

## Lab Report 10

João Luiz Teixeira de Godoy Pereira

[Degodoypereira.j@northeastern.edu](mailto:Degodoypereira.j@northeastern.edu)

Partner: Woobens Fleurime

[Fleurime.w@northeastern.edu](mailto:Fleurime.w@northeastern.edu)

Submit date: 04/17/2024

Due Date: 04/17/2024

### **Abstract**

This project illustrates the integration of a high-performance, low-power, 3-axis ADXL345 digital accelerometer with a DE1-SoC board, through using a hybrid hardware-software approach. The aim was to measure physical tilt and translate it into a digital inclination angle displayed on 7-segment displays. By harnessing the DE1-SoC's high-speed FPGA and HPS capabilities, the project effectively mapped raw acceleration data to angles between 0 and 180 degrees. The implementation of this project taught us how to combine software abstraction with hardware design in embedded systems to solve practical challenges in real-time data processing and display.

## Introduction

The lab's purpose was to use digital sensors with embedded systems while facing the challenges around building the software, setting up hardware, and integrating both systems. In this final project, we focused on interfacing the high-performance ADXL345 accelerometer with the DE1-SoC board to measure the board's inclination and display this as an angle on 7-segment displays. This required a understanding of both the hardware capabilities of the DE1-SoC and the software in C++ needed to manipulate data from the accelerometer; both concepts of which were built throughout the entire semester. Through creating hardware designs in Quartus and software in C++, the project explored the practical application of theories learned throughout the course, outlining the importance of comprehension in both hardware and software.

Before starting the lab, we first looked at the ADXL345 accelerometer data sheet to determine the values that we would use in our lab.

12:03 PM Sat Apr 13

ADXL345

Trigger mode, set the device to bypass mode and then set the device back to trigger mode. Note that the FIFO data should be read first because placing the device into bypass mode clears FIFO.

analog.com

Rev. G | 21 of 36

### Data Sheet

#### ADXL345

#### SELF-TEST

The ADXL345 incorporates a self-test feature that effectively tests its mechanical and electronic systems simultaneously. When the self-test function is enabled (via the SELF\_TEST bit in the DATA\_FORMAT register, Address 0x31), an electrostatic force is exerted on the mechanical sensor. This electrostatic force moves the mechanical sensing element in the same manner as acceleration, and it is additive to the acceleration experienced by the device. This added electrostatic force results in an output change in the x-, y-, and z-axes. Because the electrostatic force is proportional to  $V_S^2$ , the output change varies with  $V_S$ . This effect is shown in Figure 43. The scale factors shown in Table 14 can be used to adjust the expected self-test output limits for different supply voltages,  $V_S$ . The self-test feature of the ADXL345 also exhibits a bimodal behavior. However, the limits shown in Table 1 and Table 15 to Table 18 are valid for both potential self-test values due to bimodality. Use of the self-test feature at data rates less than 100 Hz or at 1600 Hz may yield values outside these limits. Therefore, the part must be in normal power operation (LOW\_POWER bit = 0 in BW\_RATE register, Address 0x2C) and be placed into a data rate of 100 Hz through 800 Hz or 3200 Hz for the self-test function to operate correctly.

Figure 43. Self-Test Output Change Limits vs. Supply Voltage

Table 14. Self-Test Output Scale Factors for Different Supply Voltages,  $V_S$

Supply Voltage, $V_S$ (V)	X-Axis, Y-Axis	Z-Axis
2.00	0.84	0.8
2.50	1.00	1.00
3.30	1.77	1.47
3.60	2.11	1.89

Table 15. Self-Test Output in LSB for 12 g, 10-Bit or Full Resolution ( $T_A = 25^\circ\text{C}$ ,  $V_S = 2.5\text{ V}$ ,  $V_{DDIO} = 1.8\text{ V}$ )

Axis	Min	Max	Unit
X	50	540	LSB
Y	-540	-50	LSB
Z	75	875	LSB

Table 16. Self-Test Output in LSB for 12 g, 10-Bit Resolution ( $T_A = 25^\circ\text{C}$ ,  $V_S = 2.5\text{ V}$ ,  $V_{DDIO} = 1.8\text{ V}$ )

Axis	Min	Max	Unit
X	25	270	LSB
Y	-270	-25	LSB
Z	38	438	LSB

Table 17. Self-Test Output in LSB for 12 g, 10-Bit Resolution ( $T_A = 25^\circ\text{C}$ ,  $V_S = 2.5\text{ V}$ ,  $V_{DDIO} = 1.8\text{ V}$ )

Axis	Min	Max	Unit
X	12	135	LSB
Y	-135	-12	LSB
Z	19	219	LSB

Table 18. Self-Test Output in LSB for 12 g, 10-Bit Resolution ( $T_A = 25^\circ\text{C}$ ,  $V_S = 2.5\text{ V}$ ,  $V_{DDIO} = 1.8\text{ V}$ )

Axis	Min	Max	Unit
X	6	67	LSB
Y	-67	-6	LSB
Z	10	110	LSB

Handwritten notes on the left page:

- 88.5 gms!
- initial value =  $x \pm 100$
- range of raw data
- $\text{angle} = (x + 100) \cdot 180$
- $(-1000, 0)$
- $(1000, 180)$
- $y = mx + b$

analog.com

Rev. G | 22 of 36

12:03 PM Sat Apr 13

ADXL345

23 of 36

### Data Sheet

#### ADXL345

#### REGISTER MAP

Table 19.

Hex	Dec	Name	Type	Reset Value	Description
0x00	0	DEVICE_ID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/W	00000000	Tap threshold
0x1E	30	OFSX	R/W	00000000	X-axis offset
0x1F	31	OFSY	R/W	00000000	Y-axis offset
0x20	32	OFSZ	R/W	00000000	Z-axis offset
0x21	33	DUR	R/W	00000000	Tap duration
0x22	34	Latent	R/W	00000000	Tap latency
0x23	35	Window	R/W	00000000	Tap window
0x24	36	THRESH_ACT	R/W	00000000	Activity threshold
0x25	37	THRESH_INACT	R/W	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/W	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/W	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/W	00000000	Free-fall threshold
0x29	41	TIME_FF	R/W	00000000	Free-fall time
0x2A	42	TAP_AXES	R/W	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/W	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/W	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/W	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/W	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupt
0x31	49	DATA_FORMAT	R/W	00000000	Data format control
0x32	50	DATA0	R	00000000	X-Axis Data 0
0x33	51	DATA1	R	00000000	X-Axis Data 1
0x34	52	DATA0	R	00000000	Y-Axis Data 0
0x35	53	DATA1	R	00000000	Y-Axis Data 1
0x36	54	DATA0	R	00000000	Z-Axis Data 0
0x37	55	DATA1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/W	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

Handwritten notes on the right page:

- the y intercept should be zero because the range of the x intercept is 1000 to 1000 to find median which is zero and zero's equal to 90.
- $y_2 - y_1 = 180 - 0$
- $x_2 - x_1 = 1000 - 1000$
- $\frac{180}{2000} = 0.09$
- $y = 180x + 90$
- use the slope equation because the two points are given and you need line equation implemented in algorithm.

analog.com

Rev. G | 23 of 36

### Data Sheet

#### ADXL345

#### REGISTER MAP

Table 19.

Calculations Before Lab, Image 2

## Calculations Before Lab, Image 1

## Part 1

In the initial phase of our final project, we interpreted raw output from the ADXL345 accelerometer, which measures acceleration across three axes in units of milli-gravities (mg). The challenge was to transform these outputs, ranging from -1000 mg to 1000 mg, into interpretable inclination angles between 0 and 180 degrees. To achieve this, we developed an accurate linear mapping algorithm inside the **mapValue()** function within

the **GSensorMain.cpp** file. This function was designed to scale the raw accelerometer data proportionally to the angle range, facilitating the understanding of where is the board's tilting.

The **mapValue()** function uses a simple mathematical approach to convert the accelerometer's readings to angles. We defined the input minimum and maximum (**in\_min** and **in\_max**) as -1000 mg and 1000 mg, and the output range (**out\_min** and **out\_max**) as 0 and 180 degrees, the function applies a linear transformation to scale and shifts the accelerometer data to the angle range. This ensures that an output of -1000 mg maps to 0 degrees, 0 mg maps to 90 degrees, and 1000 mg maps to 180 degree, which directly corresponded to the tilting of the board.

The solution portrayed a dynamic, in real-time, and accurate angle display on the 7-segment displays. The **mapValue()** method significantly enhanced the interactive capabilities of our embedded system. This solved the practical challenge of data interpretation but also showed the precision of our algorithm in real-time.

## Part 2

In Part 2 of our project, we managed the hardware interactions through using memory-mapped I/O operations. To improve our system, we placed the necessary hardware interaction logic within the **DE1SoCfpga** class. This class featured methods like the **RegisterWrite()** and **RegisterRead()**, which were created in previous labs and enabled us to directly manipulate registers. With this, we reduced the potential errors, ensuring consistent performance across various parts of the project. As in the previous lab, the method **RegisterWrite()** computed the specific register's address through adding the provided value to the base memory address and

then writing the given value to this computed address. Similarly, **RegisterRead()** retrieves data by accessing the computed address based on the register.

Our approach to software-hardware using these previously defined functions inside the **DE1SoCfpga** class simplified the development process and enhanced the system's reliability. By abstracting the low-level hardware interactions into reusable class methods, we minimized coding errors and improved effectiveness. This facilitated debugging and maintenance of our embedded system application, which illustrated how well-structured improves real-time processing and display.

### Part 3

In Part 3 of our project, we focused on implementing display mechanism for the inclination angles. We utilized the 7-segment displays on the DE1-SoC board to visualize the angles. The challenge was to ensure that the displayed angles were accurate, dynamic and responsive to real-time changes in the board's inclination.

To achieve this, we implemented an interrupt-driven mechanism that utilized the FPGA's timers to update the angle display, with a 50 Hz frequency. This ensured that the angle display was updated every 20 milliseconds, which was sufficient for real-time display of the board's tilt. Another benefit of the interrupt-driven was that it allowed the system to operate without any CPU. We had to then connect this mechanism back to the software part and test for the results, which are clear in the video and show the efficient and reliable display. By using the FPGA's timers the classes created, we abstracted the hardware manipulation, minimizing coding errors

and achieving high-performance real-time processing display. The tests and accuracy of our implementation is clear in the image below:

```
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=8 mg, Z=1092 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-40 mg, Y=36 mg, Z=1076 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-44 mg, Y=12 mg, Z=1088 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-32 mg, Y=20 mg, Z=1080 mg
X=-36 mg, Y=12 mg, Z=1092 mg
X=-36 mg, Y=12 mg, Z=1092 mg
X=-36 mg, Y=12 mg, Z=1092 mg
```

Assignment 3, Image 3

## Part 4

Part 4 of our project involved establishing a communication link between the Hard Processor System (HPS) and FPGA on the DE1-SoC board, which was crucial for the real-time data processing of accelerometer measurements. The challenge was to configure the HPS-FPGA bridge correctly to ensure reliable data transfer. We addressed this challenge by utilizing the Quartus Prime and integrated Programmable Input/Output (PIO) components. Further, this helped to send sensor data to the FPGA and perform angle calculations sending them back to the HPS. To achieve this, we used two PIO components, one for output and another for input. The output PIO was configured to handle the X-coordinate data from the G-sensor, which the FPGA

then used to compute the inclination angle. We ensured these components were correctly connected to the HPS's **h2f\_lw\_axi\_master** interface, allowing smooth and error-free data transmission. This setup ensured that the data handling between the HPS and FPGA was both precise. The configuration of these communication links taught us how to create stable connections between hardware and software using Quartus prime and portrayed the interaction between the software running on the HPS and the hardware logic on the FPGA. At this point we were able to develop a real-time data processing and display.

With this we then tested our results by running the program. These results are evident in the page below and portray our effective attempt.

Assignment 4, Images 4-6, include entire computer screenshot

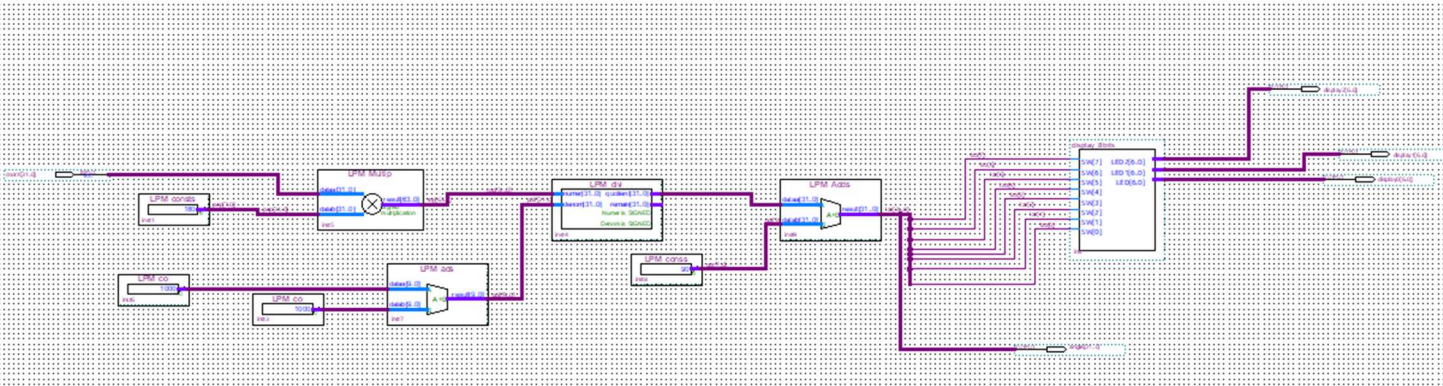


## Part 5

Part 5 of our final project was focused on the hardware design to compute the inclination angle of the board based on the X-coordinate data received from the HPS.

The challenge was to design an FPGA-based solution that could accurately calculate and then visually display the computed angle on the DE1-SoC board's 7-segment displays. Hence we developed a custom hardware component, named **angle\_calculator**, which translated accelerometer data into an interpretable angle. In our Quartus project, we created a new schematic file for the **angle\_calculator** and designed it to include various logical and arithmetic components that could process the sensor data. The math behind these arithmetic operations followed a similar approach to when we defined the `mapValue()` method earlier in the lab. The main difference was that this time, we were trying to use hardware to get these angles from the data instead of using the software approach. Therefore, we utilized adders and multipliers to formulate the calculation of the inclination angle based on the received X-coordinate, for example. These components were configured to handle 32-bit integers to ensure precision in calculations. We connected these components so that the X-coordinate input was directly processed to output the angle, which was then forwarded to both the HPS and the 7-segment displays for real-time visualization. The implementation of this hardware design was exceptionally effective, as it enabled the precise conversion of raw accelerometer data into a visually interpretable format on the 7-segment displays, reflecting the physical tilt of the board in real-time. The **angle\_calculator** component proved to be a cornerstone of the project, bridging the gap between raw data acquisition and user-friendly output, and highlighting the project's

innovative approach to embedded system design. The way we set up our schematics to ensure the correct outcome in the hardware end is clear below:



Assignment 5, Schematic, Image 7

## Part 6

Part 6 of our final project was dedicated to integrating software with the custom hardware system designed on the FPGA, which required precise control over input and output (I/O) operations through Programmable Input/Output (PIO) components. Our objective was to ensure data transfer between the software running on the HPS and the hardware logic on the FPGA. To accomplish this, we developed a **PIOControl class** that handled all interactions from the PIO components regarding the output and input functions. The **PIOControl class** extended the functionality of the pre-existing **DE1SoCfpga class**, inheriting its methods for memory-mapped I/O operations such as **RegisterWrite()** and **RegisterRead()**. This means that the **PIOControl class** was a subclass of the **DE1SoCfpga class**. These methods for the PIO components by set up the appropriate base addresses (**OUT\_BASE** and **IN\_BASE**). For example, the **WritePIOout(int value)** method in **PIOControl** processed the writing data to the output PIO register. This method took an integer value representing the angle calculated by the FPGA and then wrote it to the output register using the **RegisterWrite()** function. Similarly, the

**ReadPIOin()** method retrieved the computed angle from the input PIO register, illustrating the communication with the FPGA for both reading and writing. The idea was that by adding this functions using the **RegisterRead()/RegisterWrite()** methods we were able to create the **ReadPIOin()/ WritePIOout(int value)** which had a similar functionality but giving us control over the PIOControl and not the registers. This integration was critical for our project as it allowed the HPS to send the X-coordinate data to the FPGA and receive the calculated inclination angle back from it. Through using the hardware access logic within the **PIOControl class**, we made our software easier to manage and debug.

## Part 7

In Part 7 of our final project, we focused on integrating the software and hardware components that we had developed, enabling the full functionality of our system. Our objective was to ensure that the **G-sensor** data accurately triggered the computation of inclination angles by the FPGA, meaning that the data captured by the sensore would make the board incline correctly. This information would then be read back and displayed by the software on the HPS. The main challenge here was to ensure that all components interacted in real-time to provide accurate output. To address this challenge, we modified the main function in our **GSensorMain.cpp** program to add instances of the **PIOControl class**. Using dynamic memory allocation, we created a **new PIOControl** and managed our custom PIO components for input and output. This setup allowed us to directly store the X-coordinate data obtained from the **G-sensor** into the FPGA via the **WritePIOout()** method. Furthermore, the computed angle, which represented the board's inclination, was retrieved from the FPGA using the **ReadPIOin()** method. This method reads the angle from the input PIO register and displays it in the terminal,

providing a direct interaction between the software on the HPS and the hardware logic on the FPGA. The integration portrayed the behavior of our system during runtime, where it was displayed accurate inclination measurements based on real-time gathered sensor data. This integration depicted our ability with the software-hardware interaction, ensuring that our final project was responsive to real time in hardware and software, making it reliable in processing and displaying sensor data. To correctly test the project, some screenshots of the running system were taken and will be added below:

```
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-16 mg, Y=56 mg, Z=1068 mg
Tilt Angle: -2 degrees
Display Angle: 88 degrees
X=-4 mg, Y=40 mg, Z=1076 mg
Tilt Angle: -1 degrees
Display Angle: 89 degrees
X=-4 mg, Y=40 mg, Z=1076 mg
Tilt Angle: -1 degrees
Display Angle: 89 degrees
X=-4 mg, Y=40 mg, Z=1076 mg
Tilt Angle: -1 degrees
Display Angle: 89 degrees
^Cuser126@de1soclinux:~/lab_project/SW_project$ make clean
rm -f DE1SoChps.o DE1SoCfpga.o PIOControl.o GSensor.o GSensorMain.o SW_project
user126@de1soclinux:~/lab_project/SW_project$
```

Assignment 7, Image 8

[illegible]

Assignment 7, Image 9

[illegible]

## Assignment 7, Image 10

## Conclusion

In conclusion, this project demonstrates the integration of a high-performance ADXL345 accelerometer with the DE1-SoC board, utilizing both hardware and software approaches. By using the FPGA and HPS capabilities of the DE1-SoC board, we were able to map raw acceleration data to angles between 0 and 180 degrees and display the angles on 7-segment displays in real-time. Through the development of classes in C++ and hardware designs in Quartus Prime, we were able to portray our understanding in both the hardware and software when solving the lab's challenges. Further, this enabled us to progress our knowledge in embedded systems and get a better understanding of how it works. The successful implementation of this project illustrates our ability to combine software abstraction with hardware design to achieve accurate real-time processing display. This shifted the accelerometer data to the angle range. This transformation ensures that an output of -1000 mg maps to 0 degrees, 0 mg maps to 90 degrees, and 1000 mg maps to 180 degrees, corresponding to the physical tilting of the board. Overall, the lab taught me and my partner great skills on manipulating hardware component creation, logic and regarding software, using of classes to abstract methods and behaviors in C++. All of which are useful knowledge in this and future embedded design courses.