John Danison

ECET 32900 – Lab 10

04/18/2025

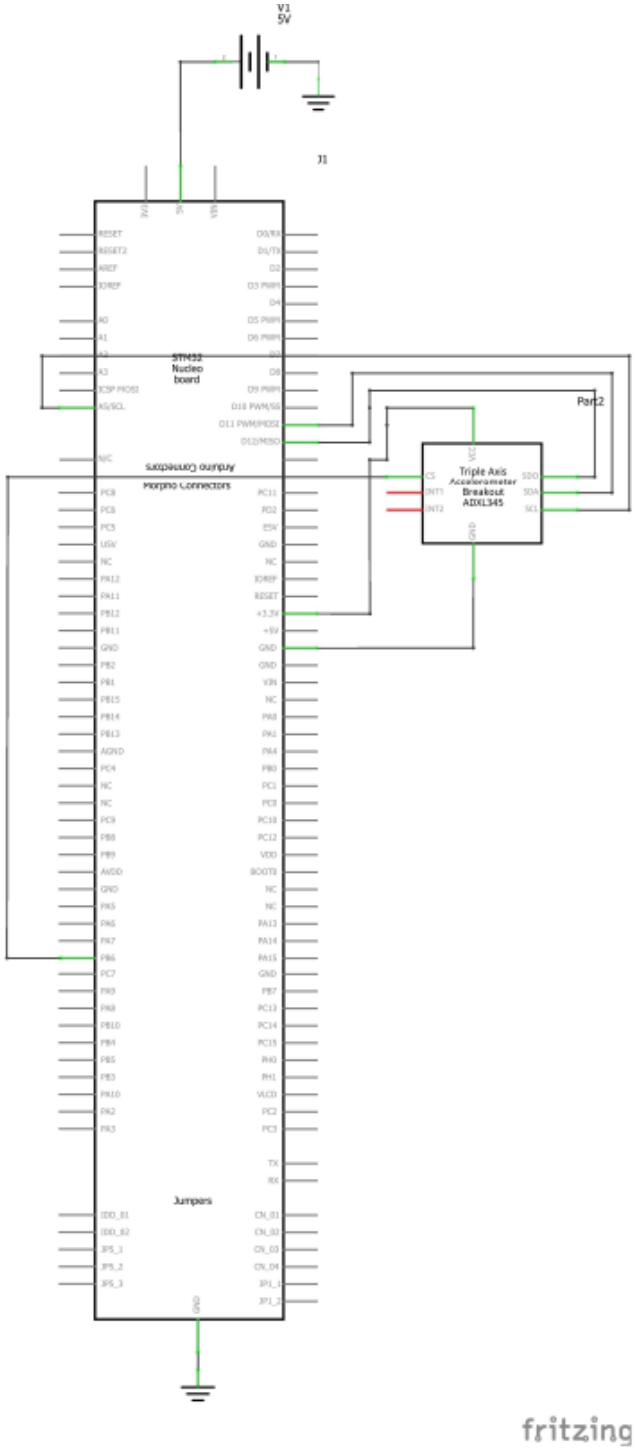
Goal:

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.

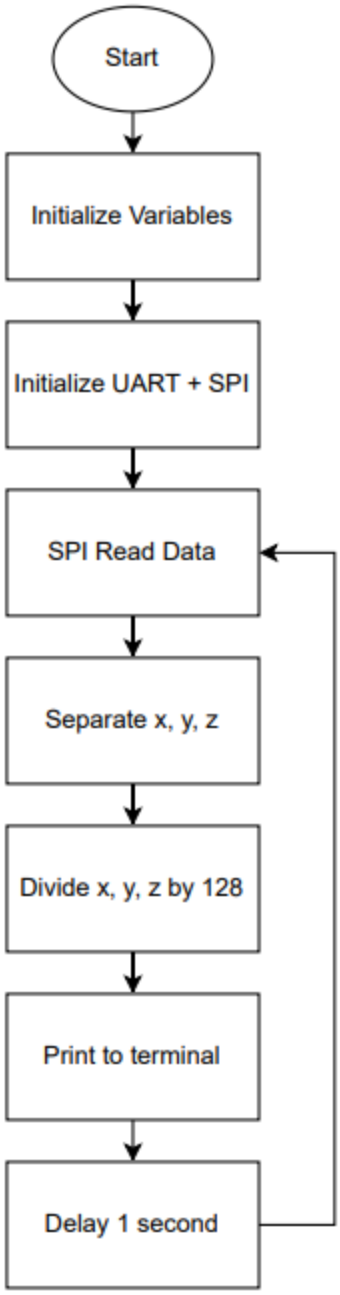
Activities:

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:



Flowchart for ADXL 345 Accelerometer:


Start
Initialize Variables
Initialize UART + SPI
SPI Read Data
Separate x, y, z
Divide x, y, z by 128
Print to terminal
Delay 1 second

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;
```

```c
  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 */
```

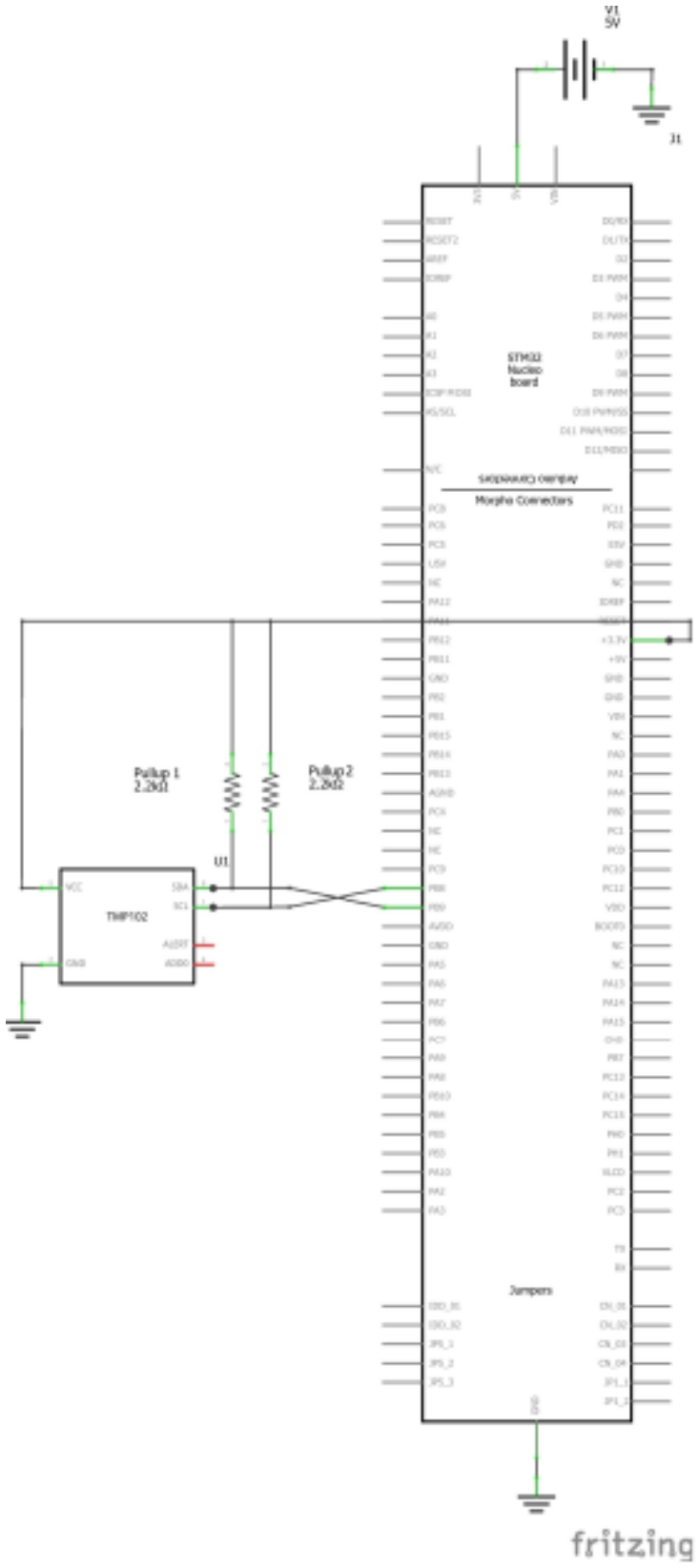
```c
}

/* 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:

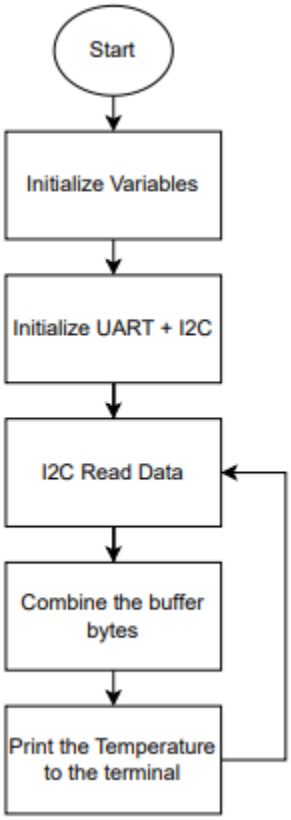
Xg	Yg	Zg
0.03	0.00	0.98
0.03	0.00	0.98
0.03	0.00	0.98
0.03	0.00	0.98
0.25	0.66	0.60
-0.14	0.20	-0.95

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



Flowchart for TMP 102 Temperature Sensor



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);
}
```

```c
/* 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)
```

```c
{

  /* 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 */
```

```c
}

/* 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

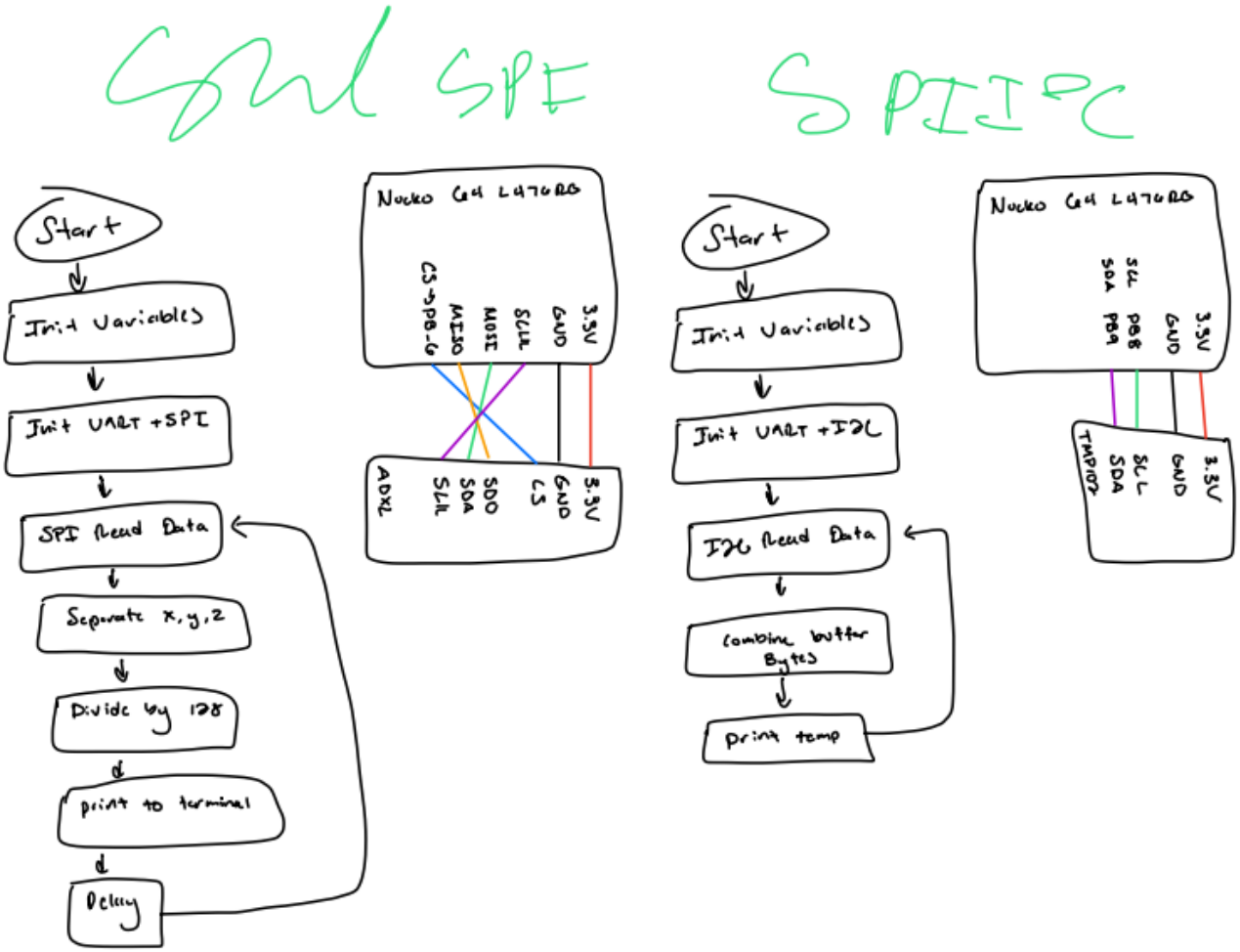
This table shows a couple samples that were returned from the TMP 102 Temperature Sensor. The returned 22.78°C average reading equates to roughly 73°F which makes sense for the abnormally warm lab room. In combination with a finger warming up the sensor, the results from the TMP 102 sensor were mainly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


SPI
SPI I²C
Start
Init Variables
Init UART +SPI
SPI Read Data
Separate x, y, z
Divide by 128
print to terminal
Delay
Nucleo G4 L476RG
CS→PB-6
MISO
MOSI
SCLK
3.3V
GND
ADXL
SCLK
SDO
SDA
CS
GND
3.3V
Start
Init Variables
Init UART +I2C
I2C Read Data
Combine buffer Bytes
print temp
Nucleo G4 L476RG
SCL
SDA
PB8
PB9
GND
3.3V
TMP102
SCL
SDA
GND
3.3V

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.