ELECTROCARDIOGRAM

ENGR 450 – FPGA Design

Final Report

Designing and building a working ECG using an FPGA

Ryan Shappa, Rachel Craig & Tom DeLine

Table of Contents

Abstract

FPGAs (Field Programmable Gate Arrays) are finding wide acceptance in medical systems because of their ability to rapidly prototype a concept that requires hardware/software co-design, for performing custom processing in parallel at high data rates, and to be programmed in the field after manufacturing. Based on the market demand, the FPGA design can be changed and no new hardware needs to be purchased which saves lots of time and money in producing a new and improved product.

Medical companies can now move over to FPGAs saving cost and delivering highly efficient upgradable systems. ECG (Electrocardiogram) is considered to be a must have feature for a medical diagnostic imaging system.

The scope of this project is to use any signal and use an FPGA as the hardware and software to detect that signal and configure some sort of system using the FPGA board and software. This project attempts at implementing ECG heart-rate attenuation in an FPGA.

First, the signal is created from two electrodes that are connected by the wrist and ankle to grab the heart-rate signal to be amplified using an instrumentation amplifier to a visible voltage level. Next, an ADC will be created using the FPGA board to create a digital signal that the FPGA board can filter from a created FIR filter. Finally, the filtered signal will be displayed from a created serial monitor to be explicitly shown.

Introduction

1. BACKGROUND

This section gives a brief introduction to ECG. Section 1.1 explains what an ECG is. Section 1.2 explains the electrical activity of the heart resulting in generation of 5 distinct waves. Finally, section 1.3 explains how ECG data is interpreted to get the heart rate from the electrical signal.

1.1 WHAT IS ECG

Electrocardiography (ECG/EKG) is a medical diagnostic test that captures the heart's electrical workings helping in understanding the rhythm of the heart and any irregularities associated with it. The result of the test is called an "Electrocardiogram". The heart muscles contract due to electrical signals received from the sinoatrial node. These electrical impulses are detected by an ECG. It is non-evasive and one of the most common procedures a person undergoes when having any trouble with breathing, chest pain, etc.

1.2 HOW DOES IT WORK

Electrical signals are triggered due to heart muscle depolarization which occurs during every heartbeat. Each heart muscle cell has a negative charge around it, which is along its membrane when the heart is at rest. During each heartbeat, a healthy heart will have an orderly progression of a wave of depolarization that is triggered by the cells coming from multiple nodes. Generally, more than two pairs of electrodes are used to detect these signals. The output of a pair of electrodes is known as a lead. Placing more than a pair of electrodes provides a better way to analyze the heart from different angles. Commonly 12 electrodes are used during an ECG, which are placed across the chest, arms, and legs. An ECG signal is shown in the figure below:

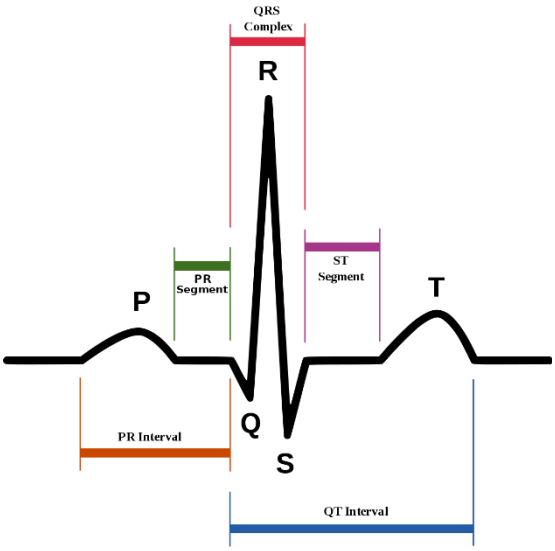

Figure 1: ECG Signal

- The P wave represents the atrial contractions.
- QRS complex represents the ventricular contractions. The R peak indicates a heartbeat.
- The T wave is the last common wave in an ECG. This electrical signal is produced when the ventricles are repolarizing.
- The letters used in the ECG signal description do not have abbreviations in medical terminology.

1.3 ECG INTERPRETATION

The output generated by an ECG is in the graph format with time on the x-axis and voltage on the y-axis. The baseline voltage is known as isoelectric line. In case of no signal detection, this would be a flat line. QRS is a combination of three graphical deflections noted on a typical ECG. Most of the time, it is the central and most visually obvious part of the trace generated. An ECG has five deflections, “P” through “T”. The Q, R and S waves occur in rapid succession and are usually considered together. They reflect a single event but do not necessarily appear in all leads. Post P wave, the Q wave is a downward deflection. It is followed by the R wave which is an upward spike. The S wave is again a downward deflection after the R wave. S wave is followed by the T wave. Modern ECG monitors convert these electrical signals into digital

values. They make use of various types of filters for signal processing. AN ECG graph as observed on a Philips ECG system is shown below:

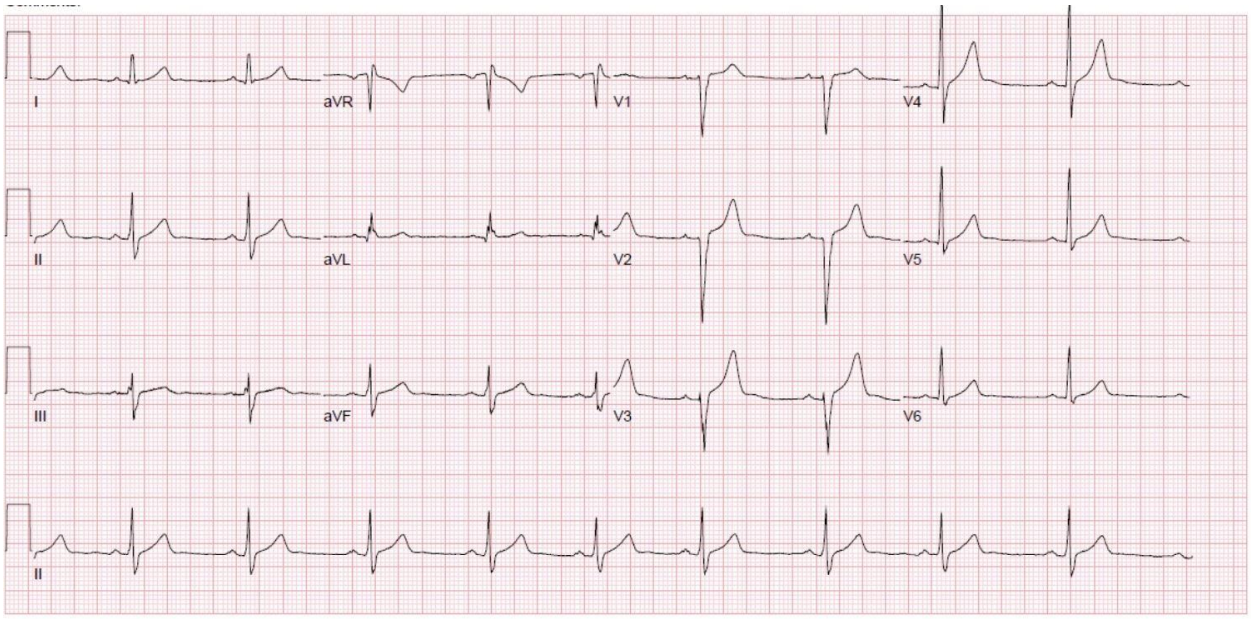

Figure 2: ECG Graph (Phillips System)

2. ECG Implementation

Our ECG will be implemented through 3 separate segments:

The first will be the external portion that was constructed to generate a human heart signal that will be generated in real time from the given electrodes sensors to pass into the created Analog to Digital Converter (ADC) that was developed with the FPGA software and board. The developed programs will convert the analog signals from the heart rate to a digital signal that the FPGA board can interpret to finally pass into a serial console. The created serial console will consist of a transmitter to serialize the digital data into fragments of 8 bits each with an established baud rate (9600) to then be ultimately plotted into the Arduino IDE's serial plotter. The complete system diagram visually instructs this explanation all on (pg. 8).

Specifications

1. Measures Heart Rate from 0bpm – 480 bpm
2. Identifies the PR interval, FR Segment, QRS complex, ST segment, and QT Interval
3. Number of Electrodes: 2
4. ADC Resolution: 10bit
5. External Communication Protocol: SPI
6. Instrumentation Amplifier Gain: 5 – 1000
7. Instrumentation Amplifier Working voltage: 2V-12V
8. May not read accurately for people with heart conditions
9. User output: console

TimelineTimeline/Work Distribution

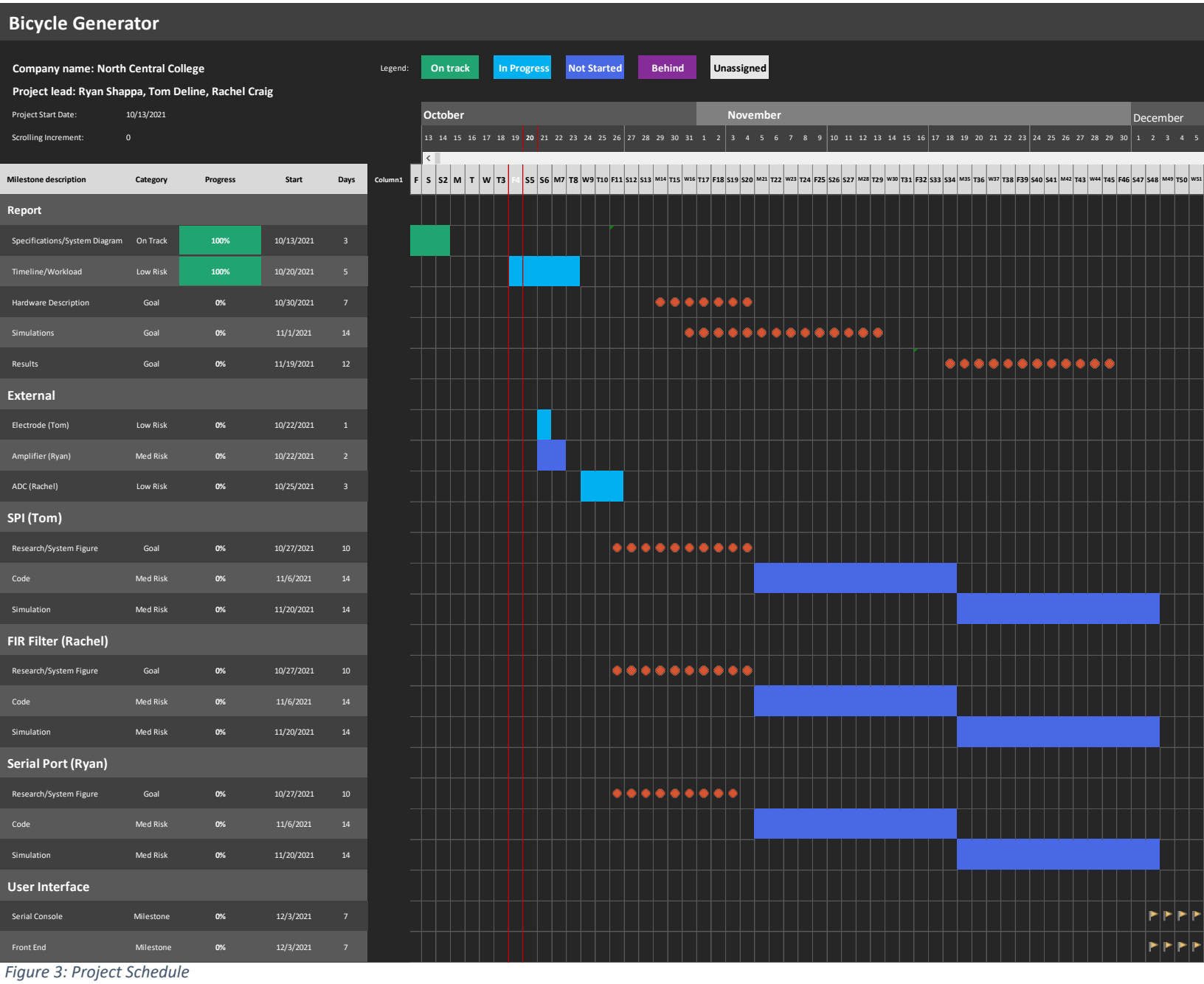

Figure 3: Project Schedule

Shown above in fig.3 is the complete timeline for the scope of the project represented into a Gantt chart. The listings on the left are the tasks that are to be completed by the denoted team member in parenthesis. Each task has a subset of inner tasks that are to be completed by a set date to finish the project by the due date. (Dec.13). Each team member has their own task with their own schedule (coinciding with each other's) to add and complete all together at the same time. The first subtask for each member has a goal to complete by the end date to timely start the next sub task and then complete the entire task. All denotations are shown in the legend.

System Diagram

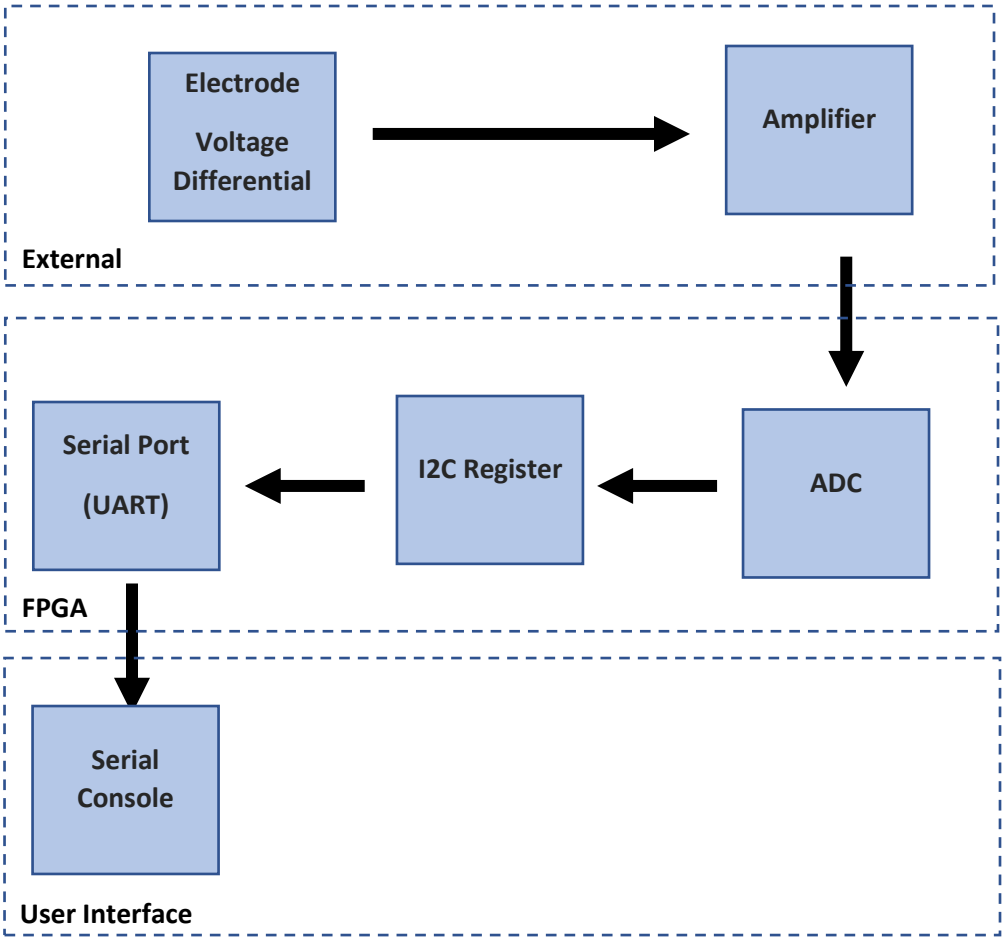

Figure 4: System Body Diagram

Shown above in fig. 4 is the chronological sequence for the scope of the EKG project depicted by a series of blocks combining to show the entire system.

Electrode – Detects the voltage differential

Amplifier – Amplifies the input signal for use with ADC

ADC – Converts analog data to digital data

I2C Register – Stores values on FPGA and communicates with ADC via I2C

FIR Filter – Filters out muscular noise and line frequency from input signal

Serial Port – Sending data to serial output

Serial Console – Reading data in for visual display

Python / C++ Front End – Display data graphically through Python GUI or Serial Plotter

Hardware Description

Instrumentation Amplifier

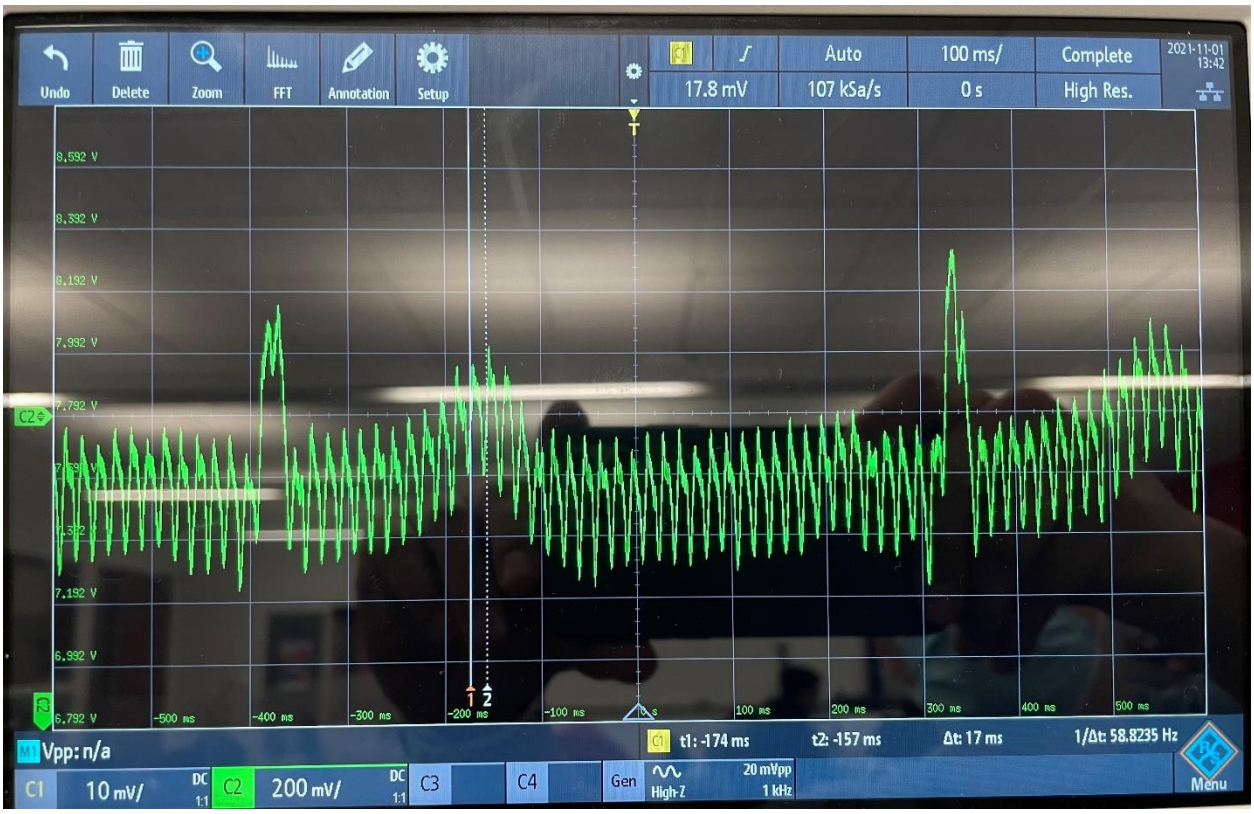

Figure 5: Instrumentation Amplifier Signal

Shown above in figure 5 is the amplified heart-signal shown on an oscilloscope. The ECG waveform is shown through the 60Hz noise that lies within the signal. The 60Hz is indicated at the bottom right of the screen per calculation taken from the oscilloscope.

The next step is to convert this analog signal into a digital signal for the FPGA board to interact with and filter out that 60Hz noise that is shown within the signal using the FIR filter from the FPGA. An LTC2308 ADC found on the FPGA board will be used for the analog to digital conversion.

ADC

Figure 6: ADC Body Diagram

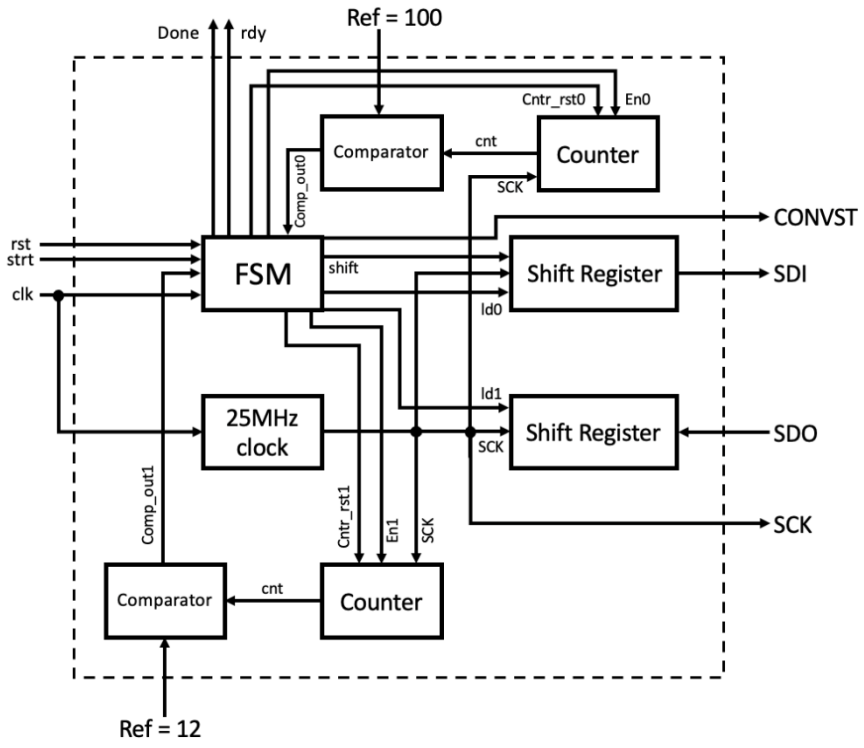

Figure 6 shows the block diagram of the ADC which implements two counters, two comparators, a serial-to-parallel shift register, a parallel-to-serial shift register, a 25MHz clock, and a finite state machine that will control each part. The state machine, as shown in figure 7, has four separate states which will operate in the following order: the state machine waits for a start signal and allows the first shift register to load with the SDI instructions; it will then stop loading the register, turn CONVST high, and will wait 100 clock cycles; next, it will then allow the SDO shift register to load, turn CONVST low, and wait 12 clock cycles; and finally, it will reset each part and return to the starting state. In its last state, a ready signal will be sent to the outside communicating hardware.

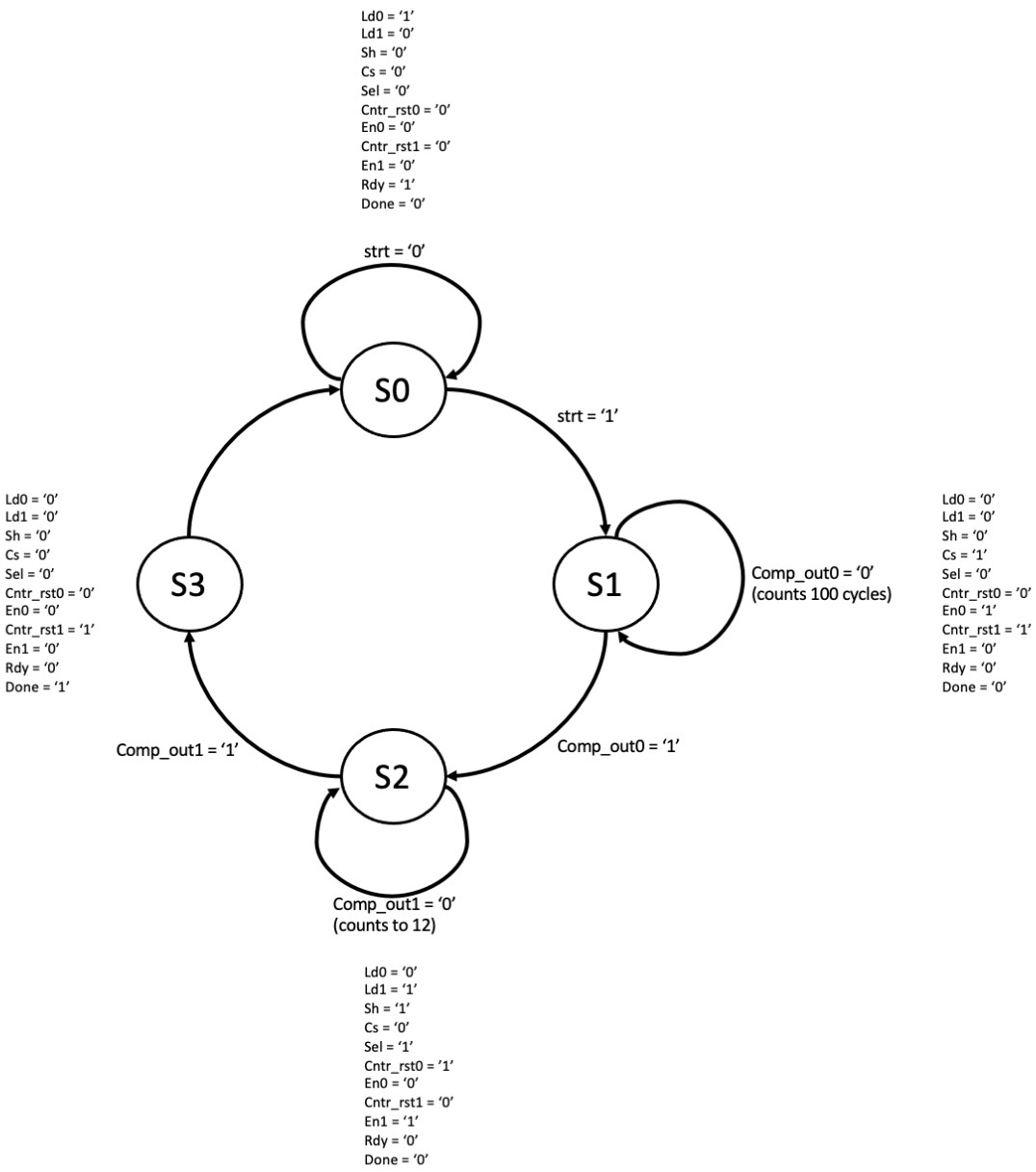

Figure 7: ADC FSM

UART

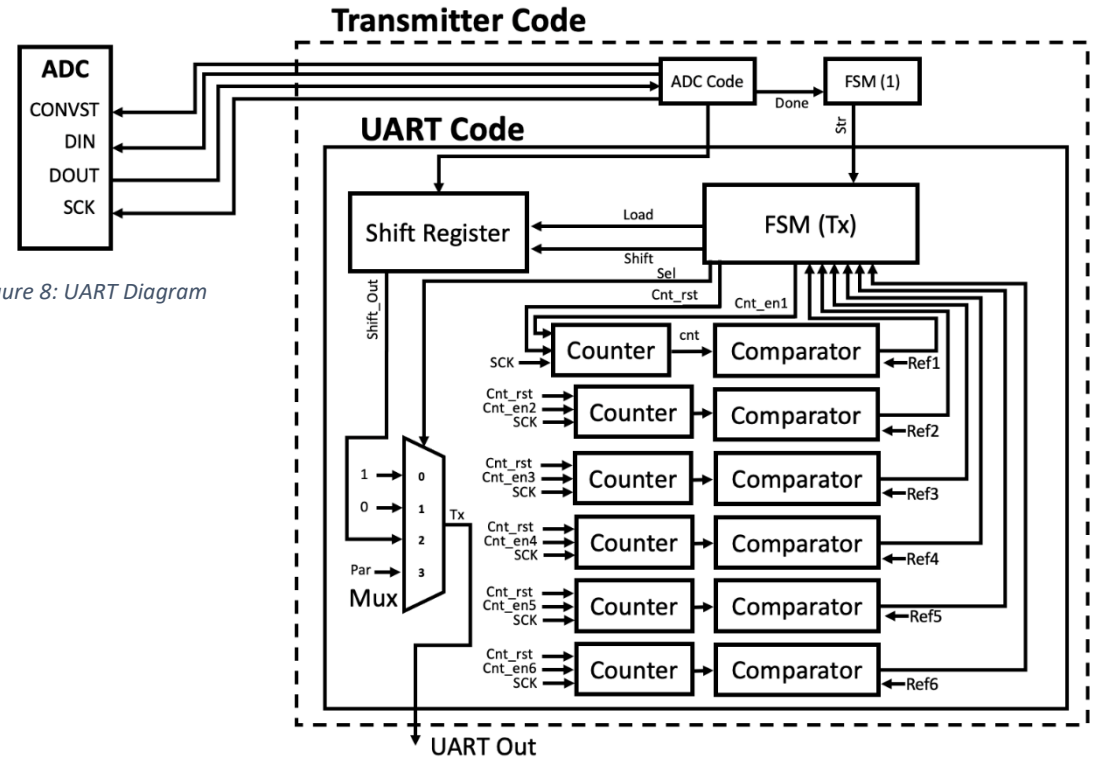

Figure 8: UART Diagram

The transmitter of figure 8 functions by waiting for the start signal from FSM (1). Once the start signal is received, the Transmit FSM can keep track of timing intervals through the use of six counters and comparators, each with their own ref value. The counters are synchronous and are activated through the Cnt_en signal from the FSM. To initiate the UART transmission, the FSM sends the Sel signal to the Mux which changes the signal from high to low. After waiting for the appropriate timing interval, the FSM can send the shift signal to the output shift register to shift out the first bit of data. The first counter is used for monitoring the amount of time elapsed since the signal went low before beginning the next bit data bit, while the remaining counters and comparators are used for outputting the remaining data at the required 9600 baud rate. The parity bit can be calculated by summing the number of 1's in the data stream. If the number of 1's is even, then the parity bit is zero. Once calculated, the parity bit is the next frame in the data packet. The FSM can select "10" on the Mux to place it on the output line.

UART Cont.

The transmission is ended once the required number of bits are sent, and the FSM selects “00” which sets the output to high where it remains until the next transmission.

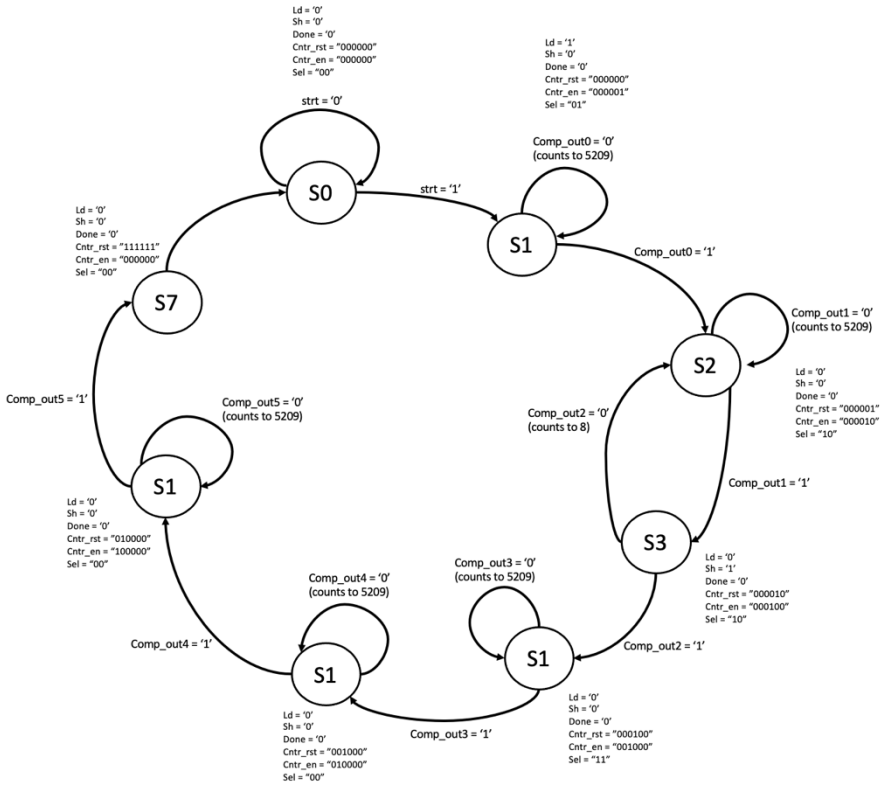

Figure 9: UART FSM

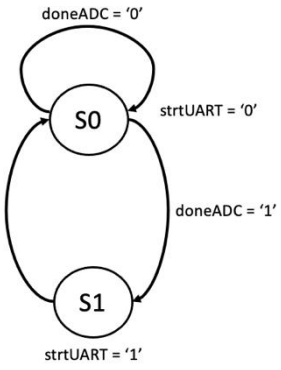

Figure 10: ECG-Transmitter FSM

Simulation

Clock Divider

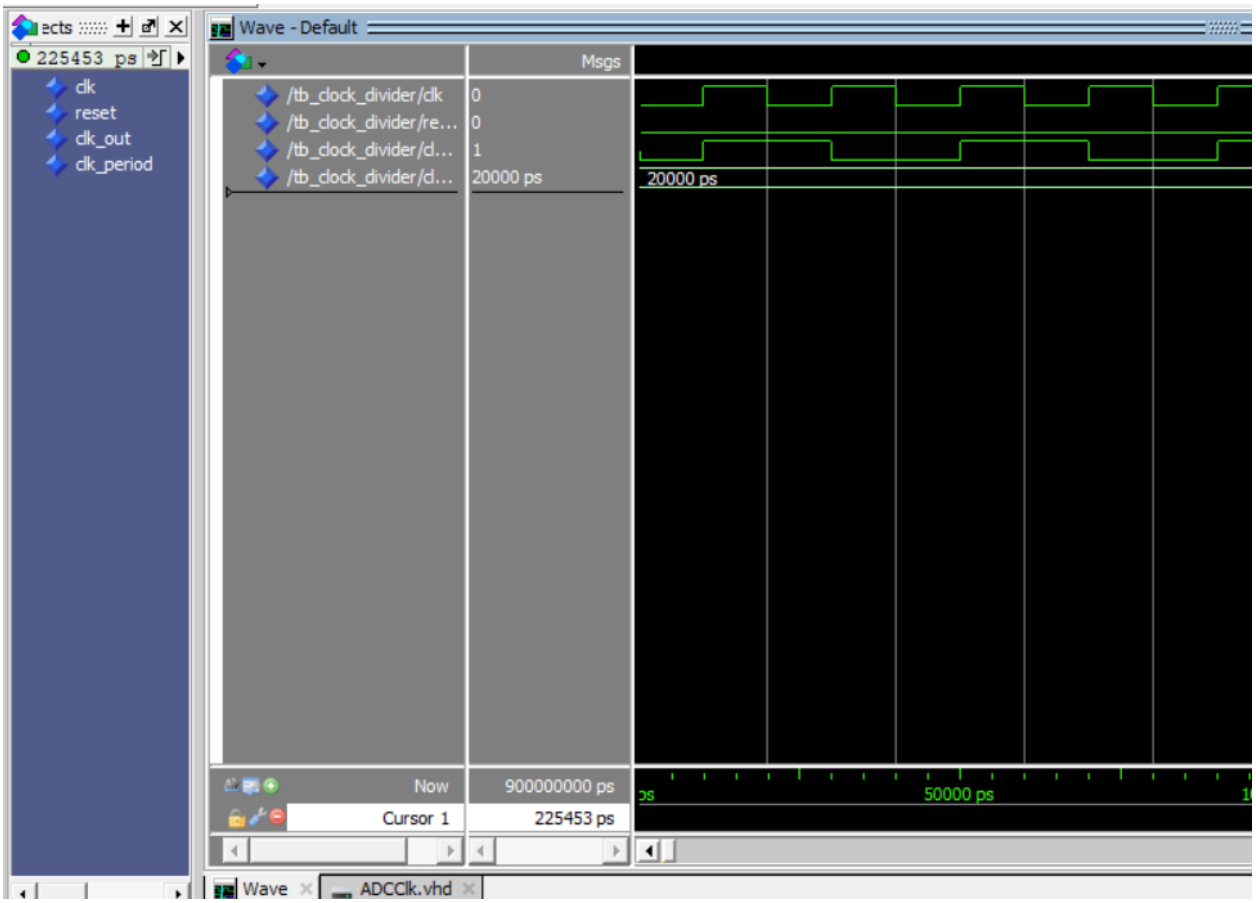

Figure 11: Clock Divider ModelSim

Shown above in fig. 11 is the simulation that was created from the clock divider program that is used for the ADC to divide the standard clock signal (40MHz) in half to a 20MHz signal as shown above. It is clearly shown from the simulation that the output signal (bottom) takes exactly two cycles to match the input signal above it. This in turn is the correct clock signal that was needed for the ADC.

ADC

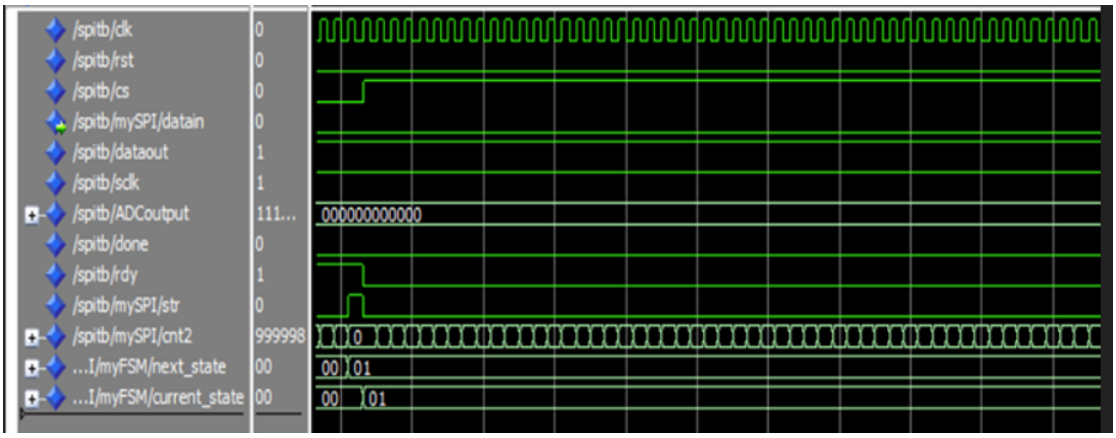

Figure 12: ADC Simulation 1

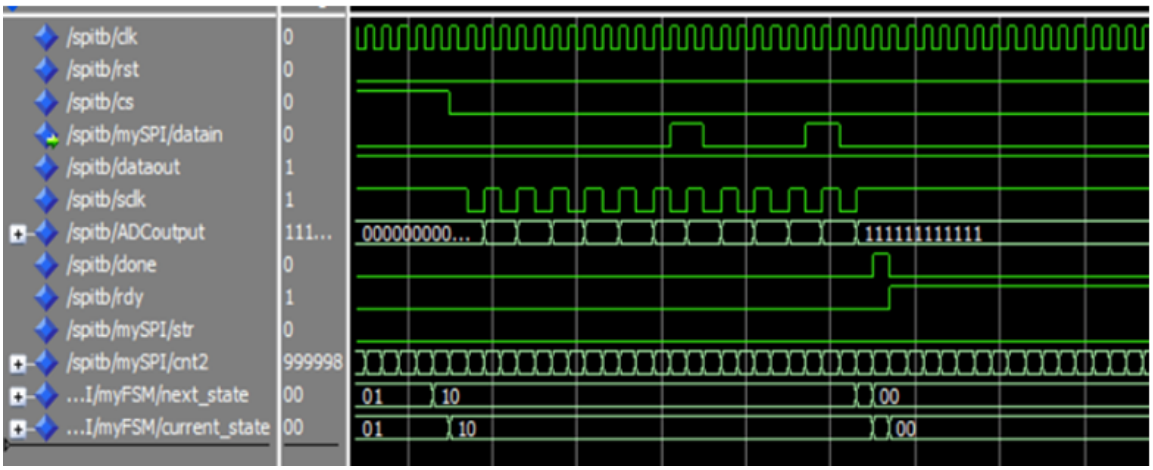

Figure 13: ADC Simulation 2

Figures 12 and 13 above are the captures taken from ModelSim that show the simulation of the working ADC. Figure 12 shows the ready signal coming in to trigger the chip select and start signal to begin counting which counts per state shown at the bottom. Figure 13 shows the serial clock running at the 25MHz as created to time the ADC output while the done signal successfully triggers at the end to loop the process yet again from the ready signal triggering high immediately after.

Transmitter

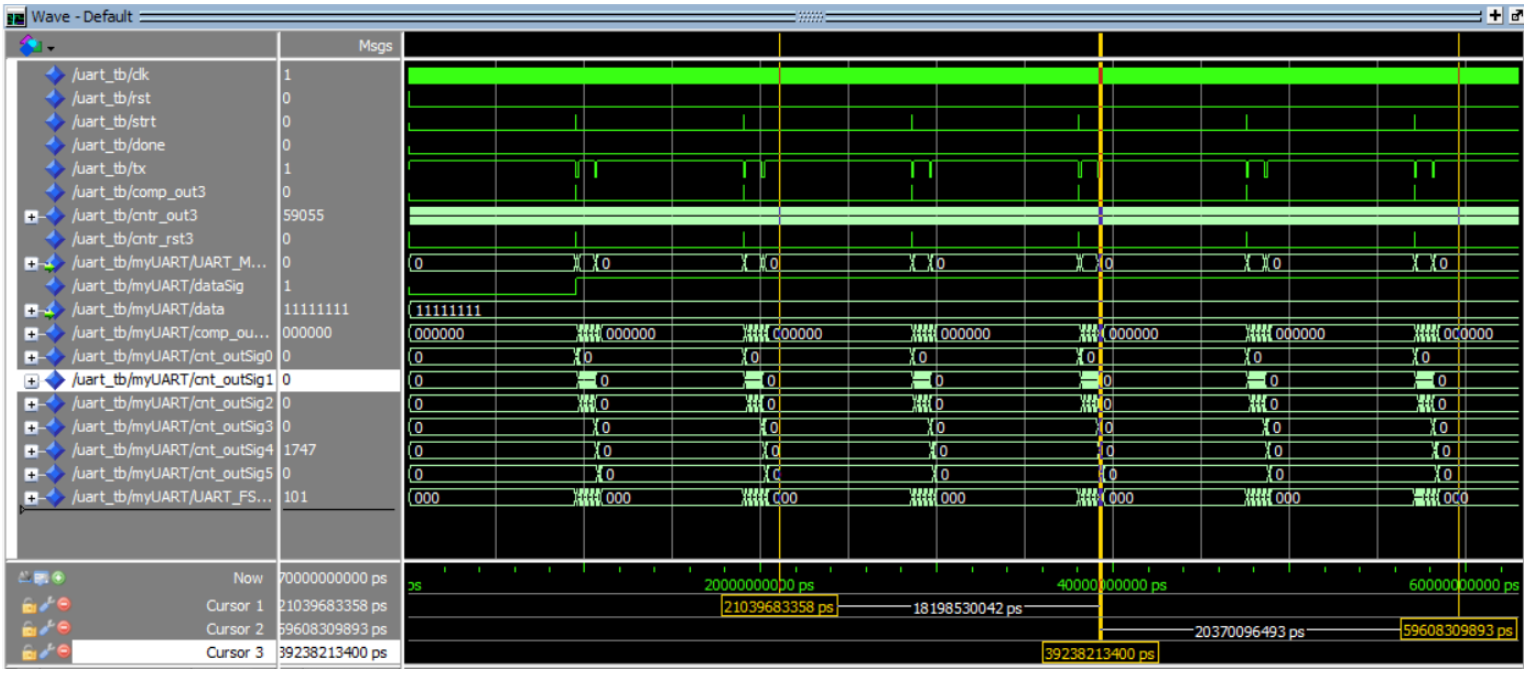

Figure 14: UART Simulation

Shown above in figure 14 is the ModelSim capture from the UART simulation. The first step in checking the implementation of the UART protocol is to verify the output timing of the data stream following the strt signal from the FSM. As shown in figure 14, when the data was set to all 1's, the tx signal can be observed as all 1's occurring at the appropriate timing intervals. Each of the timing intervals can be checked for accuracy by measuring the cnt_outSig. These signals can be seen resetting after the completion of each data transmission session.

Results

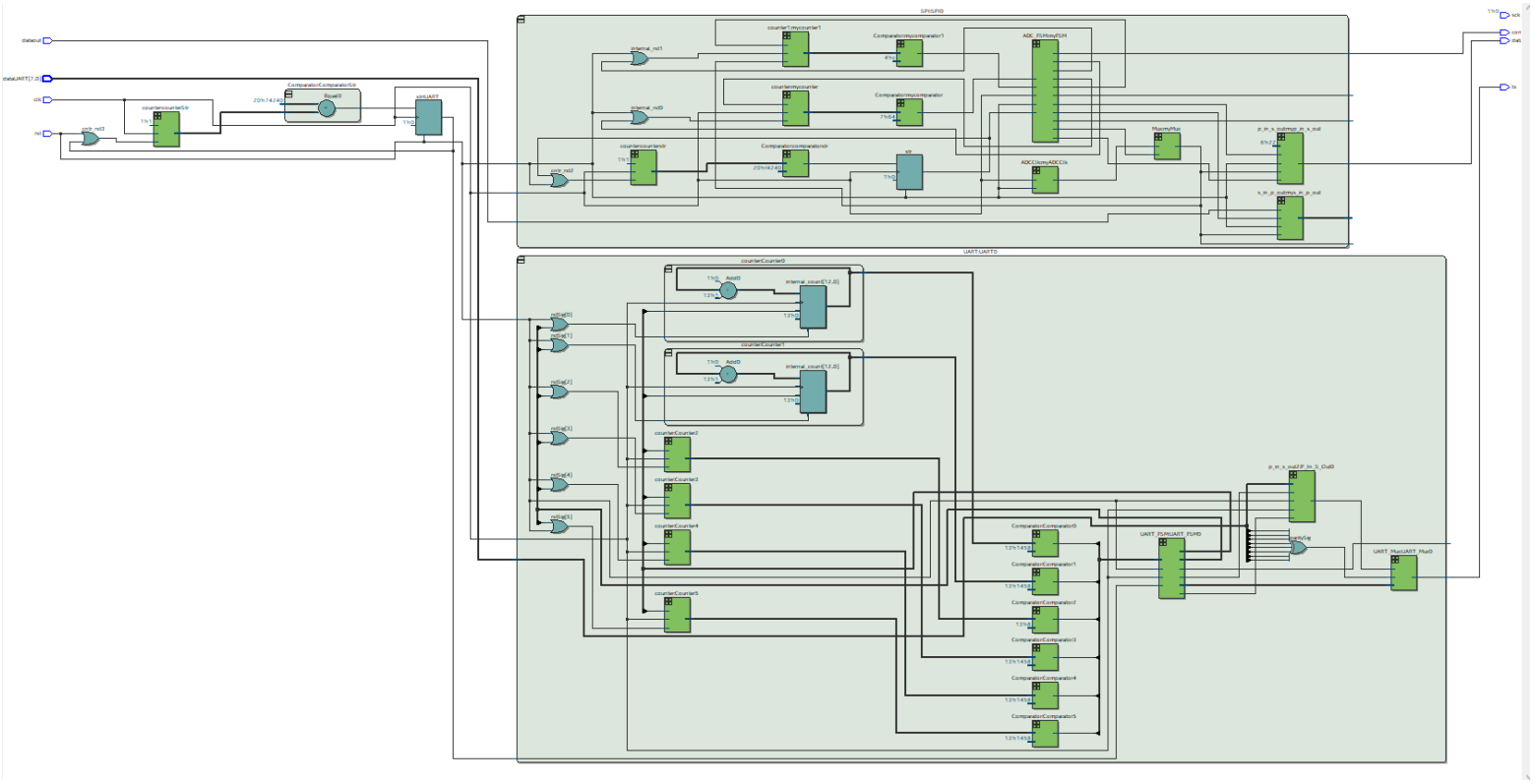

Figure 16: Overall View of System

Netlist RTL Diagram

Figure 16 shows the entire system as an RTL diagram. The smaller components to the left represent the smaller state machine found in figure 10 which controls when the UART and SPI are receiving and transmitting data. The upper green block represents the ADC connection and each of its components, and the lower block represents the UART. Most of the components' inner workings are compressed and not shown on this diagram, except for a few counters in the UART and a comparator to the left as examples.

Transmitter

The results from bypassing the ADC can be seen in figure 17. The switches SW1 – SW9 were manually set to 10101010. These values are fed into the FSM of the transmitter and output via the tx line of the UART data output. A common ground was shared between the controller and output data stream device used for capturing output via USB. A data transfer is initiated from one of the FPGA's GPIO pins when the output signal goes from high to low. Eight bits of data can be observed as the signal goes from low to high, and then high to low. This is repeated four times as seen in figure 12 data bits. Since the number of 1's in the data is even, the parity bit is shown as zero and then followed by the stop bit, which goes high, and a wait period until the next data transmission is started.

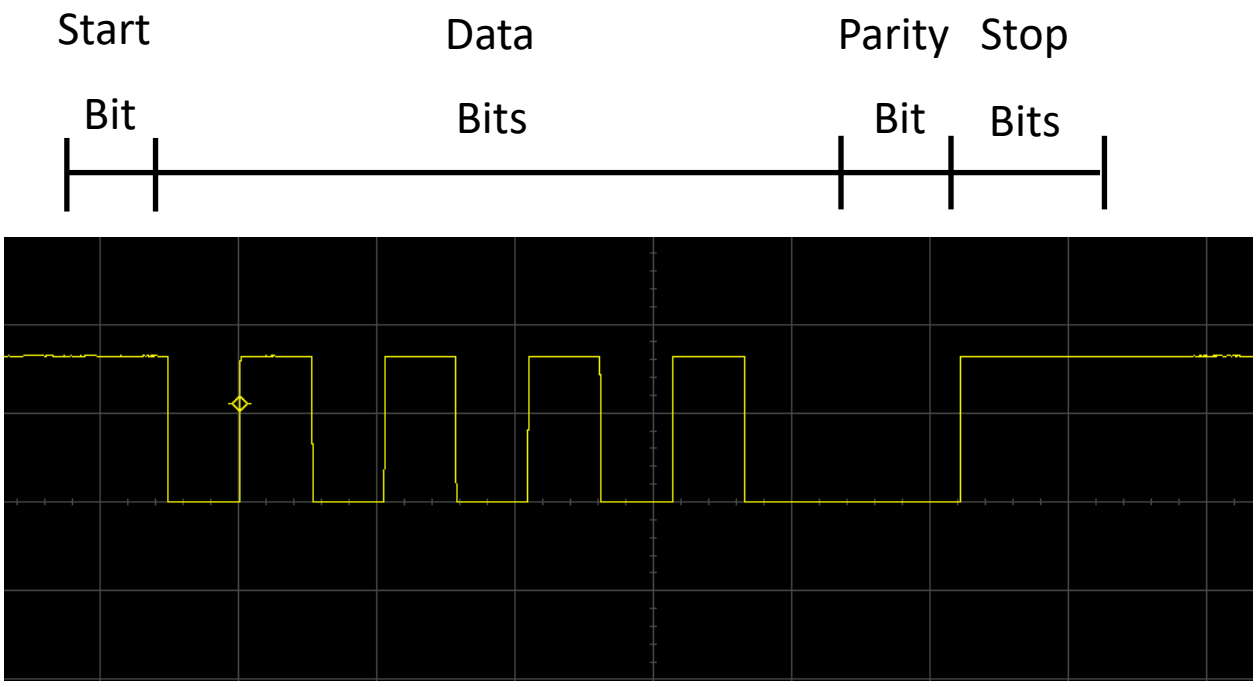

Figure 17: Tx output with data = 10101010

ADC

IMG_1402.mp4

Above is the link for a video showing the working implementation of the ADC. The video shows David Roush moving a potentiometer that is acting as the input signal from one end to the other which is attached to the LED's on the FPGA board to propagate the digital signal that is coming out of the FPGA and into the LED's.

Transmitter cont.

IMG_0663.MOV

This link above is for the transmitter implementation that was successfully done by bypassing the ADC from creating the switches on the FPGA boards as the data that is being fed into transmitter being shown onto the oscilloscope and the serial monitor on the laptop via Putty. All the ASCII characters being displayed on the serial monitor correspond to different bit output values. The decipherment between this can be found in (Ref. IIII) which will show exactly what bit values were being outputted per ascii character, showing that the UART functionality is correct.

ECG

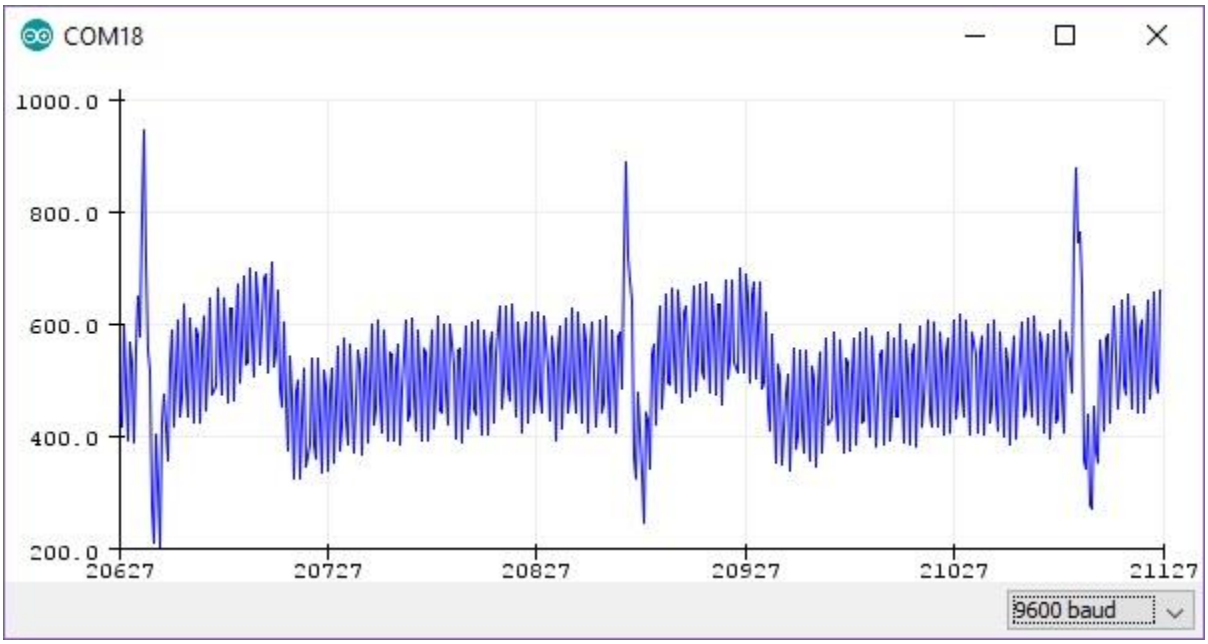

Figure 18: ECG Output

Shown above in fig. 18 is the ECG waveform displayed through the serial monitor of the Arduino IDE. The waveform does have some 'noise' being shown in between the peaks; however, the figure above does resemble the important components if referencing back to fig. 1 of the waveform showing all intervals (P, Q, R, S, and T).

Conclusion

All in all, the complete scope of the project was unattained due to the lack of time given for completing the project. As shown in previous pages, the Analog to Digital Converter was programmed, simulated, and tested for validity. As was the UART: programmed, simulated, and tested for validity. The amplified signal coming from the sensors could not be properly connected to the ADC to be transmitted and displayed via serial transmission. Multiple serial monitor programs were attempted, however none succeeded. The project was successful, however, in implementing the UART transmission protocol from one of the GPIO pins on the FPGA development board, as demonstrated in the Transmitter Results section of this report.

References

I. Desai, Vaibhav, "Electrocardiogram (ECG/EKG) using FPGA" (2012). Master's Projects. 238. DOI: https://doi.org/10.31979/etd.kk7h-c84x https://scholarworks.sjsu.edu/etd_projects/238

II. Person. "UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter." UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter | Analog Devices. Accessed December 3, 2021. https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html.

III. Weiman, D. (n.d.). ASCII Conversion Chart. Retrieved December 10, 2021, from http://web.alfredstate.edu/faculty/weimandn/miscellaneous/ascii/ascii_index.html.

IV. AD7908/AD7918/AD7928 (Rev.E) – Datasheet

V. Quartus – Software

VI. ModelSim – Software