

Lab Report 11  
(Final Lab Report)

Ryder Robbins  
robbins.r@northeastern.edu

Submit date: 4/18/2024  
Due Date: 4/18/2024

## **Abstract**

Lab 11 aimed to integrate software and hardware components from previous labs into a co-designed implementation. Central to this integration was the use of the ADXL345 digital accelerometer sensor module, or G-sensor, which measures acceleration and angular position. For all the marbles...

## **Introduction**

The primary objective of Lab 11 was to integrate the software and hardware components from all previous laboratories, culminating in a hardware/software co-designed implementation. The digital accelerometer sensor module (ADXL345), commonly referred to as the G-sensor, would be referenced and utilized throughout the lab, and most notably connected to the HPS on the DE1SoC board. This sensor, by definition, could measure the acceleration and angular position of the DE1-SoC board, providing high-resolution measurements across three axes (with convention X, Y, Z). The Lab procedure entailed processing the sensor's measurements within the FPGA and converting them into the board's inclination angle. This angle, representing the tilt of the board, would then be displayed (in decimal) across three of the 7-segment displays, and relayed back to the HPS. This, for the first instance in the Embedded Design and Enabling Robotics curriculum, would necessitate the development and programming of synchronous circuit-design-hardware and C++- program-software. The Quartus hardware design was necessary for the implementation of an algorithm that could convert G-sensor data (in mg) into the accurate angular position (in degrees) of the sensor, as well as configuring the DE1-SoC displays to display the results. The C++ program was to acquire measurements from the G-sensor, transmit the X coordinate to the FPGA, and the BDF schematic was to implement the necessary hardware circuitry to convert it into an angle. This angle will subsequently be displayed on the 7-segment display and relayed to the HPS for terminal display. This was handily the longest and most comprehensive Lab of the course.

## **Software and Hardware Used**

The Cyclone V DE1-SoC board has been used in every lab so far. Newly used software and hardware include MobaXterm, C++ coding shells, the ethernet port on the DE1-SoC, and a USB to ethernet adaptor.

Pulse width modulation (abbreviated PWM) is a means is a means of toggling across a range of states between 'OFF' and 'ON', such as dimming slider for a light. In-between values are virtually attainable through rapid successive repeating changes between 'OFF' and 'ON' states. If changes between 'OFF' and 'ON' states occur faster than the physical output-depending mechanism (such as the light from the dimming example), the resulting state is reflected by outputted power proportional to the average time over which the output value is set to 1 ('ON'). PWM is therefore the technique of modulating the width of these supplied 'ON' pulses to

virtually change a state by a percentage of 'ON'. As for the Lab 10 procedure, PWM would be used to translate input values of angular position into a servo-motor signal input consisting of 'ON' pulses of a width resulting in the positional state corresponding to the desired/inputted angle.

The ADXL345 G-sensor is a triple-axis accelerometer sensor used to detect orientation, motion, tilt, and vibration by measuring acceleration forces in three perpendicular axes (Named X, Y and Z). Acceleration values from the sensor would be retrieved via I2C0, interpreted through a larger object-oriented C++ program, and processed through the hardware setup in Quartus. The Quartus Prime Platform Designer tool would also be used for the first and only time in the course.

## **Lab Steps & Discussion**

### **Pre-Lab Assignment**

The Pre-Lab Assignment required reviewing the attached documentation. A base class needed to be created for accessing and using devices mounted on the HPS. The base address, 'HPS\_BRIDGE\_BASE', and bridge span, 'HPS\_BRIDGE\_SPAN', were provided in the *Lab 11 User Manual* (Kimani[4]). The remaining addresses needed to be declared as static constant unsigned integers (in the same header file containing the base class) and were found in the *Cyclone V HPS Technical Reference Manual* (Intel[7], 3014-3015). The *ADXL345 Datasheet* (Analog.com[8]) would provide references to the functionality of the G-sensor, and how to collect/graphically interpret data from it using a C++ program. The Lab instructions also explicitly stated, "You do not need to include anything in the report for the Pre-lab assignment." (Kimani[4]).

### **Lab Assignment 11.1**

For the first Lab Assignment 11.1, an algorithm was created for converting G-sensor readings of the X-axis in the range of [-1000 to 1000] mg to the corresponding rotational position in the range of [0 to 180] degrees. Through positive-equivalent-range translation (i.e, '+1000') and algebraic manipulation of the ratio between the span of each range (i.e,  $\ast(180/2000)$ ), this conversion could ultimately be achieved with:  $\emptyset(\text{degrees}) = (\text{coordX}(\text{mg}) + 1000) \ast 0.09$ . This algorithm would later be applied in the 'angle\_calculator' BDF schematic in the Lab Assignment 11.5 Quartus hardware setup.

## Lab Assignment 11.2

For Lab Assignment 11.2, a C++ program was to be developed providing a base class (with convention 'DE1SoChps') for initializing memory and controlling register access, paving the way for the subsequent implementation of the 'GSensor' class. In the 'DE1SoChps.h' header file (Appendix[1]), a class named DE1SoChps would be declared to encapsulate the functionality required for memory-mapped I/O operations. As per the instructions, it included a constructor to initialize memory mapping, a destructor to finalize memory mapping, a 'RegisterWrite' function to write a value into a register given its offset, and a 'RegisterRead' function to return the value read from a register given its offset. These functions would mirror the functionality of the DE1SoCfpga class from previous labs, performing register read and write operations using volatile pointers to memory addresses calculated from the base address and register offsets. The constructor and destructor functions were implemented in the DE1SoChps.cpp file (Appendix[2]). The constructor initialized memory-mapped I/O by opening /dev/mem and mapping physical addresses to virtual addresses using mmap. Similarly, the destructor finalized memory-mapped I/O by unmapping memory and closing the file descriptor. Replacing the register offsets in the header file were the HPS device offsets, I2C0 register offsets, and Pin Multiplexer register offsets as obtained in the Pre-Lab steps. The mmap and munmap functions in the constructor and destructor also were modified to use HPS\_BRIDGE\_BASE address and HPS\_BRIDGE\_SPAN (instead of the ones used by the DE1SoCfpga class). The definitions for the 'RegisterRead' and 'RegisterWrite' functions as given in Appendix[2] are pictured in Fig1.

**Fig1: Pictured below are the definitions for the 'RegisterRead' and 'RegisterWrite' functions as (given in Appendix[2]).**

```
28 void DE1SoChps::RegisterWrite(unsigned int reg_offset, int value) {
29     |(volatile unsigned int *)(pBase + reg_offset) = value;
30 }
31
32 int DE1SoChps::RegisterRead(unsigned int reg_offset) {
33     |return |(volatile unsigned int *)(pBase + reg_offset);
34 }
35
```

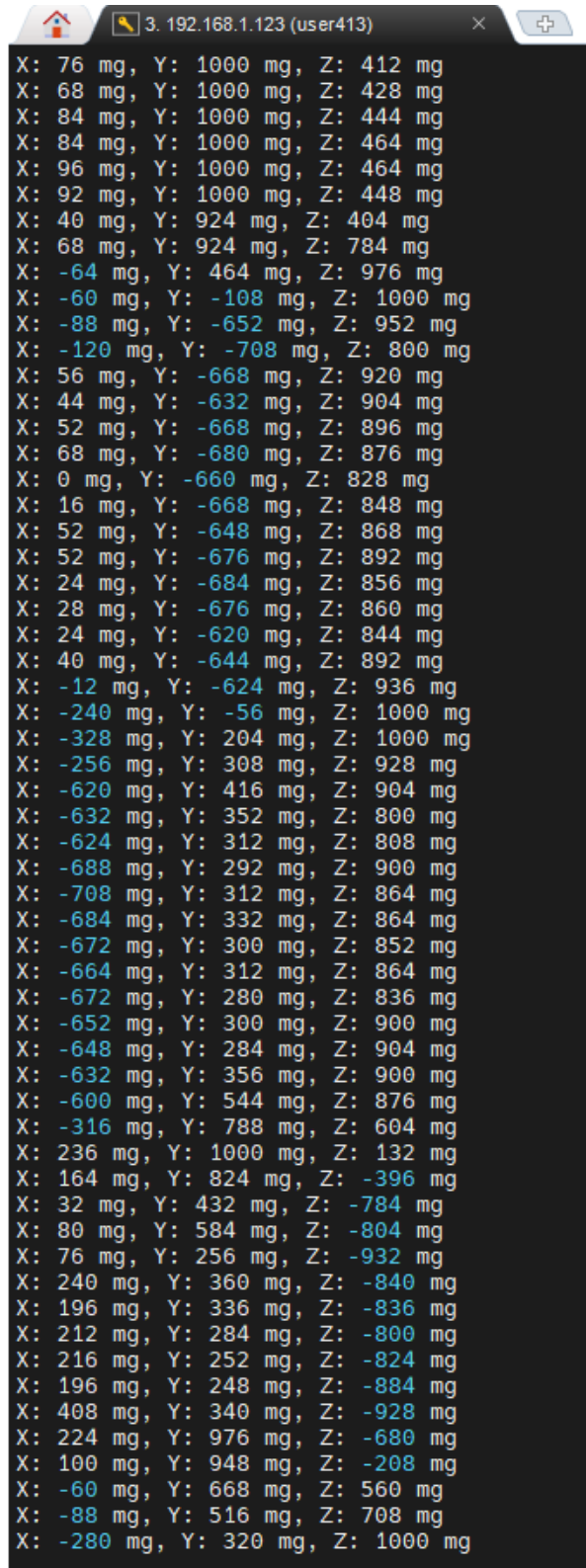
### **Lab Assignment 11.3**

For Lab Assignment 11.3, a new class named 'GSensor' (declared and defined in a respective header and source file) would be created and used for setting up the G-Sensor module for reading data. The header file 'GSensor.h' (Appendix[7]) was given in the instructions, and the class itself inherited from the 'DE1SoChps' class and 'ADXL345.h', which included functions to configure the Pin Mux and I2C0 for communication with the ADXL345. Additionally, it included functions to initialize the ADXL345 chip, read and write registers, read acceleration data, and check for new data. This header file provided the interface for interacting with the G-sensor module. The 'ADXL345.h' header file was included in the instructions, and 'GSensor.h' would inherit all relevant configurations and register addresses listed in the *ADXL345 Datasheet* (Analog.com[8]).

As for the 'GSensor.cpp' (Appendix[8]) member function definitions, the attached documentation *Using the Accelerometer on DE-SoC Boards* (Intel[9], 11-15) provided templates (written in C) for the definitions for all 'GSensor' functions. Setting and assigning target addresses, unlike in the documentation, would be done through the 'RegisterRead' and 'RegisterWrite' functions, thus necessitating the inclusion of 'DE1SoChps.h' in 'GSensor.h'. These functions implemented the configuration of Pin Mux and I2C0, initialization of the ADXL345 chip, and reading/writing of registers. This file contained the logic required to set up the G-sensor module for data acquisition.

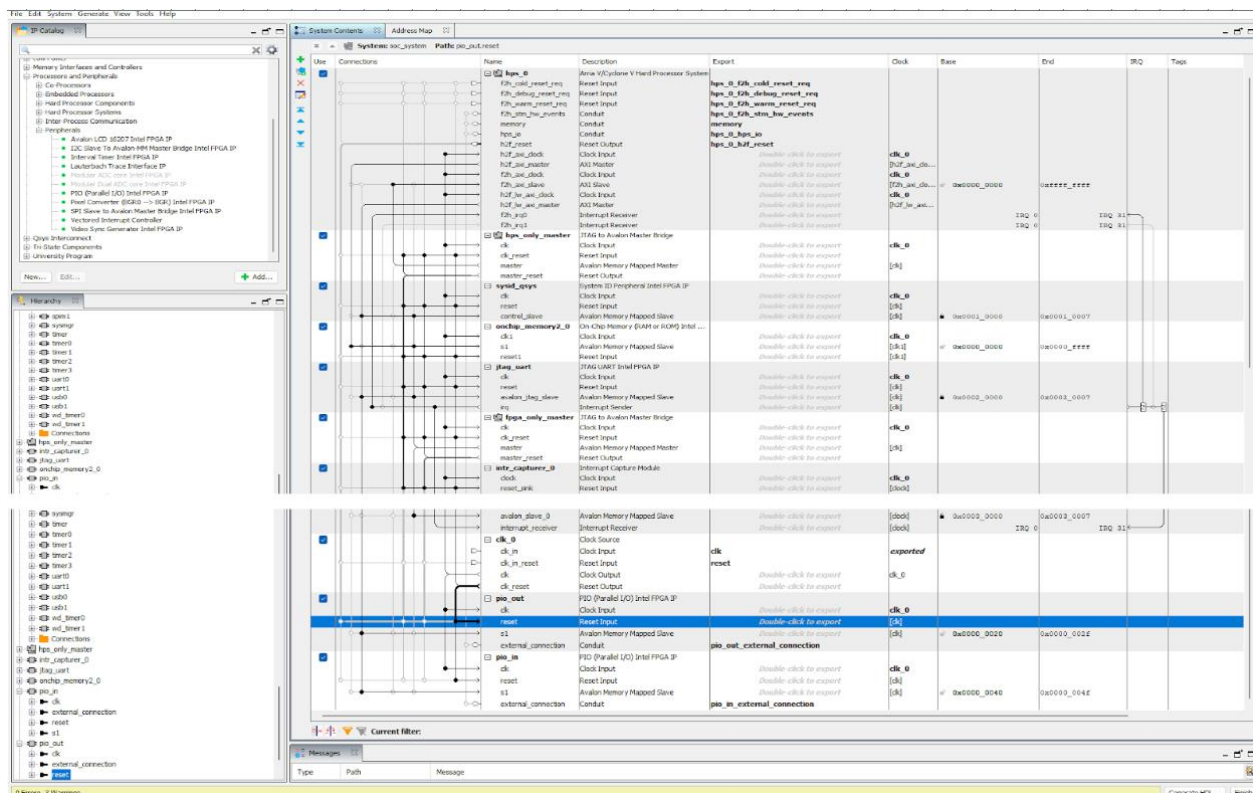
Where 'GSensor.h' and 'GSensor.cpp', were to implement a G-sensor module for reading data from the ADXL345 accelerometer (via the I2C0 line on the DE1-SoC board), the main function 'GSensorMain.cpp' (Appendix[9]) was to utilize the module, retrieving and printing live G-Sensor readings in the appropriate range. GSensorMain.cpp contained the main function where the functionality of the G-sensor module would be tested. It instantiated an object of the 'GSensor' class using dynamic memory allocation, read the ADXL345 device ID to verify correct initialization, initialized the ADXL345 module, and entered an infinite loop to continuously read and print the X, Y, Z accelerometer data. Additionally, it would control the range of the accelerometer values and interfaces with conditional statements. This file served as the entry point for testing the G-sensor. Running compiling and running this program yielded expected results, as live G-Sensor values were printed as the DE1-SoC was turned/rotated as seen in Fig2.

**Fig2: Pictured below is the Lab 11.3 program (with its newly developed main function) returning live expected G-Sensor Readings to the Linux Terminal.**



The image shows a terminal window titled "3. 192.168.1.123 (user413)". The terminal displays a continuous stream of G-sensor readings in the format "X: [value] mg, Y: [value] mg, Z: [value] mg". The readings are color-coded: X values are blue, Y values are green, and Z values are red. The data shows a sequence of readings that generally decrease in X and Y values while Z values remain relatively stable or slightly increase towards the end of the sequence.

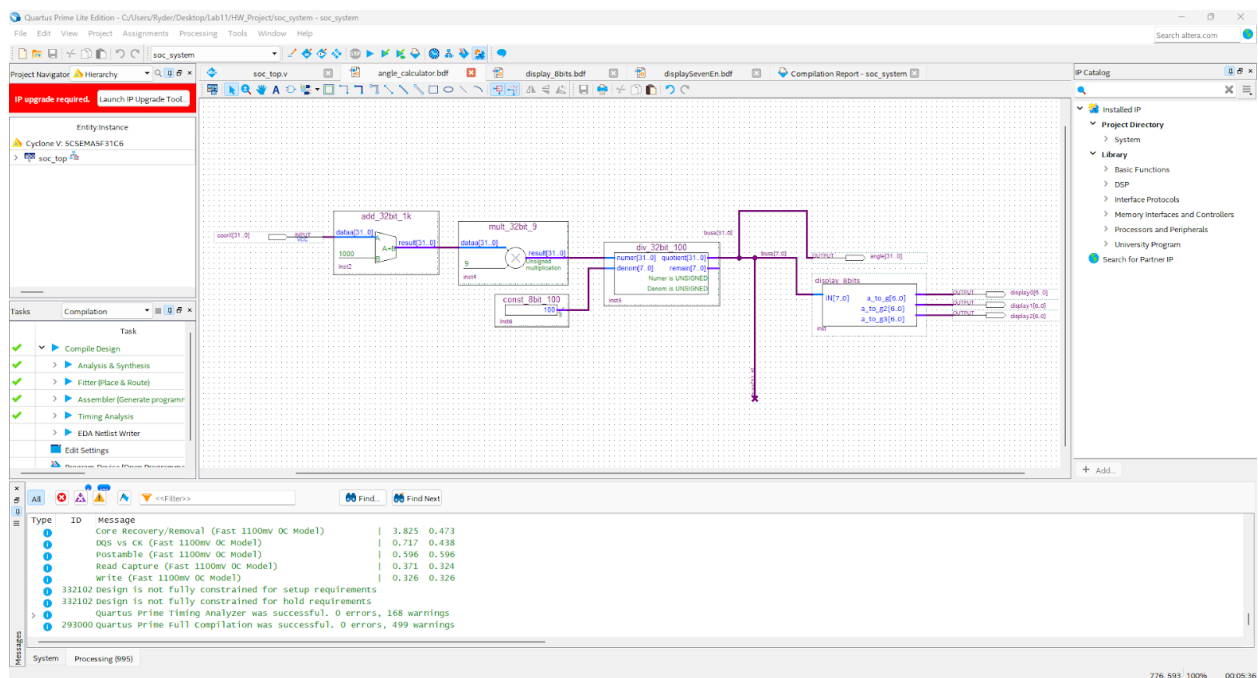
```
X: 76 mg, Y: 1000 mg, Z: 412 mg
X: 68 mg, Y: 1000 mg, Z: 428 mg
X: 84 mg, Y: 1000 mg, Z: 444 mg
X: 84 mg, Y: 1000 mg, Z: 464 mg
X: 96 mg, Y: 1000 mg, Z: 464 mg
X: 92 mg, Y: 1000 mg, Z: 448 mg
X: 40 mg, Y: 924 mg, Z: 404 mg
X: 68 mg, Y: 924 mg, Z: 784 mg
X: -64 mg, Y: 464 mg, Z: 976 mg
X: -60 mg, Y: -108 mg, Z: 1000 mg
X: -88 mg, Y: -652 mg, Z: 952 mg
X: -120 mg, Y: -708 mg, Z: 800 mg
X: 56 mg, Y: -668 mg, Z: 920 mg
X: 44 mg, Y: -632 mg, Z: 904 mg
X: 52 mg, Y: -668 mg, Z: 896 mg
X: 68 mg, Y: -680 mg, Z: 876 mg
X: 0 mg, Y: -660 mg, Z: 828 mg
X: 16 mg, Y: -668 mg, Z: 848 mg
X: 52 mg, Y: -648 mg, Z: 868 mg
X: 52 mg, Y: -676 mg, Z: 892 mg
X: 24 mg, Y: -684 mg, Z: 856 mg
X: 28 mg, Y: -676 mg, Z: 860 mg
X: 24 mg, Y: -620 mg, Z: 844 mg
X: 40 mg, Y: -644 mg, Z: 892 mg
X: -12 mg, Y: -624 mg, Z: 936 mg
X: -240 mg, Y: -56 mg, Z: 1000 mg
X: -328 mg, Y: 204 mg, Z: 1000 mg
X: -256 mg, Y: 308 mg, Z: 928 mg
X: -620 mg, Y: 416 mg, Z: 904 mg
X: -632 mg, Y: 352 mg, Z: 800 mg
X: -624 mg, Y: 312 mg, Z: 808 mg
X: -688 mg, Y: 292 mg, Z: 900 mg
X: -708 mg, Y: 312 mg, Z: 864 mg
X: -684 mg, Y: 332 mg, Z: 864 mg
X: -672 mg, Y: 300 mg, Z: 852 mg
X: -664 mg, Y: 312 mg, Z: 864 mg
X: -672 mg, Y: 280 mg, Z: 836 mg
X: -652 mg, Y: 300 mg, Z: 900 mg
X: -648 mg, Y: 284 mg, Z: 904 mg
X: -632 mg, Y: 356 mg, Z: 900 mg
X: -600 mg, Y: 544 mg, Z: 876 mg
X: -316 mg, Y: 788 mg, Z: 604 mg
X: 236 mg, Y: 1000 mg, Z: 132 mg
X: 164 mg, Y: 824 mg, Z: -396 mg
X: 32 mg, Y: 432 mg, Z: -784 mg
X: 80 mg, Y: 584 mg, Z: -804 mg
X: 76 mg, Y: 256 mg, Z: -932 mg
X: 240 mg, Y: 360 mg, Z: -840 mg
X: 196 mg, Y: 336 mg, Z: -836 mg
X: 212 mg, Y: 284 mg, Z: -800 mg
X: 216 mg, Y: 252 mg, Z: -824 mg
X: 196 mg, Y: 248 mg, Z: -884 mg
X: 408 mg, Y: 340 mg, Z: -928 mg
X: 224 mg, Y: 976 mg, Z: -680 mg
X: 100 mg, Y: 948 mg, Z: -208 mg
X: -60 mg, Y: 668 mg, Z: 560 mg
X: -88 mg, Y: 516 mg, Z: 708 mg
X: -280 mg, Y: 320 mg, Z: 1000 mg
```



## Lab Assignment 11.5

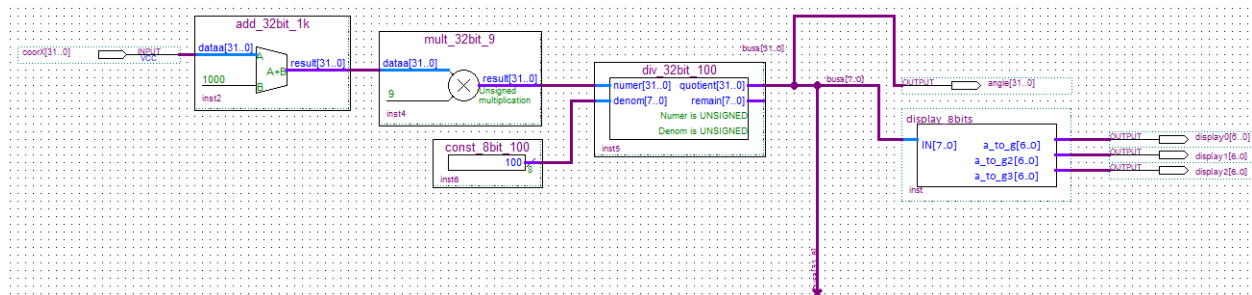
For Lab Assignment 11.5, a new Quartus schematic would be created, named 'angle\_calculator', (as a continuation of the hardware setup) aimed at converting an X coordinate from the HPS into an angle representing the inclination of the board. The schematic was designed with specific input and output pins: a 32-bit input pin named 'coorX' to receive the X coordinate from the HPS, and several output pins named 'angle[31..0]', 'display0[6..0]', 'display1[6..0]', and 'display2[6..0]' for conveying the calculated angle and displaying it on 7-segment displays HEX0, HEX1, and HEX2. The schematic incorporated necessary Arithmetic LPM IP blocks, including Adder, Multiplier, and Divider, to implement the algorithm for angle calculation:  $\theta(\text{degrees}) = (\text{coorX}(\text{mg}) + 1000) * 0.09$ . By connecting the appropriate input and output pins of these IP blocks according to the algorithm derived in the Pre-Lab, the angle calculation logic was established within the schematic. Additionally, the 'display\_8bits' block created in previous labs was utilized to display the calculated angle on the 7-segment displays. Finally, the schematic was saved and added to the project, ready for compilation. The successful compilation of the BDF is given in Fig4, and the zoomed-in schematic is given in Fig5.

**Fig4: Pictured below is the successfully compiled schematic, replicating the Pre-Lab algorithm and fulfilling Lab Assignment 11.5.**





**Fig5: Pictured below is the successfully compiled (zoomed-in) schematic, replicating the Pre-Lab algorithm and fulfilling Lab Assignment 11.5.**



### Lab Assignment 11.6

For Lab Assignment 11.6, the 'DE1SoCfpga' and 'LEDControl' classes would be adapted by means of preparing the software program for hardware integration with the PIO components. The DE1SoCfpga class, implemented in DE1SoCfpga.h (Appendix[3]) and DE1SoCfpga.cpp (Appendix[4]), included necessary functions for memory-mapped I/O, such as the constructor initializing memory and the destructor finalizing memory, along with migrated functions for 'RegisterWrite' and 'RegisterRead' to write a value into a register and read a value from a register, respectively. Additionally, the 'DE1SoCfpga' class would be updated with register offsets for the PIO components: OUT\_BASE for PIO output and IN\_BASE for PIO input, in accordance with the *Lab 11 User Manual* (Kimani[4], 11-12). The 'PIOControl' class, (created out of the previous LEDControl class from Lab 10, and) implemented in PIOControl.h (Appendix[5]) and PIOControl.cpp (Appendix[6]), included private data members 'out\_regValue' and 'in\_regValue' to represent the state of the output/input PIO registers, along with functions WritePIOout and ReadPIOin to write a value to the output PIO register and read a value from the input PIO register, respectively. The necessary modifications from the 'LEDControl' class were made to accommodate these changes, including updating function names and member variables. All required files, including 'PIOControl.h', PIOControl.cpp, DE1SoCfpga.h, and DE1SoCfpga.cpp, had been added to the 'SW\_project' folder as specified in the instructions, completing the preparation of the software program for hardware integration. The updates made to 'GsensorMain.cpp' as part of this assignment are given in Fig6.

**Fig6: Pictured below are the sole updates made to ‘GsensorMain.cpp’ as part of Lab Assignment 11.6.**

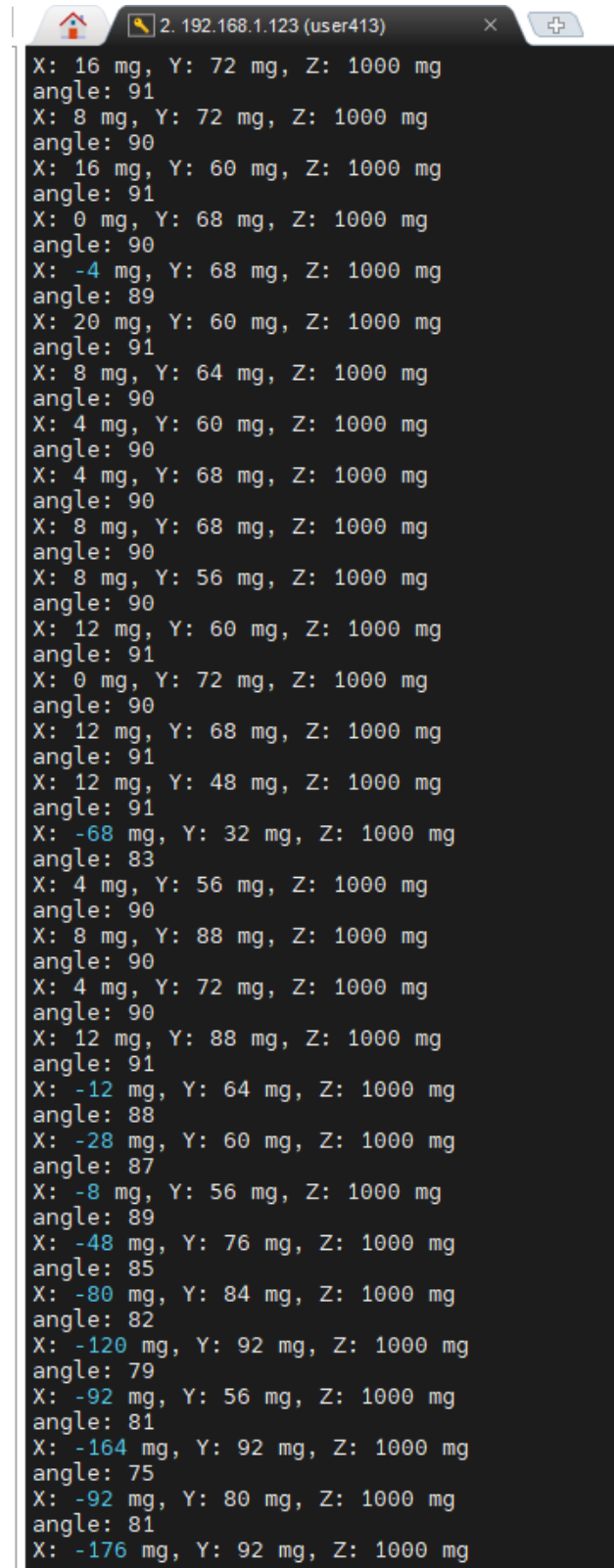
```
67      // Store X coordinate in PIO output register
68      piocontrol->WritePIOout(x_mg);
69
70      // Read angle value from PIO input register
71      int angle = piocontrol->ReadPIOin();
72
73      // Print angle
74      std::cout << "angle: " << angle << std::endl;
75
76      // Wait for a short duration before polling again
77      usleep(100000); // Sleep for 100 milliseconds (adjust as needed)
78
```

### **Lab Assignment 11.7**

For Lab Assignment 11.7, the updated ‘GSensorMain.cpp’ integrated the hardware design with the software program. First, the necessary headers were included, fulfilling the requirement to add new headers. Then, objects of the ‘GSensor’ and ‘PIOControl’ classes were instantiated using dynamic memory allocation as specified. The X coordinate obtained from the G-sensor would be stored in the PIO output register using the ‘WritePIOout’ function of the ‘PIOControl’ class, fulfilling the requirement to store the coordinate X in the PIO output register. Subsequently, the angle value representing the inclination of the board was read from the PIO input register using the ‘ReadPIOin’ function of the ‘PIOControl’ class. Finally, the angle value would be printed in the terminal, as required. Additionally, the main function included an infinite loop to continuously read and print X, Y, Z data from the G-sensor, ensuring that the program would continually update the PIO registers with the latest X coordinate and displayed the corresponding angle value. The modifications to the main function enabled the seamless integration of the hardware and software components, completing the software-hardware integration process.

After the Lab Assignment 11.5 Schematic was compiled, it would be uploaded to the Processor. Then, after compiling the program with the updated ‘GsensorMain.cpp’ using the Makefile (Appendix[10]), the program would be run. As the board was tilted and rotated, as seen in the attached video, ‘Assign7.mov’, the Program would retrieve and print X, Y and Z mg values, in addition to the ‘angle’ value, cooresponding to the X coordinate and calculated through the ‘angle\_calculator’ hardware setup. The output terminal is given in Fig7.

**Fig7:** Pictured below is a portion of the 'Assign7.mov' output terminal, displaying all correct and expected values.



```
2. 192.168.1.123 (user413) X: 16 mg, Y: 72 mg, Z: 1000 mg
angle: 91
X: 8 mg, Y: 72 mg, Z: 1000 mg
angle: 90
X: 16 mg, Y: 60 mg, Z: 1000 mg
angle: 91
X: 0 mg, Y: 68 mg, Z: 1000 mg
angle: 90
X: -4 mg, Y: 68 mg, Z: 1000 mg
angle: 89
X: 20 mg, Y: 60 mg, Z: 1000 mg
angle: 91
X: 8 mg, Y: 64 mg, Z: 1000 mg
angle: 90
X: 4 mg, Y: 60 mg, Z: 1000 mg
angle: 90
X: 4 mg, Y: 68 mg, Z: 1000 mg
angle: 90
X: 8 mg, Y: 68 mg, Z: 1000 mg
angle: 90
X: 8 mg, Y: 56 mg, Z: 1000 mg
angle: 90
X: 12 mg, Y: 60 mg, Z: 1000 mg
angle: 91
X: 0 mg, Y: 72 mg, Z: 1000 mg
angle: 90
X: 12 mg, Y: 68 mg, Z: 1000 mg
angle: 91
X: 12 mg, Y: 48 mg, Z: 1000 mg
angle: 91
X: -68 mg, Y: 32 mg, Z: 1000 mg
angle: 83
X: 4 mg, Y: 56 mg, Z: 1000 mg
angle: 90
X: 8 mg, Y: 88 mg, Z: 1000 mg
angle: 90
X: 4 mg, Y: 72 mg, Z: 1000 mg
angle: 90
X: 12 mg, Y: 88 mg, Z: 1000 mg
angle: 91
X: -12 mg, Y: 64 mg, Z: 1000 mg
angle: 88
X: -28 mg, Y: 60 mg, Z: 1000 mg
angle: 87
X: -8 mg, Y: 56 mg, Z: 1000 mg
angle: 89
X: -48 mg, Y: 76 mg, Z: 1000 mg
angle: 85
X: -80 mg, Y: 84 mg, Z: 1000 mg
angle: 82
X: -120 mg, Y: 92 mg, Z: 1000 mg
angle: 79
X: -92 mg, Y: 56 mg, Z: 1000 mg
angle: 81
X: -164 mg, Y: 92 mg, Z: 1000 mg
angle: 75
X: -92 mg, Y: 80 mg, Z: 1000 mg
angle: 81
X: -176 mg, Y: 92 mg, Z: 1000 mg
```

## Results

All results came in from the main Linux command window in MobaXterm as well as a secure C++ shell (SSH) and the Quartus Prime Software. All code would be read, interpreted and run through the DE1-SoC board.

Reiterated results from the MobaXterm Linux command window and Quartus Compilation report are pictured below:

**Fig1: Pictured below are the definitions for the ‘RegisterRead’ and ‘RegisterWrite’ functions as (given in Appendix[2]) .**

```
28 void DE1SoChps::RegisterWrite(unsigned int reg_offset, int value) {
29     *(volatile unsigned int *)(pBase + reg_offset) = value;
30 }
31
32 int DE1SoChps::RegisterRead(unsigned int reg_offset) {
33     return *(volatile unsigned int *)(pBase + reg_offset);
34 }
35
```

**Fig2: Pictured below is the Lab 11.3 program (with its newly developed main function) returning live expected G-Sensor Readings to the Linux Terminal.**

```
3. 192.168.1.123 (user413) X +
X: 76 mg, Y: 1000 mg, Z: 412 mg
X: 68 mg, Y: 1000 mg, Z: 428 mg
X: 84 mg, Y: 1000 mg, Z: 444 mg
X: 84 mg, Y: 1000 mg, Z: 464 mg
X: 96 mg, Y: 1000 mg, Z: 464 mg
X: 92 mg, Y: 1000 mg, Z: 448 mg
X: 40 mg, Y: 924 mg, Z: 404 mg
X: 68 mg, Y: 924 mg, Z: 784 mg
X: -64 mg, Y: 464 mg, Z: 976 mg
X: -60 mg, Y: -108 mg, Z: 1000 mg
X: -88 mg, Y: -652 mg, Z: 952 mg
X: -120 mg, Y: -708 mg, Z: 800 mg
X: 56 mg, Y: -668 mg, Z: 920 mg
X: 44 mg, Y: -632 mg, Z: 904 mg
X: 52 mg, Y: -668 mg, Z: 896 mg
X: 68 mg, Y: -680 mg, Z: 876 mg
X: 0 mg, Y: -660 mg, Z: 828 mg
X: 16 mg, Y: -668 mg, Z: 848 mg
X: 52 mg, Y: -648 mg, Z: 868 mg
X: 52 mg, Y: -676 mg, Z: 892 mg
X: 24 mg, Y: -684 mg, Z: 856 mg
X: 28 mg, Y: -676 mg, Z: 860 mg
X: 24 mg, Y: -620 mg, Z: 844 mg
X: 40 mg, Y: -644 mg, Z: 892 mg
X: -12 mg, Y: -624 mg, Z: 936 mg
X: -240 mg, Y: -56 mg, Z: 1000 mg
X: -328 mg, Y: 204 mg, Z: 1000 mg
X: -256 mg, Y: 308 mg, Z: 928 mg
X: -620 mg, Y: 416 mg, Z: 904 mg
X: -632 mg, Y: 352 mg, Z: 800 mg
X: -624 mg, Y: 312 mg, Z: 808 mg
X: -688 mg, Y: 292 mg, Z: 900 mg
X: -708 mg, Y: 312 mg, Z: 864 mg
X: -684 mg, Y: 332 mg, Z: 864 mg
X: -672 mg, Y: 300 mg, Z: 852 mg
X: -664 mg, Y: 312 mg, Z: 864 mg
X: -672 mg, Y: 280 mg, Z: 836 mg
X: -652 mg, Y: 300 mg, Z: 900 mg
X: -648 mg, Y: 284 mg, Z: 904 mg
X: -632 mg, Y: 356 mg, Z: 900 mg
X: -600 mg, Y: 544 mg, Z: 876 mg
X: -316 mg, Y: 788 mg, Z: 604 mg
X: 236 mg, Y: 1000 mg, Z: 132 mg
X: 164 mg, Y: 824 mg, Z: -396 mg
X: 32 mg, Y: 432 mg, Z: -784 mg
X: 80 mg, Y: 584 mg, Z: -804 mg
X: 76 mg, Y: 256 mg, Z: -932 mg
X: 240 mg, Y: 360 mg, Z: -840 mg
X: 196 mg, Y: 336 mg, Z: -836 mg
X: 212 mg, Y: 284 mg, Z: -800 mg
X: 216 mg, Y: 252 mg, Z: -824 mg
X: 196 mg, Y: 248 mg, Z: -884 mg
X: 408 mg, Y: 340 mg, Z: -928 mg
X: 224 mg, Y: 976 mg, Z: -680 mg
X: 100 mg, Y: 948 mg, Z: -208 mg
X: -60 mg, Y: 668 mg, Z: 560 mg
X: -88 mg, Y: 516 mg, Z: 708 mg
X: -280 mg, Y: 320 mg, Z: 1000 mg
```

IP Catalog
System Catalog
Address Map

**IP Catalog**

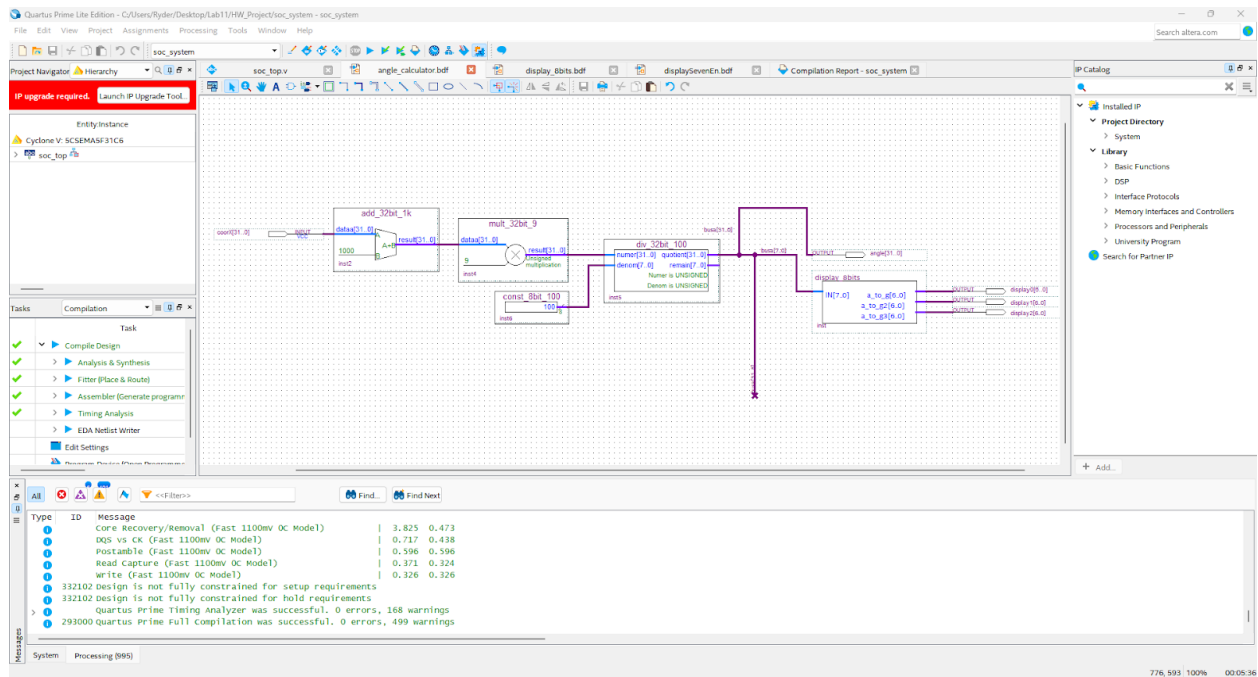
- Hardware
  - Processors and Peripherals
  - CPU Processors
  - Unembedded Processors
  - Hard Processor Components
  - Hard Processor Systems
  - Inter-Process Communication
  - Interfacing
    - Avnet iC2000 Dual FPGA IP
    - DC Slave To Avalon-MM Master Bridge Intel FPGA IP
    - Internal Timer Intel FPGA IP
    - Layer2/3 Traffic Interface IP
    - Modbus ASCII over SPI Intel FPGA IP
    - Modbus RTU over SPI Intel FPGA IP
    - PSD (Parallel) I/O Intel FPGA IP
    - Reset Controller (RSTC) Intel FPGA IP
    - SPI Slave to Avalon-MM Master Bridge Intel FPGA IP
    - Universal Interrupt Controller
    - Value Sync Generator Intel FPGA IP
- QoS Interconnect
- For SoC Components
- University Program

New... Edit...

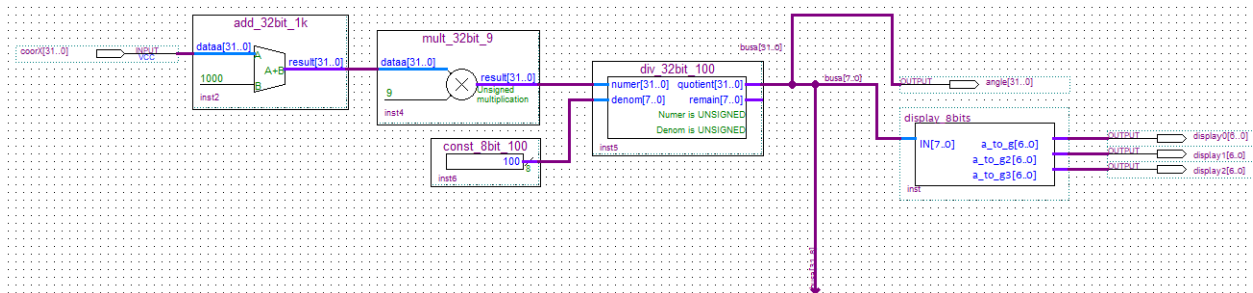
**Search**

- sym0
- sym1
- sym2
- sym3
- sym4
- sym5
- sym6
- sym7
- sym8
- sym9
- sym10
- sym11
- sym12
- sym13
- sym14
- sym15
- sym16
- sym17
- sym18
- sym19
- sym20
- sym21
- sym22
- sym23
- sym24
- sym25
- sym26
- sym27
- sym28
- sym29
- sym30
- sym31
- sym32
- sym33
- sym34
- sym35
- sym36
- sym37
- sym38
- sym39
- sym40
- sym41
- sym42
- sym43
- sym44
- sym45
- sym46
- sym47
- sym48
- sym49
- sym50
- sym51
- sym52
- sym53
- sym54
- sym55
- sym56
- sym57
- sym58
- sym59
- sym60
- sym61
- sym62
- sym63
- sym64
- sym65
- sym66
- sym67
- sym68
- sym69
- sym70
- sym71
- sym72
- sym73
- sym74
- sym75
- sym76
- sym77
- sym78
- sym79
- sym80
- sym81
- sym82
- sym83
- sym84
- sym85
- sym86
- sym87
- sym88
- sym89
- sym90
- sym91
- sym92
- sym93
- sym94
- sym95
- sym96
- sym97
- sym98
- sym99
- sym100
- sym101
- sym102
- sym103
- sym104
- sym105
- sym106
- sym107
- sym108
- sym109
- sym110
- sym111
- sym112
- sym113
- sym114
- sym115
- sym116
- sym117
- sym118
- sym119
- sym120
- sym121
- sym122
- sym123
- sym124
- sym125
- sym126
- sym127
- sym128
- sym129
- sym130
- sym131
- sym132
- sym133
- sym134
- sym135
- sym136
- sym137
- sym138
- sym139
- sym140
- sym141
- sym142
- sym143
- sym144
- sym145
- sym146
- sym147
- sym148
- sym149
- sym150
- sym151
- sym152
- sym153
- sym154
- sym155
- sym156
- sym157
- sym158
- sym159
- sym160
- sym161
- sym162
- sym163
- sym164
- sym165
- sym166
- sym167
- sym168
- sym169
- sym170
- sym171
- sym172
- sym173
- sym174
- sym175
- sym176
- sym177
- sym178
- sym179
- sym180
- sym181
- sym182
- sym183
- sym184
- sym185
- sym186
- sym187
- sym188
- sym189
- sym190
- sym191
- sym192
- sym193
- sym194
- sym195
- sym196
- sym197
- sym198
- sym199
- sym200
- sym201
- sym202
- sym203
- sym204
- sym205
- sym206
- sym207
- sym208
- sym209
- sym210
- sym211
- sym212
- sym213
- sym214
- sym215
- sym216
- sym217
- sym218
- sym219
- sym220
- sym221
- sym222
- sym223
- sym224
- sym225
- sym226
- sym227
- sym228
- sym229
- sym230
- sym231
- sym232
- sym233
- sym234
- sym235
- sym236
- sym237
- sym238
- sym239
- sym240
- sym241
- sym242
- sym243
- sym244
- sym245
- sym246
- sym247
- sym248
- sym249
- sym250
- sym251
- sym252
- sym253
- sym254
- sym255
- sym256
- sym257
- sym258
- sym259
- sym260
- sym261
- sym262
- sym263
- sym264
- sym265
- sym266
- sym267
- sym268
- sym269
- sym270
- sym271
- sym272
- sym273
- sym274
- sym275
- sym276
- sym277
- sym278
- sym279
- sym280
- sym281
- sym282
- sym283
- sym284
- sym285
- sym286
- sym287
- sym288
- sym289
- sym290
- sym291
- sym292
- sym293
- sym294
- sym295
- sym296
- sym297
- sym298
- sym299
- sym300
- sym301
- sym302
- sym303
- sym304
- sym305
- sym306
- sym307
- sym308
- sym309
- sym310
- sym311
- sym312
- sym313
- sym314
- sym315
- sym316
- sym317
- sym318
- sym319
- sym320
- sym321
- sym322
- sym323
- sym324
- sym325
- sym326
- sym327
- sym328
- sym329
- sym330
- sym331
- sym332
- sym333
- sym334
- sym335
- sym336
- sym337
- sym338
- sym339
- sym340
- sym341
- sym342
- sym343
- sym344
- sym345
- sym346
- sym347
- sym348
- sym349
- sym350
- sym351
- sym352
- sym353
- sym354
- sym355
- sym356
- sym357
- sym358
- sym359
- sym360
- sym361
- sym362
- sym363
- sym364
- sym365
- sym366
- sym367
- sym368
- sym369
- sym370
- sym371
- sym372
- sym373
- sym374
- sym375
- sym376
- sym377
- sym378
- sym379

**Fig4:** Pictured below is the successfully compiled schematic, replicating the Pre-Lab algorithm and fulfilling Lab Assignment 11.5.



**Fig5:** Pictured below is the successfully compiled (zoomed-in) schematic, replicating the Pre-Lab algorithm and fulfilling Lab Assignment 11.5.

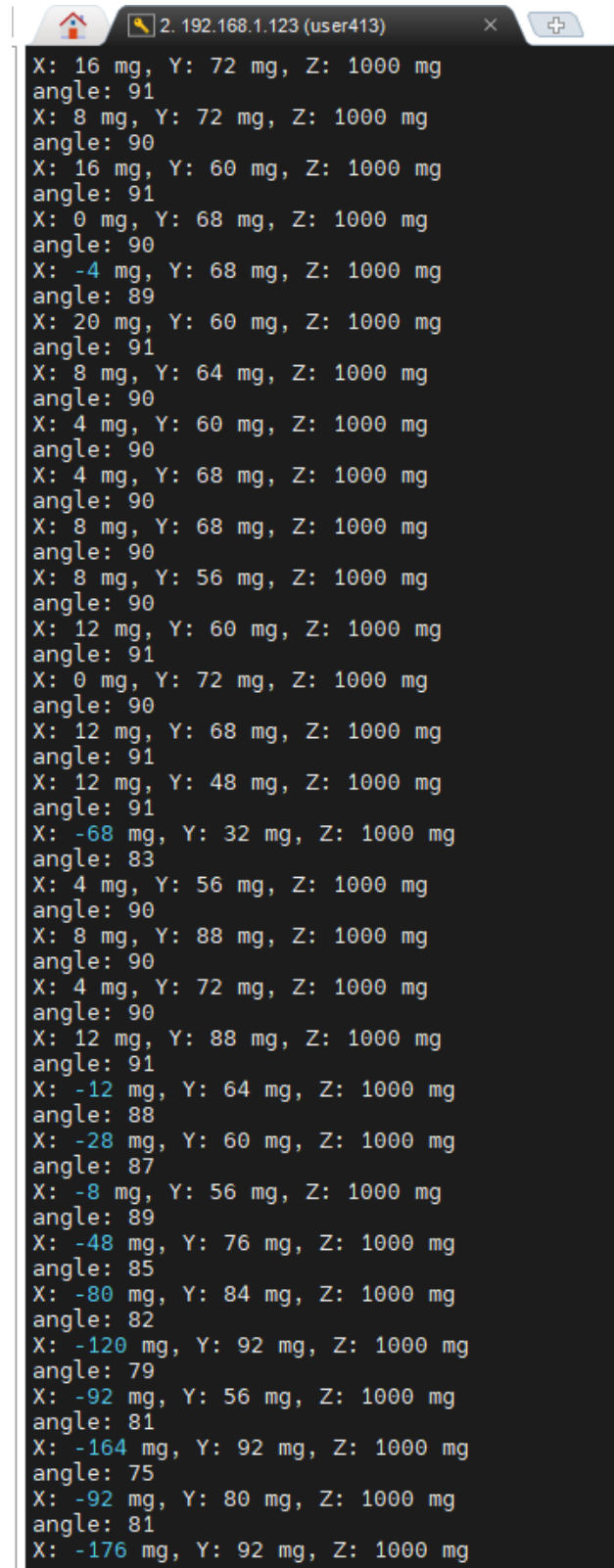


**Fig6: Pictured below are the sole updates made to 'GsensorMain.cpp' as part of Lab Assignment 11.6.**

```
67      // Store X coordinate in PIO output register
68      piocontrol->WritePIOout(x_mg);
69
70      // Read angle value from PIO input register
71      int angle = piocontrol->ReadPIOin();
72
73      // Print angle
74      std::cout << "angle: " << angle << std::endl;
75
76      // Wait for a short duration before polling again
77      usleep(100000); // Sleep for 100 milliseconds (adjust as needed)
78  }
```



**Fig7:** Pictured below is a portion of the 'Assign7.mov' output terminal, displaying all correct and expected values.



```
2. 192.168.1.123 (user413) X: 16 mg, Y: 72 mg, Z: 1000 mg
angle: 91
X: 8 mg, Y: 72 mg, Z: 1000 mg
angle: 90
X: 16 mg, Y: 60 mg, Z: 1000 mg
angle: 91
X: 0 mg, Y: 68 mg, Z: 1000 mg
angle: 90
X: -4 mg, Y: 68 mg, Z: 1000 mg
angle: 89
X: 20 mg, Y: 60 mg, Z: 1000 mg
angle: 91
X: 8 mg, Y: 64 mg, Z: 1000 mg
angle: 90
X: 4 mg, Y: 60 mg, Z: 1000 mg
angle: 90
X: 4 mg, Y: 68 mg, Z: 1000 mg
angle: 90
X: 8 mg, Y: 68 mg, Z: 1000 mg
angle: 90
X: 8 mg, Y: 56 mg, Z: 1000 mg
angle: 90
X: 12 mg, Y: 60 mg, Z: 1000 mg
angle: 91
X: 0 mg, Y: 72 mg, Z: 1000 mg
angle: 90
X: 12 mg, Y: 68 mg, Z: 1000 mg
angle: 91
X: 12 mg, Y: 48 mg, Z: 1000 mg
angle: 91
X: -68 mg, Y: 32 mg, Z: 1000 mg
angle: 83
X: 4 mg, Y: 56 mg, Z: 1000 mg
angle: 90
X: 8 mg, Y: 88 mg, Z: 1000 mg
angle: 90
X: 4 mg, Y: 72 mg, Z: 1000 mg
angle: 90
X: 12 mg, Y: 88 mg, Z: 1000 mg
angle: 91
X: -12 mg, Y: 64 mg, Z: 1000 mg
angle: 88
X: -28 mg, Y: 60 mg, Z: 1000 mg
angle: 87
X: -8 mg, Y: 56 mg, Z: 1000 mg
angle: 89
X: -48 mg, Y: 76 mg, Z: 1000 mg
angle: 85
X: -80 mg, Y: 84 mg, Z: 1000 mg
angle: 82
X: -120 mg, Y: 92 mg, Z: 1000 mg
angle: 79
X: -92 mg, Y: 56 mg, Z: 1000 mg
angle: 81
X: -164 mg, Y: 92 mg, Z: 1000 mg
angle: 75
X: -92 mg, Y: 80 mg, Z: 1000 mg
angle: 81
X: -176 mg, Y: 92 mg, Z: 1000 mg
```

## **Conclusion**

In conclusion, Lab 11 marked the culmination of efforts to integrate software and hardware components into a cohesive system. Through a series of assignments, ranging from hand-written algorithm Pre-Lab preparations to C++ software and Quartus-virtual-circuit-design hardware development, hands-on experience in designing and implementing a complex embedded system was undoubtedly acquired. The successful completion of Lab 11.7 demonstrated the seamless integration of the G-sensor hardware with the software program, achieving the main goal of the lab. With the Final Lab coming to its conclusion, Embedded Design and Enabling Robotics, overall, provided valuable insights into the intricacies of software-hardware integration, providing practical knowledge and experience with embedded systems development.

## **References**

- [1] Altera University, “DE1-SoC User Manual,” Terasic Technologies, January 26, 2019
- [2] Intel® FPGA IP, “IP Catalog Library,” Intel® Quartus®, April 17, 2019
- [3] EETech Media, LLC. “Contact ‘Bounce,’” All About Circuits, May 3, 2001
- [4] John Kimani, “Lab 11 User Manual” Northeastern University, January 31, 2024.
- [5] John Kimani, “Lab 5 User Manual” Northeastern University, January 31, 2024.
- [6] Jacob Peddicord, “Unix/Linux Command Cheat Sheet,” FOSSwire.com, August 2, 2007.
- [7] Intel®, “Cyclone V HPS Technical Ref. Manual,” Terasic Technologies, April 3, 2007
- [8] Analog Devices, “ADXL345 Datasheet,” Analog.com, September 4, 2007
- [9] Intel Corporation, “Using the Accelerometer on DE-SoC Boards,” Intel University Program, June 20, 2017

## Appendix

### 1. DE1SoChps.h:

```
2. #ifndef DE1SOCHPS_H
3. #define DE1SOCHPS_H
4. #include <iostream>
5. #include <unistd.h>
6. #include <fcntl.h>
7. #include <sys/mman.h>
8. #include <cstdlib>
9.
10. class DE1SoChps {
11. private:
12.     char *pBase;
13.     int fd;
14.
15. public:
16.     DE1SoChps();
17.     ~DE1SoChps();
18.
19.
20.     // Function to write a value into a register given its offset
21.     void RegisterWrite(unsigned int reg_offset, int value);
22.
23.     // Function to return the value read from a register given its offset
24.     int RegisterRead(unsigned int reg_offset);
25.
26.     // Cyclone V HPS Devices
27.     static const unsigned int HPS_BRIDGE_BASE = 0xFF700000;
28.     static const unsigned int HPS_BRIDGE_SPAN = 0x06FFFFFF;
29.
30.     // I2C0 Peripheral Registers
31.     static const unsigned int I2C0_BASE = 0x00504000; // I2C0 Base Address
32.     static const unsigned int I2C0_CON = 0x00504000; // Control Register
33.     static const unsigned int I2C0_TAR = I2C0_BASE + 0x04; // Target Address
34.     static const unsigned int I2C0_DATA_CMD = I2C0_BASE + 0x10; // Tx Rx Data
35.     static const unsigned int I2C0_FS_SCL_HCNT = I2C0_BASE + 0x1C; // Fast
36.     static const unsigned int I2C0_FS_SCL_LCNT = I2C0_BASE + 0x20; // Fast
37.     static const unsigned int I2C0_CLR_INTR = I2C0_BASE + 0x40; // Combined
    and Individual Interrupt Register
```

```
38.     static const unsigned int I2C0_ENABLE = I2C0_BASE + 0x6C; // Enable
      Register
39.     static const unsigned int I2C0_TXFLR = I2C0_BASE + 0x74; // Transmit FIFO
      Level Register
40.     static const unsigned int I2C0_RXFLR = I2C0_BASE + 0x78; // Receive FIFO
      Level Register
41.     static const unsigned int I2C0_ENABLE_STATUS = I2C0_BASE + 0x9C; //
      Enable Status Register
42.
43.     // The Pin Multiplexer selection
44.     static const unsigned int PIN_MUX_GEN_IO7 = 0x0060849C; // GENERALIO7 reg
      offset
45.     static const unsigned int PIN_MUX_GEN_IO8 = 0x006084A0; // GENERALIO8 reg
      offset
46.     static const unsigned int PIN_MUX_GPLMUX55 = 0x006086B0; // GPLMUX55 reg
      offset
47.     static const unsigned int PIN_MUX_GPLMUX56 = 0x006086B4; // GPLMUX56 reg
      offset
48.     static const unsigned int PIN_MUX_I2C0USEFPGA = 0x00608704; //
      I2C0USEFPGA reg offset
49.
50.     // I2C0 Peripheral Registers
51.     //static const unsigned int I2C0_BASE = 0x00504000; // I2C0 Base Address
52.     //static const unsigned int I2C0_CON = 0x00504000; // Control Register
53.     //static const unsigned int I2C0_TAR = 0x00504004; // Target Address
      Register
54.     //static const unsigned int I2C0_DATA_CMD = 0x00504010; // Tx Rx Data and
      Command Register
55.     //static const unsigned int I2C0_FS_SCL_HCNT = 0x0050401C; // Fast Spd
      Clock SCL HCNT Register
56.     //static const unsigned int I2C0_FS_SCL_LCNT = 0x00504020; // Fast Spd
      Clock SCL LCNT Register
57.     //static const unsigned int I2C0_CLR_INTR = 0x0050402C; // Combined and
      Individual Interrupt Register
58.     //static const unsigned int I2C0_ENABLE = 0x0050406C; // Enable Register
59.     //static const unsigned int I2C0_TXFLR = 0x00504078; // Transmit FIFO
      Level Register
60.     //static const unsigned int I2C0_RXFLR = 0x0050407C; // Receive FIFO
      Level Register
61.     //static const unsigned int I2C0_ENABLE_STATUS = 0x0050409C; // Enable
      Status Register
62.
63.     // The Pin Multiplexer selection
64.     //static const unsigned int PIN_MUX_GEN_IO7 = 0x005050EC; // GENERALIO7
      reg offset
```

```
65.    //static const unsigned int PIN_MUX_GEN_I08 = 0x005050F0; // GENERALI08
      reg offset
66.    //static const unsigned int PIN_MUX_GPLMUX55 = 0x0050517C; // GPLMUX55
      reg offset
67.    //static const unsigned int PIN_MUX_GPLMUX56 = 0x00505180; // GPLMUX56
      reg offset
68.    //static const unsigned int PIN_MUX_I2C0USEFPGA = 0x005051B4; //
      I2C0USEFPGA reg offset
69.
70.};
71.
72.#endif // DE1SOCHPS_H
73.
```

## 2. DE1SoChps.cpp:

```
3.  #include "DE1SoChps.h"
4.
5.  DE1SoChps::DE1SoChps() {
6.      // Open /dev/mem to give access to physical addresses
7.      fd = open("/dev/mem", (O_RDWR | O_SYNC));
8.      if (fd == -1) {
9.          std::cerr << "ERROR: could not open /dev/mem..." <<
std::endl;
10.         exit(1);
11.     }
12.
13.     // Get a mapping from physical addresses to virtual addresses
14.     pBase = (char *)mmap(NULL, HPS_BRIDGE_SPAN, (PROT_READ |
PROT_WRITE), MAP_SHARED, fd, HPS_BRIDGE_BASE);
15.     if (pBase == MAP_FAILED) {
16.         std::cerr << "ERROR: mmap() failed..." << std::endl;
17.         close(fd); // close memory before exiting
18.         exit(1);  // Returns 1 to the operating system;
19.     }
20. }
21.
22. DE1SoChps::~DE1SoChps() {
23.     if (munmap(pBase, HPS_BRIDGE_SPAN) != 0) {
24.         std::cerr << "ERROR: munmap() failed..." << std::endl;
25.         exit(1);
26.     }
27.     close(fd); // close memory
28. }
29.
30. void DE1SoChps::RegisterWrite(unsigned int reg_offset, int value) {
31.     *(volatile unsigned int *) (pBase + reg_offset) = value;
32. }
33.
34. int DE1SoChps::RegisterRead(unsigned int reg_offset) {
35.     return *(volatile unsigned int *) (pBase + reg_offset);
36. }
37.
```

## 3. DE1SoCfpga.h:

```
4.  #ifndef DE1SOCFPGA_H
5.  #define DE1SOCFPGA_H
6.
7.  #include <iostream>
8.  #include <unistd.h>
9.  #include <fcntl.h>
10. #include <sys/mman.h>
11. #include <cstdlib>
12.
13. class DE1SoCfpga {
14. private:
15.     char *pBase;
16.     int fd;
17.
18. public:
19.     DE1SoCfpga();
20.     ~DE1SoCfpga();
21.     void RegisterWrite(unsigned int reg_offset, int value);
22.     int RegisterRead(unsigned int reg_offset);
23.
24.     // Constants for DE1-SoC FPGA devices
25.     static const unsigned int OUT_BASE = 0x00000020;
26.     static const unsigned int IN_BASE = 0x00000040;
27.     static const unsigned int LW_BRIDGE_BASE = 0xFF200000;
28.     static const unsigned int LW_BRIDGE_SPAN = 0x00005000;
29.     //const unsigned int LEDR_BASE;
30.     //const unsigned int SW_BASE;
31.     //const unsigned int KEY_BASE;
32.     //const unsigned int HEX3_HEX0_BASE;
33.     //const unsigned int HEX5_HEX4_BASE;
34.     //const unsigned int JP2_BASE;
35.     //const unsigned int JP2_DDR;
36. };
37.
38. #endif // DE1SOCFPGA_H
39.
```



## 4. DE1SoCfpga.cpp:

```
5.  #include "DE1SoCfpga.h"
6.
7.  DE1SoCfpga::DE1SoCfpga() {
8.      // Open /dev/mem to give access to physical addresses
9.      fd = open("/dev/mem", (O_RDWR | O_SYNC));
10.     if (fd == -1) {
11.         std::cerr << "ERROR: could not open /dev/mem..." <<
std::endl;
12.         exit(1);
13.     }
14.
15.     // Get a mapping from physical addresses to virtual addresses
16.     pBase = (char *)mmap(NULL, LW_BRIDGE_SPAN, (PROT_READ |
PROT_WRITE), MAP_SHARED, fd, LW_BRIDGE_BASE);
17.     if (pBase == MAP_FAILED) {
18.         std::cerr << "ERROR: mmap() failed..." << std::endl;
19.         close(fd); // close memory before exiting
20.         exit(1);  // Returns 1 to the operating system;
21.     }
22. }
23.
24. DE1SoCfpga::~DE1SoCfpga() {
25.     if (munmap(pBase, LW_BRIDGE_SPAN) != 0) {
26.         std::cerr << "ERROR: munmap() failed..." << std::endl;
27.         exit(1);
28.     }
29.     close(fd); // close memory
30. }
31.
32. void DE1SoCfpga::RegisterWrite(unsigned int reg_offset, int value) {
33.     *(volatile unsigned int *)(pBase + reg_offset) = value;
34. }
35.
36. int DE1SoCfpga::RegisterRead(unsigned int reg_offset) {
37.     return *(volatile unsigned int *)(pBase + reg_offset);
38. }
39.
```

5. PIOControl.h:

```
6.  #ifndef PIOCONTROL_H
7.  #define PIOCONTROL_H
8.
9.  #include "DE1SoCfpga.h"
10.
11.  class PIOControl : public DE1SoCfpga {
12.  private:
13.      unsigned int out_regValue;
14.      unsigned int in_regValue;
15.
16.  public:
17.      PIOControl();
18.      ~PIOControl();
19.      void WritePIOout(int value);
20.      int ReadPIOin();
21.  };
22.
23.  #endif // PIOCONTROL_H
24.
```

6. PIOControl.cpp:

```
7.  #include "PIOControl.h"
8.
9.  PIOControl::PIOControl() {
10.      out_regValue = 0;
11.  }
12.
13.  PIOControl::~~PIOControl() {
14.      std::cout << "Closing PIOs..." << std::endl;
15.  }
16.
17.  void PIOControl::WritePIOout(int value) {
18.      out_regValue = value;
19.      RegisterWrite(OUT_BASE, out_regValue);
20.  }
21.
22.  int PIOControl::ReadPIOin() {
23.      return RegisterRead(IN_BASE);
24.  }
25.
```

## 7. GSensor.h:

```
8.  /**
9.   * @file    GSensor.h
10.  * @brief   Process accelerometer input from from the DE1-SoC
11.  * Reads the XYZ accelerometer data from ADXL345 via the I2C0 line
12.  */
13.
14.  #ifndef GSENSOR_H
15.  #define GSENSOR_H
16.
17.  // project includes
18.  #include <stdint.h>
19.  #include "ADXL345.h"
20.  #include "DE1SoChps.h"
21.
22.  class GSensor : public DE1SoChps
23.  {
24.  private:
25.      /* Prototypes for functions used to configure Pin Mux and I2C0
26.      */
27.      /**
28.       * Configure I2C Pin Multiplexer to use I2C for ADXL345.
29.       * - Writes the settings to the corresponding pin_mux registers
30.       */
31.      void PinmuxConfig();
32.
33.      /**
34.       * Configure I2C0 for communication with ADXL345
35.       * - Writes the settings to the corresponding I2C0 registers
36.       */
37.      void I2C0_Init();
38.  public:
39.      /**
40.       * Constructor Initializes Pin Multiplexer I2C0
41.       * by calling the two functions
42.       */
43.      GSensor();      // Constructor
44.      /**
45.       * Destructor to finalize G-Sensor module
46.       * Does nothing other than print a finalizing message
47.       */
48.      ~GSensor();     // Destructor
49.  }
```

```
50.      /* Prototypes for functions used to configure ADXL345 */
51.      /**
52.       * Initialize the ADXL345 chip
53.       *- Writes the settings to the corresponding I2C0 registers
54.       */
55.      void ADXL345_Init();
56.
57.      /**
58.       * Read value from internal register at address
59.       *
60.       * @param address  Address of register to read.
61.       * @param value     Address of value to read
62.       */
63.      void ADXL345_RegRead(uint8_t address, uint8_t *value);
64.
65.      /**
66.       * Write value to internal register at address
67.       * @param address  Address of register to write to.
68.       * @param value     Value to write to register
69.       */
70.      void ADXL345_RegWrite(uint8_t address, uint8_t value);
71.
72.      /**
73.       * Reads values from multiple internal registers
74.       *
75.       * @param address  Address of the first register to read.
76.       * @param values   Buffer to read data into
77.       * @param len      Number of registers to read from
78.       */
79.      void ADXL345_RegMultiRead(uint8_t address, uint8_t values[],
80.      uint8_t len);
81.
82.      /**
83.       * Read acceleration data of all three axes
84.       *
85.       * @param szData16  Buffer to read data into
86.       * @param mg_per_lsb  Resolution multiplier factor
87.       */
88.      void ADXL345_XYZ_Read(int16_t szData16[3], int16_t mg_per_lsb);
89.
90.      /**
91.       * Return true if there is new data
92.       *
93.       * @param None
94.       * @return None
```

```
94.      */
95.      bool ADXL345_IsDataReady();
96.  };
97.
98.  #endif
99.
```

## 8. GSensor.cpp:

```
9.  #include "GSensor.h"
10.
11.  GSensor::GSensor() {
12.      std::cout << "GSensor constructor called!" << std::endl;
13.      PinmuxConfig();
14.      I2C0_Init();
15.  }
16.
17.  GSensor::~GSensor() {
18.      std::cout << "Finalizing G-Sensor module..." << std::endl;
19.      // Any necessary cleanup code here
20.  }
21.
22.  void GSensor::PinmuxConfig() {
23.      // Configure Pin Mux to connect the ADXL345's I2C wires to I2C0
24.      // Access the necessary registers using RegisterRead and
RegisterWrite
25.
26.      // Set I2C0USEFPGA register to 0
27.      RegisterWrite(PIN_MUX_I2C0USEFPGA, 0);
28.
29.      // Set GENERALIO7 and GENERALIO8 registers to 1
30.      RegisterWrite(PIN_MUX_GEN_IO7, 1);
31.      RegisterWrite(PIN_MUX_GEN_IO8, 1);
32.  }
33.
34.  void GSensor::I2C0_Init() {
35.      // Configure I2C0 for communication with ADXL345
36.      // Write the necessary settings to the corresponding I2C0
registers
37.
38.      // Abort any ongoing transmits and disable I2C0
39.      RegisterWrite(I2C0_ENABLE, 2);
40.
41.      // Wait until I2C0 is disabled
42.      while (RegisterRead(I2C0_ENABLE_STATUS) & 0x1) {}
43.
44.      // Configure the config reg with the desired setting
45.      RegisterWrite(I2C0_CON, 0x65); // (act as a master, use 7bit
addressing, fast mode (400kb/s))
46.
47.      // Set target address (disable special commands, use 7bit
addressing)
```

```
48.     RegisterWrite(I2C0_TAR, 0x53);
49.
50.     // Set SCL high/low counts
51.     RegisterWrite(I2C0_FS_SCL_HCNT, 60 + 30); // 0.6us + 0.3us
52.     RegisterWrite(I2C0_FS_SCL_LCNT, 130 + 30); // 1.3us + 0.3us
53.
54.     // Enable the controller
55.     RegisterWrite(I2C0_ENABLE, 1);
56.
57.     // Wait until controller is powered on
58.     while (!(RegisterRead(I2C0_ENABLE_STATUS) & 0x1)) {}
59. }
60.
61. void GSensor::ADXL345_Init() {
62.     // Initialize the ADXL345 chip
63.     // Write the settings to the corresponding I2C0 registers
64.
65.     ADXL345_RegWrite(ADXL345_REG_DATA_FORMAT, XL345_RANGE_16G |
XL345_FULL_RESOLUTION); // +- 16g range, full resolution
66.     ADXL345_RegWrite(ADXL345_REG_BW_RATE, XL345_RATE_200); // Output
Data Rate: 200Hz
67.
68.     // Configure thresholds and interrupts
69.     ADXL345_RegWrite(ADXL345_REG_THRESH_ACT, 0x04); // Activity
threshold
70.     ADXL345_RegWrite(ADXL345_REG_THRESH_INACT, 0x02); // Inactivity
threshold
71.     ADXL345_RegWrite(ADXL345_REG_TIME_INACT, 0x02); // Time for
inactivity
72.     ADXL345_RegWrite(ADXL345_REG_ACT_INACT_CTL, 0xFF); // Enables AC
coupling for thresholds
73.     ADXL345_RegWrite(ADXL345_REG_INT_ENABLE, XL345_ACTIVITY |
XL345_INACTIVITY); // Enable interrupts
74.
75.     // Stop measure
76.     ADXL345_RegWrite(ADXL345_REG_POWER_CTL, XL345_STANDBY);
77.
78.     // Start measure
79.     ADXL345_RegWrite(ADXL345_REG_POWER_CTL, XL345_MEASURE);
80. }
81.
82. void GSensor::ADXL345_RegRead(uint8_t address, uint8_t *value) {
83.     // Read value from internal register at address
84.     // Write the register address to initiate the read operation
```

```
85.     RegisterWrite(I2C0_DATA_CMD, address + 0x400); // Send reg
address (+0x400 to send START signal)
86.
87.     // Write the read signal to receive the data
88.     RegisterWrite(I2C0_DATA_CMD, 0x100); // Send read signal
89.
90.     // Wait for the response (data) to be available in the RX buffer
91.     while (RegisterRead(I2C0_RXFLR) == 0) {}
92.
93.     // Read the response (data) from the RX buffer
94.     *value = RegisterRead(I2C0_DATA_CMD);
95. }
96.
97. void GSensor::ADXL345_RegWrite(uint8_t address, uint8_t value) {
98.     // Write value to internal register at address
99.     RegisterWrite(I2C0_DATA_CMD, address + 0x400); // Send reg
address (+0x400 to send START signal)
100.    RegisterWrite(I2C0_DATA_CMD, value); // Send value
101. }
102.
103. void GSensor::ADXL345_RegMultiRead(uint8_t address, uint8_t
values[], uint8_t len) {
104.    // Reads values from multiple internal registers
105.    RegisterWrite(I2C0_DATA_CMD, address + 0x400); // Send reg
address (+0x400 to send START signal)
106.
107.    // Send read signal len times
108.    for (int i = 0; i < len; i++)
109.        RegisterWrite(I2C0_DATA_CMD, 0x100);
110.
111.    // Read the bytes
112.    int nth_byte = 0;
113.    while (len) {
114.        if ((RegisterRead(I2C0_RXFLR)) > 0) {
115.            values[nth_byte] = RegisterRead(I2C0_DATA_CMD);
116.            nth_byte++;
117.            len--;
118.        }
119.    }
120. }
121.
122. void GSensor::ADXL345_XYZ_Read(int16_t szData16[3], int16_t
mg_per_lsb) {
123.    // Read acceleration data of all three axes
124.    uint8_t szData8[6];
```



```
125.     ADXL345_RegMultiRead(0x32, (uint8_t *)&szData8,  
sizeof(szData8));  
126.  
127.     szData16[0] = (szData8[1] << 8) | szData8[0];  
128.     szData16[1] = (szData8[3] << 8) | szData8[2];  
129.     szData16[2] = (szData8[5] << 8) | szData8[4];  
130. }  
131.  
132. bool GSensor::ADXL345_IsDataReady() {  
133.     // Return true if there is new data  
134.     bool bReady = false;  
135.     uint8_t data8;  
136.  
137.     ADXL345_RegRead(ADXL345_REG_INT_SOURCE, &data8);  
138.     if (data8 & XL345_ACTIVITY)  
139.         bReady = true;  
140.  
141.     return bReady;  
142. }  
143.
```

## 9. GSensorMain.cpp:

```
10.  #include <iostream>
11.  #include <unistd.h> // For usleep
12.  #include "GSensor.h"
13.  #include "DE1SoChps.h"
14.  #include "PIOControl.h"
15.
16.  int main() {
17.      std::cout << "Starting GSensorMain main function!" << std::endl;
18.
19.      // Instantiate GSensor object using dynamic memory allocation
20.      std::cout << "Instantiating GSensor object using dynamic memory
allocation!" << std::endl;
21.      GSensor *gsensor = new GSensor();
22.      PIOControl *piocontrol = new PIOControl();
23.
24.      // Read the ADXL345 Device ID (Reg 0x00) and verify correct ID
25.      std::cout << "Reading the ADXL345 Device ID (Reg 0x00) and
verify correct ID!" << std::endl;
26.      uint8_t deviceId;
27.      gsensor->ADXL345_RegRead(ADXL345_REG_DEVID, &deviceId);
28.      if (deviceId != 0xE5) {
29.          std::cerr << "Error: Incorrect device ID!" << std::endl;
30.          delete gsensor; // Clean up allocated memory
31.          return 1;
32.      } else {
33.          std::cerr << "Device ID valid!" << std::endl;
34.      }
35.
36.      // Initialize the ADXL345 module
37.      std::cout << "Initializing the ADXL345 module!" << std::endl;
38.      gsensor->ADXL345_Init();
39.
40.      // Set the resolution factor (mg_per_lsb)
41.      float mg_per_lsb = 4; // Approximate multiplier
42.
43.      // Infinite loop to read and print X, Y, Z data
44.      std::cout << "Starting the infinite loop to read and print X, Y,
Z data: " << std::endl;
45.      std::cout << "Move the board around to obtain valid data within
the range of about -1000 mg to 1000 mg on each axis." << std::endl;
46.      while (true) {
47.          int16_t data[3];
48.          gsensor->ADXL345_XYZ_Read(data, mg_per_lsb);
```

```
49.         int16_t x_mg = data[0] * mg_per_lsb;
50.         int16_t y_mg = data[1] * mg_per_lsb;
51.         int16_t z_mg = data[2] * mg_per_lsb;
52.
53.         // Ensure X value is within desired range
54.         if (x_mg > 1000) {
55.             x_mg = 1000;
56.         } else if (x_mg < -1000) {
57.             x_mg = -1000;
58.         }
59.
60.         // Ensure Y value is within desired range
61.         if (y_mg > 1000) {
62.             y_mg = 1000;
63.         } else if (y_mg < -1000) {
64.             y_mg = -1000;
65.         }
66.
67.         // Ensure Z value is within desired range
68.         if (z_mg > 1000) {
69.             z_mg = 1000;
70.         } else if (z_mg < -1000) {
71.             z_mg = -1000;
72.         }
73.
74.         std::cout << "X: " << x_mg << " mg, Y: " << y_mg << " mg, Z:
" << z_mg << " mg" << std::endl;
75.
76.         // Store X coordinate in PIO output register
77.         piocontrol->WritePIOout(x_mg);
78.
79.         // Read angle value from PIO input register
80.         int angle = piocontrol->ReadPIOin();
81.
82.         // Print angle
83.         std::cout << "angle: " << angle << std::endl;
84.
85.         // Wait for a short duration before polling again
86.         usleep(100000); // Sleep for 100 milliseconds (adjust as
needed)
87.     }
88.
89.     delete gsensor; // Clean up allocated memory
90.     return 0;
91. }
```

10. Makefile:

```
11.  # Makefile
12.
13.  # Compiler
14.  CC := g++
15.
16.  # Compiler flags
17.  CFLAGS := -std=c++0x -Wall -Wextra
18.
19.  # Source files
20.  SRCS := GSensor.cpp GSensorMain.cpp DE1SoChps.cpp PIOControl.cpp
    DE1SoCfpga.cpp
21.
22.  # Object files
23.  OBJS := $(SRCS:.cpp=.o)
24.
25.  # Executable name
26.  TARGET := GSensorMain
27.
28.  .PHONY: all clean
29.
30.  all: $(TARGET)
31.
32.  # Compile source files into object files
33.  %.o: %.cpp
34.      $(CC) $(CFLAGS) -c $< -o $@
35.
36.  # Link object files into executable
37.  $(TARGET): $(OBJS)
38.      $(CC) $(OBJS) -o $(TARGET)
39.
40.  # Clean up object files and executable
41.  clean:
42.      rm -f $(OBJS) $(TARGET)
43.
```