Lab Report 10

João Luiz Teixeira de Godoy Pereira

Degodoypereira.j@northeastern.edu

Partner: Woobens Fleurime

Fleurime.w@northeastern.edu

Submit date: 04/17/2024

Due Date: 04/17/2024

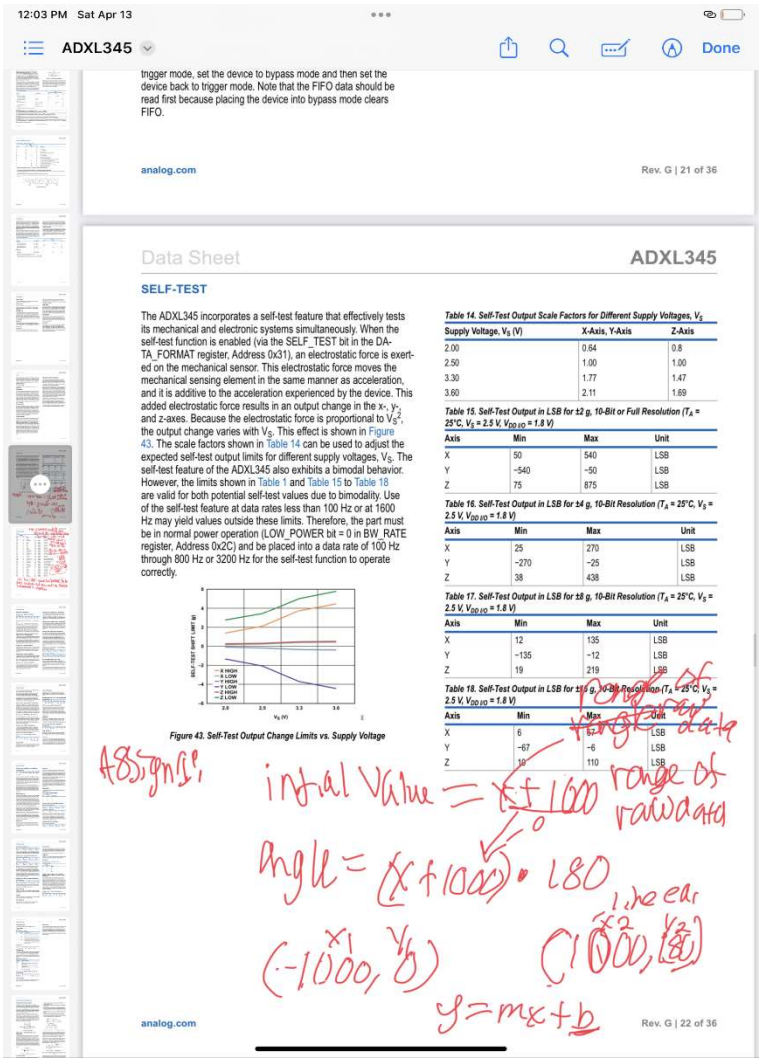
Abstract

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 taugh us how to combine software abstraction with hardware design in embedded systems to solve practical challenges in real-time data processing and display.

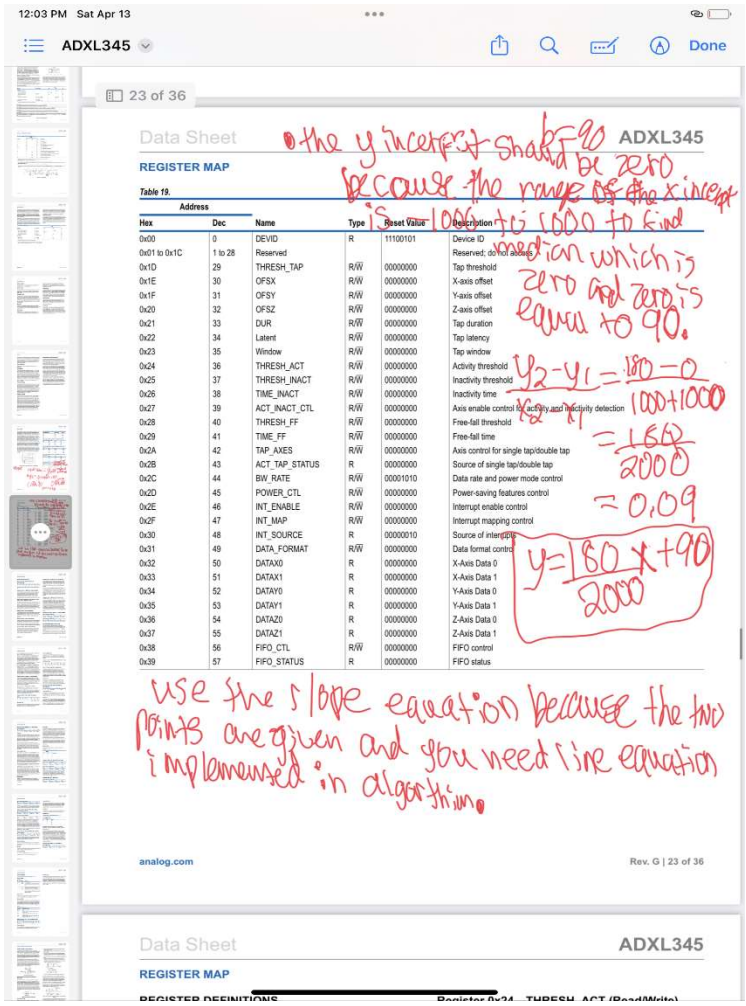
Introduction

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.



Calculations Before Lab, Image 1



Calculations Before Lab, Image 2

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.

Oue 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:

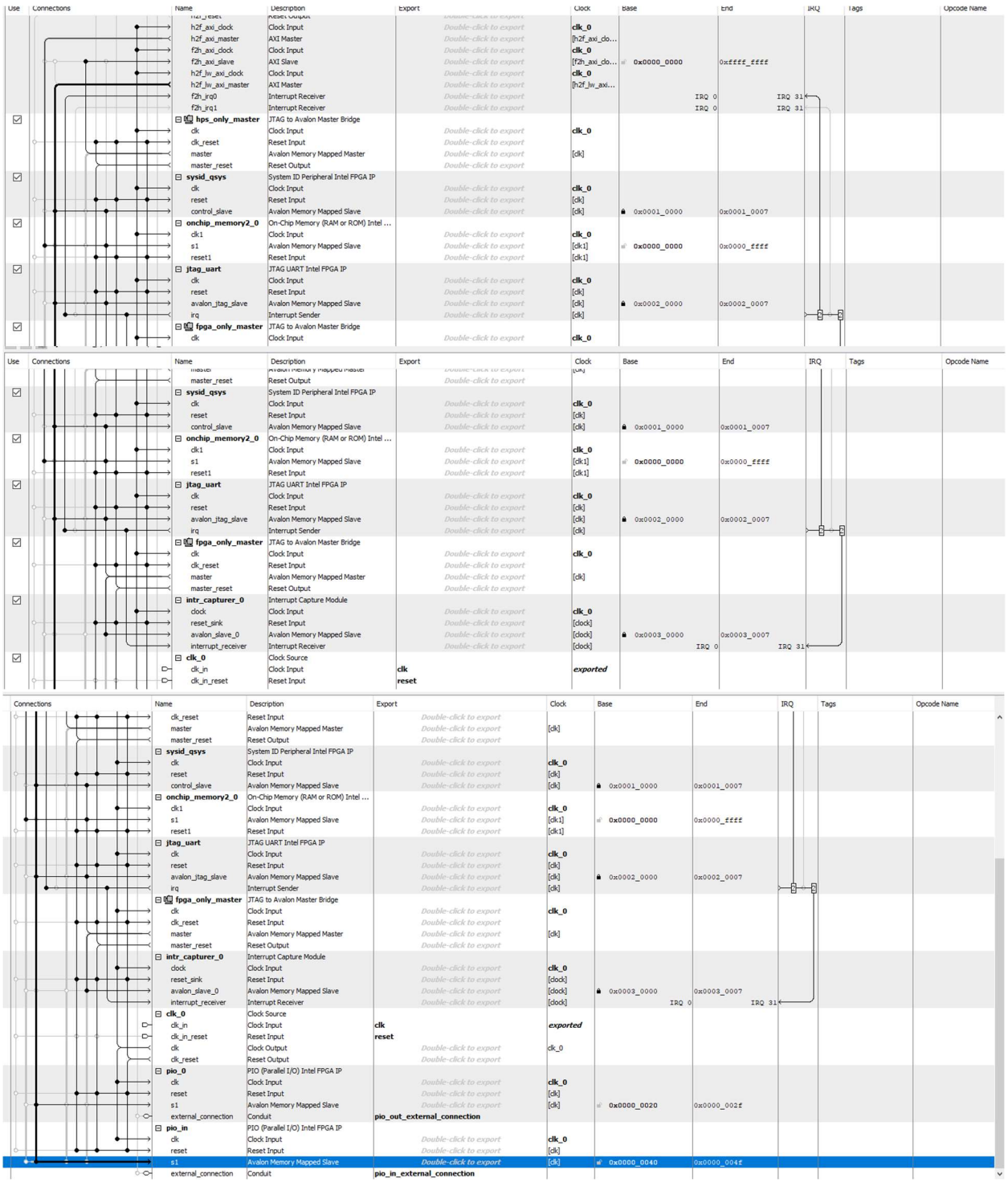
Assignment 3, Image 3

Part 4

Part 4 of our project involved establishing a communication link between the Hard Processor System (HPS) and FPGAon 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 price 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.



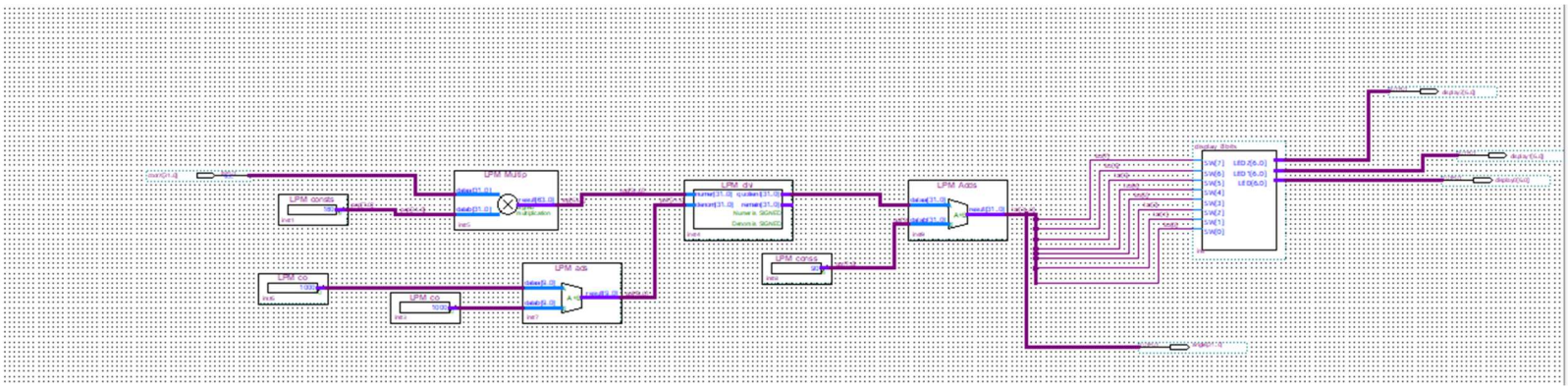
Assignment 4, Images 4-6, include entire computer screenshot

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

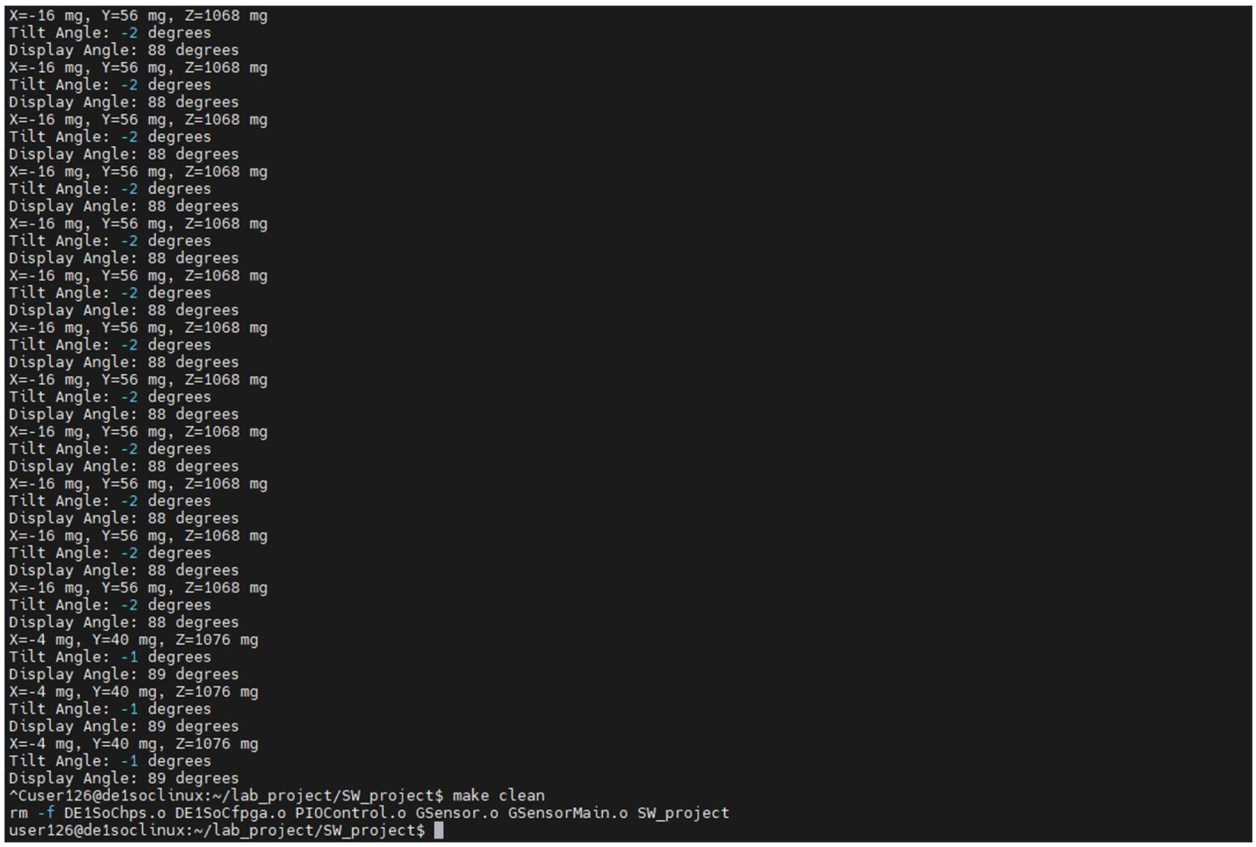
Part 6 of our final project was dedicated to integrating software with the custom hardware system designed on the FPGA, which required precise control over input and output (I/O) operations through Programmable Input/Output (PIO) components. Our objective was to ensure data transfer between the software 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:



Assignment 7, Image 8

```
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=772 mg, Y=8 mg, Z=688 mg
Tilt Angle: 69 degrees
Display Angle: 159 degrees
X=792 mg, Y=0 mg, Z=720 mg
Tilt Angle: 71 degrees
Display Angle: 161 degrees
X=792 mg, Y=0 mg, Z=720 mg
Tilt Angle: 71 degrees
Display Angle: 161 degrees
X=792 mg, Y=0 mg, Z=720 mg
Tilt Angle: 71 degrees
Display Angle: 161 degrees
X=792 mg, Y=0 mg, Z=720 mg
Tilt Angle: 71 degrees
Display Angle: 161 degrees
X=792 mg, Y=0 mg, Z=720 mg
Tilt Angle: 71 degrees
Display Angle: 161 degrees
X=792 mg, Y=0 mg, Z=720 mg
Tilt Angle: 71 degrees
Display Angle: 161 degrees
X=792 mg, Y=0 mg, Z=720 mg
```

Assignment 7, Image 9

```
X=-616 mg, Y=-124 mg, Z=876 mg
Tilt Angle: -56 degrees
Display Angle: 34 degrees
X=-616 mg, Y=-124 mg, Z=876 mg
Tilt Angle: -56 degrees
Display Angle: 34 degrees
X=-616 mg, Y=-124 mg, Z=876 mg
Tilt Angle: -56 degrees
Display Angle: 34 degrees
X=-616 mg, Y=-124 mg, Z=876 mg
Tilt Angle: -56 degrees
Display Angle: 34 degrees
X=-616 mg, Y=-124 mg, Z=876 mg
Tilt Angle: -56 degrees
Display Angle: 34 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
X=-628 mg, Y=-136 mg, Z=876 mg
Tilt Angle: -57 degrees
Display Angle: 33 degrees
```

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.