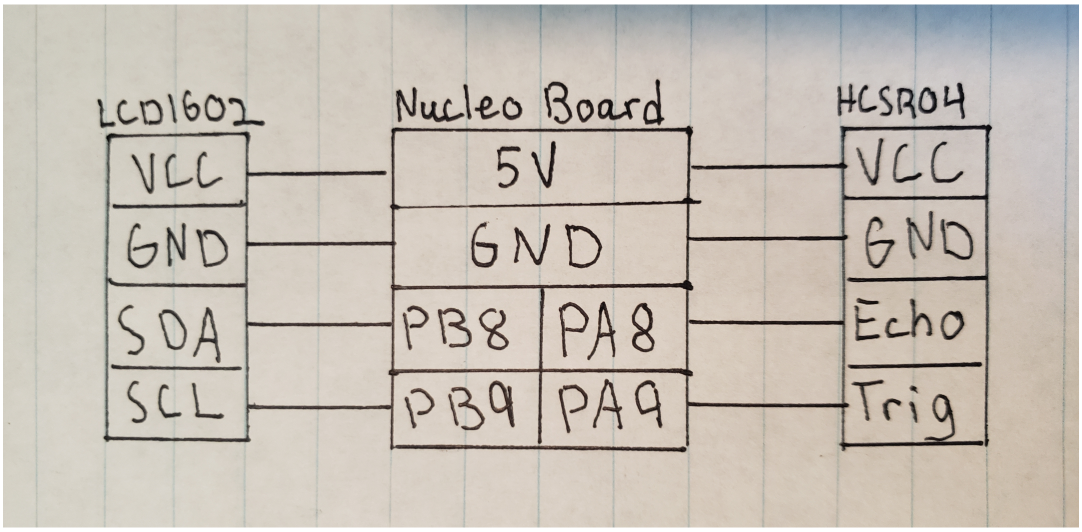

Figure 1: Schematic diagram for the Lab.

3 Test Plan and Test Results

In this test plan we will do: Power-On Test, Static Object Distance Measurement, Dynamic Object Distance Measurement,Max Distance Measurement, Sensor Response to Non-reflective Objects

3.1 Test Plan Procedure

- Test Scenario #1
  - Step 1: Connect the setup to power supply
  - Step 2: Turn on the power
- Test Scenario #2
  - Step 1: Place a static object at a known distance (say 10cm) in front of the sensor.
  - Observe the LCD display
- Test Scenario #3
  - Step 1: Start moving an object towards the sensor from a distance of 50cm at a steady pace.
  - Step 2: Observe the LCD display as the object moves closer.
- Test Scenario #4
  - Step 1: Place an object at a distance just within the maximum range of the sensor (say, 400cm).
  - Step 2: Observe the LCD display.
- Test Scenario #5
  - Step 1: Place a non-reflective object like a fluffy blanket in front of the sensor.
  - Step 2: Observe the LCD display

3.2 Expected and Observed Results

This section should include the the expected and actual results of each test.

- Test Scenario #1
  - Expected Result: The LCD screen should light up and display the distance message with a possible varying distance.
  - Actual Result: The LCD screen lit up and displayed the initial message with a varying distance.
- Test Scenario #2

- Expected Result: The LCD should display the distance close to the actual distance (10 cm), taking into consideration the +- 1cm accuracy of the sensor.
- Actual Result: When a static object was placed at a distance of 10cm, the LCD displayed a distance of 10 cm, verifying the system's accurate distance measurement.

- Test Scenario #3
  - Expected Result: The LCD display should update the distance as the object moves closer to the sensor. The readings should decrease.
  - Actual Result: As the object moved closer to the sensor from a distance of 50cm, the LCD display updated the distance. The readings decreased from 50cm to 2 cm as the object moved as close as it could.
- Test Scenario #4
  - Expected Result: The sensor should be able to detect the object and the LCD should display the distance close to the actual distance. If the object is placed beyond the sensor's range, the LCD should display the max distance of 400cm.
  - Actual Result: The sensor was able to detect an object at a max of 202 cm. When it was placed past 202 cm, the display stayed at 202 cm.
- Test Scenario #5
  - Expected Result: The sensor should show a larger margin of error when measuring the distance of non-reflective objects due to the object absorbing the sound waves.
  - Actual Result: When a stuffed animal was placed in front of the sensor, it seemed to have absorbed all of the sound waves, because the display was showing 202cm, which is the max distance of the sensor.

4 Code

The following section contains the main.c code for the project. I built it around the STM32Cube HAL library, which provides high level interacting with the microcontroller. I also implemented a I2C library located under User Code begin 0. The logic for the sensor is also included under User Code 0, and I then implement it with the display in the main loop.

4.1 Code for main.c

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 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 ------------------------------------------------------------*/
```

```c
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
#define I2C_ADDR 0x27 // I2C address of the PCF8574
#define RS_BIT 0 // Register select bit
#define EN_BIT 2 // Enable bit
#define BL_BIT 3 // Backlight bit
#define D4_BIT 4 // Data 4 bit
#define D5_BIT 5 // Data 5 bit
#define D6_BIT 6 // Data 6 bit
#define D7_BIT 7 // Data 7 bit
#define LCD_ROWS 2 // Number of rows on the LCD
#define LCD_COLS 16 // Number of columns on the LCD

#define TRIG_PIN GPIO_PIN_9
#define TRIG_PORT GPIOA

/* USER CODE END PM */

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

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim16;

/* USER CODE BEGIN PV */
uint8_t backlight_state = 1;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM16_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

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


//////////////////////////////////////////////////////////
/*This is the delay for the trigger pulse*/
void delay (uint16_t time)
{
```

```c
  __HAL_TIM_SET_COUNTER(&htim1, 0);
  while (__HAL_TIM_GET_COUNTER (&htim1) < time);
}
/////////////////////////////////////////////////////////


///////////////////////////////////////////////////////////////
//This is the input capture callback function
uint32_t IC_Val1 = 0;
uint32_t IC_Val2 = 0;
uint32_t Difference = 0;
uint8_t Is_First_Captured = 0;
uint8_t Distance =0;

// This function gets called when the timer capture interrupt occurs
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    // if the interrupt source is channel1
  if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
  {
        // if the first value has not yet been captured
    if (Is_First_Captured==0)
    {
            // Read the first value from the timer's capture register
      IC_Val1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);

            // Set the flag to indicate that the first value has been captured
      Is_First_Captured = 1;

            // Now change the input channel's capture polarity to falling edge
      __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);
    }

        // If the first value has already been captured
    else if (Is_First_Captured==1)
    {
            // Read the second value from the timer's capture register
      IC_Val2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);

            // Reset the timer's counter
      __HAL_TIM_SET_COUNTER(htim, 0);

            // If the second value is greater than the first one
      if (IC_Val2 > IC_Val1)
      {
                // The difference is a simple subtraction
        Difference = IC_Val2-IC_Val1;
      }

            // If the first value is greater than the second one
      else if (IC_Val1 > IC_Val2)
      {
```

```
                // The difference is calculated by taking into account the timer's maximum value (0xffff)
          Difference = (0xffff - IC_Val1) + IC_Val2;
        }

            // Convert the timer difference into distance (in cm) using the speed of sound (0.034 cm/us / 2 for round-trip)
        Distance = Difference * .034/2;

            // Reset the flag so that the first value can be captured on the next interrupt
        Is_First_Captured = 0;

            // Set the input channel's capture polarity back to rising edge for the next cycle
        __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);

            // Disable the capture compare interrupt, so the interrupt doesn't trigger again until it's enabled elsewhere
        __HAL_TIM_DISABLE_IT(&htim1, TIM_IT_CC1);
      }
    }
}

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


//////////////////////////////////////////////////////////////////////
/* This is the function that creates a trig pulse, HIGH for 10 seconds
 * and then set back to LOW*/
void HCSR04_Read (void)
{
  HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_SET);  // pull the TRIG pin HIGH
  delay(10);  // wait for 10 us
  HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET);  // pull the TRIG pin low

  __HAL_TIM_ENABLE_IT(&htim1, TIM_IT_CC1);
}
/////////////////////////////////////////////////////////////////////////


///////////////////////////////////////////////////////////////////////////
/*These are the I2C LCD functions/library from class*/
void lcd_write_nibble(uint8_t nibble, uint8_t rs) {
uint8_t data = nibble << D4_BIT;
data |= rs << RS_BIT;
data |= backlight_state << BL_BIT; // Include backlight state in data
data |= 1 << EN_BIT;
HAL_I2C_Master_Transmit(&hi2c1, I2C_ADDR << 1, &data, 1, 100);
HAL_Delay(1);
data &= ~(1 << EN_BIT);
```

```
HAL_I2C_Master_Transmit(&hi2c1, I2C_ADDR << 1, &data, 1, 100);
}
void lcd_send_cmd(uint8_t cmd) {
uint8_t upper_nibble = cmd >> 4;
uint8_t lower_nibble = cmd & 0x0F;
lcd_write_nibble(upper_nibble, 0);
lcd_write_nibble(lower_nibble, 0);
if (cmd == 0x01 || cmd == 0x02) {
HAL_Delay(2);
}
}
void lcd_send_data(uint8_t data) {
uint8_t upper_nibble = data >> 4;
uint8_t lower_nibble = data & 0x0F;
lcd_write_nibble(upper_nibble, 1);
lcd_write_nibble(lower_nibble, 1);
}

void lcd_init() {
HAL_Delay(50);
lcd_write_nibble(0x03, 0);
HAL_Delay(5);
lcd_write_nibble(0x03, 0);
HAL_Delay(1);
lcd_write_nibble(0x03, 0);
HAL_Delay(1);
lcd_write_nibble(0x02, 0);
lcd_send_cmd(0x28);
lcd_send_cmd(0x0C);
lcd_send_cmd(0x06);
lcd_send_cmd(0x01);
HAL_Delay(2);
}
void lcd_write_string(char *str) {
while (*str) {
lcd_send_data(*str++);
}
}
void lcd_set_cursor(uint8_t row, uint8_t column) {
uint8_t address;
switch (row) {
case 0:
address = 0x00;
break;
case 1:
address = 0x40;
break;
default:
address = 0x00;
}
address += column;
lcd_send_cmd(0x80 | address);
}
void lcd_clear(void) {
```

```
lcd_send_cmd(0x01);
HAL_Delay(2);
}
void lcd_backlight(uint8_t state) {
if (state) {
backlight_state = 1;
} else {
backlight_state = 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_TIM1_Init();
  MX_I2C1_Init();
  MX_TIM16_Init();
  /* USER CODE BEGIN 2 */

  HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_1);


  // I2C pull-up resistors
  GPIOB->PUPDR |= 0b01 << (8*2);
  GPIOB->PUPDR |= 0b01 << (9*2);
```

```
  // Initialize the LCD
  lcd_init();
  lcd_backlight(1); // Turn on backlight
  // Write a string to the LCD
  lcd_write_string("Dist= ");


  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
    HCSR04_Read();
    char str_distance[4]; // buffer for the distance string. Increase the size
     if you have bigger numbers
    sprintf(str_distance, "%d", Distance); // convert the Distance to a string
    lcd_clear(); // clear the LCD before writing
    lcd_write_string("Dist= ");
    lcd_write_string(str_distance); // write the distance string
    lcd_write_string(" cm");
    HAL_Delay(200);

  }
  /* 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
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** 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 = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  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_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != 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 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x10909CEC;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }

```

```
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != 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_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 72;
  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_IC_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief TIM16 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM16_Init(void)
{

  /* USER CODE BEGIN TIM16_Init 0 */

  /* USER CODE END TIM16_Init 0 */

  /* USER CODE BEGIN TIM16_Init 1 */

  /* USER CODE END TIM16_Init 1 */
  htim16.Instance = TIM16;
  htim16.Init.Prescaler = 0;
  htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim16.Init.Period = 65535;
  htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim16.Init.RepetitionCounter = 0;
  htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM16_Init 2 */

  /* USER CODE END TIM16_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
```

```c
  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_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(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);

  /*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);

/* 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 */
```

5 Conclusion

In this lab I successfully measured the data from a ultrasonic sensor onto a display. Starting with the basics, I explored how the sensor works. Basically, it sends out a sound wave that bounces off a nearby object. The sensor then measures the time it takes for the echo to return, and using the speed of sound, it calculates the distance to the object.

Reflecting on the lab, there were several challenges. The main one was successfully reading the data from the sensor. The logic for the IC Capture was the hardest part for me to understand. I understood what was suppose to happen, but it took a long time to realize how to code it out and debug it. I probably spent around 20 hours trying to read the data on the sensor. This challenge helped me learn a lot more about how ultrasonic sensors work but also how to deal with the ISR and how to work the timer. It gave me good knowledge of how sound speed, distance calculation, and interrupts can be applied. I am excited to take what I have learned and apply it to upcoming projects. Now that I have foundational skills to use the sensor there is all sorts of projects I could work on.