ECE 4240 – Lab 3 Report

Getting Stuff Done with I^2C

Course: ECE 4240 – Microprocessor Interfacing

Instructor: Dr. Dean McNeill

Student: Dayaveer Gill (7969885)

Lab Partner: Severyn Shved

Date Performed: October 20, 2025

Date Submitted: October 29, 2025

University of Manitoba

Department of Electrical and Computer Engineering

4.0 - Laboratory Experiments

Part 1

For the first part of the lab we begun by programming the board as instructed in part 1, and adding all the files to their necessary locations. Then we began by adding all the include statements, then adding the required code for the floating to digit function which is shown in figure 2 in the appendix. The code firstly subtracts the whole part from the float then multiplying it by the scaler, and then rounding it using a if-else statement to get rounded results. Then we called the generate sine wave to get a signal, which was then sent to the convert float to digit function to turn back the whole and fraction parts. Then we displayed the current step count on the LCD, and then cleared the whole LCD, then displayed the sine value, the whole and fraction parts, after which the step count increase by one and the loop repeats. The code for this part is shown in figure 1 in the appendix. Our results matched the expected behavior shown in the lab handout, and the results were shown to a TA.

Part 2

In Part 2 we started off by adding a include statement for the AHT20_Driver.h file. Then we read through the provided data sheet and determined that the I^2C bus address to be 0x38, and changing the pins to the next available pins which were PB7 and PB8. After having all the pins configured we attached the AHT20 temperature sensor to our circuit from part 1. After having everything set up we began writing code in the main c file, which is shown in figures 3 and 4 in the appendix. This code starts by updating the previous readings so the 2nd last becomes last, previous becomes 2nd last and the current becomes the previous. Then we get a reading by calling the getAHT20_Temperature function and passing it the reference to the temp variable which will store the temperature read from the sensor and reading is the result returned from the function. The sensor is OK if the result is 0, if its anything else the temp is not accurate and can not be used for processing, that's why we check is the reading is okay if it is only then the current temperature is updated to the temperature read from the sensor. Then after every one second the the LCD is updated to display the step count first, then clear the LCD, then display the 3 previous values and the current value in brackets. Our results matched the expected behavior shown in the lab handout, and the results were shown to a TA.

Part 3

In part 3, we started off by making a moving average function. The functions initially starts with temperature array of size 16 filled with 0's, and this function takes in the temperature reading. In this function a while runs 15 times to get rid of the data in position 0, and move all

the other data up, such as position 1 data takes space of position 0 and so on it also adds the value to a total. After the loop runs the temperature reading which is given as the input to the function is added to position 15, and the total. Then total divided by 16 is returned which is the average. In the while loop we start by getting the calling the getAHT20_Temperature function which updates the temp to the temperature reading and returns reading as if the reading was valid or not. Then we use a if statement to determine if the reading was valid if it was we send the temperature reading the the moving average which returns the average which is then sent to the covert floating to digits functions and then the step count is printed on the LCD then the LCD is cleared and then the average temperature is printed on the LCD and step count is increased by 1. Our results matched the expected behavior shown in the lab handout, and the results were shown to a TA.

Additional Questions

Question 1:

I would change the resistors pull up resistors for the SDA and SCL with smaller values. which would reduce the RC time constant which would then make the rise time fall below 1 μs ensuring it follows I^2C requirements and make it more stable.

Question 2:

From section 3.2.2 of the lab handout we see that each 4x increase in sampling is equivalent to 1 extra effective bit. Which one would assume for a 16x increase would be equivalent to 4 extra effective bits, but section 3.2.2 tells us thats not true and that 16x oversampling only results in 2 extra effective bits due to noise averaging effects. We also know that our board uses a 12 bit ADC [3] so this would make the ENOB 14 bits.

Question 3:

After researching I was able to find out that with an ”single MOS-FET a bi-directional level shifter circuit can be realised to connect devices with different supply voltages of e.g. 5 Volt and 3.3 Volt to one I^2C-bus system”[4]. This is exactly what we need so that neither device is stressed by the over voltage on their inputs.

Question 4:

After reading the data sheet for BME280 atmospheric sensor, I found out you can connect two identical sensors to the same I^2C bus by changing their least significant bit of their address by

connecting the SDO pin to either ground for 0 or VDD for 1 [2]. Then I read the data sheet for AS6212 sensor and found out that the AS6212 digital temperature sensor allows up to 8 of the same sensors to be connected to the same I^2C bus, it allows this by having 2 address select pins on the sensor and depending on the configuring of the 2 pin connections to SCL, SDA, VSS or VDD it can select between 8 different addresses [1].

Question 5.A:

I chose the BME280 atmospheric sensor, and to connect this with an STM32 project, you would need to connect VDD to 3.3V, GND to GND, SDA to I^2C SDA which was P9 for us, SCL to I^2C SCL which was P8 for us, SDO to GND or VDD and then CSB to VDD which disables SPI mode and enables I^2C mode for the sensor. We would also need to add the pull up resistors like we did in this lab. We would also need to set the humidity register, which is 0xF2 to a oversampling value of 1 which is 0x01, then also set the temperature, pressure, and sensor mode register, which is 0xF4 to over sampling of 1 for temperature and pressure and mode to normal which has the value of 0x27 [2].

Question 5.B:

I2CWrite(Sensor adress, register address(0xF2), register value to set(0x01)
I2CWrite(Sensor adress, register address(0xF4), register value to set(0x27)

dataHumidty = I2CRead(Sensor adress, register address(0xF2))
dataTempPressure = I2CRead(Sensor adress, register address(0xF4))

Citations

[1] SparkFun, *AS621x Temperature Sensor Datasheet*, Available: https://cdn.sparkfun.com/assets/f/a/6/c/8/AS621x_DS000677_2-00.pdf, accessed Oct. 29, 2025.

[2] Bosch Sensortec, *BME280: Final Data Sheet, Revision 1.1*, Available: https://cdn.sparkfun.com/assets/learn_tutorials/4/1/9/BST-BME280_DS001-10.pdf, accessed Oct. 29, 2025.

[3] STMicroelectronics, *This is information on a product in full production*, Nov. 2024. Available: https://www.st.com/resource/en/datasheet/DM00037051.pdf, accessed Oct. 30, 2025.

[4] NXP Semiconductors, *Application Note: Bi-directional Level Shifter for I2C-bus and Other Systems*, Available: https://cdn-shop.adafruit.com/datasheets/an97055.pdf, accessed Oct. 29, 2025.

Appendix:

```
  CLCD_Init(&hi2c1);
  CLCD_ClearScreen();
  CLCD_SetCursor(0,0);

  int stepCount = 0;
  char strBuffer[17];
  int32_t whole = 0;
  int32_t frac = 0;


/* USER CODE END 2 */


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

    float signal = generate_sine_wave();
    convert_float_to_digits(signal, &whole, &frac, 3);

    sprintf(strBuffer, "P1 Step # %d", stepCount);
    CLCD_SetCursor(0,0);
    CLCD_WriteString(strBuffer);

    CLCD_SetCursor(1,0);
    sprintf(strBuffer, "                ");
    CLCD_WriteString(strBuffer);
    sprintf(strBuffer, "Sine Val %ld.%03ld", (long)whole, (long)frac);
    CLCD_SetCursor(1,0);
    CLCD_WriteString(strBuffer);

     stepCount++;
     HAL_Delay(1000);
}
```

Figure 1: Part 1 Code.

```c
void convert_float_to_digits(float value, int32_t *wholePart, int32_t *fracPart, uint8_t precision){
    // Calculate scaling factor based on desired precision
       int32_t scale = pow(10, precision);

       // Calculate the integer (whole) part by casting from float to int
       *wholePart = (int32_t)value;

       // Calculate the fractional part with rounding by:
       //   Subtracting the whole part from the float
       //  Multiplying by the scaling factor
       //  Conditionally Adding 0.5 or -0.5 to round the result.
       *fracPart = (value -(*wholePart))*scale;

       if(*fracPart >= 0.0f){
        *fracPart += 0.5f;
       } else{
        *fracPart -= 0.5f;
       }
       //<<  Complete the code by implementing the description above!  >>

       //<<     Implement your code above!       >>

       // Make fractional part positive if necessary, avoids printing multiple negative signs
       if (*fracPart < 0) {
           *fracPart = -*fracPart;
       }

       // Edge case: If rounding makes fracPart equal to scale, adjust wholePart and reset fracPart
       if (*fracPart >= scale) {
           *wholePart += (value >= 0) ? 1 : -1;
           *fracPart = 0;
       }
```

Figure 2: Convert digits function, part 1 & 2 Code.

```c
CLCD_Init(&hi2c1);
CLCD_ClearScreen();
CLCD_SetCursor(0,0);
AHT20_Init(&hi2c1);

int stepCount = 0;
char strBuffer[17];
int t_3last = 0;
int t_2last = 0;
int t_1last = 0;
float t_curr = 0;
float temp = 0.0;
uint32_t start = HAL_GetTick();
```

Figure 3: Part 2 Code.

```
while (1)
{
  /* USER CODE END WHILE */
    t_3last = t_2last;
    t_2last = t_1last;
    t_1last = t_curr;

    uint32_t reading = getAHT20_Temperature(&temp);

    if(reading == 0){
      t_curr = (int)temp;
    }
    else{
      t_curr = t_1last;
    }
    if ((HAL_GetTick()- start)>=1000){

      start = HAL_GetTick();
      sprintf(strBuffer, "P2 Loop # %d", stepCount);
      CLCD_SetCursor(0,0);
      CLCD_WriteString(strBuffer);

      CLCD_SetCursor(1,0);
      sprintf(strBuffer, "                ");
      CLCD_WriteString(strBuffer);
      sprintf(strBuffer, "%d, %d, %d, [%d]", t_3last, t_2last, t_1last,t_curr);
      CLCD_SetCursor(1,0);
      CLCD_WriteString(strBuffer);

      stepCount++;

    }

}
```

Figure 4: Part 2 Code.

```
CLCD_Init(&hi2c1);
CLCD_ClearScreen();
CLCD_SetCursor(0,0);
AHT20_Init(&hi2c1);

float t_curr = 0;
float temp = 0.0;
int32_t whole = 0;
int32_t frac = 0;
int stepCount = 0;
char strBuffer[17];
uint32_t start = HAL_GetTick();
float tempArray[16] = {0};

float movingAverage(float inputTemp){
    int i = 0;
    float total = 0;
    while (i < 15){
        tempArray[i]= tempArray[i+1];
        total += tempArray[i];
        i++;
    }
    total += inputTemp;
    tempArray[15] =  inputTemp;
    total = total/16;
    return total;
}
```

Figure 5: Part 3 Code.

```
while (1)
{
    if ((HAL_GetTick()- start)>1000){

        start = HAL_GetTick();
        uint32_t reading = getAHT20_Temperature(&temp);

        if(reading == 0){
            t_curr = movingAverage(temp);
            convert_float_to_digits(t_curr, &whole, &frac, 3);

            sprintf(strBuffer, "P3 Loop # %d", stepCount);
            CLCD_SetCursor(0,0);
            CLCD_WriteString(strBuffer);

            CLCD_SetCursor(1,0);
            sprintf(strBuffer, "                ");
            CLCD_WriteString(strBuffer);
            sprintf(strBuffer, "Temp: %ld.%03ld %cC", (long)whole, (long)frac, 0b11011111);
            CLCD_SetCursor(1,0);
            CLCD_WriteString(strBuffer);

            stepCount++;
            HAL_Delay(1000);
        }
    }
```

Figure 6: Part 3 Code.