John Danison

ECET 32900 – Lab 10

04/18/2025

**Goal:**

The goal of this lab was to design programming logic and implement the program to interface with two different types of sensors over two different types of connections. The first was an accelerometer using SPI connection and the second was a temperature sensor using I2C.

**Activities:**

In this lab experiment, during each checkpoint I was follow lab instructions. I made sure that I meticulously read through the datasheets for each sensor respectively, thoroughly understanding the wiring schematic and the different communication types. SPI consisted of 4 wires, while the I2C only consisted of 2 wires to be able to communicate from master to slave. Upon completion of reading the data sheet, I transitioned to the book to read each respective SPI and I2C chapter for further understanding of the Nucleo 64 – L476RG board. I then completed a handwritten flowchart and wiring diagram before diving the STM32 Cube IDE project setup. I then carefully set up each project to the specific requirements and completed the code. I then troubleshooted any errors that occurred within this process.

**Electrical Schematic for ADXL 345 Accelerometer:**

A diagram of a measuring device

AI-generated content may be incorrect.

**Flowchart for ADXL 345 Accelerometer:**

A diagram of a data flow

AI-generated content may be incorrect.

**Source Code for ADXL 345 Accelerometer:**

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* USER CODE BEGIN Includes \*/

#include "string.h"

#include "stdio.h"

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

SPI\_HandleTypeDef hspi1;

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\_SPI1\_Init(void);

/\* USER CODE BEGIN 0 \*/

/\* SPI Write Function \*/

void adxl\_write (uint8\_t Reg, uint8\_t data)

{

    uint8\_t writeBuf[2];

    writeBuf[0] = Reg|0x40;  // multibyte write enabled

    writeBuf[1] = data;

    HAL\_GPIO\_WritePin (GPIOB, GPIO\_PIN\_6, GPIO\_PIN\_RESET); // pull the cs pin low to enable the slave

    HAL\_SPI\_Transmit (&hspi1, writeBuf, 2, 100);  // transmit the address and data

    HAL\_GPIO\_WritePin (GPIOB, GPIO\_PIN\_6, GPIO\_PIN\_SET); // pull the cs pin high to disable the slave

}

/\* SPI Read Function \*/

void adxl\_read (uint8\_t Reg, uint8\_t \*Buffer, size\_t len)

{

    Reg |= 0x80;  // read operation

    Reg |= 0x40;  // multibyte read

    HAL\_GPIO\_WritePin (GPIOB, GPIO\_PIN\_6, GPIO\_PIN\_RESET);  // pull the cs pin low to enable the slave

    HAL\_SPI\_Transmit (&hspi1, &Reg, 1, 100);  // send the address from where you want to read data

    HAL\_SPI\_Receive (&hspi1, Buffer, len, 100);  // read 6 BYTES of data

    HAL\_GPIO\_WritePin (GPIOB, GPIO\_PIN\_6, GPIO\_PIN\_SET);  // pull the cs pin high to disable the slave

}

/\* ADXL Initialization Function \*/

void adxl\_init (void)

{

    uint8\_t chipID=0;

    adxl\_read(0x00, &chipID, 1);

    if (chipID == 0xE5)

    {

        adxl\_write (0x2d, 0x00);  // reset all bits; standby

        adxl\_write (0x2d, 0x08);  // measure=1 and wake up 8hz

        adxl\_write (0x31, 0x01);  // 10bit data, range= +- 4g

    }

}

/\* Main Loop\*/

int main(void)

{

  /\* Initializations\*/

  HAL\_Init();

  SystemClock\_Config();

  MX\_GPIO\_Init();

  MX\_USART2\_UART\_Init();

  MX\_SPI1\_Init();

  adxl\_init();

  /\* Infinite loop \*/

  while (1)

  {

    /\* USER CODE END WHILE \*/

        /\* Read ADXL Data \*/

        uint8\_t RxData[6];

        adxl\_read (0x32, RxData, 6);

        /\* Seperate Data into x, y, z\*/

        int16\_t x = ((RxData[1]<<8)|RxData[0]);

        int16\_t y = ((RxData[3]<<8)|RxData[2]);

        int16\_t z = ((RxData[5]<<8)|RxData[4]);

        /\* Convert into gravitational values \*/

        float xg = (float)x/128;

        float yg = (float)y/128;

        float zg = (float)z/128;

        char buf[50];

        /\* Print to Termainal \*/

        sprintf(buf, "X: %+5.2f  Y: %+5.2f  Z: %+5.2f", xg, yg, zg);

        HAL\_UART\_Transmit(&huart2, (uint8\_t\*)buf, strlen(buf), HAL\_MAX\_DELAY);

        HAL\_UART\_Transmit(&huart2, (uint8\_t\*)"\r\n", 2, HAL\_MAX\_DELAY);

        /\* Delay 1 Second \*/

        HAL\_Delay(1000);

  }

}

/\*\*

  \* @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 SPI1 Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_SPI1\_Init(void)

{

  /\* USER CODE BEGIN SPI1\_Init 0 \*/

  /\* USER CODE END SPI1\_Init 0 \*/

  /\* USER CODE BEGIN SPI1\_Init 1 \*/

  /\* USER CODE END SPI1\_Init 1 \*/

  /\* SPI1 parameter configuration\*/

  hspi1.Instance = SPI1;

  hspi1.Init.Mode = SPI\_MODE\_MASTER;

  hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

  hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;

  hspi1.Init.CLKPolarity = SPI\_POLARITY\_HIGH;

  hspi1.Init.CLKPhase = SPI\_PHASE\_2EDGE;

  hspi1.Init.NSS = SPI\_NSS\_SOFT;

  hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_16;

  hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

  hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

  hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

  hspi1.Init.CRCPolynomial = 7;

  hspi1.Init.CRCLength = SPI\_CRC\_LENGTH\_DATASIZE;

  hspi1.Init.NSSPMode = SPI\_NSS\_PULSE\_DISABLE;

  if (HAL\_SPI\_Init(&hspi1) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\* USER CODE BEGIN SPI1\_Init 2 \*/

  /\* USER CODE END SPI1\_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;

  huart2.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

  huart2.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

  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\_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(GPIOB, GPIO\_PIN\_6, 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 : PB6 \*/

  GPIO\_InitStruct.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(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 \*/

**Sample for ADXL 345 Accelerometer:**

|  |  |  |
| --- | --- | --- |
| **Xg** | **Yg** | **Zg** |
| 0.03 | 0.00 | 0.98 |
| 0.03 | 0.00 | 0.98 |
| 0.03 | 0.00 | 0.98 |
| 0.03 | 0.00 | 0.98 |
| 0.25 | 0.66 | 0.60 |
| -0.14 | 0.20 | -0.95 |

This table shows a couple samples that were returned from the ADXL 345 Accelerometer. The Zg value is roughly the force due to gravity while Xg and Yg values are in terms of it moving around. For the first 4 data points, the sensor was stationary but then was lifted and turned upside down causing change in the Xg and Yg readings. Upon being upside down, the sensor read a negative Zg gravity reading, indicating that the sensor was properly reading.

**Electrical Schematic for TMP 102 Temperature Sensor**

A diagram of a circuit

AI-generated content may be incorrect.

**Flowchart for TMP 102 Temperature Sensor**

A diagram of a computer system

AI-generated content may be incorrect.

**Source Code for TMP 102 Temperature Sensor**

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private includes ----------------------------------------------------------\*/

#include "string.h"

#include "stdio.h"

/\* USER CODE BEGIN PTD \*/

uint8\_t I2C\_ADDRESS = 0x48 << 1;

uint8\_t Temp\_Reg = 0x00;

/\* UART Send New Line \*/

void UART\_SEND\_NL(UART\_HandleTypeDef \*huart)

{

    HAL\_UART\_Transmit(huart, (uint8\_t\*)"\n\r", 2, HAL\_MAX\_DELAY);

}

/\* UART Send Text \*/

void UART\_SEND\_TXT(UART\_HandleTypeDef \*huart, char buffer[], int m)

{

    HAL\_UART\_Transmit(huart, (uint8\_t\*) buffer, strlen(buffer), HAL\_MAX\_DELAY);

    if(m == 1) HAL\_UART\_Transmit(huart, (uint8\_t\*)"\n\r", 2, HAL\_MAX\_DELAY);

}

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

I2C\_HandleTypeDef hi2c1;

UART\_HandleTypeDef huart2;

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_I2C1\_Init(void);

static void MX\_USART2\_UART\_Init(void);

/\* Main Function \*/

int main(void)

{

    /\* Declare Variables \*/

    uint8\_t buf[7];

    int16\_t comb;

    float temperature, LSB = 0.0625;

    /\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

    HAL\_Init();

    /\* Configure the system clock \*/

    SystemClock\_Config();

    /\* Initialize all configured peripherals \*/

    MX\_GPIO\_Init();

    MX\_I2C1\_Init();

    MX\_USART2\_UART\_Init();

    HAL\_I2C\_Init(&hi2c1); // start I2C

    /\* Infinite loop \*/

    while (1)

    {

        /\* Connect to & Read TMP 102 Sensor \*/

        buf[0] = Temp\_Reg;

        HAL\_I2C\_Master\_Transmit(&hi2c1, I2C\_ADDRESS, buf, 1, HAL\_MAX\_DELAY);

        HAL\_I2C\_Master\_Receive(&hi2c1, I2C\_ADDRESS, buf, 2, HAL\_MAX\_DELAY);

        /\* Store Data \*/

        comb = ((int16\_t)buf[0] << 4) | (buf[1] >> 4);

        /\* Combine separate bytes \*/

        if (comb > 0x7FF ) { // If negative

            comb = (~comb) & 0xFFF;

            comb = comb + 1;

            temperature = -comb \* LSB;

        }

        else {

            temperature = comb \* LSB;

        }

        /\* Print temperature to the terminal \*/

        sprintf((char\*)buf, "%+5.2f", temperature);

        UART\_SEND\_TXT(&huart2, "Temperature = ", 0);

        HAL\_UART\_Transmit(&huart2, buf, strlen((char\*)buf), HAL\_MAX\_DELAY);

        /\* Send a New Line \*/

        UART\_SEND\_NL(&huart2);

        /\* Delay 1 Second \*/

        HAL\_Delay(1000);

    }

}

/\*\*

  \* @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\_MSI;

  RCC\_OscInitStruct.MSIState = RCC\_MSI\_ON;

  RCC\_OscInitStruct.MSICalibrationValue = 0;

  RCC\_OscInitStruct.MSIClockRange = RCC\_MSIRANGE\_6;

  RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

  RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_MSI;

  RCC\_OscInitStruct.PLL.PLLM = 1;

  RCC\_OscInitStruct.PLL.PLLN = 40;

  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 = 0x10D19CE4;

  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 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\_RX;

  huart2.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

  huart2.Init.OverSampling = UART\_OVERSAMPLING\_16;

  huart2.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

  huart2.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

  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)

{

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

  /\* GPIO Ports Clock Enable \*/

  \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

  \_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\* 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 \*/

**Sample Dasta for TMP 102 Temperature Sensor**

|  |  |
| --- | --- |
| **Reading** | **Unit** |
| 22.81 | °C |
| 22.88 | °C |
| 22.88 | °C |
| 22.88 | °C |
| 22.69 | °C |
| 22.56 | °C |

This table shows a couple samples that were returned from the TMP 102 Temperature Sensor. The returned 22.78°C average reading equates to roughly 73°F which makes sense for the abnormally warm lab room. In combination with a finger warming up the sensor, the results from the TMP 102 sensor were mainly accurate.

**Conclusion:**

During this lab activity I learned the difference between a SPI and I2C connection and the different coding protocols you must follow to properly set up and initialize the Nucleo-64 L476RG board. I also learned some important troubleshooting techniques as there was difficulty getting the SPI portion of the lab to work on my setup.   
  
I also learned how to use two new sensors, the ADXL 345 Accelerometer and the TMP 102 Temperature Sensor and now have the knowledge to use these in personal projects if needed in the future.

**Appendix:**

A diagram of a flowchart

AI-generated content may be incorrect.

References

Analog Devices. (2010). *ADXL345: Digital accelerometer data sheet*.

<https://www.analog.com/media/en/technical-documentation/data-sheets/adxl345.pdf>

Texas Instruments. (2015). *TMP102: Low-power digital temperature sensor with SMBus and*

*two-wire interface*. <https://www.ti.com/lit/ds/symlink/tmp102.pdf>

Ibrahim, D. (2020). *Nucleo boards programming with the STM32CubeIDE: Hands-on in more*

*than 50 projects*. Elektor International Media B.V.

Peers at Purdue University, School of Engineering Technology. (2025). *ECET 32900 Lab*

*collaboration*.

Purdue University. (2025). *ECET 32900 Lab 10 Instructional Documents*. Purdue University.