CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

# Biosignals Real-Time Measurement

A graduate project submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

By

Luis Rivera

December 2022

The graduate project of Luis Rivera is approved:

| | |
|---|---|
| Dr. David Hawkins | Date |
| Dr. Xiaojun Geng | Date |
| Dr. Ramin Roosta, Chair | Date |

California State University Northridge

ACKNOWLEDGEMENTS

Words cannot express my gratitude to my professor and chair of my committee Dr. Ramin Roosta for his invaluable experience, patience, and feedback. I also could not have undertaken this journey without the help of Dr. David Hawkins, who generously provided knowledge, subject matter expertise and constant motivation. Additionally, this endeavor would not have possible without the generous support from Dr. Xiaojun Geng.

Finally, I want to thank my family and friends, for their belief and unconditional support that has kept my spirit and motivation high.

Table of Contents

List of Figures

List of Tables

Abstract

Biosignals Real-Time
Measurement

By
Luis Rivera

Master of Science in Computer Engineering

Biosignals are physiological and physical measurements from living beings. The biosensor developed in this graduate project measures human biosignals. Each biosignal is a continuous function of time, and provides useful information of the human body's function such as Electrocardiogram, Galvanic skin response, Electromyograph, and Electroencephalogram among others. This project implements real time analysis on multiple sensors connected to a Digilent Zybo Z7-20 development board. Digilent Pmod sensors were used with the development board to demonstrate the measurement of human biosignals in real-time. The Digilent development board and Pmod sensors were used to measure biosignals and then the software running on the Zynq processor was used to process the sensor data. The outcome of this project is a prototype wearable device that measures biosignals of the user and identify stress using a stress level algorithm.

# 1. Theory

## 1.1 Motivation

For many years, electronic devices have helped diagnose patients and perform analysis of biometric data. This required expensive diagnostic equipment that only medical practitioners could afford. With advancements in technology, biometric sensors and processors have significantly reduced in size and area, to the extent that we can now wear these devices on our wrists. By nature, people have become curious to monitor their own health, from the comfort of home with such wearable devices.

Being able to determine one's biometric data such as BPM, skin conductance and temperature can be utilized to help understand one's medical or emotional state. For example, smartwatches from companies such as Apple, Fossil, Fitbit and others, record user's biosignals such as BPM, sleeping patterns, level of oxygen in blood, GPS tracking for exercise and other features. Wearable medical or fitness devices can help people improve health and fitness goals.

## 1.2 Analysis

The following sections provide details on the sensors used to measure human biosignals. An understanding of how each sensor measures a biosignal, and interfaces to electronics, will be useful knowledge for implementation and integration. Knowing how the sensor functions, will give the necessary background in determining the logic and filtering needed to calculate each respective measurement.

### 1.2.1. Electrocardiography

An electrocardiograph (ECG) measures the electrical signals in a heart for examining heart problems and monitoring health. ECG signals are converted to a voltage using a pulse sensor Sen-11574 to determine the patient heart rate or BPM [1]. Figure 1 shows how a pulse sensor works; a green LED shines light on the finger and measures the amount of light reflected using a photosensor. The sensor provides a measurement of blood flow during each heartbeat. The analog signal generated from the photosensor usually is small and noisy, therefore an RC filter and Op Amp are used. The RC filter is used to remove noise while the Op Amp is used to amplify the few millivolts output, before being sampled by the analog-to-digital converter (ADC).

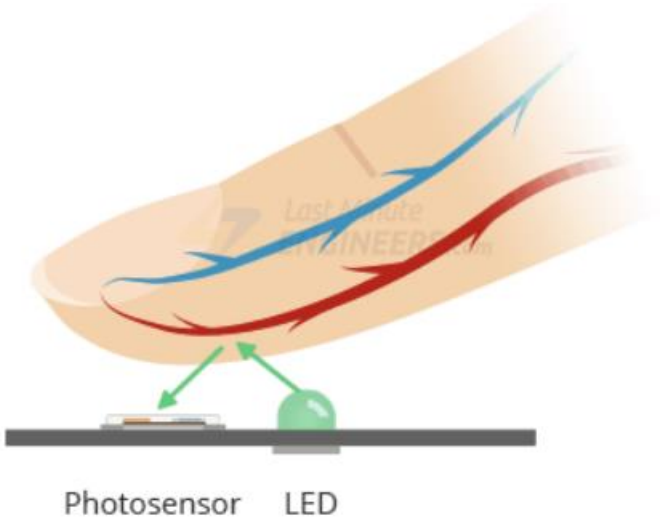


*Figure 1. Pulse Sensor Concept*

For this project's purpose, understanding when the heart rate increases or rate of change is increasing, is sufficient to measure BPM. Figure 2 demonstrates the typical ECG waveform that is composed of the P, Q, R, S and T wave. The most

prominent wave in an ECG, is the QRS complex that consists only of Q, R and S wave. The QRS complex is responsible for reflecting ventricular contractions and the time of occurrence. The increasing part of the QRS complex happens to be from the Q to R wave. By measuring the time between QR rising-edges, we can measure a person's BPM. Many different approaches exist for measuring BPM with an ECG signal but the Q to R wave is one of the simplest and most effective methods to understand and implement.

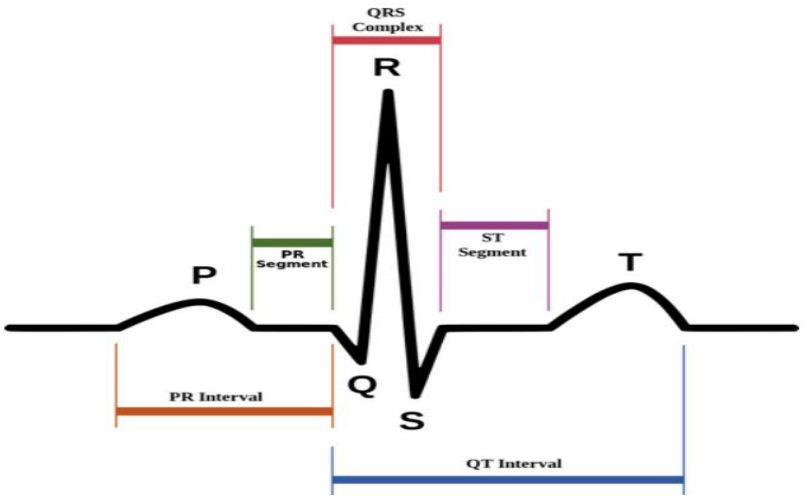


*Figure 2. ECG Signal*

### 1.2.2. Galvanic Skin Response

The galvanic skin response (GSR) refers to the electrical conductance of human skin, that is related to sweat gland activity or emotional arousal. The conductance of the skin changes due to sweat gland activity or emotional state. The biosensor used the Grove-GSR to measures electrical conductance of the skin [2]. This electrodermal activity is strongly related to moods such as happy, sad, afraid, nervous or others that can be related to stress. Figure 3 describes how the GSR sensor works by using two electrodes that apply a small constant voltage usually 0.5V and

measuring the variations in voltage through the electrodes. Changes in the voltage are related to sweat gland activity or moods.

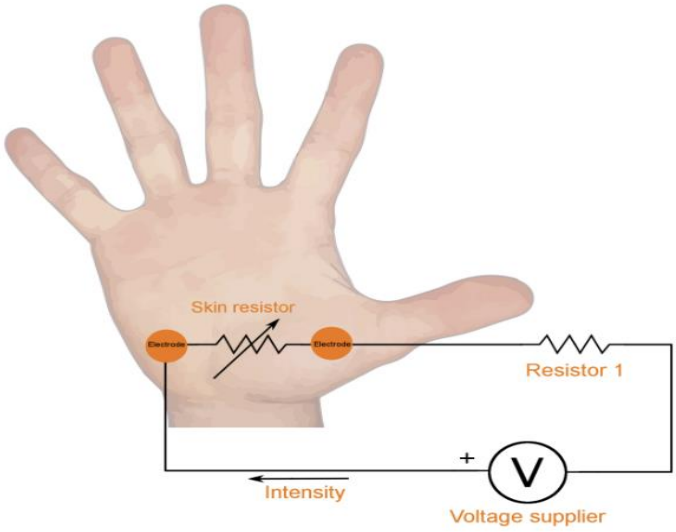


*Figure 3. GSR Circuit Diagram*

When placing two electrodes of the GSR sensor on a hand as shown in Figure 3, the circuit closes as your hand behaves as a resistor. The output of the GSR sensor is a voltage that is converted to conductance, which is the inverse of resistance. Conductance (Siemens) is inversely proportional to resistance (Ohms) and calculating conductance can approximate for sweat gland activity. This is assuming the subject does not have sweaty hands in the beginning or in an environment that causes the fingers to be moist, therefore fingers should be wiped dry before experimentation. Measuring GSR has been shown to be an extremely accurate method to determine emotional stress [3].

### 1.2.3. Temperature

For measuring temperature, an ambient temperature sensor was used to measure the environment temperature that can be extra parameter for the project. Note, this is not an accurate approach since this assumes the person is a similar

temperature to the environment. The sensor used was the Pmod TMP3 [4], which works by generating a current over the bandgap, measuring the voltage, and then generates a different current over it to measures new voltage. The voltages are processed and then converted to temperature. A Sigma-Delta ADC is used to convert voltage to a digital word that relates to the transistor temperature. The output of this sensor is a digital signal encoded with the temperature. The Pmod TMP3 was used during the initial development of the biosensor, since the temperature sensor physical packaging was not suitable for measuring human body temperature. So, the decision was made to upgrade the Pmod TMP3 for a sensor with contact leads or that is more appropriate for body temperature or holding.

A more suitable sensor for measuring human body temperature is the Pmod TC1 [5] that is a K-type thermocouple with wide temperature range -73°C to 482°C and has accuracy of $\pm$2°C. Figure 4 illustrates how the thermocouple works by measuring the cold junction that consists of two different electrical conductors that form a thermal junction. As the temperature changes at the junction, the voltage measured can be translated to a corresponding temperature. The output of thermocouples, is a small voltage in the mV range. Thermocouples J, K and T type are the most common and at room temperature, their voltage varies from 52µV/°C, 41 µV/°C, and 41 µV/°C while other less-common types are even smaller.

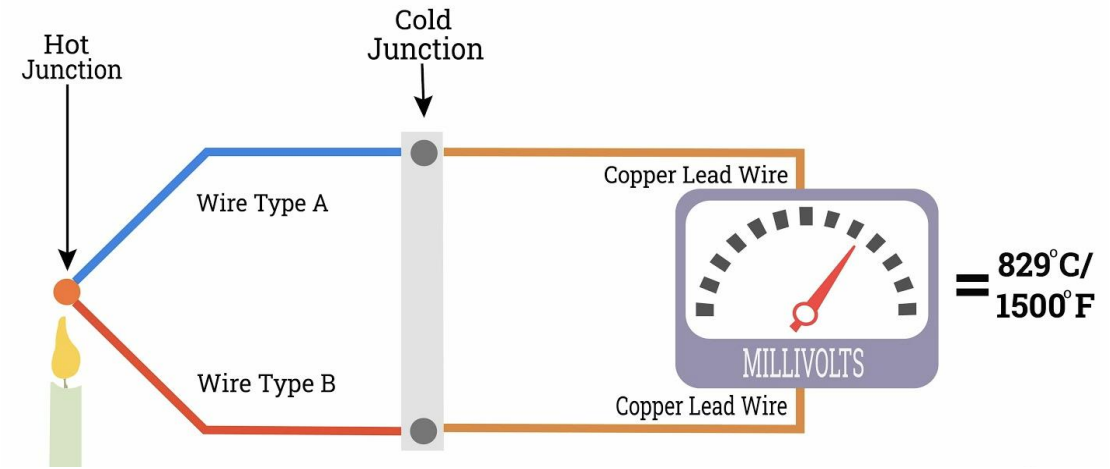


*Figure 4. K-type Thermocouple Concept*

To meet input requirements of FPGA system and ADC, an amplifier, filter and cold junction compensation sensor as shown in Figure 5 are used to interface with thermocouple. There are several modules such as the Pmod TC1 that are packaged with a temperature sensor to measure the reference junction of the k-type thermocouple, amplifier to increase voltage and filter to shape signal.

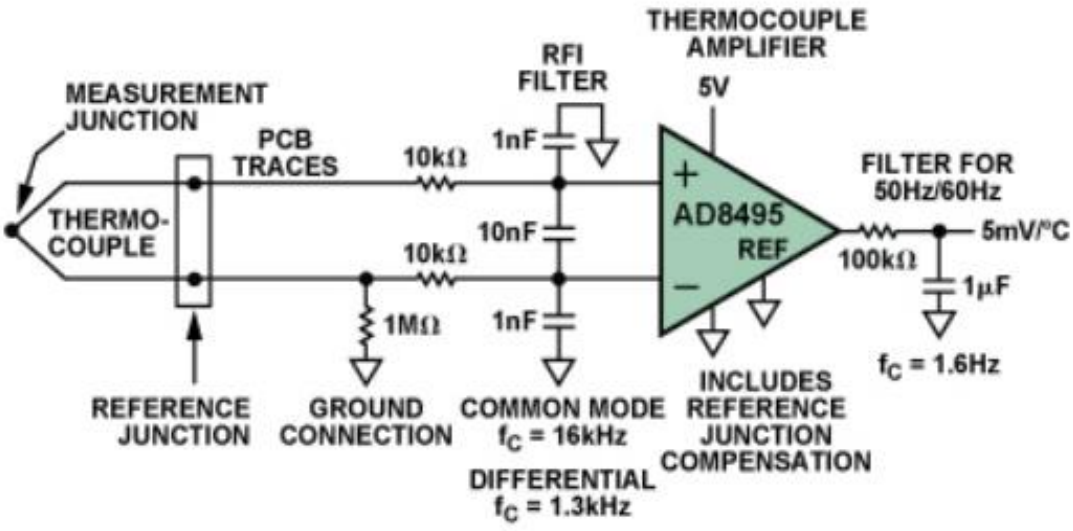


*Figure 5. K-Type Thermocouple Circuit*

### 1.3 Stress Analysis

The biosensor software executing on the Zynq processor implements a stress level algorithm that utilizes the measurements from ECG, GSR and temperature sensor. Each person has a unique emotional response, so it is difficult to interpret the same set of sensor readings on two different people as corresponding to the same stress level. To accommodate the unique features of each user, testing different scenarios on the subject for different stress levels are needed.

By observing the sensor outputs, if the heart rate is 110-130 BPM, then it can be perceived as you are exercising, normally have a fast heartbeat, fever, or other conditions. Now if the temperature is high like above 100ºF or 37.78°C, then maybe the subject is in a really warm environment, ill, intense physical activity or other factors. With the two results, it's not bad judgement to say the subject is stressed since BPM and temperature measurements are higher than normal. The stress level algorithm receives measurements from the sensors as inputs and will report the stress level to the wearer. To improve the algorithm, a machine learning algorithm can be implemented to learn from previous subjects and still learn from new patients by storing data in memory.

### 1.4 OLED Display

A Pmod OLED [6] display was used for the project, the measurements from the sensors are displayed on the OLED. Results are displayed on the Vitis GUI terminal but to make the embedded system more user friendly, results are printed on OLED display.

### 1.5 Processors and Field Programmable Gate Arrays

A System on Chip (SoC) [7] combines the software programmability of ARM processors with the hardware programmability of a Field Programmable Gate Array (FPGA) [8], illustrated in Figure 6. An FPGA is a reprogrammable logic semiconductor device containing a matrix of configurable logic blocks (CLBs) that are interconnected with wires. CLBs [9] are essential to the FPGA as they contain the design logic. FPGAs can be programmed using Hardware Description Language (HDL) such as VHDL, Verilog and System Verilog while ARM processors and connected peripherals can be programmed using C/C++.

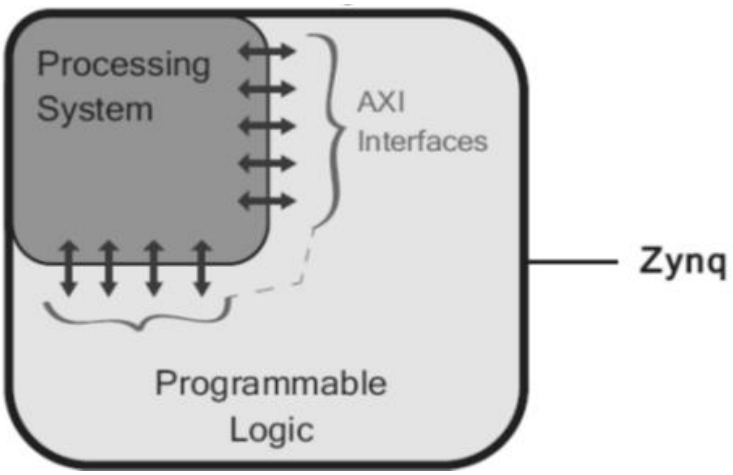


*Figure 6. Conceptual Zynq SoC*

CLBs allows the device to implement combinational and sequential logic using Flip-Flops (FFs), Look-up Tables (LUTs), and Multiplexers, illustrated in Figure 7. CLBs can be broken down to two main structures called slices, SLICLEL and SLICEM. Each slice consists of LUTs, FFs, multiplexers and depending on the FPGA family, the elements may vary. The two slices are SLICEL which can only be used for combinational logic, while SLICEM can be configured to implement distributed RAM or shift registers. Key distinction

between the two slices is one has memory capability while the other doesn't.

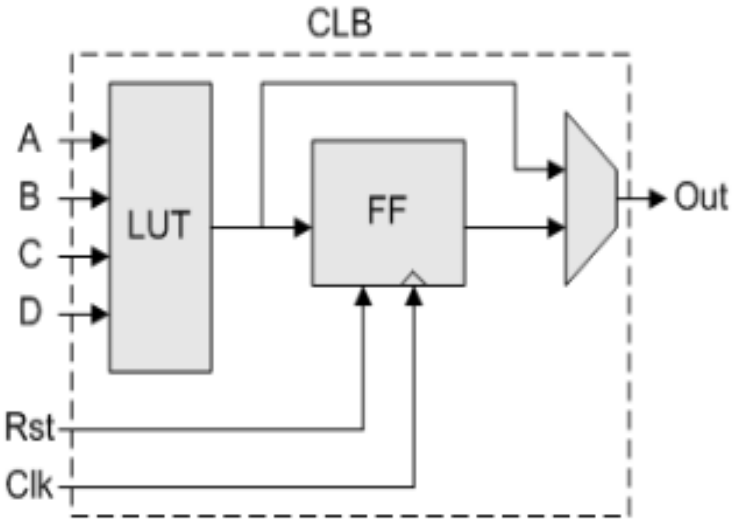


*Figure 7. Conceptual Configurable Logic Block*

### 1.6 Zynq Architecture

The Digilent Zybo Z7-20 development board [10], [11] contains a Xilinx Zynq-7000 [12] device that has a dual-core ARM Cortex-A9 processor [13] and an FPGA. Figure 6 illustrates the general architecture of a Zynq device that consists of two parts: the Processing System (PS) and Programmable Logic (PL). Figure 8 provides greater insight to the peripherals a Zynq device consists of via PS (processor) and PL (FPGA fabric). PS is better used for dynamic tasks, general purpose sequential tasks, operating system, and complicated logic controls assuming the processor can keep up, while the PL is good for static parallel tasks, peripheral controls, and intensive data computations. For more details on the Xilinx Zynq-7000 architecture see references [14], [15].

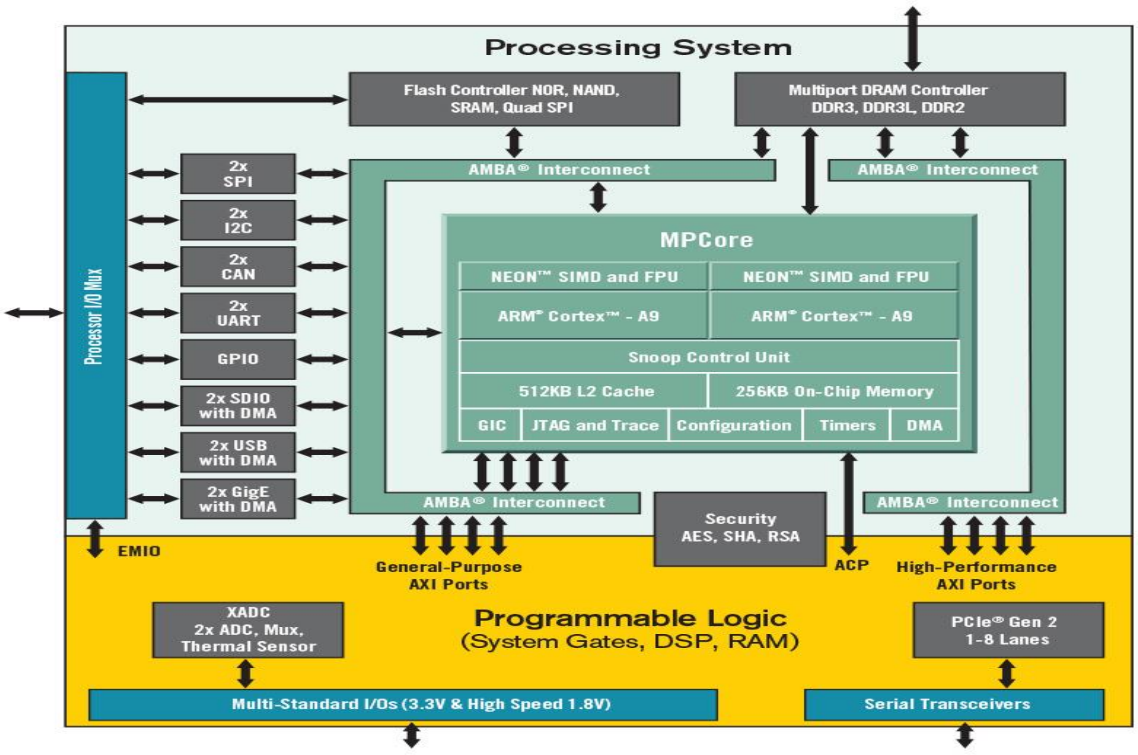


*Figure 8. Zynq SoC*

The PS can communicate directly with external interfaces such as sensors via Multiplexed Input/Output (MIO) that provide 54 pins of flexible connectivity. Extended MIO (EMIO) connects the PS to the fabric and can be used to connect to external devices through the FPGA PL pins. Figure 9 describes the peripherals in the PS such as the ARM Cortex-A9, peripheral interfaces, cache memory, memory interfaces, clock generation, interconnects and more. The I/O peripherals on the PS include communication protocols such as SPI, I2C, CAN, UART, SD, USB, GigE and General-Purpose Input/Output (GPIO) to control buttons, switches and LEDs.

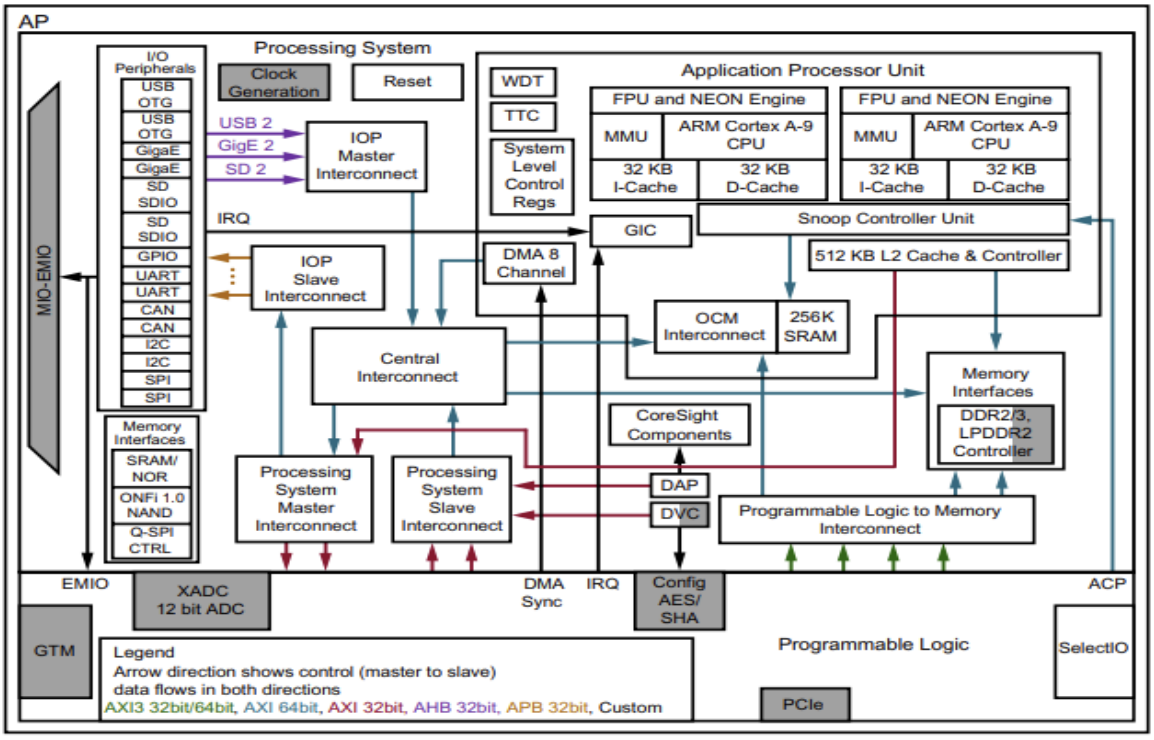


*Figure 9. Zynq SoC Architecture*

Figure 10 illustrates primary resources of the PL such as general-purpose FPGA logic fabric that is composed of components like Block RAM (BRAM) for embedded memory, DSP48E1 slice for high-speed arithmetic operations, PCIe general purpose serial interconnect and XADC for analog-to-digital signal conversion. The Input/Output blocks are GPIO interfaces permitting up to 3.3V that contain a pad, to provide a physical connection to the outside world for a single input or output signal.

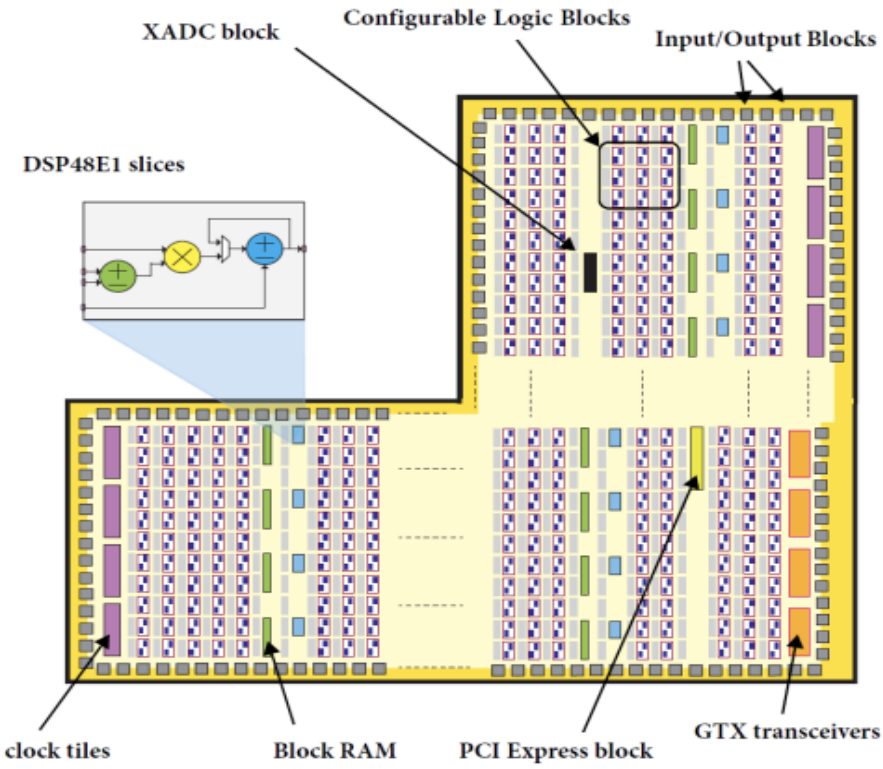


*Figure 10. Zynq PL Architecture*

### 1.7 AXI Interface

Figure 11 illustrates PS and PL blocks connected; Advanced eXtensible Interface (AXI) [16] buses are used to connect PS to other blocks on the PL. This protocol enhances the performance and utilization of the interconnect when used by multiple masters, allowing better bandwidth and low latency [17]. There are three types of AXI4 bus protocols: AXI4, AXI4-Lite, and AXI4-Stream. AXI4 is a memory-mapped link, data arrives first with up to 256 data words and then address is supplied; while AXI4-Lite is a simpler version that supplies an address, and a single data word can be transferred. AXI4-Stream is the only nonmemory-mapped link that supports burst transfers of any size.

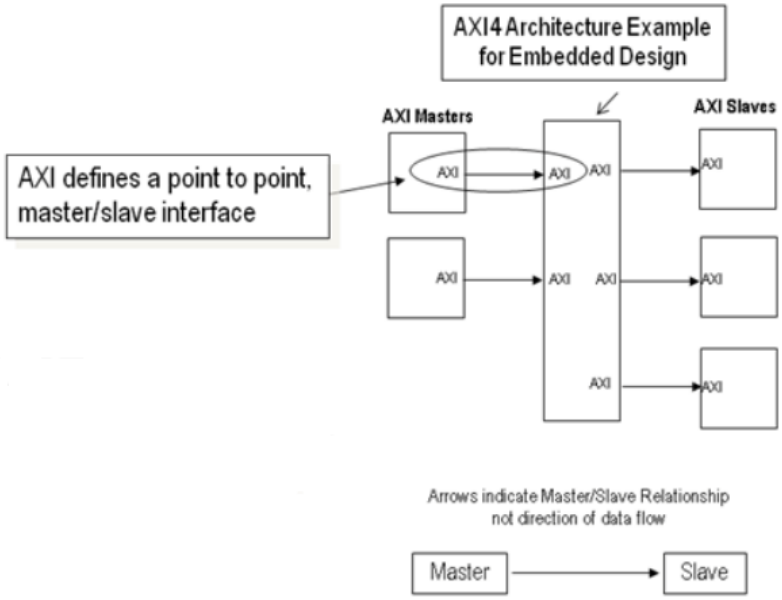


*Figure 11. AXI Architecture Design*

# 2. Methodology

## 2.1 Sensors & Communication Protocols

The following sections provide details of the communication protocols for each sensor such as Inter-Integrated Circuit (I2C) and Serial Peripheral Interconnect (SPI). For the Zynq-7000 to communicate with the sensors, the Zynq-7000 must implement the same communications protocol.

### 2.1.1. Pmod AD2

The heart rate and galvanic skin response sensor both have an analog output. To communicate or interface with either sensor, an ADC must be used to convert voltage to digital. The Zynq-7000 has an internal System Monitor called the "XADC" [18], however, the input range is limited to 1V.

The voltage from heart rate and GSR sensor is 1-1.5V, therefore if the XADC is to be utilized, the output of both sensors must be attenuated using a voltage divider. Interfacing the ECG and GSR sensors to the XADC would have required custom interface circuits, therefore an alternative ADC was required. The Pmod AD2 input range is 0 to 3.3V meet the sensor voltage requirements, so Pmod AD2 was purchased.

The Pmod AD2 [19], [20] contains an AD7991 IC chip [21] shown in Figure 12, which can read up to four channels sequentially with up to 12-bit resolution. The development board must use I2C to send/receive data from the Pmod AD2, which is connected to the sensors. I2C is a serial communication protocol that transfers data, bit by bit sequentially, while only using three wires to transmit/receive data (SDA), serial clock (SCL), and ground. Note, SDA and

SCL must be connected to pull up resistors because I2C devices are open drain requiring resistors to generate the logic high level on the bus, otherwise the bus does not work.

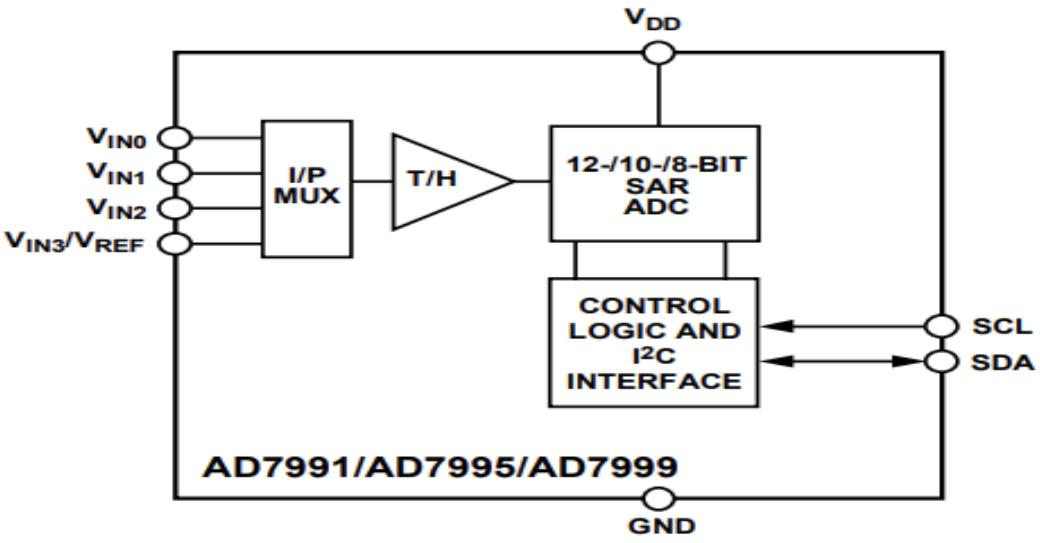


*Figure 12. AD7991 Functional Block Diagram*

Figure 13 illustrates a timing diagram of the I2C protocol [22] start/stop conditions, 7-bit or 10-bit slave address, read/write bit, ACK/NACK bit and the data bits. The master device can initiate start to signal the start of a data transfer and stop for the end of transmission. I2C allows for multiple masters and slaves as seen in Figure 14 and some devices can behave as both master and slave. To distinguish between devices, a 7-bit or 10-bit address is needed at the start of each data transfer to identify the slave in communication. A read/write bit is used to identify if the master is reading from or writing to the slave. Lastly, when a byte of data is being transferred, there is an ACK/NACK bit that allows the receiver to communicate to the transmitter that the byte was successfully received and another byte may be sent.

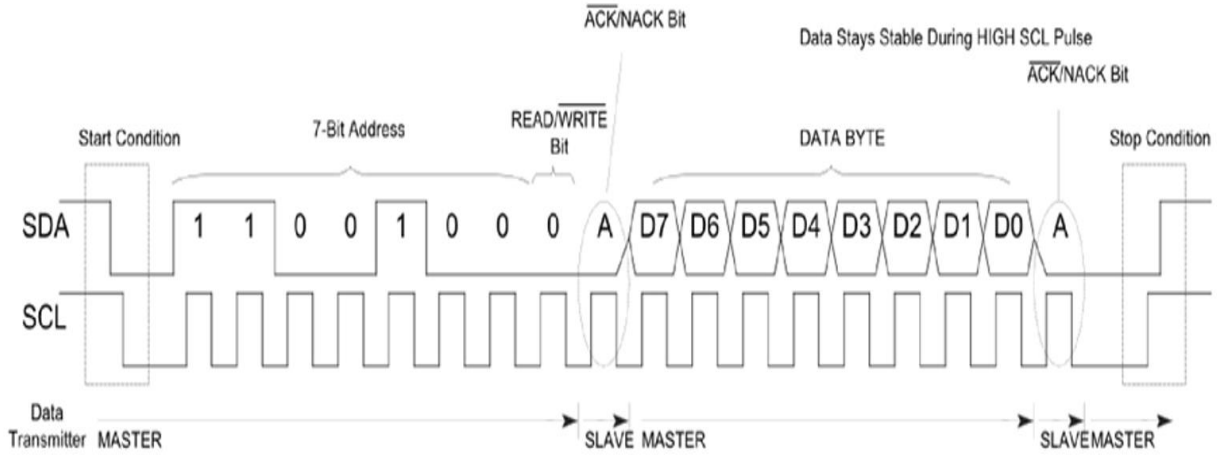


*Figure 13. I2C Timing Diagram*

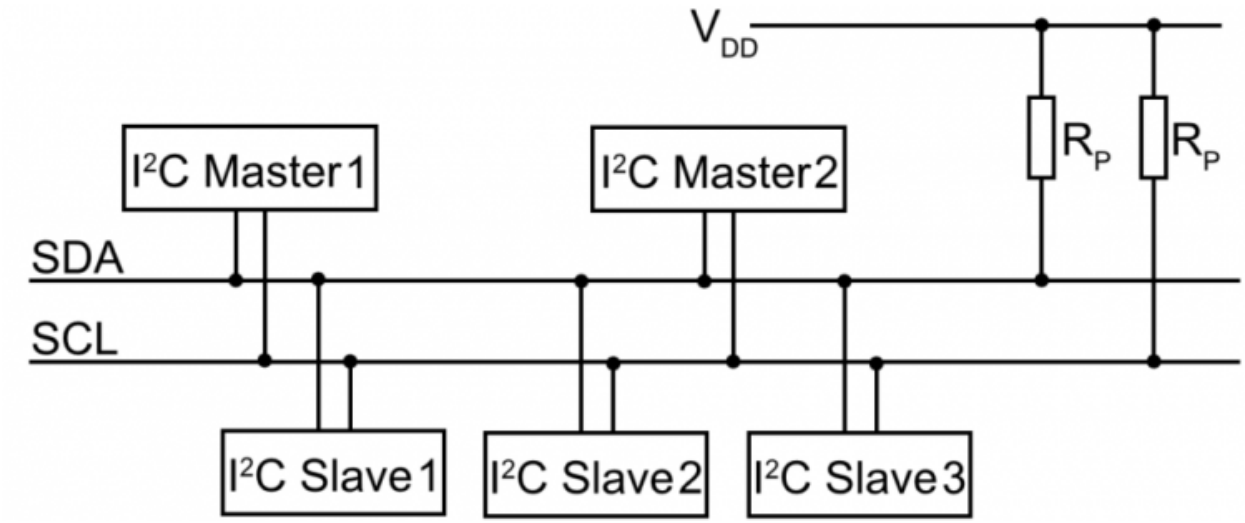


*Figure 14. I2C Connections Diagram*

### 2.1.2. Pmod TMP3

The Pmod TMP3 [23], [24] uses the TCN75A IC chip [25] shown in Figure 15 and output is a signed 12-bit digital signal via I2C. For the project, a sensor that is more appropriate for body temperature is needed. The outcome is to prototype a wearable device and the Pmod TMP3 is not suitable for attaching to a Velcro strap, glove nor an accurate measurement for body temperature. Not much will be discussed about this Pmod, and communication protocol is same as Pmod AD2.

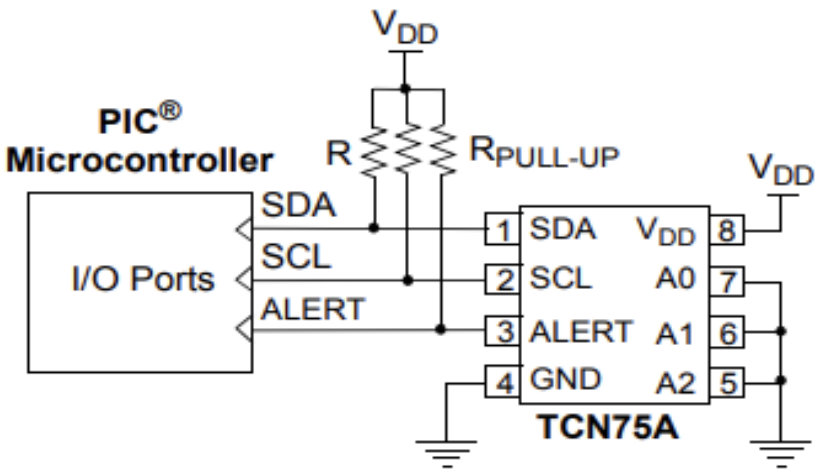


*Figure 15. TCN75A Typical Application Circuit*

### 2.1.3. Pmod TC1

The Pmod TC1 [26], [27] is implemented using the MAX31855 IC chip [28] shown in Figure 16, that performs cold junction compensation with 14-bit ADC and output is a signed 14-bit digital signal accessible over SPI protocol [29]. The data is transmitted from the low noise amplifier to ADC, which connects to a digital controller that is SPI compatible.

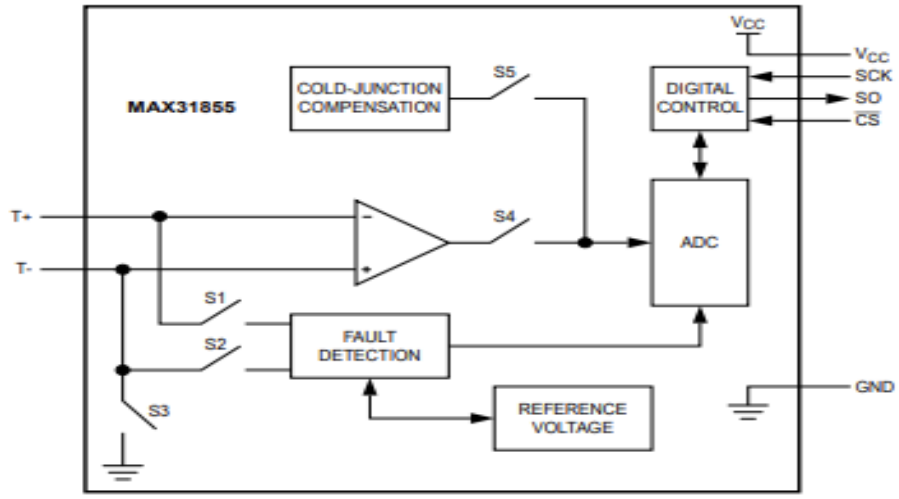


*Figure 16. MAX31855 Block Diagram*

SPI can be connected to multiple slaves if they each have unique slave select signals, as shown in Figure 17. The devices tri-state their outputs, when not selected. This protocol is synchronous, therefore there is a clock signal (SCLK) and chip select (CS) is used to enable device for reading/writing. Transmission of data from master to slave is via the Master Output Slave Input (MOSI) bus while sending data from slave to master is via Master Input Slave Output (MISO). Slaves are controlled by 3 inputs SCLK, MOSI, CS# and output MISO. The master is denoted by the controller device that drives the SCLK, CS and MOSI pins. Note, SPI is much faster compared to UART and I2C for short distances.

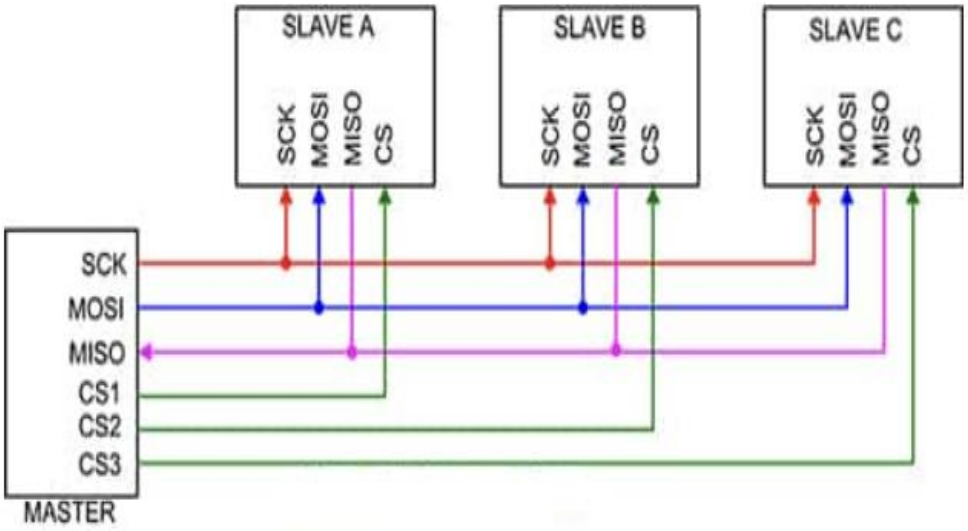


*Figure 17. SPI Connections Diagram*

### 2.1.4. Pmod OLED

For a wearable device, features like Bluetooth, WIFI or screen display are needed to send or print results. The simplest approach is using an OLED display to print the sensor measurements instead of using Bluetooth or WIFI. Figure 18 illustrates an IC SSD1306 [30] display controller found on the Pmod OLED [31],

[32]. The OLED display is an SPI peripheral that uses a display controller chip. The embedded display only supports SPI write and has a variety of features such as turn on individual pixels, print predefined characters or load bitmaps onto the screen.

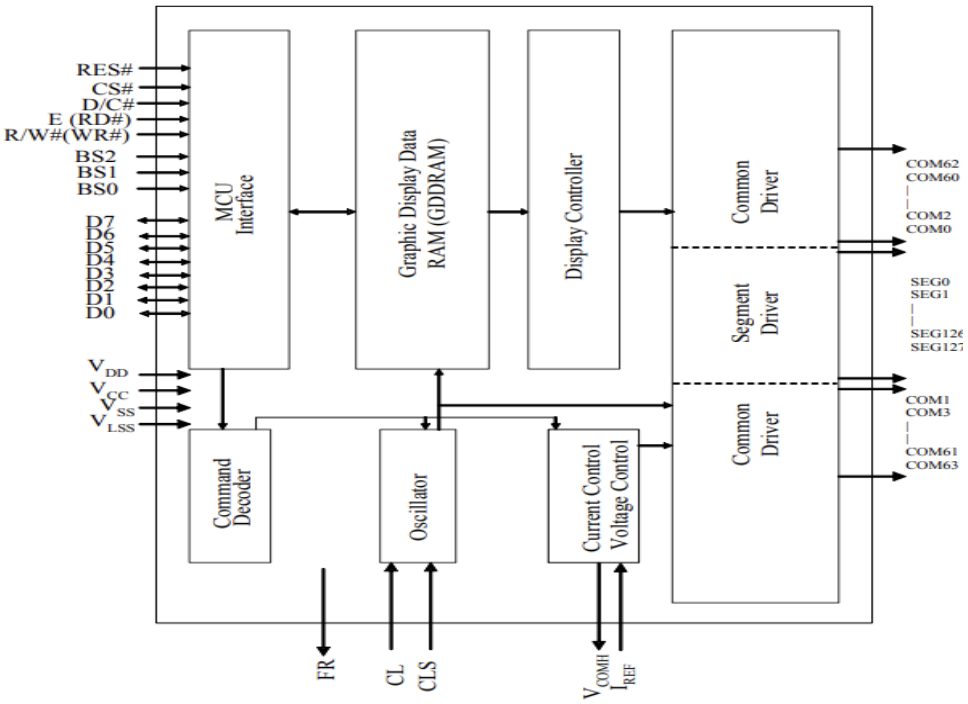


*Figure 18. SSD1306 Block Diagram*

## 2.2 Hardware Setup

Like mentioned in Sensors & Communication Protocols, the heart rate and GSR sensor will be connected to a Pmod AD2 that uses I2C, while the Pmod TC1 uses SPI and connects directly to the FPGA Pmod connectors. For the development board to interface with sensors, the Zynq-7000 must implement I2C and SPI interfaces. There are various implementations as each interface can be implemented via the PS hard-IP or PL logic to implement controllers. Implementing the communication peripherals will allow the board to read data from the sensors and once the data is retrieved, the processing can begin.

## 2.3 Software Flow

The following sections provide flowcharts and details regarding the individual algorithms for each sensor. The software configured each of the sensors for the operating modes required for the biosensor project.

### 2.3.1. Heart Rate

Figure 19 shows a flow-chart of the BPM algorithm; the signal processing performed on the ADC samples to isolate the pulses and detect peaks or rising edges. To interface with biosensor, the Pmod must be initialized, configured to read from channel 1, start ADC conversion, filter signal and calculate BPM. The I2C sensor reading is a 16-bit value, but only the LSB 12-bits are valid. A 0xFFF mask is used to preserve 12-bits from the ADC conversion.

To determine the DC average of the ECG signal, a moving average filter was used that behaves as a lowpass filter. The high frequency content is of interest, so by using the moving average result, it is used as a threshold to determine the rising-edge of the QRS complex. In an ECG signal, the pulse or peak is the highest value or point in the signal. By looking at the digital value or voltage representation, the moving average can easily be used in a conditional statement for filtering by comparing the ADC conversion with moving average. During implementation, it was noticed that simply detecting rising edges it was not efficient enough.

The ECG rising-edge is detected by thresholding ADC samples with moving average and checking that previous ADC sample is below the threshold. After detecting two rising edges or pulses, BPM can be calculated by dividing 60 with

the elapsed time between those two pulses (BPM = 60/elapsed_time). This can further be improved by adding another conditional logic to filter any elapsed time calculation less than 0.4 seconds. An elapsed time of less than 0.4 seconds results in a BPM greater than 150, which is not of interest to have such range. MATLAB was used to threshold the ECG signal more appropriately by using a histogram, more details in 4.1.

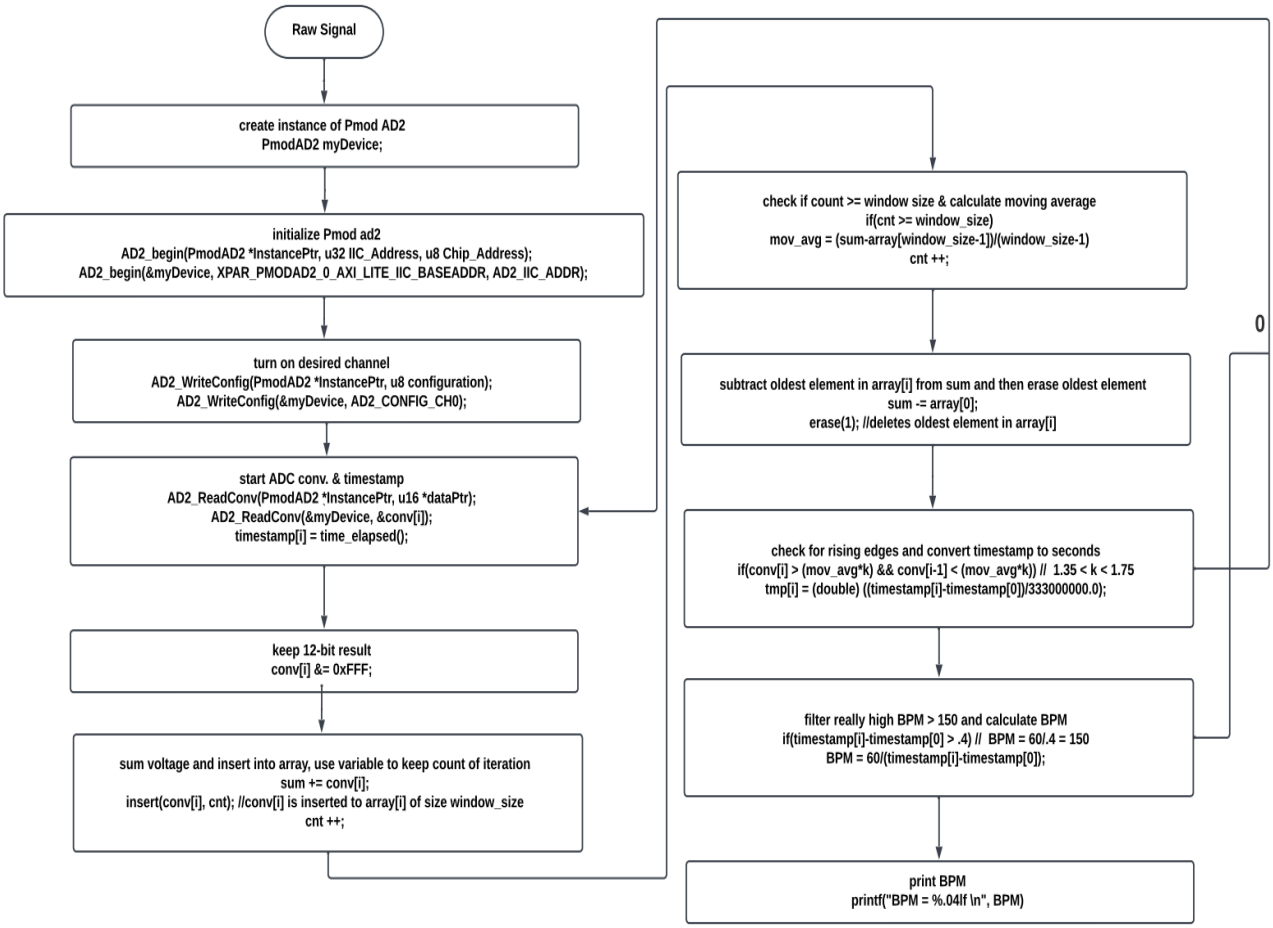


*Figure 19. Pulse Sensor using Pmod AD2 Flowchart*

### 2.3.2. GSR

Figure 20 illustrates a flow-chart of the GSR algorithm; the Pmod ADC samples are converted to voltage and then resistance is calculated to determine conductance. To calculate skin resistance, the calculation needs to be interpolated using two voltages from Figure 41 , that is from previous research [33] and used the same sensor. The voltage and resistance relationship are not linear since each voltage range increments by different resistances. MATLAB was used to determine the incremental resistance of each voltage range, more details in 4.3.

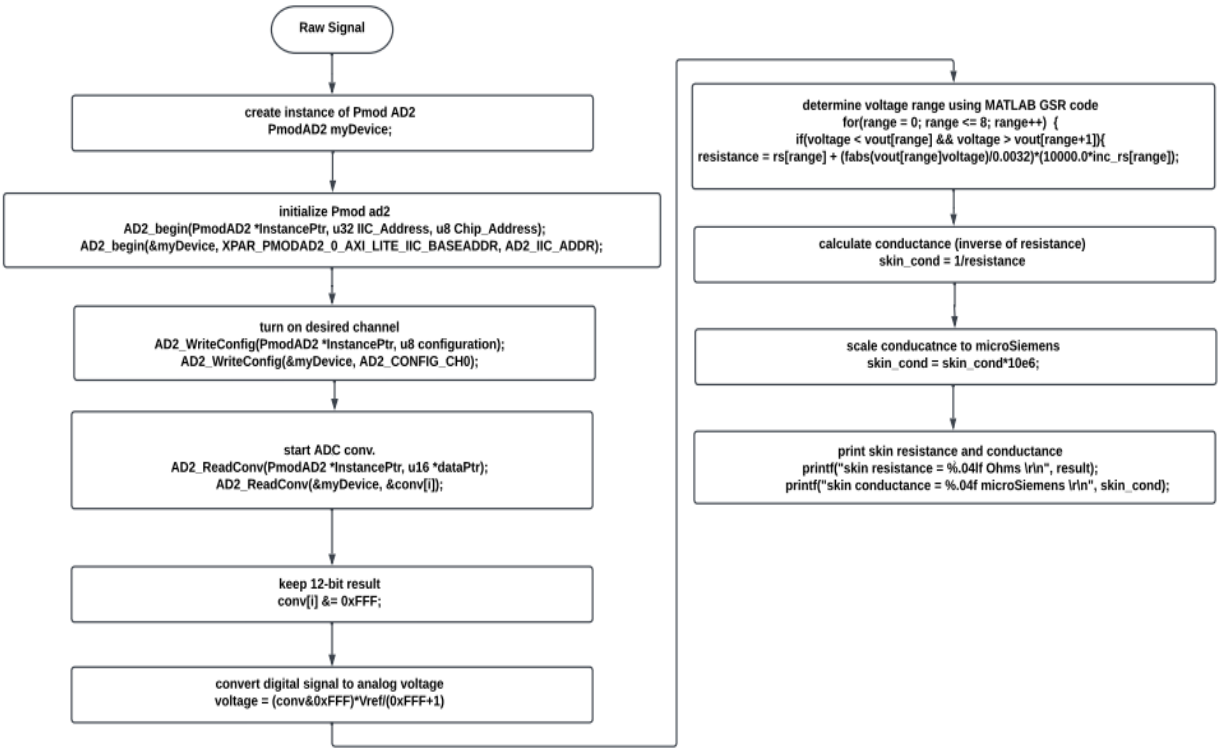


*Figure 20. GSR Sensor using Pmod AD2 Flowchart*

### 2.3.3. Temperature

The software implementation for temperature is much easier compared to the others. Figure 21 describes the few Pmod TC1 steps: creating instance of

Pmod device, initializing Pmod TC1, and computing Celsius or Fahrenheit measurement.

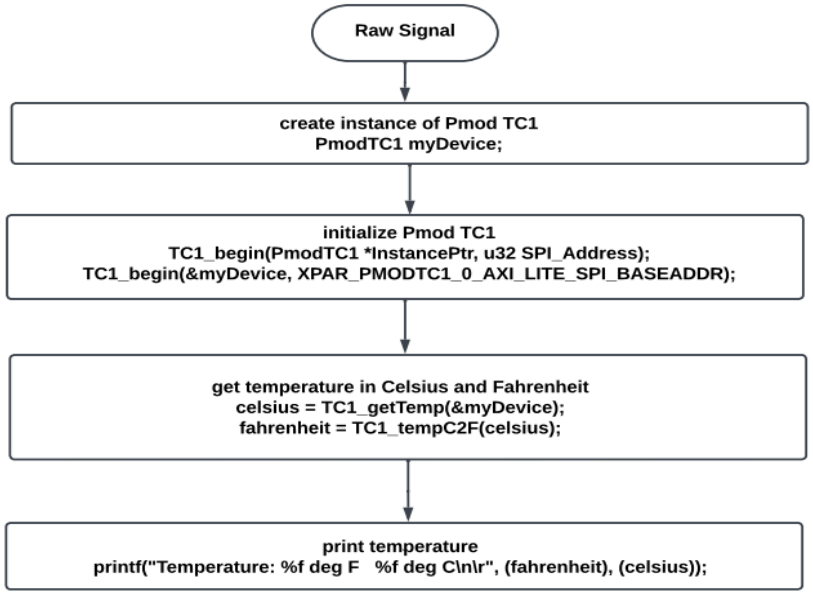


*Figure 21. Pmod TC1 Flowchart*

### 2.3.4. OLED Display

The implementation for the OLED display is illustrated in Figure 22; the OLED contains a very flexible controller that requires configuration before the display can be used. The configurations include setting the display orientation, background color, choosing and setting fill pattern, size of characters, and setting cursor to specific row and column. After creating an instance of the Pmod OLED and initializing device, the display is configured with default settings to simplify the process. To write to display, the display should be cleared and then the Pmod OLED functions can be used to point to a row and column and print characters to display.

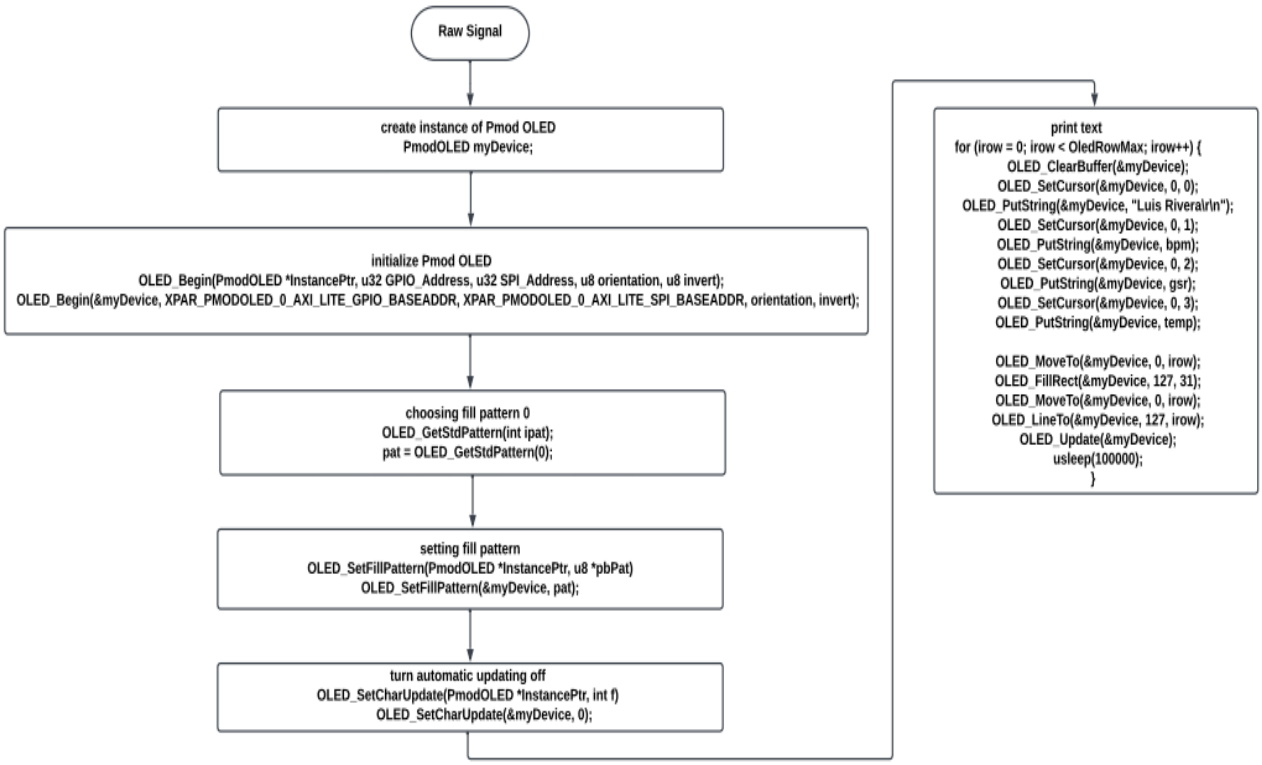


*Figure 22. Pmod OLED Flowchart*

# 3. Implementation

## 3.1 Xilinx Zynq-7000 Design

To configure the SoC, Vivado is used to create a block design that implements the desired PS and PL blocks. The PL implements SPI and I2C Master controllers to communicate with external sensors Pmod AD2, TC1 and OLED display. The biosensor software runs on one of the ARM cores within the Zynq-7000 PS, and the implemented interfaces can be connected to an AXI Interconnect that allows communication between PS-PL.

The Pmod IP blocks are custom IPs designed by Digilent with specific configurations for each specific Pmod. Digilent [34] has a Vivado library where all their IP repos can be downloaded. Figure 23 is a simplified block design of the Pmod IPs, since PmodTC_v1_0 and PmodOLED_v1_0 are SPI controllers and PmodAD2_v1_0 is an I2C controller. Similar design can be implemented using two SPI controllers and a single I2C controller. Note, IP repos must be added to Vivado IP repository for usage.

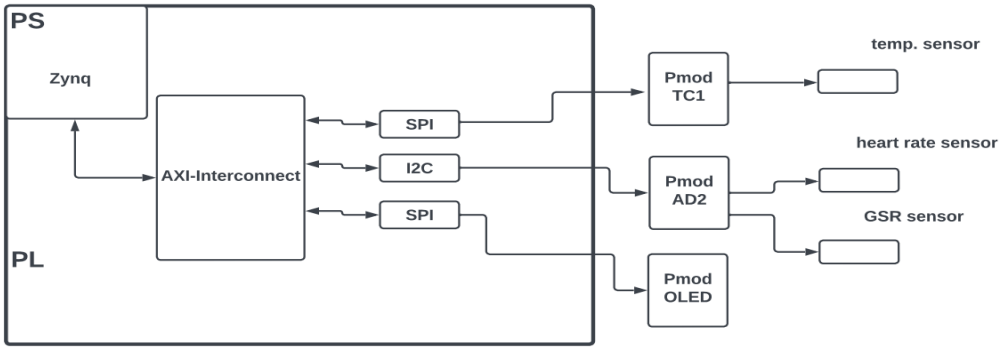


*Figure 23. Simplified block design*

Six logic blocks are used for this system in which two are generated by the development environment and the others need to be implemented from the IP catalog as shown in Figure 26. Once all IPs are added to block design, block

automation and connection automation tool are used for connecting blocks together and creating ports such as DDR and Fixed_IO that connect directly to the PS, more details on Xilinx and Digilent tutorials [35], [36], [37], [38]. The blocks generated are the Processor System Reset and AXI Peripherals Interconnect.

**(a) Zynq Processing System**

This is the instantiation of the PS that must be included in every project for the ARM processor to run the application. The PS block has configuration options that can be set at the hardware level such as enabling interfaces, defining clock frequency, setting interrupts, managing MIO configurations, and many more, however, the PS is controlled by the software. For this project, the following configurations were done:

- Clock configuration: After using block automation provided by Vivado and configuring the PLLs correctly based on the Zybo hardware design, the CPU clock is set to 667MHz along with a PL fabric clock at 50MHz. The PL fabric clock sets the PS and PL to the same clock frequency.
- MIO Configuration: UART1 interface is enabled with IO property set to MIO.
- PS-PL Configuration: UART1 should be set to a baud rate of 115,200.

**(b) Processor System Reset**

By adding the PS to a hardware design and using connection automation, the block is generated which provides resets to the entire system such as peripherals, interconnects and the processor. All connections for this block are automated using block and connection automation tool.

### (c) AXI Peripherals Connect

This block is generated when a PS and PL block are not connected in a hardware design and connection automation is used. This block allows peripherals to be connected using AXI interconnect which provides communication between PS-PL. Connections are automated as well.

### (d) Pmod AD2

The Pmod AD2 needs to be implemented from the IP catalog for usage, which will perform ADC conversions. The Pmod AD2 has four channels with up-to 12-bit resolution and reads from the channels sequentially. The interface output port (Pmod_out) of the PmodAD2_0 must be made into an external connection as shown in Figure 24, so the IP block can be mapped to the FPGA pins that connect to the physical Pmod AD2.

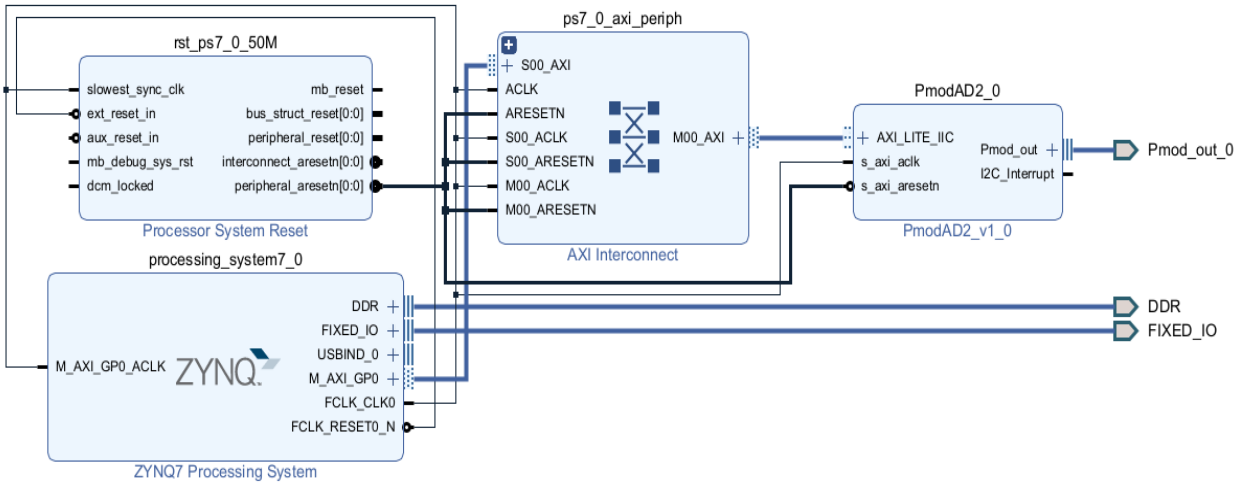


*Figure 24. Block Design w/ Pmod AD2*

After integrating and testing the system, the BPM calculation from the heart rate sensor were compared with the BPM from an Apple smartwatch. The results were nearly identical; however, the project outputs a BPM continuously while

the Apple smartwatch is every minute or so. When two sensors such as heart rate and GSR are connected to the Pmod AD2, the software starts to read from the channels sequentially. This creates a small delay that can corrupt the BPM calculations because reading two different channels and using the same interface rate, halves the rate at which one channel is read.

Initially the UART baud rate was 9,600 which was sufficient for reading from a single channel. To resolve this problem, the baud rate is increased to 115,200 which is 12x faster. This prints the measurement much quicker, which results in the next sequential read happening sooner. Therefore, the baud rate for the UART needs to be set to 115,200 otherwise the BPM calculation will be wrong.

**(e) Pmod OLED**

The OLED display is implemented from the IP catalog that allows writing to the display such as printing measurements, text, or image. The interface output port (Pmod_out) of the PmodOLED_0 needs to be made into an external connection as shown in Figure 25, allowing the pins to be mapped to the FPGA pins.

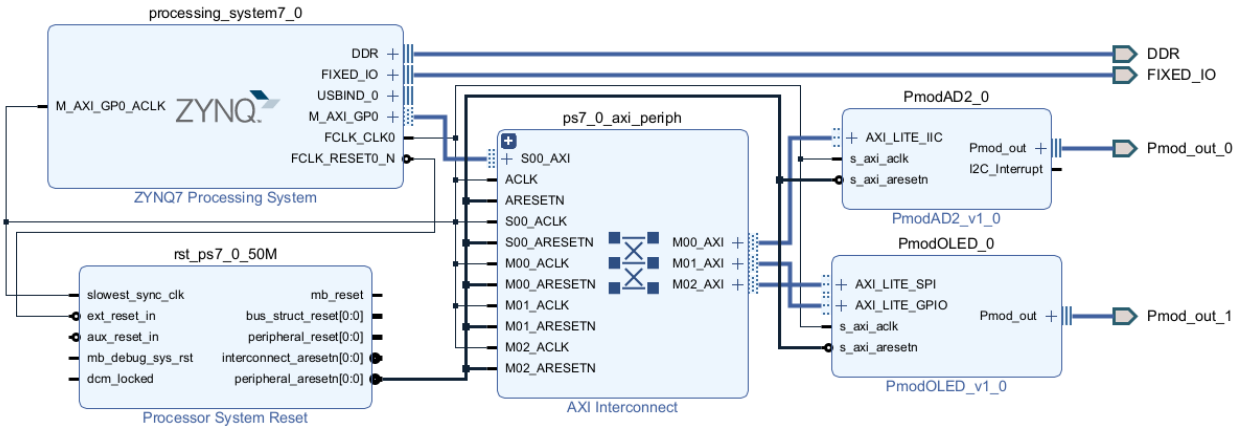


*Figure 25. Block Design w/ Pmod AD2 & Pmod OLED*

### (f) Pmod TC1

Just like previous Pmod's, this IP block is implemented from the IP catalog. The Pmod TC1 is a read-only device meaning that the Zynq (master) can only read from the module. Unlike the previous Pmod's, only four ports of the interface output (Pmod_out) of the PmodTC1_0 will be made into an external connection as shown in Figure 26, such as Pmod_out_pin1_o, Pmod_out_pin2_o, Pmod_out_pin3_o and Pmod_out_pin4_o. I renamed the external ports as CS, MOSI, MISO, and SCK (pin1-4) to map the FPGA pins easier.

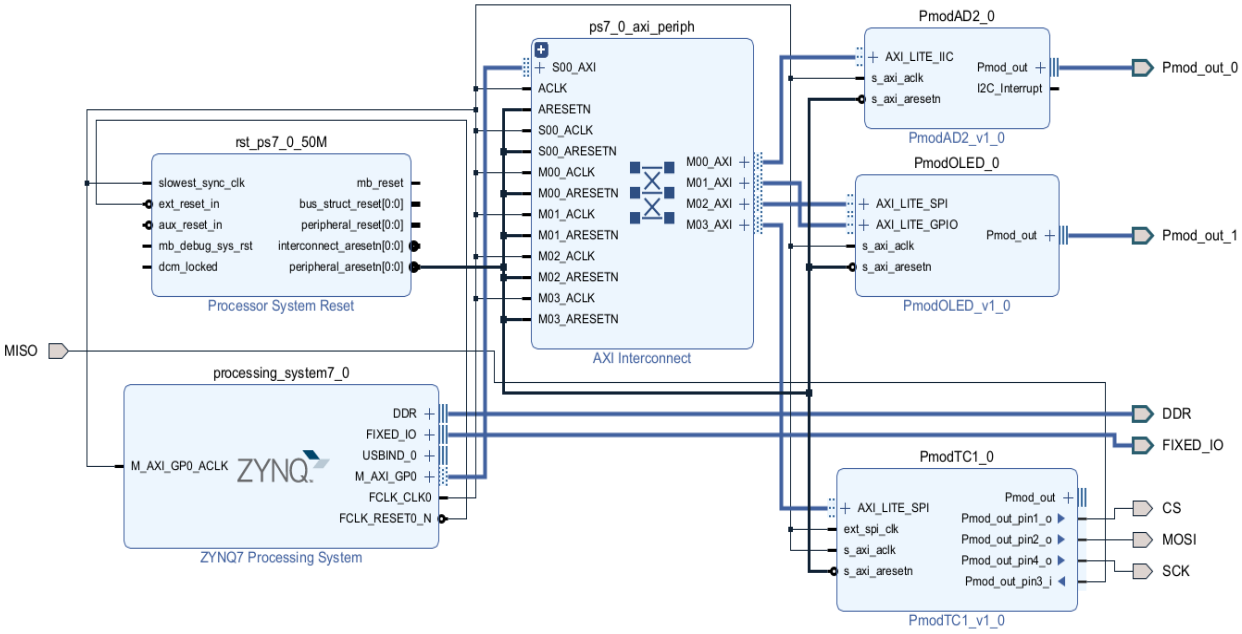


*Figure 26. Block Design w/ Pmod AD2, Pmod OLED & Pmod TC1*

## 3.2 Constraints

The Zybo Z7-20 has six Pmod ports with a variety of capabilities such as high speed, standard, MIO and XADC illustrated by Figure 27. The High-Speed Pmod ports are not necessary for the biosensor project, but to dedicate a Pmod sensor

to each Pmod connector, high speed ports JB-JD are used. These ports allow for maximum switching speeds by routing the data signals as impedance matched differential signals.

| | Pmod JA | Pmod JB* | Pmod JC | Pmod JD | Pmod JE | Pmod JF |
|---|---|---|---|---|---|---|
| Pmod Type | XADC | High-Speed | High-Speed | High-Speed | Standard | MIO |
| Pin 1 | N15 | V8 | V15 | T14 | V12 | MIO-13 |
| Pin 2 | L14 | W8 | W15 | T15 | W16 | MIO-10 |
| Pin 3 | K16 | U7 | T11 | P14 | J15 | MIO-11 |
| Pin 4 | K14 | V7 | T10 | R14 | H15 | MIO-12 |
| Pin 7 | N16 | Y7 | W14 | U14 | V13 | MIO-0 |
| Pin 8 | L15 | Y6 | Y14 | U15 | U17 | MIO-9 |
| Pin 9 | J16 | V6 | T12 | V17 | T17 | MIO-14 |
| Pin 10 | J14 | W6 | U12 | V18 | Y17 | MIO-15 |

*Figure 27. Zybo Z7-20 Pmod Pinout*

The Zybo Z7-20 pins follow the convention of Figure 28 and the interface output ports must be mapped to corresponding Pmod ports to interface with sensors and modules correctly. When mapping Pmod or I/O ports, note that the Zybo Pmod interfaces are general-purpose, so they do not include the pull-up resistors required. This leaves it up to the Pmod device to implement those pull-up resistors required by the I2C standard. These details can be found in the datasheet of the Pmod to determine whether those resistors are always present or whether jumpers need to be installed.

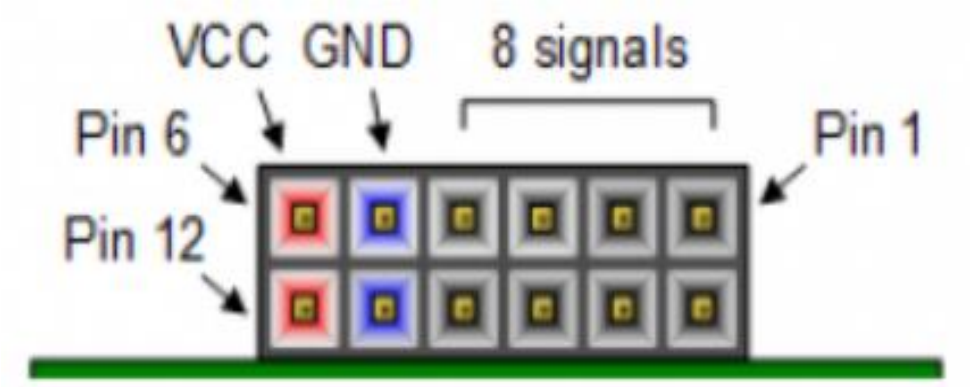


*Figure 28. Pmod Connectors*

After creating HDL wrapper of block design and successfully synthesizing design, the external output interfaces and ports are mapped creating a constraint file or configuring the I/O ports from IO planning layout. I synthesized the design and used the IO planning layout shown in Figure 29, since the graphical user interface (GUI) and constraint file, can have bugs like not setting the pins to correct Pmod port.

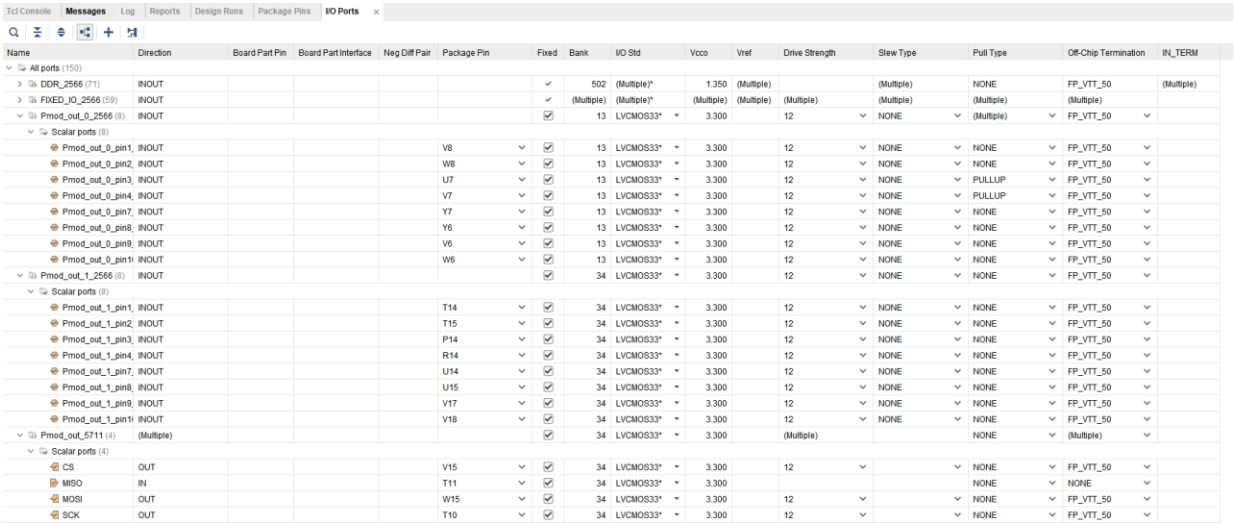


*Figure 29. I/O Ports setting on I/O Planning Layout*

The Pmod TMP3 has jumper pins that allow implementing pull-up or pull-down resistors, other modules provide an actual resistor ranging from 4.7k-10k, and other cases, they don't provide the resistor like the Pmod AD2. Note, the I/O planning layout displays all options for configuring the I/O ports like setting the I/O standard, drive strength, slew type, and pull type among others. Since the Pmod AD2 does not have pull up resistors, the pull type setting found in the I/O ports, is used to set pull up resistors for the Pmod ports associated with SCLK and SDA of the Pmod AD2.

Pmod ports JB-JD were utilized, and JB was mapped to Pmod AD2 (I2C), JC to Pmod TC1 (SPI), and JD to Pmod OLED (SPI). The I/O standard for all ports should be set to LVCMOS33 along with the Bank VCCO voltage set to 3.3V. The Pmod ports are GPIO's that can be used as either input, output or bidirectional ports. All constraints for the Pmod's are listed Table 1, Table 2, and Table 3 that map the ports to a designated FPGA Pmod pin; they are also listed on Figure 29. Note, constraints are used to map the Zynq PL I/O pins to the FPGA Pmod pins.

*Table 1. Constraints for Pmod AD2*

| Block Diagram Port: Pmod_out_0, Figure 24 | Package pin | Pmod pin |
|---|---|---|
| Pmod_out_0_pin1 | V8 | JB1 |
| Pmod_out_0_pin2 | W8 | JB2 |
| Pmod_out_0_pin3 | U7 | JB3 |
| Pmod_out_0_pin4 | V7 | JB4 |
| Pmod_out_0_pin7 | Y7 | JB7 |
| Pmod_out_0_pin8 | Y6 | JB8 |
| Pmod_out_0_pin9 | V6 | JB9 |
| Pmod_out_0_pin10 | W6 | JB10 |

*Table 2. Constraints for Pmod TC1*

| Block Diagram Port: Figure 25 | Package pin | Pmod pin |
|---|---|---|
| CS | V15 | JC1 |
| MOSI | W16 | JC2 |
| MISO | T11 | JC3 |
| SCK | T10 | JC4 |

*Table 3. Constraints for Pmod OLED*

| Block Diagram Port: Pmod_out_1, Figure 26 | Package pin | Pmod pin |
|---|---|---|
| Pmod_out_0_pin1 | T14 | JD1 |
| Pmod_out_0_pin2 | T15 | JD2 |
| Pmod_out_0_pin3 | P14 | JD3 |
| Pmod_out_0_pin4 | R14 | JD4 |
| Pmod_out_0_pin7 | U14 | JD7 |
| Pmod_out_0_pin8 | U15 | JD8 |
| Pmod_out_0_pin9 | V17 | JD9 |
| Pmod_out_0_pin10 | V18 | JD10 |

# 4. Verification

## 4.1 MATLAB

A Zynq processor application was created that read from the ECG ADC and wrote samples to the UART. The ADC samples were copied from the serial console into a text file. MATLAB was used to read the text file, plot the data as illustrated on Figure 30 and determine an appropriate threshold value for the BPM algorithm using a histogram. The very first samples plotted, had peaks saturating near the peak pulse or flat in other words. The typical ECG signal does not have flat peaks or pulses but by zooming in on Figure 30, the signal has the main characteristics of an ECG signal with flat peaks. The flat peaks could be happening due to probing issues, light interfering with photosensor, cheap sensor or the ADC code is at maximum. Despite the flat peaks, the BPM algorithm can still calculate a reasonable BPM.

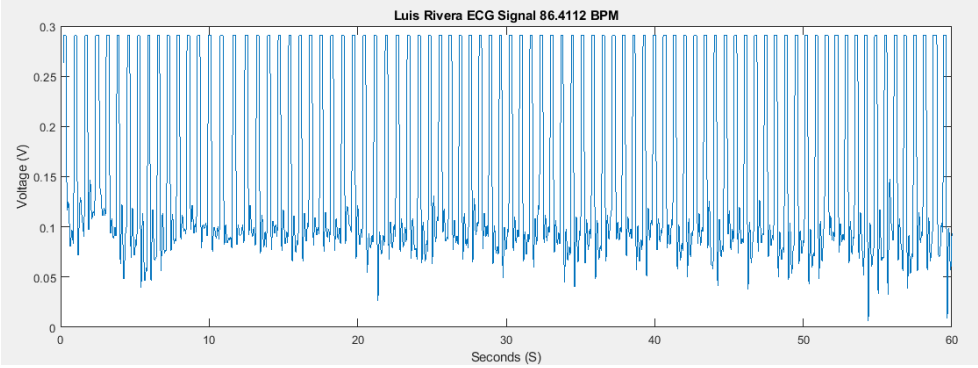


*Figure 30. ADC Data from Pulse Sensor Example 1 (Flat Peaks)*

MATLAB has useful built-in functions that can simplify calculating BPM by using findpeaks() and diff(). The findpeaks() function finds the local peaks in a data vector and can return the index of each peak. Since the x-axis contains the time information in seconds, any two neighboring peaks are sufficient to calculate a BPM. By using the diff() function, all elements in a data vector can

be subtracted (x[n]-x[n-1]) for sample n. The ECG signal rising-edge to rising-edge, provides the amount of ADC samples per pulse. Using the time between ADC samples, the samples per pulse is converted to seconds per pulse, and then that is converted to BPM. The average can be taken of all BPM calculation to determine the average BPM in that 1-minute duration. Comparing the results from MATLAB vs Zynq-7000 software for BPM, they are nearly identical every run and this is even with clipped data.

To optimize the BPM algorithm, a histogram plot was used that provided a visualization of the data and helped determine a reasonable threshold for filtering false peaks or bad data. The histogram plots the data and displays the occurrence of each ADC code. This helps visualize the max and mean values in the data, which help optimize filter performance by selecting a threshold that is always above the mean and below the max. If the threshold value is always greater than the mean and about 5-30% less than the max, this filters false peaks. Figure 31 illustrates the histogram of the ADC data from Figure 30, the pink represents the mean, green represents optimal threshold and red is the max value.

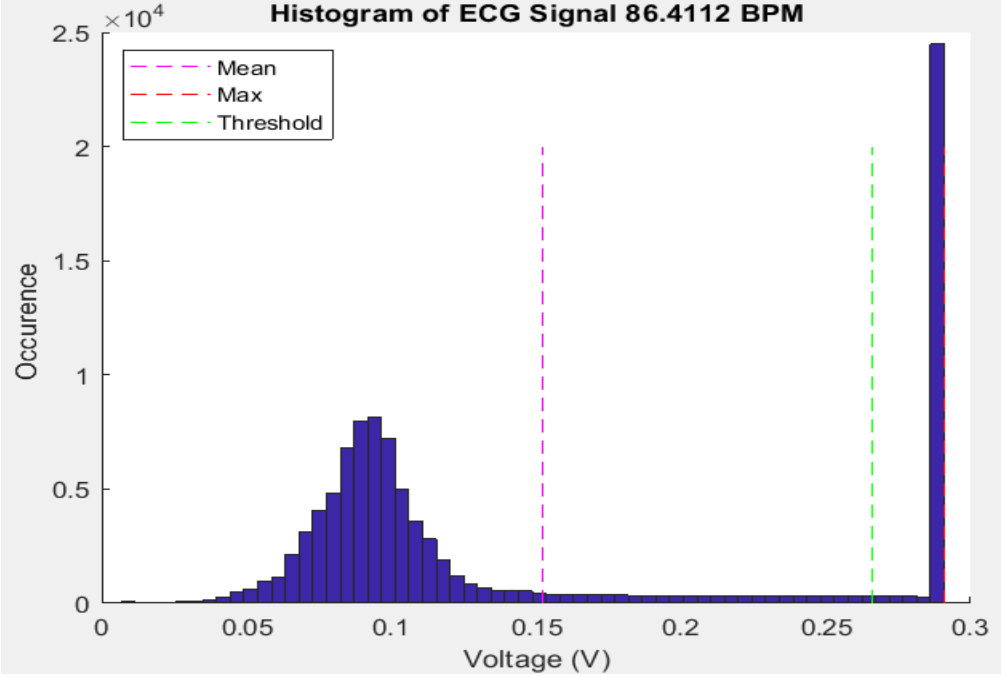


*Figure 31. Histogram of Pulse Sensor Example 1*

Figure 32 illustrates a new set of ADC samples plotted on MATLAB, note, this data does not have any flat peaks. Comparing BPM algorithm for the flat data and clean data, results are identical due to the threshold selected from the histogram. Figure 33 illustrates the histogram applied to Figure 32 and histogram is right-skewed just like previous Figure 30. Note, the new histogram does not have many occurrences of the max value, implying the data does not have flat peaks near the max.

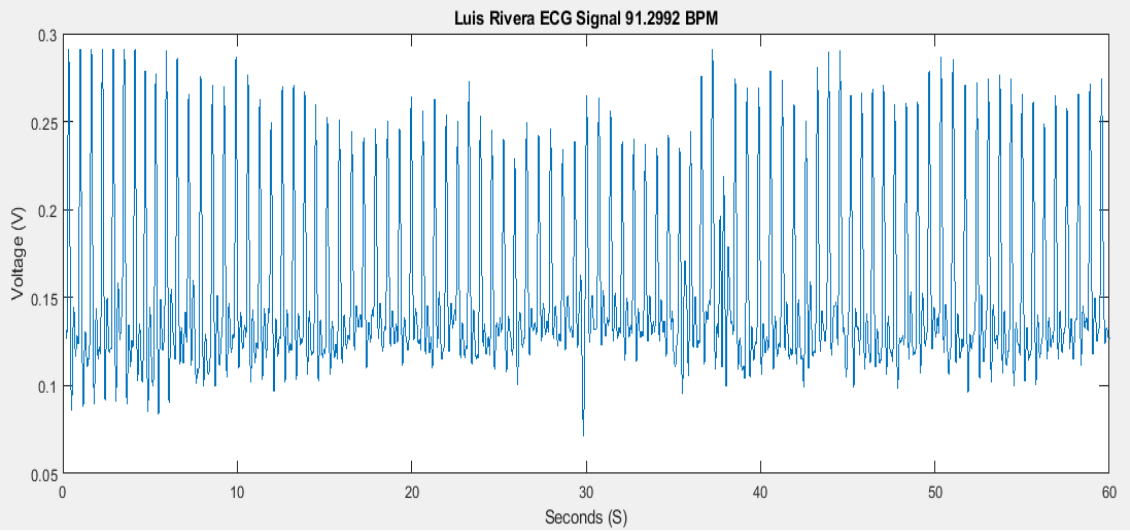


*Figure 32. ADC Data from Pulse Sensor Example 2 (Clean Data)*

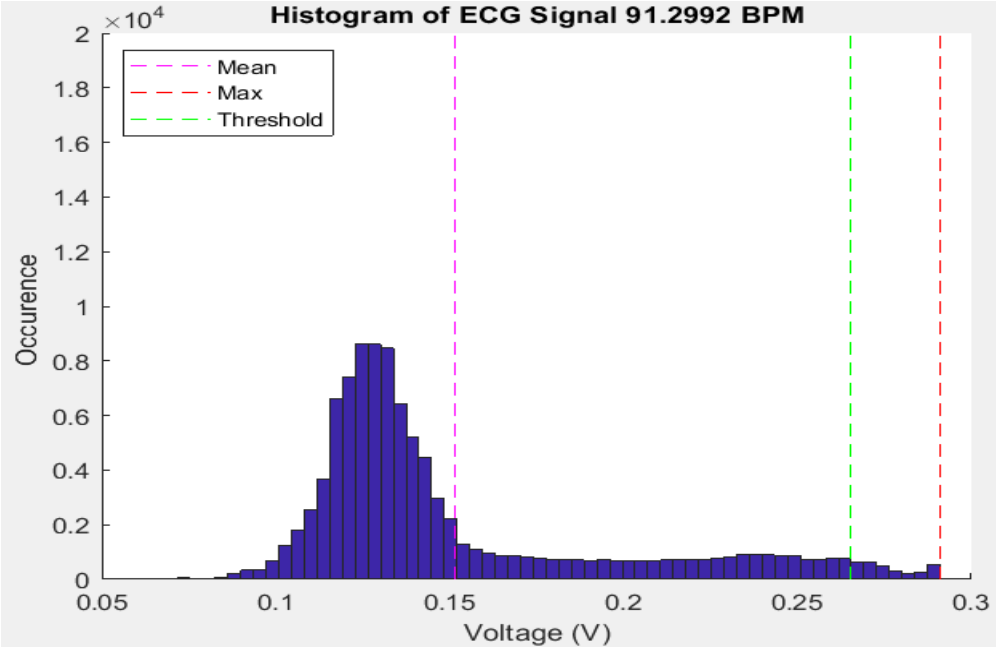


*Figure 33. Histogram of Pulse Sensor Example 2*

The first 30 seconds of the ADC data are illustrated by Figure 34, and it can be seen, that the ADC data processed by the development board can output a clean ECG signal.

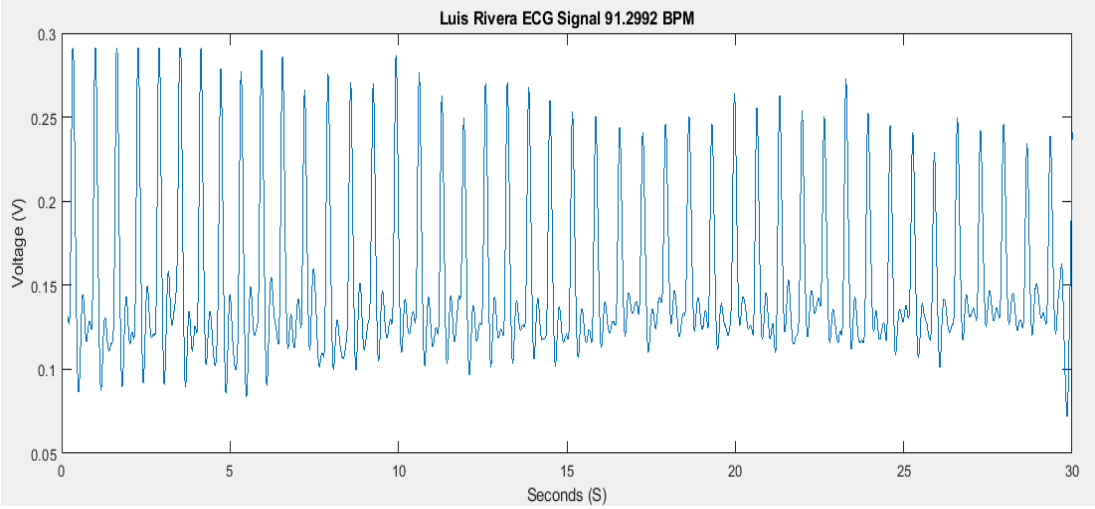


*Figure 34. First 30 seconds of ADC Data from Pulse Sensor Example 1*

The power spectrum displayed on Figure 35 plots the signal power against the frequencies. The spectrum can be calculated by performing the DFT or FFT of the ADC samples. The use of this graph helps select the appropriate sampling rate. Note FFT is the preferred operation instead of DFT, since it requires less resources; hence, the computation is less intensive and more efficient [39].

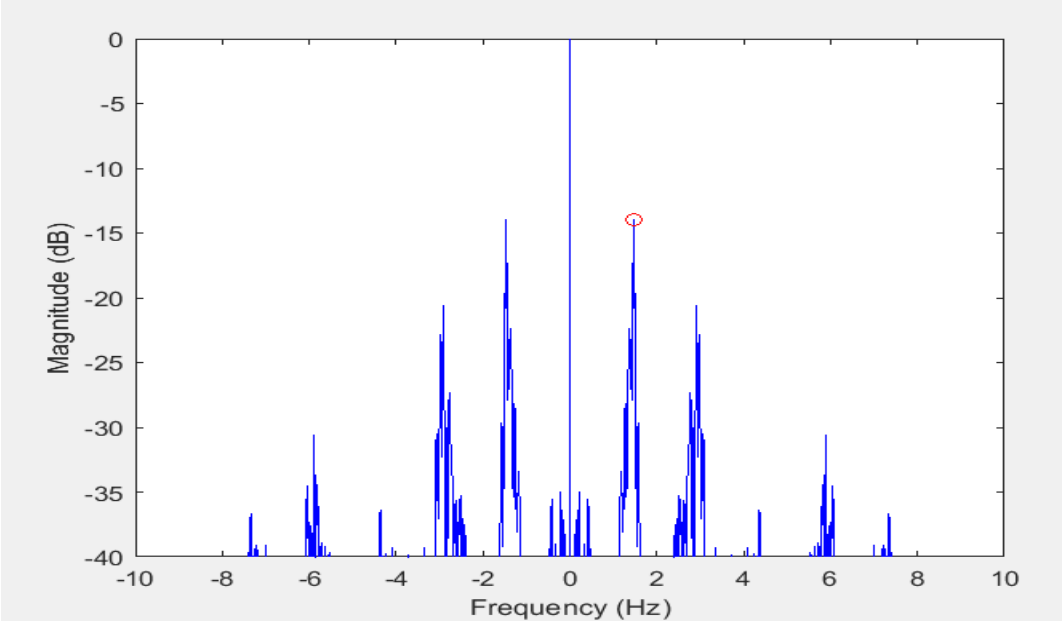


*Figure 35. Power Spectrum of ADC Data from Pulse Sensor Example 1*

### 4.2 Testing Heart Rate

The sensors were tested in stand-alone configurations as shown in Figure 36, and then they were integrated together to create the biosensor design. Initially the BPM algorithm was tested for reasonable measurements and after testing, a reference sensor was used to verify each measurement along with MATLAB to debug the algorithm. The external reference sensor is a Samsung Galaxy Watch5 that can measure BPM [40] . MATLAB is used to plot the raw ECG signal, which allows us to probe the signal and observe the signal characteristics.

Figure 36 displays my BPM measurement on Smartwatch and also shows BPM algorithm results. Unlike the smartwatch, the BPM algorithm outputs a continuous measurement every rising-edge detected, while the smartwatch has intervals or must be set manually. In this experiment, the results were nearly identical (smartwatch 86 BPM v. algorithm 75-86 BPM). Figure 37 illustrates another test created on my girlfriend with an Apple Smartwatch [41] and results were nearly identical as well.

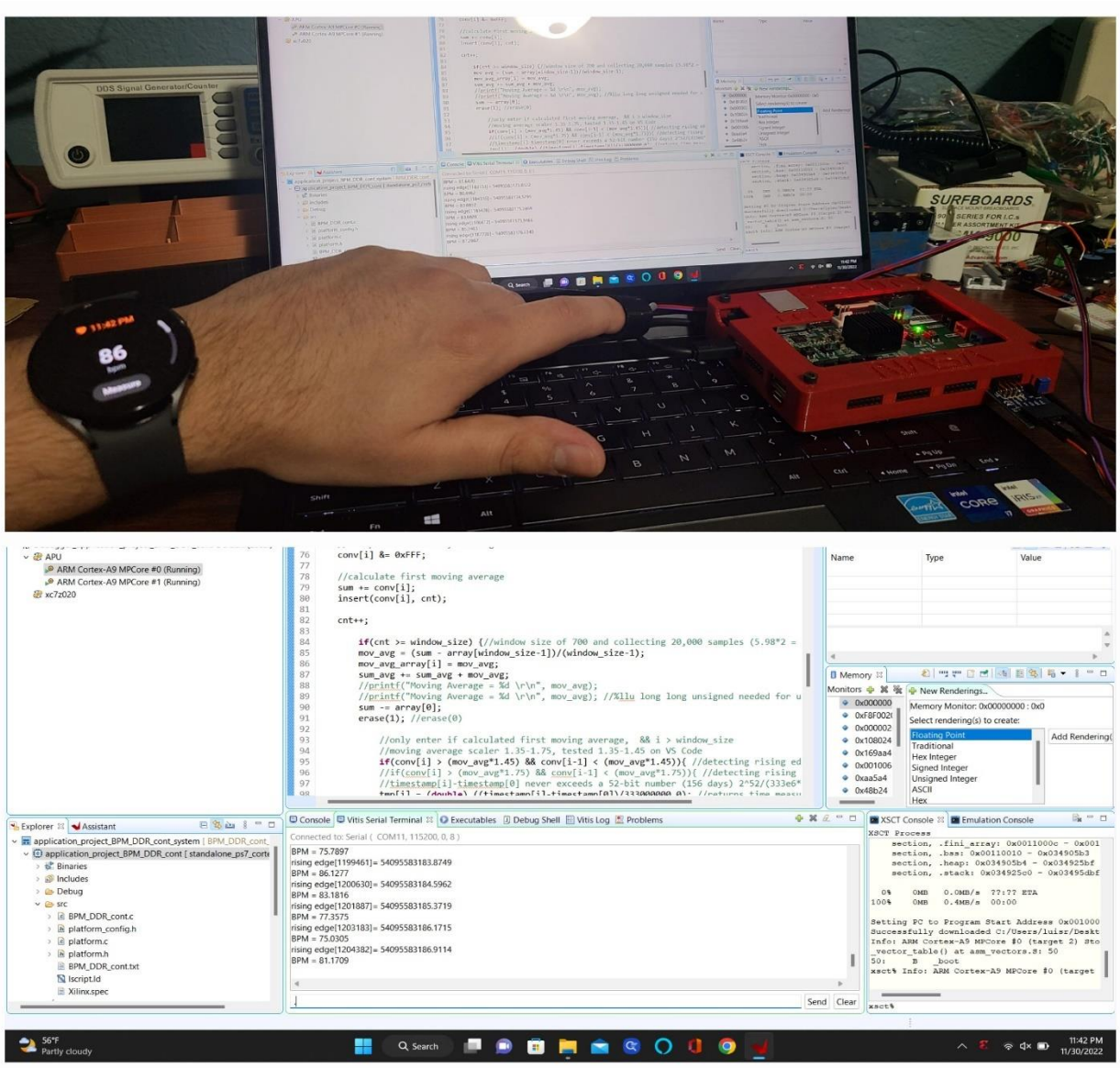


*Figure 36. Pulse Sensor Results (Resting) Example 1*

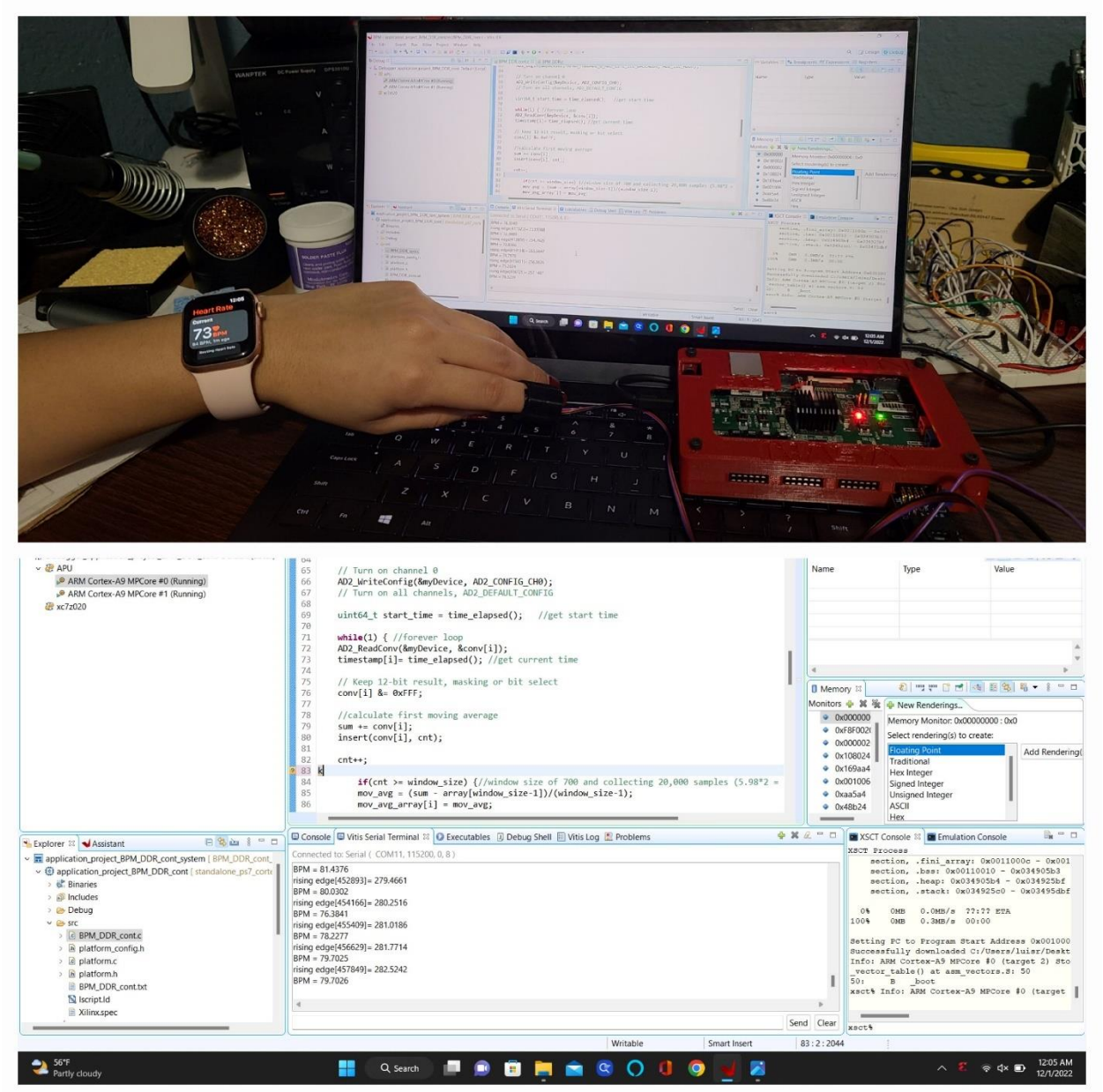


*Figure 37. Pulse Sensor Results (Resting) Example 2*

Figure 38 and Figure 39 illustrate the resting BPM before doing a mini random workout that consisted of squats, jumping jacks, pushups and running to elevate BPM. In both figures, the resting heart rate is less than 100 BPM and then after exercising, BPM increases to over 120 BPM. Measurements from the BPM algorithm are consistent with Smartwatch.

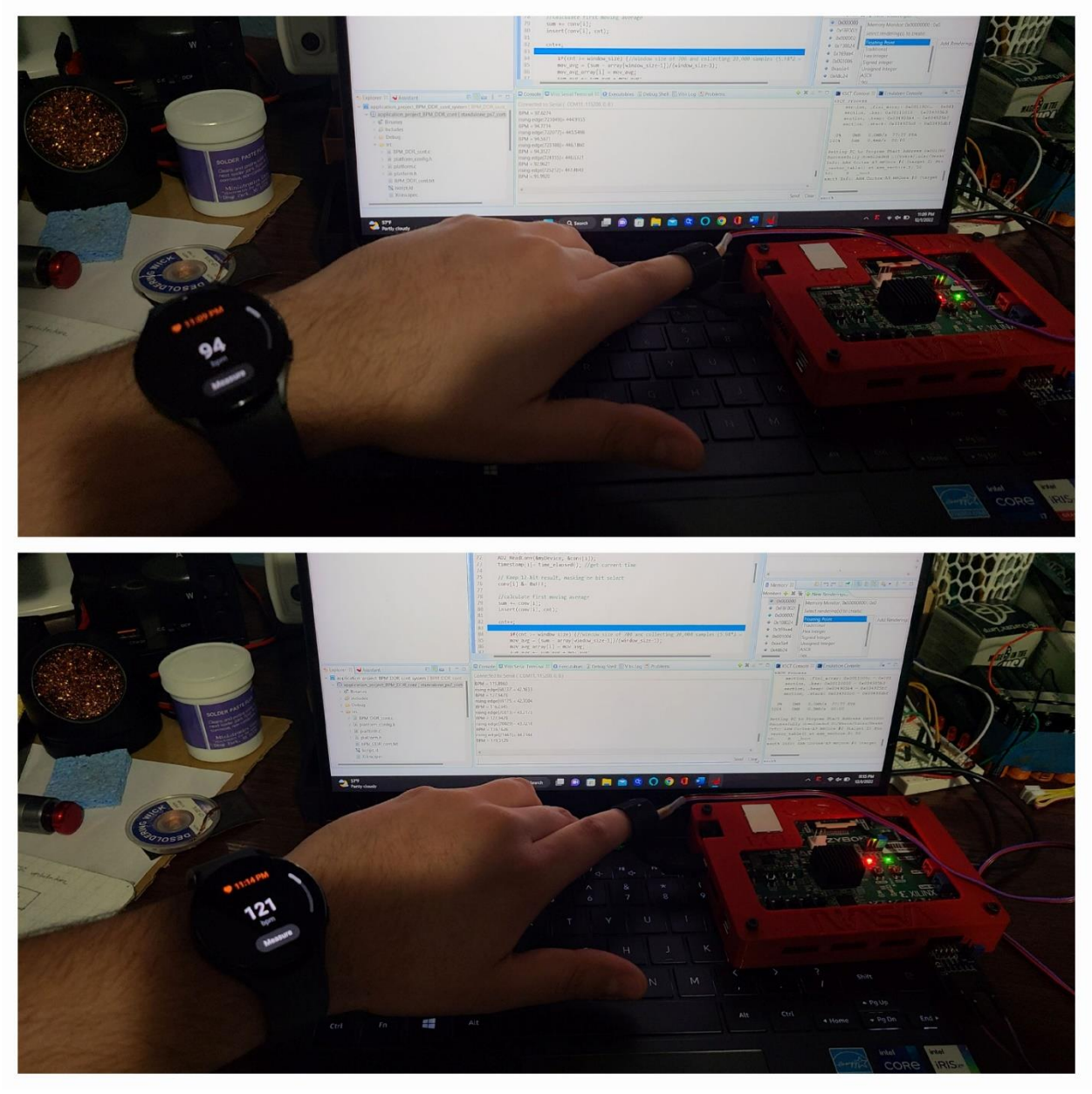


*Figure 38. Pulse Sensor Results (Exercise) Example 1*

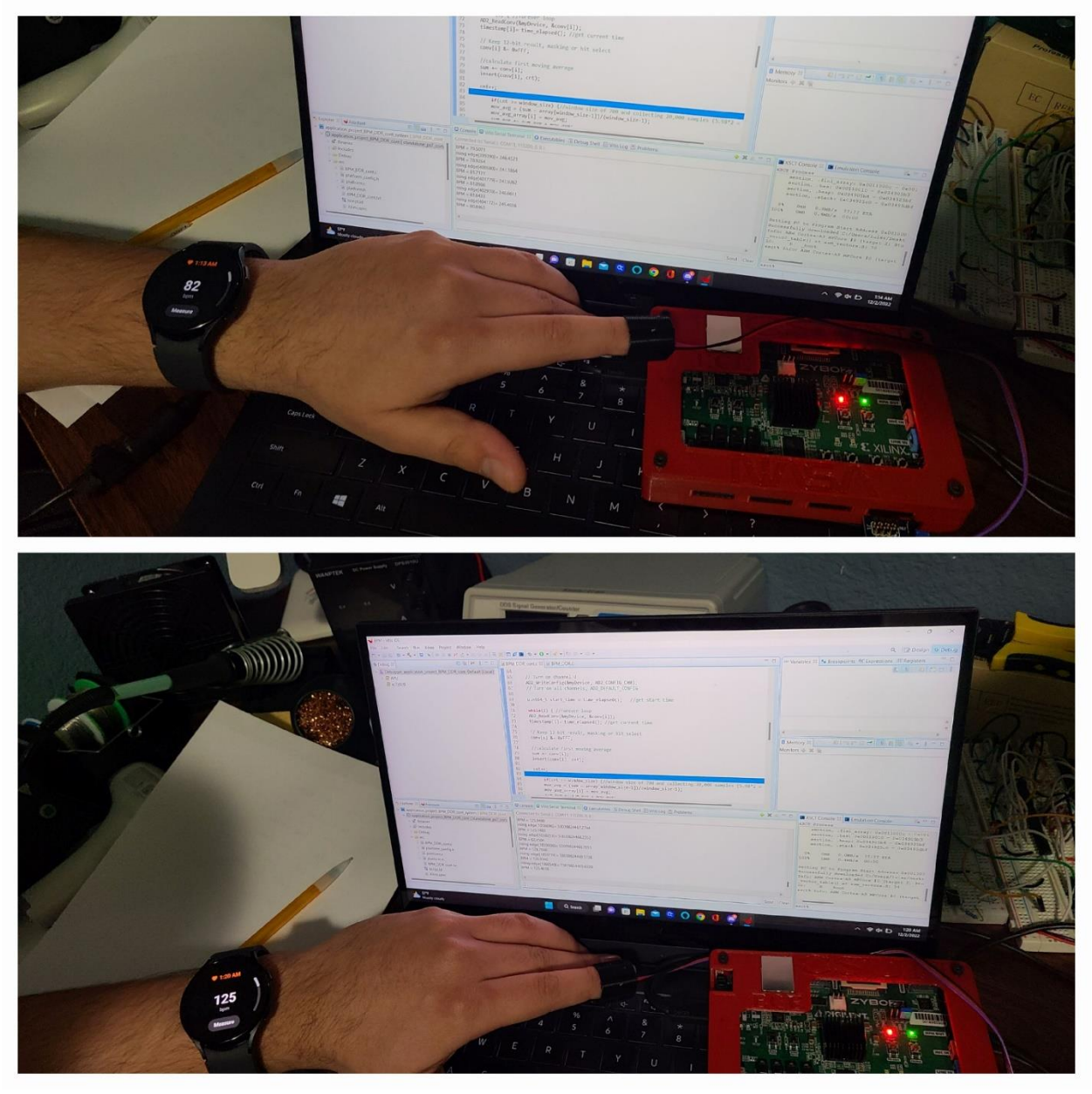


*Figure 39. Pulse Sensor Results (Exercise) Example 2*

Data was collected from the exercise experiments and every minute; the BPM were recorded from the Smartwatch and algorithm. Figure 40 plots all the BPM results from Smartwatch and algorithm to compare measurements.

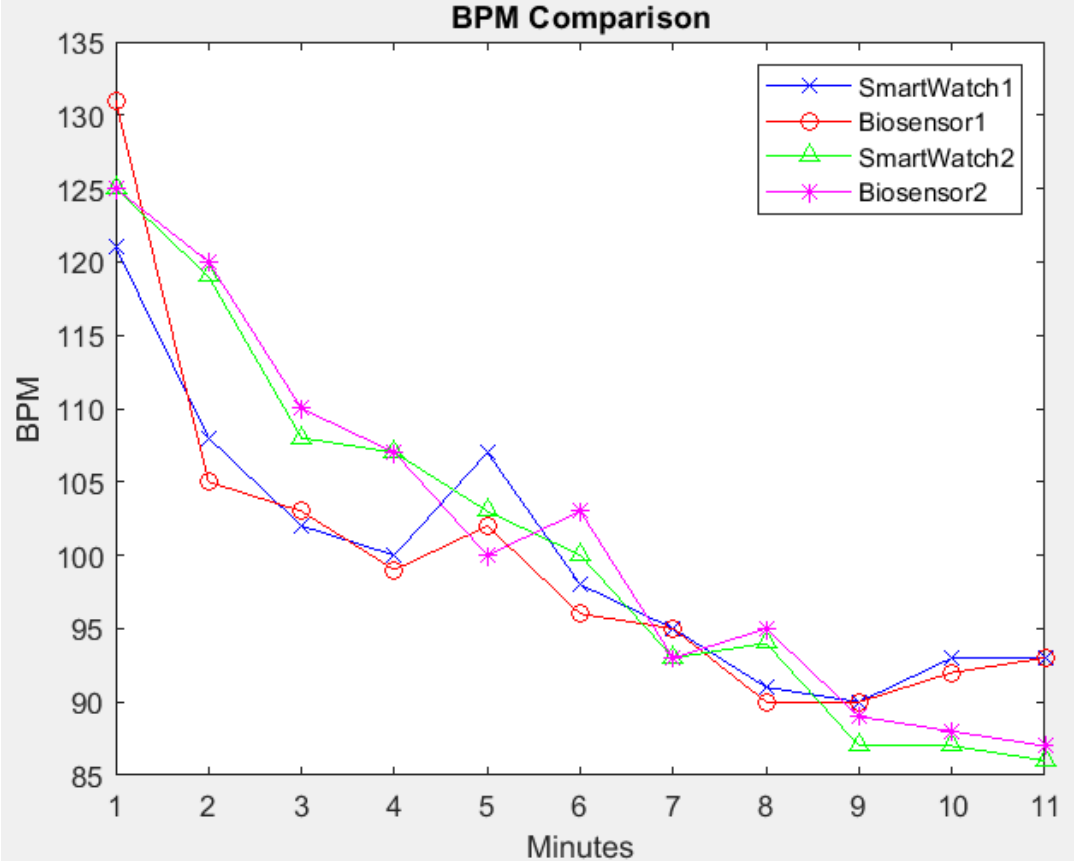


*Figure 40. Comparing BPM Results (Exercise)*

Another set of verification is an empirical observation by using the plotted ADC data. If we look at the first 10 seconds and measure the pulses in this 10 second interval, if there are 15 pulses, it can easily be determined that the expected pulses in a minute, would be (15 pulses per 10 seconds) x (6) is equivalent to 1 minute of pulses or in other words, 90 BPM. By using this empirical method, the Zynq-7000 software and MATLAB algorithm are identical.

### 4.3 Testing GSR

For testing GSR results, the voltage from the GSR sensor is interpolated into a resistance using the table from Figure 41 and then conductance can be calculated by applying the inverse. An algorithm on MATLAB was developed to

determine the incrementing resistance in each voltage range. With the incremental resistance for each voltage range calculated by MATLAB, an equation can be created to determine the skin resistance.

| Rs | Vout |
|---|---|
| 10 k | 1.755 V |
| 49.5 k | 1.677 V |
| 100 k | 1.587 V |
| 200 k | 1.434 V |
| 560 k | 1.054 V |
| 760 k | 0.923 V |
| 1 M | 0.813 V |
| 3.3 M | 0.357 V |
| 9.93 M | 0.136 V |

*Figure 41. GSR Voltage and Resistance Table*

Two methods for testing were used such as validating the GSR measurement with table and using an external reference sensor. The external reference sensor is a GSR logger sensor made by Neulog [42]. By using the table method, results are as expected since the algorithm uses the table to perform calculation. However, when using the Neulog reference sensor, the GSR algorithm and Neulog datalogger program have different results.

Figure 42 illustrates the results from the GSR algorithm in comparison to the Neulog datalogger. From testing, the GSR algorithm was always about 5-10 times greater than the Neulog datalogger results. With the given time, I was not able to determine why the results are much different. One speculation is that the potentiometer from the GSR sensor is damaged. Reading the online user guide

from seeed studio [43], the onboard resistor needs to be calibrated to a set a value but when calibrating, the ADC range gets skipped. Therefore, I adjusted the resistor to the closest value but even then, results are different.

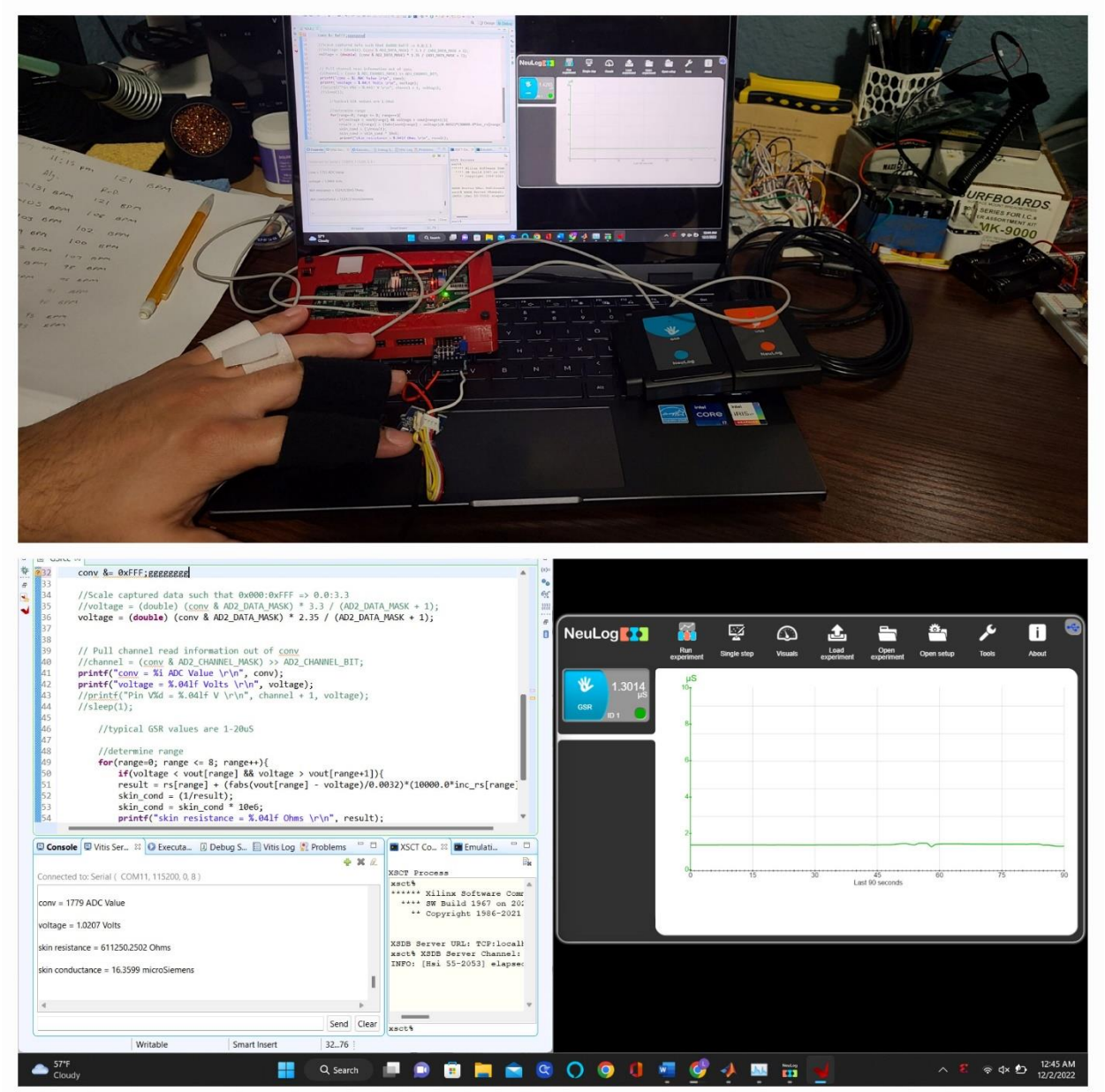


*Figure 42. GSR Sensor Results*

### 4.4 Testing Temperature

A non-contact thermometer was used as an external reference sensor to verify the temperature measurements. Figure 43, Figure 44, and Figure 45 illustrate the different scenarios tested on the Pmod TC1 such as ambient, warm, and cold environments. For testing warm, I was holding the sensor since I was indoor and am relatively warm, while testing for cold required a small Ziploc bag with lots of ice. The ambient environment is simply the room temperature. In all scenarios, the temperature adjusted appropriately, and output was nearly identical to thermometer measurement.

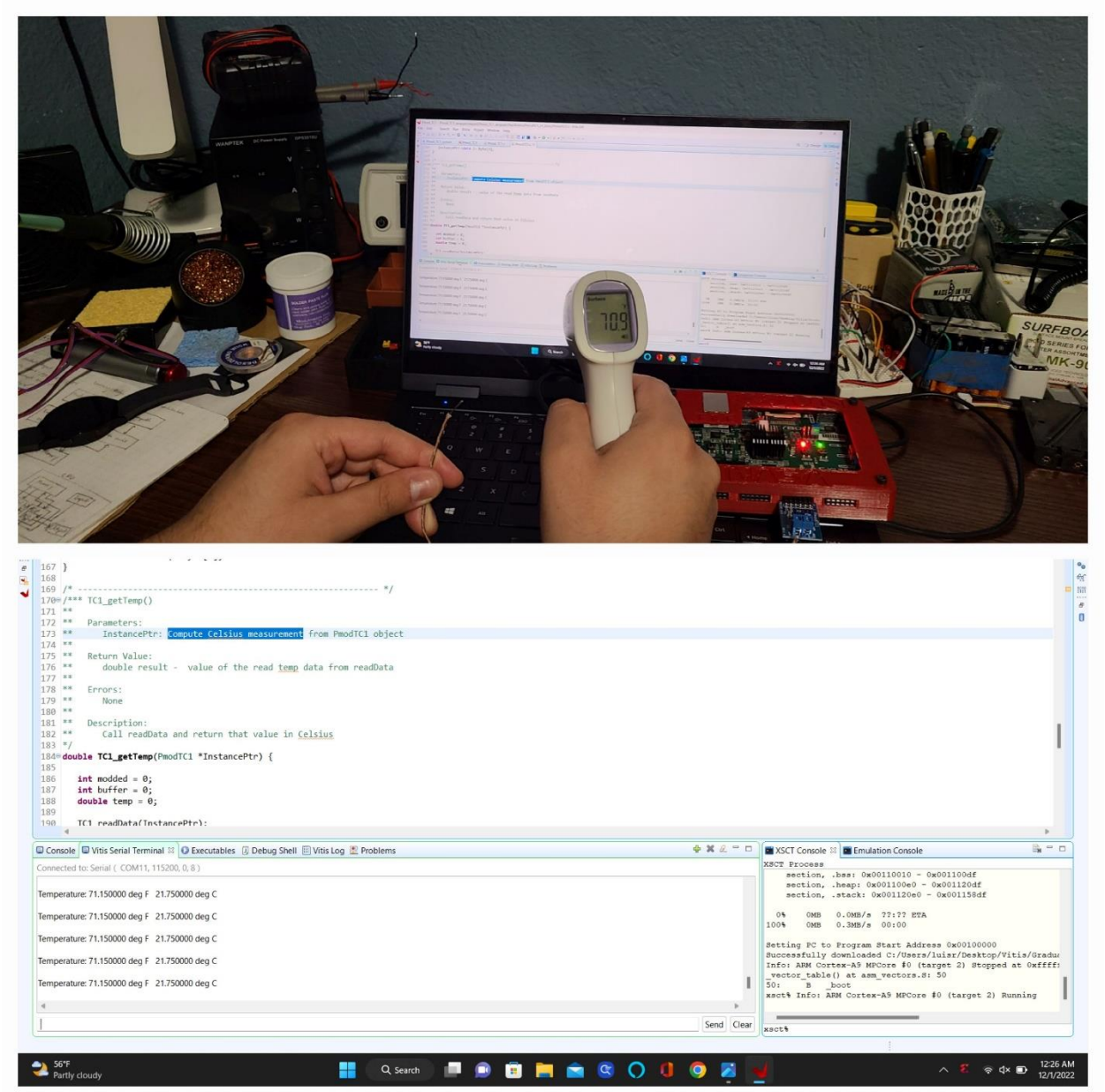


*Figure 43. Temperature Sensor Results (Ambient Temp.)*

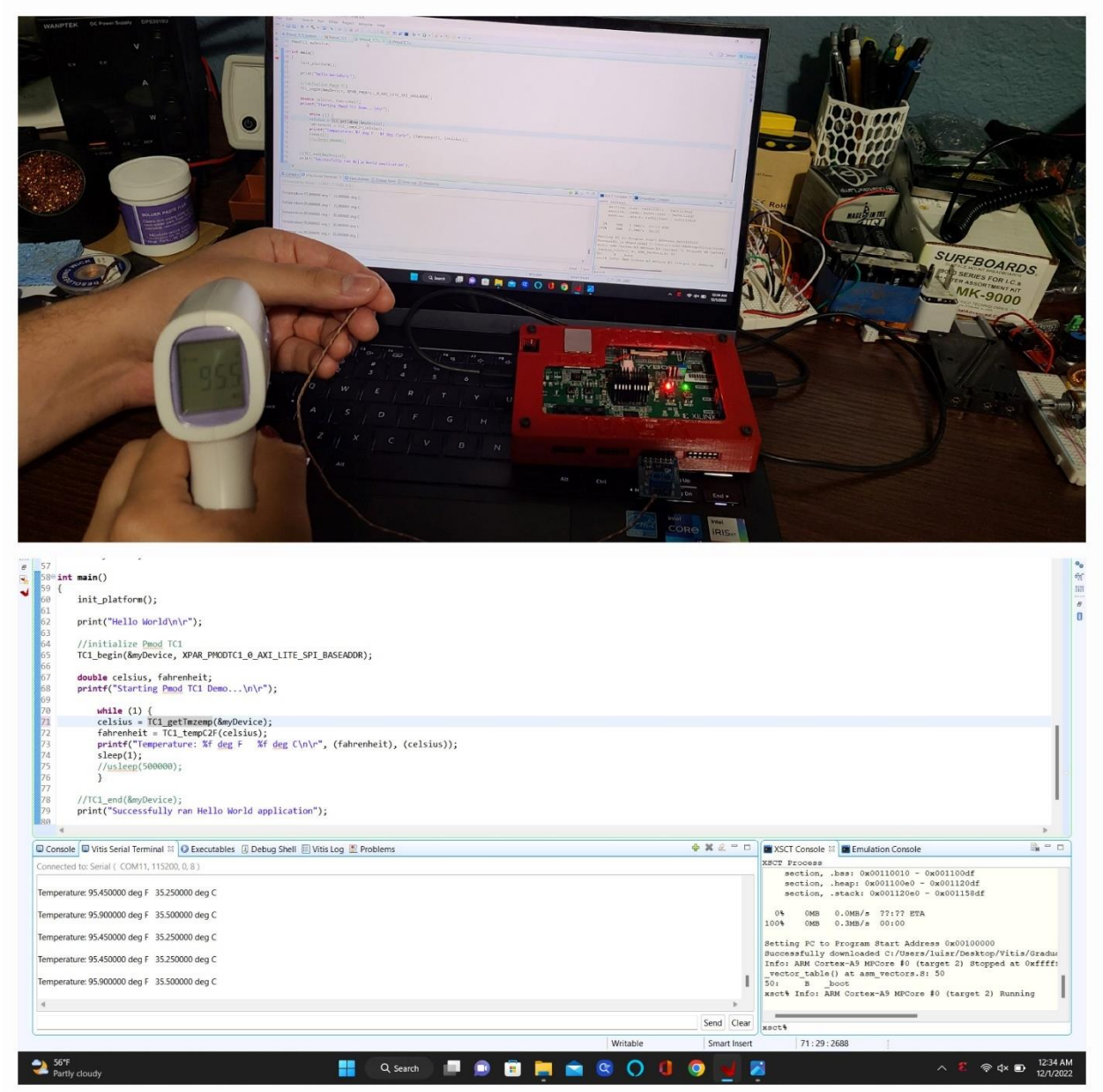


*Figure 44. Temperature Sensor Results (Body Temp.)*

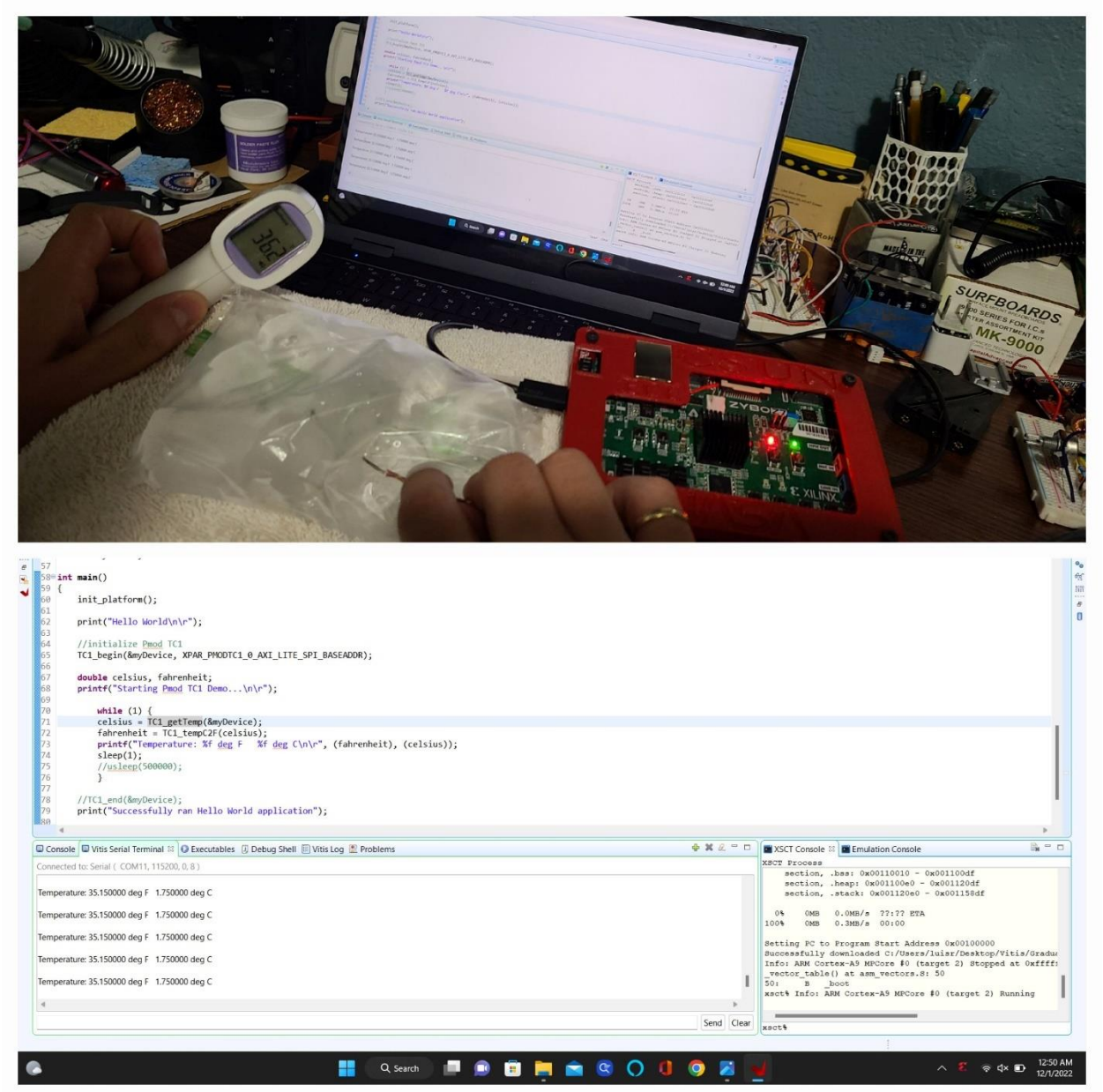


*Figure 45. Temperature Sensor Results (Cold Temp.)*

### 4.5 Testing OLED Display

Figure 46 demonstrates the OLED is working and this is the simplest module to test functionality.

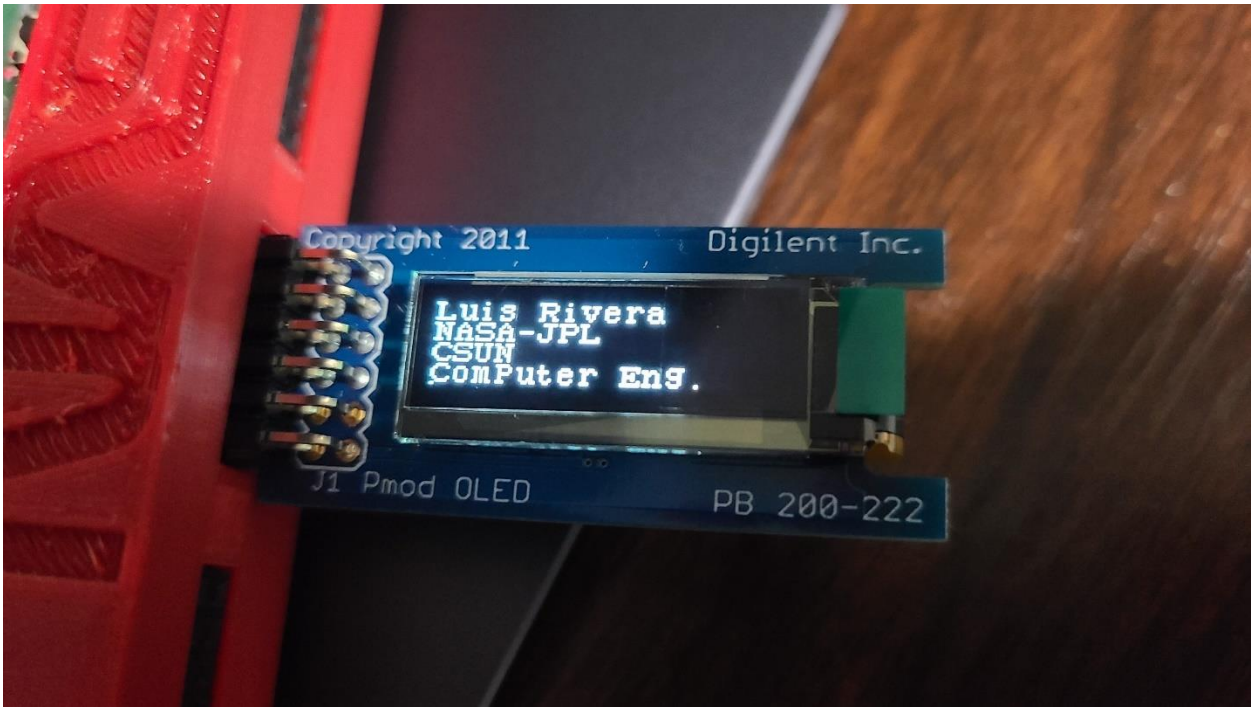


*Figure 46. OLED Display Results*

### 4.6 Results

Each sensor was tested stand-alone before integration and results for each sensor are reasonable and within range of the external reference sensor except for the GSR. The objective of the stand-alone testing was to demonstrate that each of the sensors operated correctly. If problems were experienced in the integrated biosensor system, then the sensor could be re-tested in stand-alone mode to determine if the sensor had been damaged, or whether the sensor algorithm was the source of the problem when operating in the integrated system. Figure 47 illustrates the hardware setup for being connected to the biosensor system and

Figure 48 provides a closer look at the Zybo-Z7 connected to the sensors.

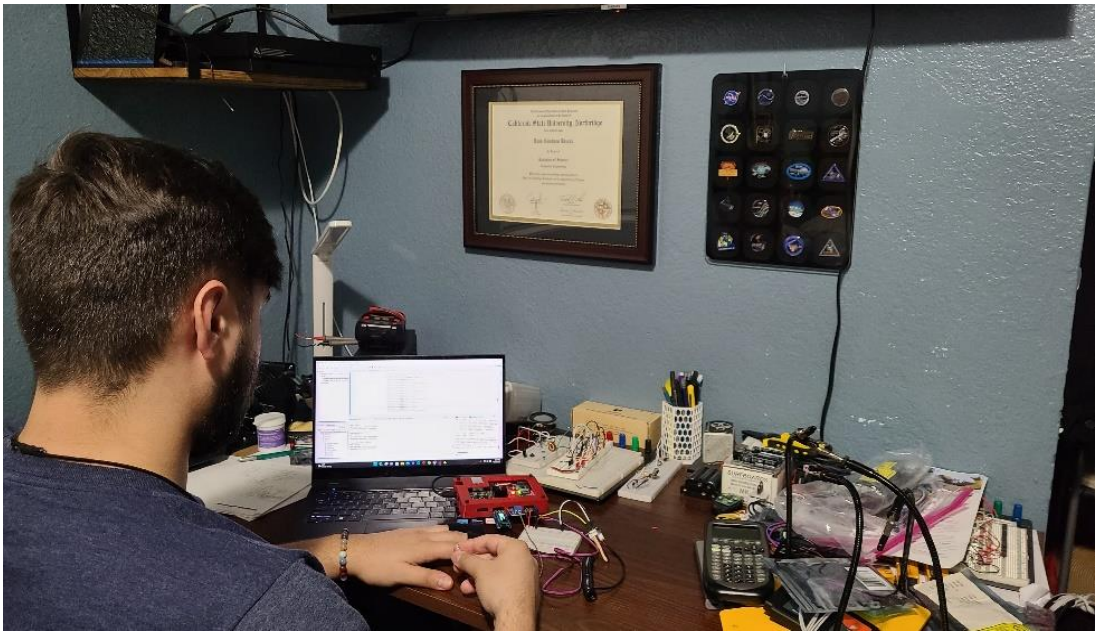

*Figure 47. Image of project in action*

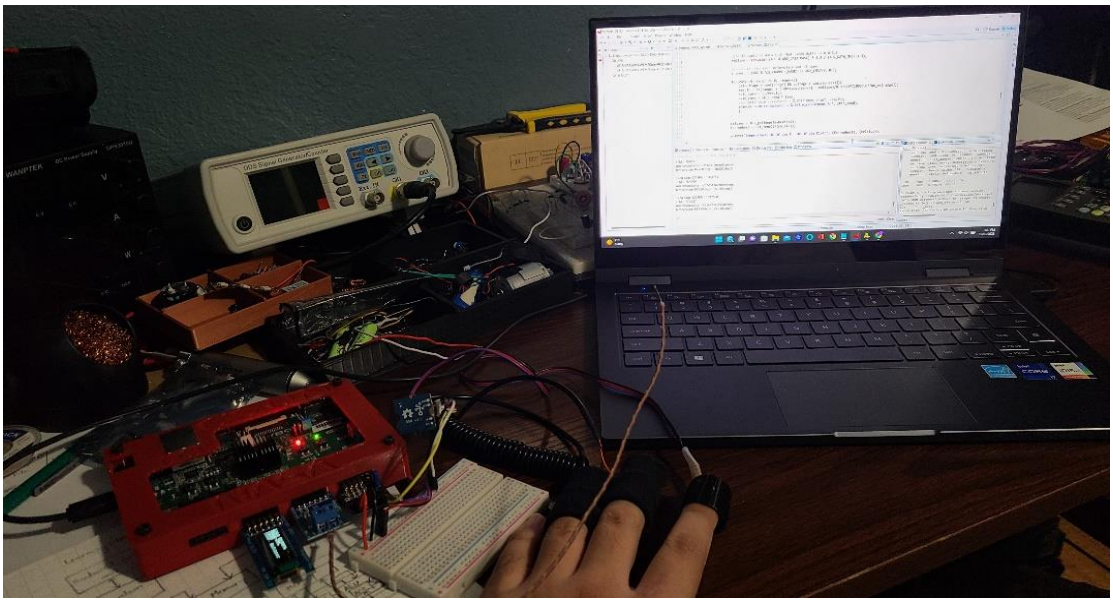

*Figure 48. Zybo-Z7 connected to sensors*

All sensors even after integration still had accurate measurements but it was

noticed that the BPM would be wrong every other calculation. During integrating & testing, the problem of the Pmod AD2 reading sequentially from two channels, explained why the integrated biosensor was having wrong BPM. The method to resolve this was increasing the UART baud rate, explained in Pmod AD2. The results for the project demonstrate that the biosensor system.

### 4.7 Proof-of-concept

To understand how to use the PL (FPGA) to create custom logic, a proof-of-concept implementation was created for integrating a stress level algorithm. The stress level algorithm was implemented in Register Transfer Level (RTL) code, allowing the algorithm to be converted as a logic block, so it can be added to the biosensor system. Integration of a custom RTL module required understanding how to use Vivado to add custom IP to block design.

Creating custom IP's and interfacing with other PS and PL peripherals requires an understanding of communication protocols. IO ports on a custom IP block can't be connected unless they are AXI compatible or using the same communication protocol. So, when using AXI stream or any communication protocols, they need to be implemented in the source code. Figure 49 illustrates the GPIOs used to drive and poll the signals since the stress level algorithm was not compatible. For high-speed processing, GPIO method should be avoided since the processor accesses to AXI4-Lite GPIOs are slow.

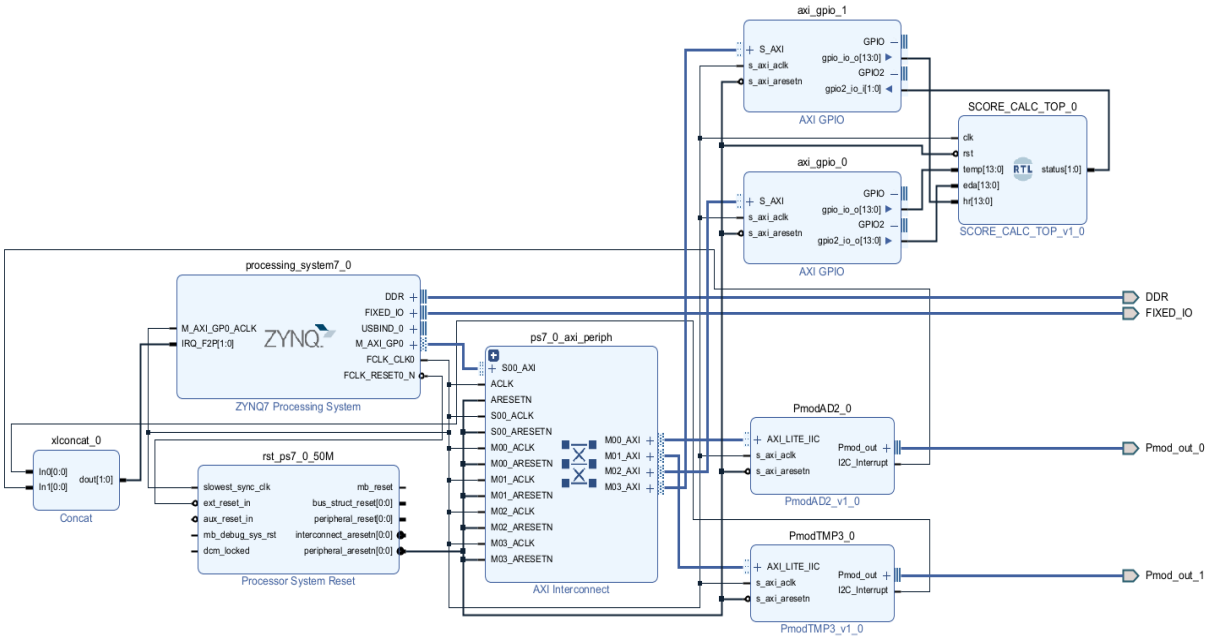


*Figure 49. Sensors w/ Stress Level Algorithm*

Figure 50 is a simplified block design of the Pmod IPs, since PmodTMP3_0 is an SPI controller, PmodAD2_0 is an I2C controller and the stress level algorithm is connected via GPIOs. Similar design can be implemented using an SPI and I2C controller, and two GPIO controllers.

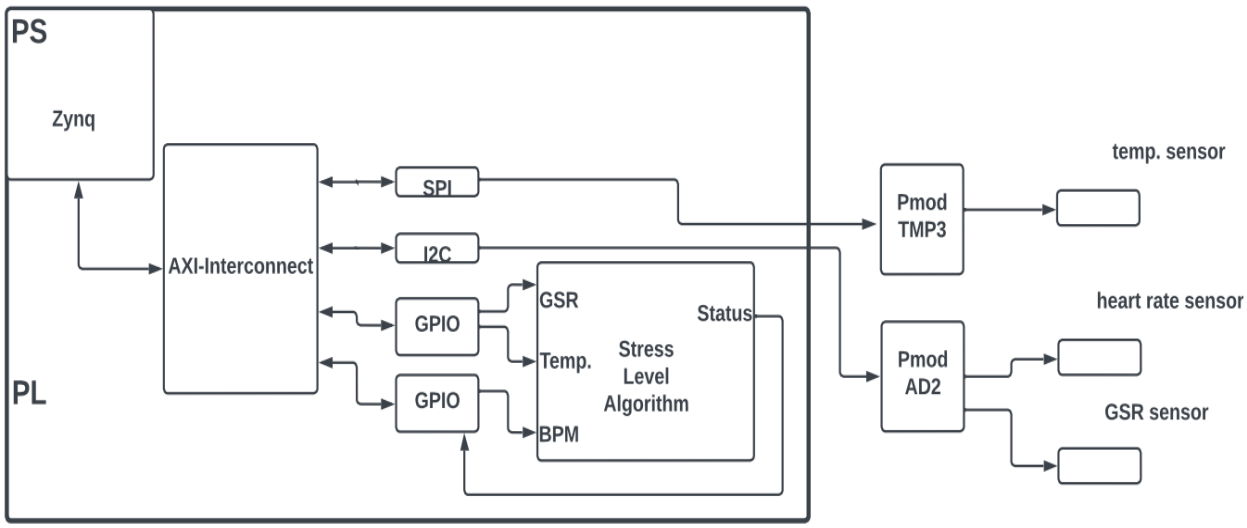


*Figure 50. Simplified block design*

No testing was done on this implementation since the goal was to determine if RTL code can be integrated to biosensor system. The stress level algorithm has an output result that determines whether you are stressed or not. Figure 51 illustrates the results from the integrated stress level algorithm with a result of "not stressed". From the minor testing done, the two subjects tested were never stressed. This doesn't prove anything aside from that a stress level algorithm can be added to perform analysis on biometric data.

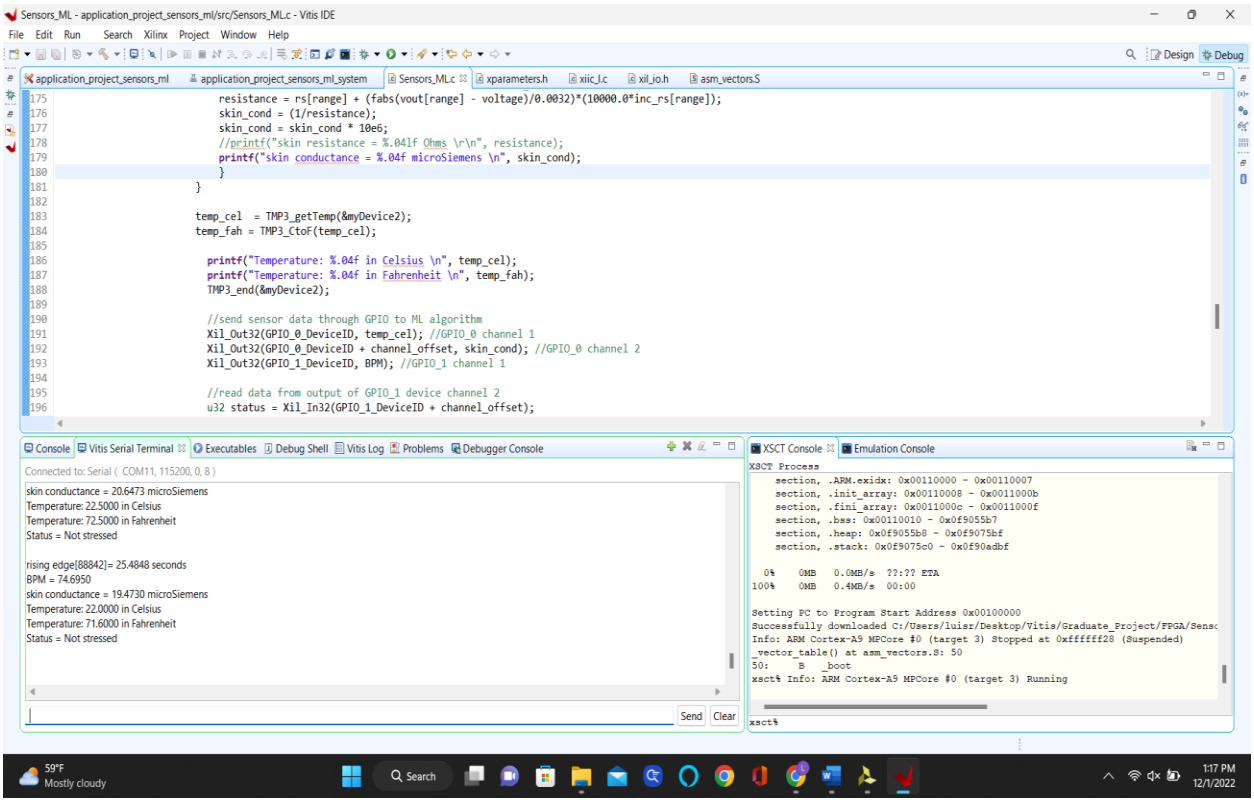


*Figure 51. Sensors w/ Stress Level Algorithm Results*

# 5. Future Work

## 5.1 Bluetooth

Just like the measurements can be sent from the FPGA to an OLED display, similarly Bluetooth can be integrated to send data using radio waves to a nearby Bluetooth device. Bluetooth is another communication protocol similar to UART but instead better, wireless UART. These devices tend to be low-power Internet-of-things (IoT) devices since they require less energy to transmit a signal. A limitation to take in consideration is that transmitting and receiving devices need to be within proximity of 30ft.

## 5.2 WIFI

Another popular choice for creating an IoT device is WIFI as it offers similar capabilities like Bluetooth. Unlike Bluetooth, WIFI can send data to devices at much longer distances like over 150ft, meanwhile a WIFI access point is available. WIFI is incredibly much faster than Bluetooth, better security, longer usable range but has a higher power consumption.

## 5.3 BPM Algorithm

For learning purposes and possible improvements, the BPM algorithm on the Zynq-7000 software implements a moving average filter which can be done using a hardware filter on the FPGA. It was also noticed that a circular buffer for holding the data in the window can be implemented instead of always shifting the elements in the window to calculate new moving average. If a hardware filter is implemented, no circular buffer will be needed and the Zynq-7000 software will be reduced allowing for faster execution time.

### 5.3.1. Hardware Filter

For the BPM algorithm, the digital signal processing was done on the software side. The output from the heart rate sensor was not sufficient to calculate an effective BPM, therefore filtering was used to eliminate the discrepancies. Figure 52 illustrates me using an oscilloscope to probe my ECG signal and measures a pulse that is roughly 1-2Hz using the heart rate sensor. My ECG pulse corresponds to a BPM ranging from 60-120 but more importantly, the pulse is active for a duration of roughly 300ms that translates to a frequency of 3.3Hz. For this signal, the high frequency content is the pulse and to isolate the signal, a high pass filter can be used to block or attenuate low frequencies.

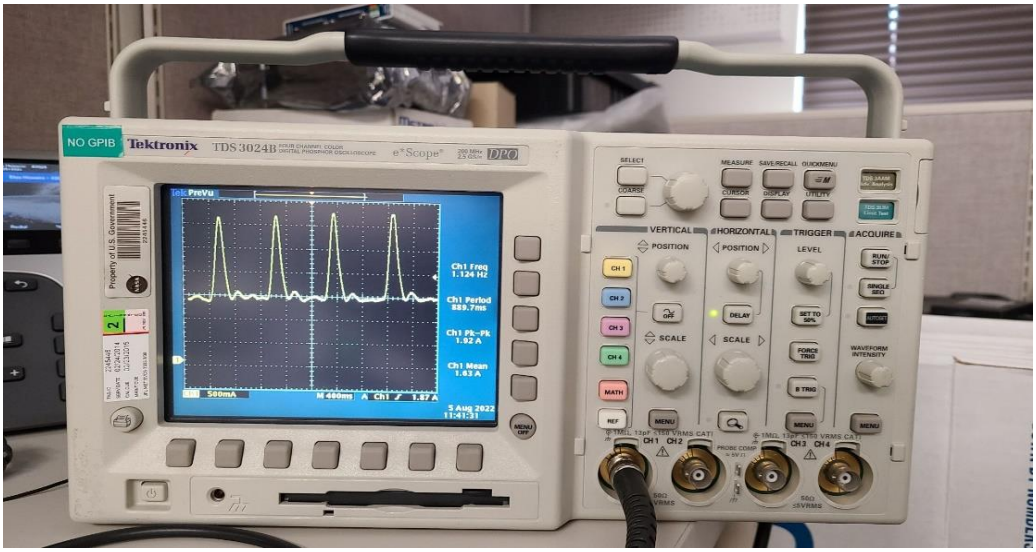


*Figure 52. Oscilloscope Reading of Pulse Sensor*

The software implementation filtered the signal by applying a moving average filter, which behaves like a low pass filter. Using the moving average of the ECG signal, you can filter the signal to not include any pulses less than the moving average. This implementation can be improved by using FPGA resources such as digital filters. The FPGA has DSP slices that allow high speed arithmetic or

filtering. By understanding the ECG pulse and frequency of interest, a Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filter can be used to attenuate the signal.

We are interested in the edge's; therefore, a high pass filter can be used but the same functionality can be achieved using a low pass and simple arithmetic operation. For example, if the low pass filter is used on the ECG signal the filtered output can be subtracted from the original ECG signal and the high frequency content will be remaining.

### 5.3.2. Circular Buffer

The filter needed an array or window to store the raw signal from the Pmod AD2 to calculate a continuous moving average. To calculate a new moving average, the array holding the current elements needs to be shifted by one element every calculation. Since only one data value changes per moving average calculation, the window should be shifted by 1 to create a new space for the incoming signal. This process can be very slow and in fact, operates at O(n) since n operations are needed to shift all elements. A circular buffer behaves like a Firs In, First Out (FIFO) data structure by releasing oldest elements first and can a be single or fixed size [44]. This can enhance the performance of the algorithm as each moving average calculation won't require the data be shuffled around but instead just update head or tail pointer. If data is added, the head pointer advances while the tail pointer advances when data is consumed. If you reach the end of the buffer, the pointers wrap around.

### 5.4 Stress Level Algorithm

A stress level algorithm template on GitHub, was found with suitable parameters to create a custom IP block with stress level logic. This allowed for easy integration to the system, but improvements can be made. The stress level module can be improved with the use of machine learning algorithm by adding memory and a feedback loop to learn from previous data.

### 5.5 Prototype Glove

One of the goals of the project was to develop a wearable device using the sensors, like an Apple smartwatch. Not enough time was spent designing a wearable device, but a Velcro glove could be a simple protype with all sensors attached. The sensors are contact type, which allows for them to be embedded or sewed onto the Velcro.

## 6. Conclusion

The project was a challenge but served its purpose in developing a biosensor system, learning fundamentals of enabling peripherals via PS or PL, configuring IP's using Vivado or Zynq software to access register, creating custom IP by using RTL code, using global timer to benchmark execution time of code, using Pmod's, implementing communication protocols, integrating a system that can read from multiple sensors, reading datasheets for details like timing diagrams that explain how to read and write to module or registers that control peripherals and the bits needed to enable features for that peripheral, and many more.

By analyzing the results from the project, the Digilent Zybo Z7-20 development board was used to demonstrate that the CONCEPT was feasible, but the hardware used was NOT appropriate for a wearable device. The development board can be connected to multiple sensors and perform signal processing for accurate measurements of biometric data. While the size may not be ideal for commercial wearable devices, this approach provides insight of the development board capabilities.

Development boards are used in applications ranging from Aerospace & Defense, ASIC Prototyping, Automotive, High-Performance Computing and Data Storage, Machine Learning & Vision, and Medical. The most important aspect of the Digilent Zybo Z7-20 development board is reprogrammable. Based on the research, engineers should consider a modular development board, research other sensors that can be used, integrate stress level analysis, and optimize algorithm among others.

## Appendix A: Source Code

I tried formatting the top-level biosensor algorithm to Appendix but Word has poor formatting of code. The GitHub repo will be provided instead that includes all source code with proper formatting, Vivado block designs, pin constraints, datasheets of all sensors and development board, and graduate report.

Graduate Project GitHub Repo

https://github.com/CSUN-Masters-Project/Zybo-Z720

## Bibliography

[1] "Pulse Sensor - SEN-11574 - SparkFun Electronics."

https://www.sparkfun.com/products/11574.

[2]"Electronic Components Distributor - Mouser Electronics."

https://www.mouser.com/catalog/specsheets/Seeed_101020052.pdf.

[3] "Galvanic Skin Response - an overview | ScienceDirect Topics."

https://www.sciencedirect.com/topics/computer-science/galvanic-skin-response.

[4] "Pmod TMP3: Digital Temperature Sensor - Digilent."

https://digilent.com/shop/pmod-tmp3-digital-temperature-sensor/.

[5] "PmodTC1™ Board Reference Manual - Digilent."

https://digilent.com/reference/_media/reference/pmod/pmodtc1/pmodtc1_rm.pdf.

[6] "PmodOLED™ Reference Manual - Digilent."

https://digilent.com/reference/_media/reference/pmod/pmodoled/pmodoled_rm.pdf.

[7] "Zynq-7000 SoC - Xilinx."

https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[8] "What is an FPGA? Field Programmable Gate Array - Xilinx."

https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html.

[9] "FPGA – Configurable Logic Block – Digilent Blog."

https://digilent.com/blog/fpga-configurable-logic-block/.

[10] "Zybo Z7 - Digilent Reference."

https://digilent.com/reference/programmable-logic/zybo-z7/start.

[11] "Zybo Reference Manual - Digilent Reference."

https://digilent.com/reference/programmable-logic/zybo/reference-manual.

[12] "Zynq-7000 SoC Technical Reference Manual (UG585) - Xilinx."

https://docs.xilinx.com/v/u/en-US/ug585-Zynq-7000-TRM.

[13] "Cortex-A9 Technical Reference."

https://vdocument.in/cortex-a9-technical-reference-manual.html.

[14] "The Zynq Book."

http://www.zynqbook.com/.

[15] "Design Hubs - Xilinx."

https://www.xilinx.com/support/documentation-navigation/design-hubs.html.

[16] "AXI Basics 1 - Introduction to AXI - Xilinx." 13 Oct. 2021,

https://support.xilinx.com/s/article/1053914?language=en_US.

[17] "LogiCORE IP AXI Interconnect (v1.05.a) - Xilinx."

https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/axi_interconnect/v1_05_a/ds768_axi_interconnect.pdf

[18] "XADC Overview - Xilinx."

https://docs.xilinx.com/r/en-US/ug480_7Series_XADC/XADC-Overview.

[19] "Pmod AD2: 4-channel 12-bit A/D Converter - Digilent."

https://digilent.com/shop/pmod-ad2-4-channel-12-bit-a-d-converter/.

[20] "PmodAD2™ Reference Manual - Digilent."

https://digilent.com/reference/_media/reference/pmod/pmodad2/pmodad2_rm.pdf

.

[21] "AD7991 Datasheet."

https://www.analog.com/en/products/ad7991.html.

[22] "Understanding the I2C Bus - Texas Instruments."

https://www.ti.com/lit/an/slva704/slva704.pdf.

[23] "Pmod TMP3: Digital Temperature Sensor - Digilent."

https://digilent.com/shop/pmod-tmp3-digital-temperature-sensor/.

[24]"PmodOLED™ Reference Manual - Digilent."

https://digilent.com/reference/_media/reference/pmod/pmodoled/pmodoled_rm.pdf.

[25] "TCN75S 2-Wire Serial Temperature Sensor Data Sheet."

https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/21935D.pdf.

[26] "Pmod TC1: K-Type Thermocouple Module with Wire - Digilent."

https://digilent.com/shop/pmod-tc1-k-type-thermocouple-module-with-wire/.

[27] "PmodTC1™ Board Reference Manual - Digilent."

https://digilent.com/reference/_media/reference/pmod/pmodtc1/pmodtc1_rm.pdf.

[28] "MAX31855 Cold-Junction Compensated Thermocouple-to-Digital Converter Datasheet."

https://datasheets.maximintegrated.com/en/ds/MAX31855.pdf.

[29] "Introduction to SPI Interface | Analog Devices." https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html.

[30] "SSD1306 Display Controller Datasheet."

https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf.

[31] "Pmod OLED: 128 x 32 Pixel Monochromatic OLED Display - Digilent."

https://digilent.com/shop/pmod-oled-128-x-32-pixel-monochromatic-oled-display/.

[32]"PmodTMP3™ Reference Manual - Digilent."

https://digilent.com/reference/_media/reference/pmod/pmodtmp3/pmodtmp3_rm.pdf.

[33] "Wearable Sensors for Human Activity Monitoring: A Review." https://www.researchgate.net/publication/273393907_Wearable_Sensors_for_Human_Activity_Monitoring_A_Review.

[34] "GitHub - Digilent/vivado-library." https://github.com/Digilent/vivado-library.

[35] "Creating a Block Design - 2022.2 - Xilinx." https://docs.xilinx.com/r/en-US/ug994-vivado-ip-subsystems/Creating-a-Block-Design.

[36] "MiniZed Tutorials." https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/minized/.

[37] "Programmable Logic Tutorials - Digilent Reference." https://digilent.com/reference/learn/programmable-logic/tutorials/start.

[38] "The Zynq Book Tutorials" http://zynqbook.com/download-tuts.html.

[39] "Difference Between FFT and DFT." 29 Sept. 2021, http://www.differencebetween.net/technology/difference-between-fft-and-dft/.

[40] "Galaxy Watch5 Smartwatch | Samsung US." https://www.samsung.com/us/watches/galaxy-watch5/.

[41]"Apple Watch Series 5 - Technical Specifications." 23 Mar. 2021, https://support.apple.com/kb/SP808?&locale=en_US.

[42] "GSR logger sensor NUL-217 | NeuLog Sensors." https://neulog.com/gsr/.

[43] "Grove - GSR Sensor - Seeed Wiki."

https://wiki.seeedstudio.com/Grove-GSR_Sensor/.

[44] "Circular buffer - Wikipedia."

https://en.wikipedia.org/wiki/Circular_buffer.