# FPGA in Quantum

## Segmented Averager Semester Project

Husnu Erman Nadas

Final Submission Report

## Table of Contents

# A. Algorithm Verification in MATLAB

## A.1. Introduction

In this subtask of the project, the focus is on creating the test signals for the Segmented Averaging implementation and testing a basic algorithm implementation on MATLAB. The main points are to generate a signal of #R repetitions, formed of #L different segments with #N samples. Since these signals will be used to validate an experiment setup, each segment is preferred to be of various forms to increase variation and reduce the probability of false positives. Covering this ground, a different noise is added to each segment to replicate real-life flaws and nonidealities. Once the test signals are generated, averaging must be performed to suppress flaws in the signal. Averaging may also be used for statistical interpretations of quantum systems. The primary motivation behind using Segmented Averaging is breaking down a large amount of data into smaller, more manageable parts and performing calculations in a more resourceful manner. When utilizing the faster processing power of the FPGAs, the larger memory capacity of computers is sacrificed. Thus, in this project, Segmented Averaging is a better fit. One other point to consider is the resolution of the instruments. Since signals must be generated digitally from an FPGA, they must be converted to analog signals via a DAC. A fixed-point representation of the signals must also be implemented to cover this ground. With all constraints covered, a basic implementation of Segmented Averaging might be designed. For the following parts of the project, the signals should be exported in a form that can be read from an FPGA.

## A.2. Design Flow

### A.2.1. Generating Test Signals

Various test signals will be generated to examine the experiment setup. This process is done in MATLAB. In this implementation, a signal catalog of different frequency-independent shapes is used. Without any dependency on the sample count, various signal shapes are generated. These signals are then concatenated to form a signal of length #N * #L * #R samples.

### A.2.2. Fixed Point Conversion and Export

In real-world cases, incoming analog signals can only be read into an FPGA through a digital-to-analog converter. In this case, the resolution of the DAC of the RedPitaya FPGA is 14 bits. This equates to 13 bits of resolution per volt, considering the input full-scale voltage range is ±1V.

The signal is digitized to -8192, +8191 for ±1V, then converted to signed binary in MATLAB using the dec2bin() function. Finally, the generated binary representation is written in a text file and later imported into the testbench module.

### A.2.3. Segmented Averaging

A simple algorithm for segmented averaging is implemented in MATLAB. A “Segments Average” variable stores the accumulation, and in each iteration of the for loop, the next segment is added to the current accumulation. Lastly, dividing the final accumulated signal by the number of repetitions #R yields the averaged signal.

## A.3. Implementation and the Top Level File

Every process mentioned in the A.2. Design Flow chapter of the report is made into a function. These functions are finally implemented in a top-level file. A critical attribute of these functions is that they are tunable by parameters #N, #L, and #R. This way, the desired modularity is simplified for the end user through a simple act of assigning values to three variables. For further modularity, a parameter “offset” is used, which defines the number of samples for which the generated shapes are separated. A flow chart of the final implementation is provided in Figure A.3.1.

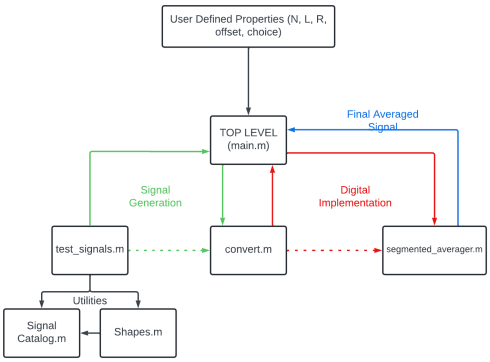



Figure A.3.1: Flow Diagram of the Implementation

The final resulting graphs of the analog and fixed-point implementations are provided in Figures A.3.2 and A.3.3.

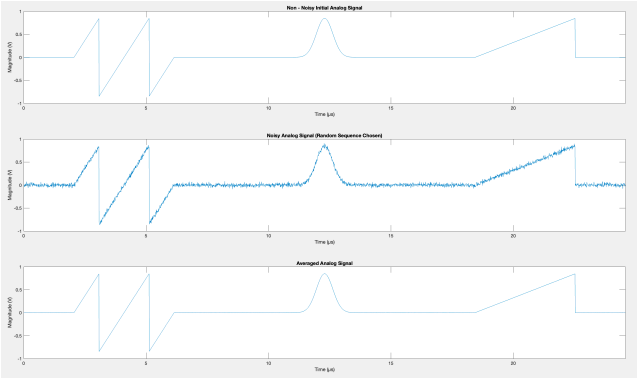


Figure A.3.2: Final Plot of the Analog Implementation

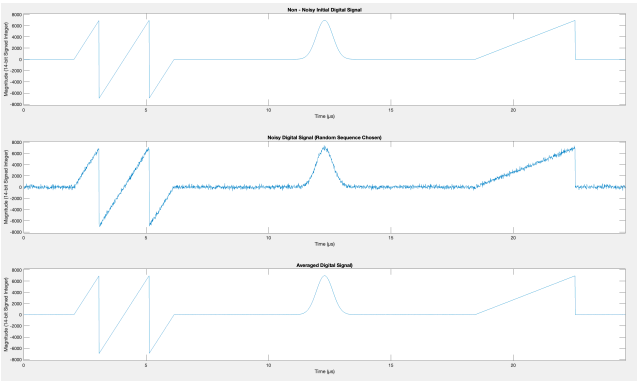


Figure A.3.3: Final Plot of the Fixed Point Implementation

# B. FPGA Implementation

## B.1.Introduction

With the grounds set on why the Segmented Averaging is a better fit for this project and how it will be implemented in A.Introduction chapter, this subtask focuses on the actual FPGA implementation. Since the division by numbers not of 2's powers on an FPGA is not efficient and precise, only the accumulation part is implemented in the programmable logic device. The FPGA should start accumulating the signal when a trigger is received and output the final accumulated signal when the specified number of sequence repetitions are done.

## B.2. Implementation

### B.2.1. Start and Trigger Detection

A rising-edge detector and latch functionality implement the trigger and start mechanism. Rising edge detection is done by delaying the input signal for one clock cycle and then checking if the current signal is high when the delayed signal is low. Since every update cycle is constrained by the clock, the mentioned situation can only occur with the rising edge of a signal. When a rising edge is detected, a latch is activated. For the trigger, the latch lasts for the number of samples + 3 for reasons mentioned in Chapter B.2.5. For the start pulse, the latch lasts until the accumulation is completed.

### B.2.2. Storing the Accumulated Signal

A two-port Block Memory is implemented to store the accumulated signal. One port is reserved for reading and one for writing the data. Chapter B.2.4 elaborates on how these ports are used. The size of the Block Memory is chosen as $2048 \times 10$. This design choice allows the program to be reconfigurable with any number of segments, a maximum of 10, and any counts of samples, a maximum of 2048.

### B.2.3. Counters

Internal counters for samples, segments, and sequences are implemented. These are used in several design aspects, such as how long the trigger latch should stay on, addressing memory, and checking if the program is complete. The counter mechanics are straightforward. The sample counter starts with 1 and counts up to the number of samples #N. If the next trigger comes right at the end of sampling, the sample counter resets to 1. But if the next trigger is not present exactly periodically with the number of samples times clock ticks, the counter counts two more times for the reasons mentioned in Chapter B.2.5. At the rising edge of every trigger, a segment is counted, and if the program is at the last segment, a sequence is counted. When the sequence count reaches the number of repetitions, the program goes into a completed state and starts outputting the accumulated signal.

### B.2.4. Storing the Accumulated Signal

As mentioned before, the signal accumulates in the block memory. Read and write operations without collisions at the same address in a block memory take as long as the amount of read latency plus one clock tick. Initially, the memory is read, and in the next clock cycle, the new information containing the old information from the same address is written. The information from the same address is the previous accumulation of that particular sample, which is added to the current sample from the signal to be written on that same address. The memory is addressed using the internal counters.

### B.2.5. Synchronization, Delays, and the Accumulation Program Flow

The signal must be synchronized with every operation for an accurate implementation. From rising edge detection to block memory read and write operations, each operation takes time to yield results. The trigger's rising edge detection introduces a single clock tick of delay. Counters are coupled with the trigger's rising edge. Memory readout should be done prior to the writein as the readout information is required for accumulation. The readout address is coupled to the counters to simplify the design, introducing another clock tick of delay. The readout address could be coupled to the rising edge of the trigger, but this design choice would further complicate the design. The read latency of the implemented block memory is a single clock tick, meaning the readout lags the counter by two clock ticks. Finally, the writein introduces another clock tick of delay. Since the writein uses data from the readout, this delay is inevitable. Taking these aspects into account, from receiving the signal to writing the accumulation into the block memory, takes four clock ticks or 32 ns.

### B.2.6. Validating and Outputting the Accumulated Signal

Accumulating data is always volatile and susceptible to overflows. This is checked by a simple if statement that checks whether the added signal's magnitude is larger than the memory's bit depth. The program will warn the user if an overflow happens, even for a single sample.

Once specified repetitions of sequences are accumulated, the program will start outputting the accumulated signal with corresponding segment numbers. Output comes directly from the block memory, and readout is addressed using a simple output counter.

## B.3. Synthesis

The implementation is synthesized with FPGA constraints. Resource utilization is given in Table B.3.1.

| Resource | Estimation | Available | Utilization % |
|---|---|---|---|
| LUT | 498 | 17600 | 2.83 |
| FF | 243 | 35200 | 0.69 |
| BRAM | 20 | 60 | 33.33 |
| DSP | 1 | 80 | 1.25 |
| IO | 98 | 100 | 98 |
| BUFG | 1 | 32 | 3.13 |

Table B.3.1: Resource Utilization

Other than Block RAM and I/O, resource utilization is observed to be frugal.

I/O is utilized at 98%. As all the inputs required for this implementation are declared, the design does not require any reconsiderations.

One-third of the available Block RAM is utilized. As the accumulated signal is the only significant amount of data that must be stored, this does not require any design reconsiderations.

## B.4. Block Diagram of the Architecture

A flow diagram is provided below to further elaborate on the module's architecture.

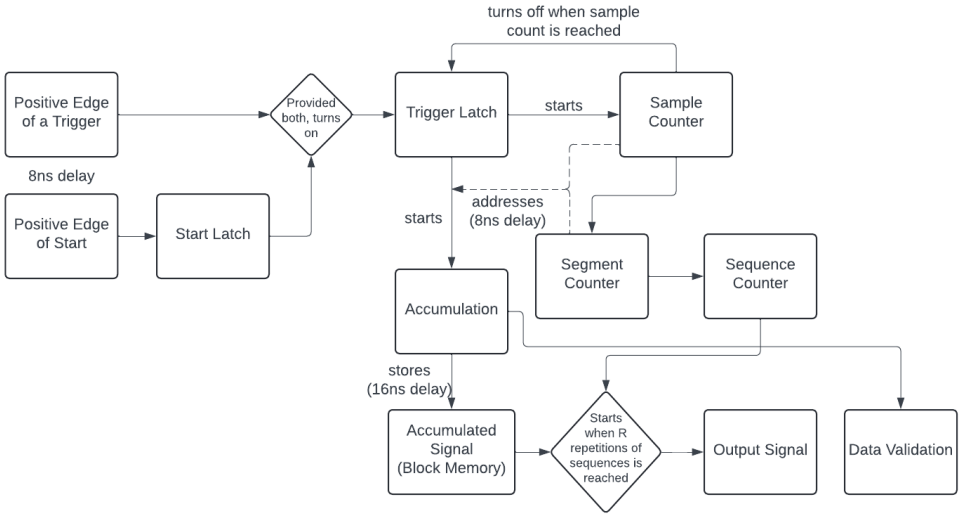



Figure B.4.2: Block Diagram of the Architecture

# C.Functional Verification

## C.1. Introduction

In Subtask B, a synthesizable implementation of the accumulator was made. In this subtask, the focus is on validating the implementation with test signals. A testbench configurable by the initially mentioned parameters #N, #L, and #R will be used for validation. This testbench is responsible for generating the reset signal, the clock, trigger, and start pulses, sending the signal, receiving the accumulated signal, and exporting the accumulated signal. The testbench's implementation introduces a new parameter named #TP that defines the trigger period. This approach replicates the case in which different sequences or segments of the signal are not received back to back.

## C.2. Implementation

### C.2.1. Clock Generation

A clock of 125 MHz or with a period of 8 ns is generated and fed to the module.

### C.2.2. Start Signal Generation

A trigger counter is used to generate the start pulse with a tunable width. A latch mechanism coupled to the start pulse is used to ease the process of when to create triggers. A start delay parameter defines when the start signal will be sent.

### C.2.3. Trigger Generation

A trigger counter is used to generate periodic triggers with a tunable width. When the counter starts, the trigger is sent, and the counter counts up to TP. Once the trigger period is reached, the counter resets to generate the subsequent trigger.

### C.2.4. Reading the Signal From a Text File

The generated signal from MATLAB is read into the testbench using the $fscanf() function. This function outputs the data asynchronously, so the asynchronous signal must be synchronized with the clock by assigning it to a synchronously updated register.

### C.2.5. Writing the Generated Output to a Text File

When the completed signal is received, it is written into a text file using $fwrite(). This text file is then used the compare the FPGA simulation results with the MATLAB results. To match the delay intentionally added for the module to output the signal after going into a completed state, the start of the output counter is delayed for two clock ticks.

## C.3. Verification Process

### C.3.1. Behavioral Simulation

Averaged signals from the FPGA implementation simulation and MATLAB algorithm implementation were compared. The resulting maximum difference was found to be 27.389, or %0.17, compared to a total magnitude of 16384. The waveforms of the behavioral simulation are provided in Figure C.3.1.1, and a plot of the averaged simulation signal on top of the averaged MATLAB signal is provided in Figure C.3.1.2.

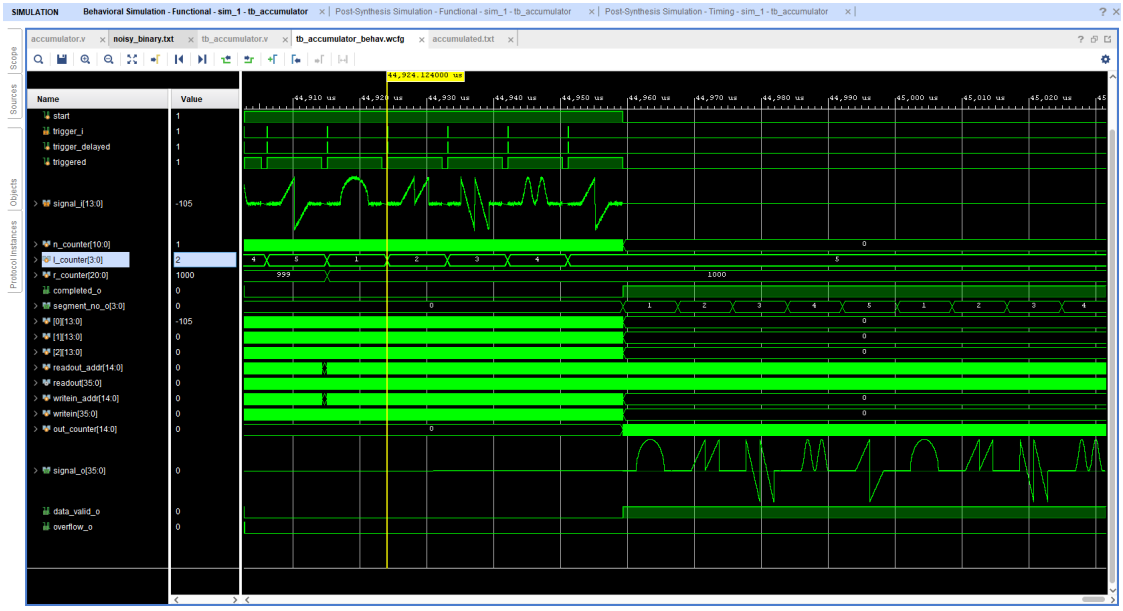



Figure C.3.1.1: Behavioral Simulation Output

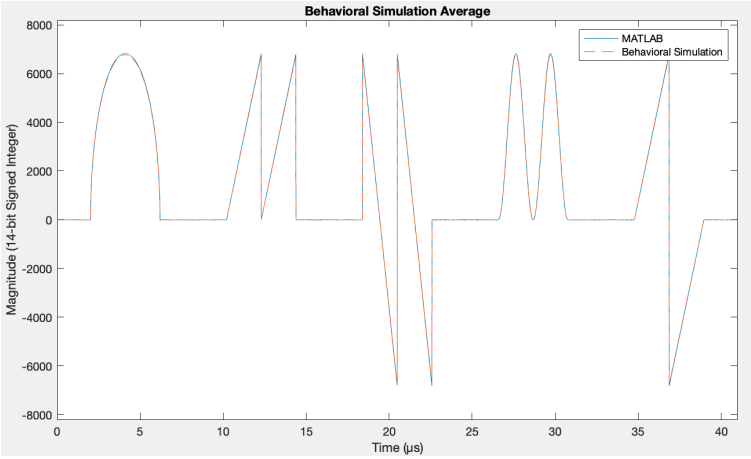



Figure C.3.1.2: Behavioral Simulation vs MATLAB Averaging

### C.3.2. Post Synthesis Simulations

Following the behavioral simulation, post-synthesis functional and timing simulations are made with fewer sample counts and fewer repetitions. As can be observed from figures C.3.2.1 and C.3.2.2, post-synthesis simulation results match behavioral simulation results, revealing that no FPGA constraints are violated and tuned parameters match precisely with the MATLAB Algorithm.

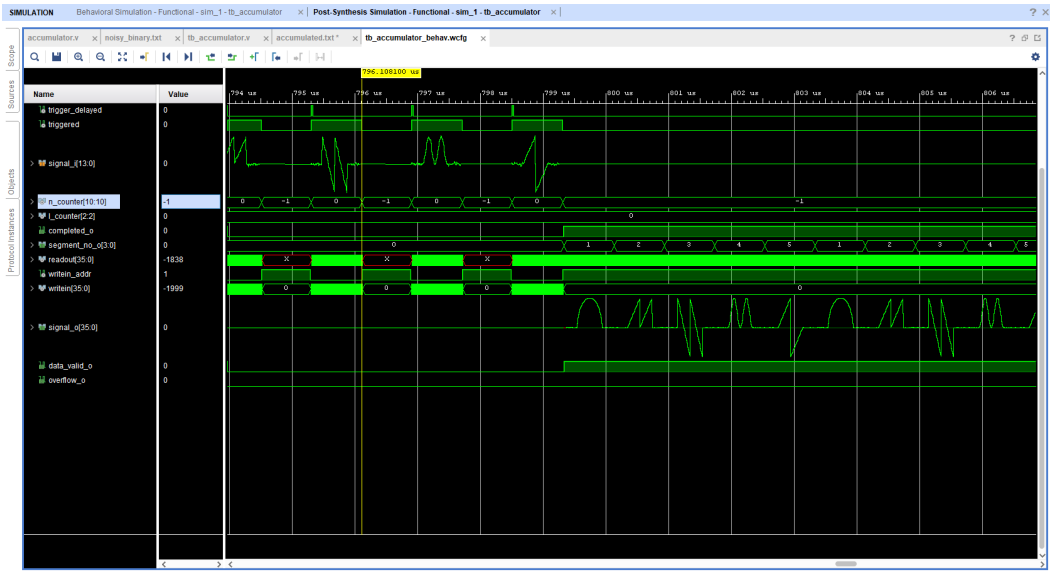



Figure C.3.2.1: Post-Synthesis Functional Simulation Output

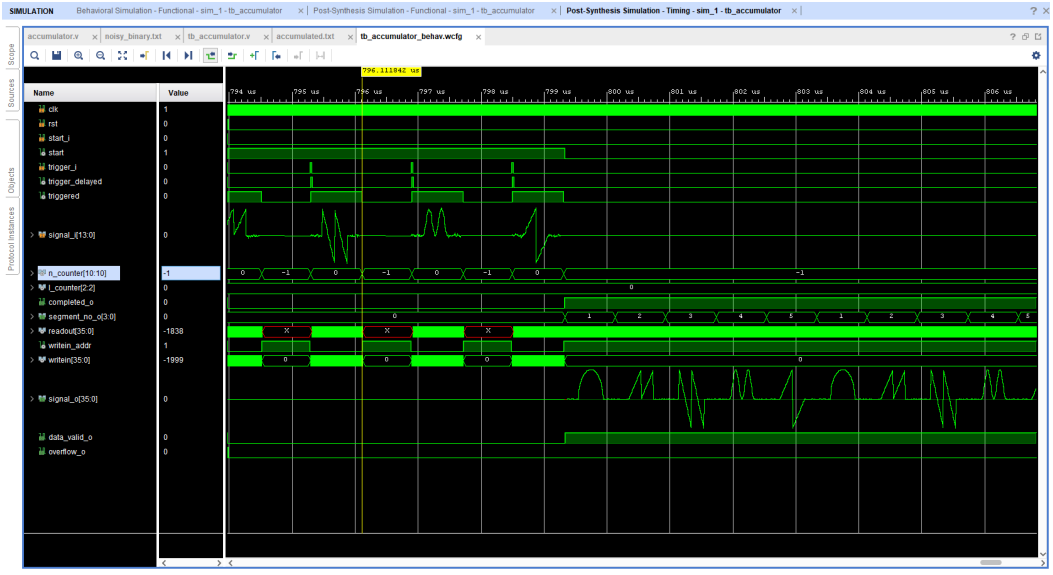



Figure C.3.2.2: Post-Synthesis Timing Simulation Output

Averaged signals from the FPGA implementation simulation and MATLAB algorithm implementation were compared. The resulting maximum difference was found to be 54.14, or %0.33, compared to a total magnitude of 16384. This is higher compared to the simulation done with more repetitions, as expected, but still low enough of an error to obtain valid information. A plot of the averaged simulation signal on top of the averaged MATLAB signal is provided in Figure C.3.2.3.

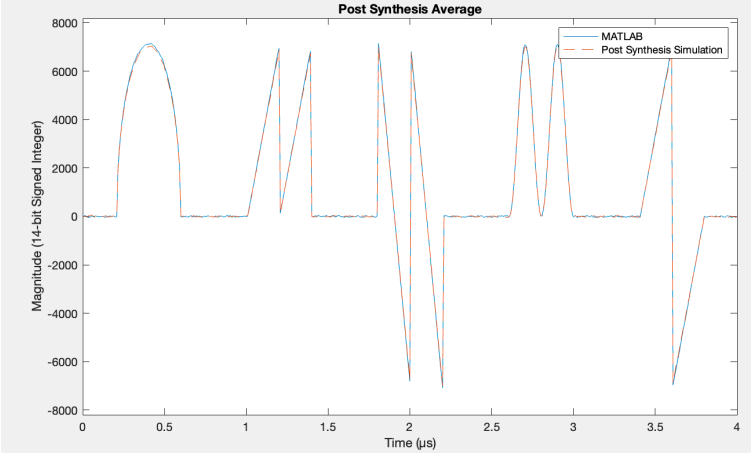



Figure C.3.2.3: Post-Synthesis Simulation vs MATLAB Averaging

This minuscule error occurs because MATLAB only works with integers or decimal numbers, but simulation averaging includes converting to and from binary numbers. Albeit, the module implementation in Subtask B is deemed valid based on the simulations.

# D. Test With Quantum Signals and Optimizations

## D.1. Introduction

In the previous subtasks, the accumulator module was implemented and tested using test signals generated from MATLAB. After testing, the design was deemed valid in Subtask C. In this subtask, real quantum experiment signals will be used to test the module. Optimizations to meet the expected performance criteria will be made if required. Finally, a bitstream generation will be done using a previously generated system wrapper to examine if any violations occur.

## D.2. Verification With Real Experiment Measurements

Measurement data of a qubit's excited and ground state is provided. In MATLAB, this data is converted into a signal with a form matching the previously generated test signals. In the generated signal, the sample count #N is 300, the sequence repetition count #R is 10000, and the segment count #L is 2, with the first segment being the ground state and the second segment being the excited state. The generated experiment signal is fed to the module, and the simulation averaged data is compared with MATLAB algorithm averaged data.

### D.2.1. Behavioral Simulation

Behavioral simulation was run using the provided test signals. The output waveforms of the simulation are provided in Figure D.2.1.1.

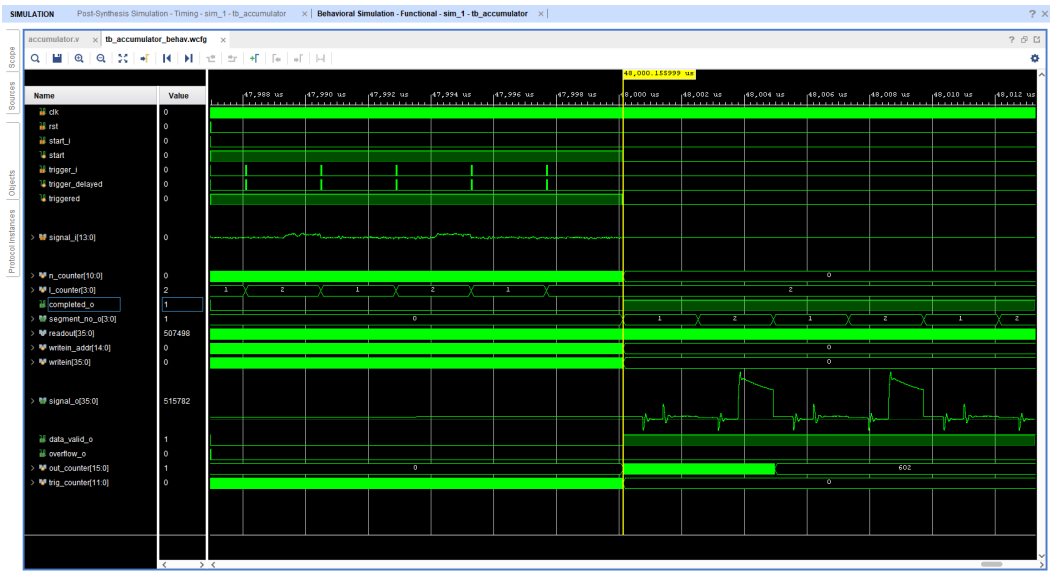



Figure D.2.1.1: Behavioral Simulation Output

The resulting maximum difference was found to be 52.55, or %0.32, compared to a total magnitude of 16384. A plot of the averaged simulation signal on top of the averaged MATLAB signal is provided in Figure D.2.1.2.

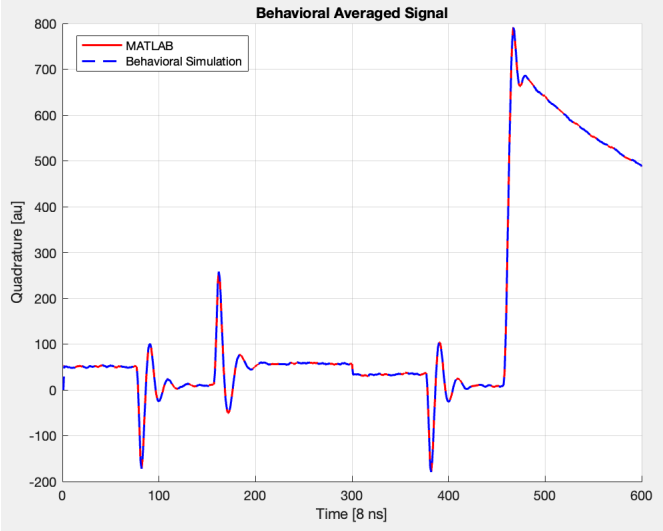



Figure D.2.1.2: Behavioral Simulation vs MATLAB Averaging

## D.2.2. Post-Synthesis Functional Simulation

Post-synthesis functional simulation was run using the provided test signals. The output waveforms of the simulation are provided in Figure D.2.2.1.

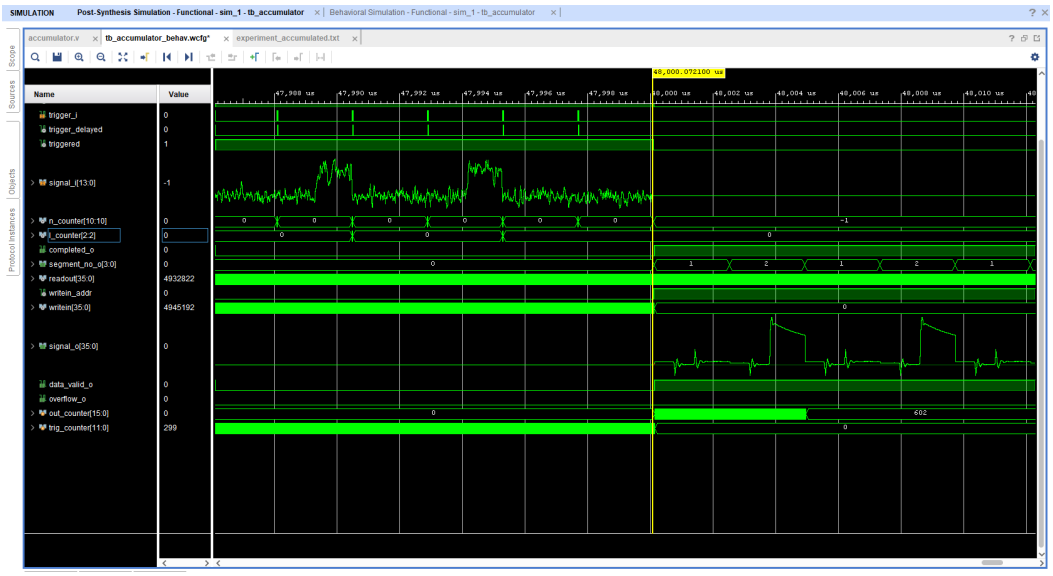



Figure D.2.2.1: Post-Synthesis Functional Simulation Output

The resulting maximum difference was found to be 52.55, or %0.32, compared to a total magnitude of 16384. A plot of the averaged simulation signal on top of the averaged MATLAB signal is provided in Figure D.2.2.2.

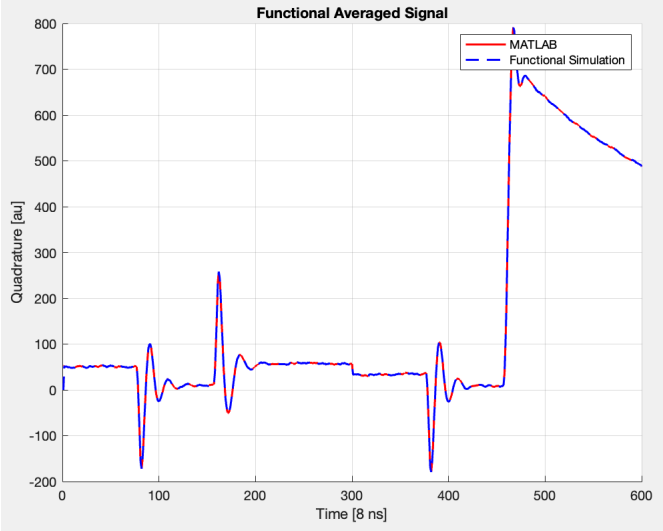



Figure D.2.2.2: Functional Simulation vs MATLAB Averaging

## D.2.3. Post-Synthesis Timing Simulation

Post-synthesis timing simulation was run using the provided test signals. The output waveforms of the simulation are provided in Figure D.2.3.1.

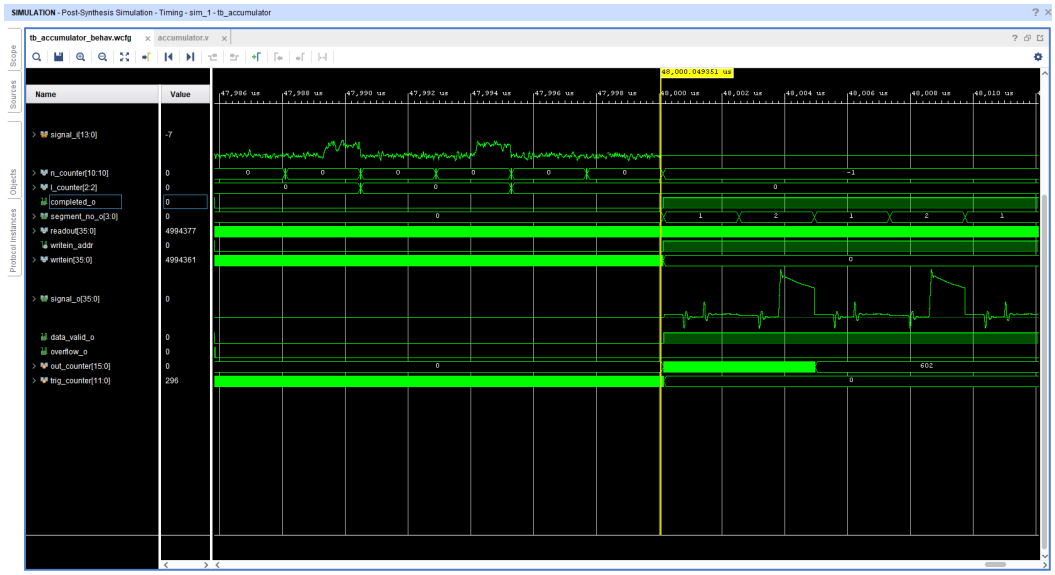



Figure D.2.3.1: Post-Synthesis Timing Simulation Output

The resulting maximum difference was found to be 52.55, or %0.32, compared to a total magnitude of 16384. A plot of the averaged simulation signal on top of the averaged MATLAB signal is provided in Figure D.2.3.2.

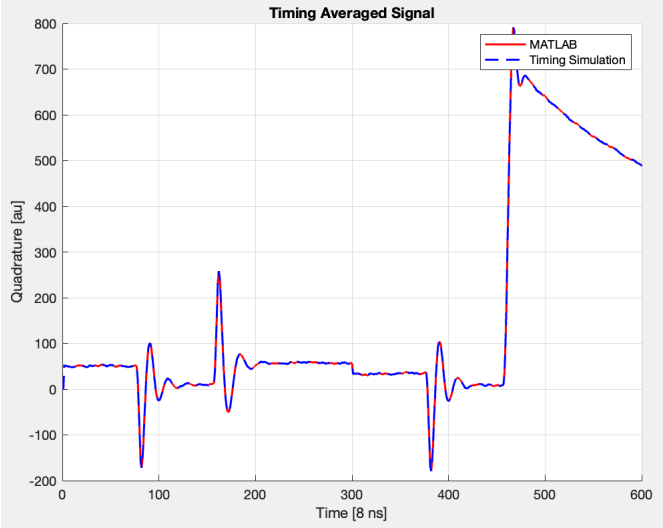



Figure D.2.3.2: Timing Simulation vs MATLAB Averaging

The correct averaging results with less than one-percent error were acquired for quantum experiment data.

## D.3. Pin Routing and Bit-Stream Generation

Once the module was validated, it was placed in a functional system wrapper for the RedPitaya FPGA. Then, the input signal of the module was routed to analog-to-digital converter channel 1. The start and trigger pulses are connected to external IO positive pins 1 and 0, respectively.

Once a valid routing was made, the bitstream was generated. The timing reports indicated no timing violations.

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 1.711 ns | Worst Hold Slack (WHS): | 0.024 ns | Worst Pulse Width Slack (WPWS): | 1.845 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 8553 | Total Number of Endpoints: | 8509 | Total Number of Endpoints: | 4645 |

**All user specified timing constraints are met.**

Figure D.3.1: Timing Report Summary

A positive WNS indicates that a signal may arrive earlier than expected, which infers no problems.

# E. Verification Using One Segment and a Rectangular Waveform

## E.1. Introduction

In the Midterm Project, an accumulator module for a segmented averager was implemented. The implementation was verificated via simulations. In the Final Project, this module will be verificated through an implementation of the module on the Red Pitaya FPGA. Using several kinds of waveforms generated on the FPGA, the validity of the module will be assessed. In this part, a single segment rectangular waveform will be used. This waveform will be generated on the FPGA and first will be routed without any external connections. Following, the signal will be routed outside the FPGA with the trigger signals routed from the external I/O pins.

## E.2. Implementation

### E.2.1. Without Using ADC/DAC Channels (Using Only Internal Connections)

Signal Generation module that generates rectangular waveforms of 1000 repetitions with a 50% duty cycle, an amplitude 0.61V (5000 per magnitude resolution), a period 4µs (500 samples per a 125 MHz clock) is implemented for this part of the assessment. The waveforms are sent with every trigger and the trigger period is determined to match the signal period at 4µs. Simple counters are used to implement this functionality. Figure E.2.1.1 provides a diagram for the architecure of this module.

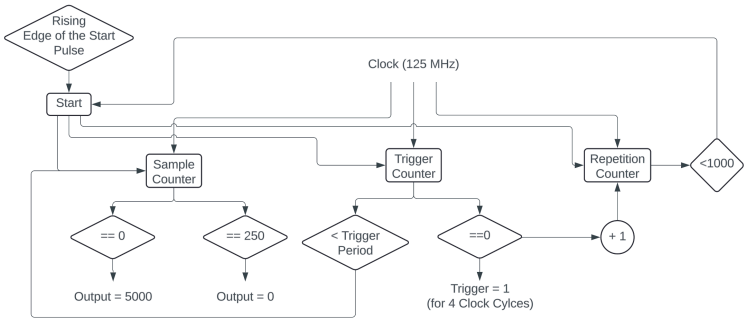



Figure E.2.1.1: Architecture of the Signal Generation Module

The generated module is routed inside a signal analysis using internal wire declarations.

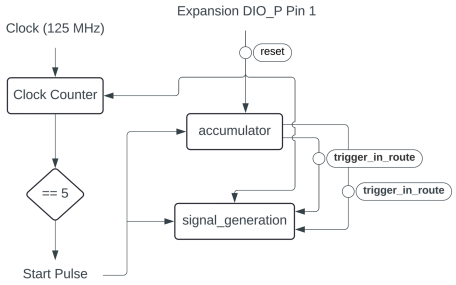



Figure E.2.1.2: Module Connections and Signal Generation Module Architecure

An active high reset signal is generated from outside the FPGA thorugh external I/O pins. Start signal is generated automatically after the reset period is over through a simple clock counter.

Resource utilization is provided in table E.2.1.1.

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 3816 | 17600 | 21.68 |
| LUTRAM | 452 | 6000 | 7.53 |
| FF | 5480 | 35200 | 15.57 |
| BRAM | 47.5 | 60 | 79.17 |
| DSP | 1 | 80 | 1.25 |
| IO | 79 | 100 | 79 |
| BUFG | 5 | 32 | 15.63 |
| MMCM | 1 | 2 | 50 |

Table E.2.1.1: Post-Implementation Resource Utilization

Other than Block RAM and I/O, resource utilization is observed to be frugal.

I/O is utilized at 79%. As all the inputs required for this implementation are declared, the design does not require any reconsiderations. 80% of the available Block RAM is utilized. At this point of the project this does not require any reconsiderations.

Timing report is provided in Figure E.2.1.3.

**Setup**

| | |
|---|---|
| Worst Negative Slack (WNS): | 0.522 ns |
| Total Negative Slack (TNS): | 0.000 ns |
| Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 12678 |

**Hold**

| | |
|---|---|
| Worst Hold Slack (WHS): | 0.032 ns |
| Total Hold Slack (THS): | 0.000 ns |
| Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 12634 |

**Pulse Width**

| | |
|---|---|
| Worst Pulse Width Slack (WPWS): | 1.845 ns |
| Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 6401 |

**All user specified timing constraints are met.**

Figure E.2.1.3: Timing Report

A positive WNS indicates that a signal may arrive earlier than expected, which infers no problems.

Results obtained from the ILA are provided in figures E.2.1.4 and E.2.1.5.

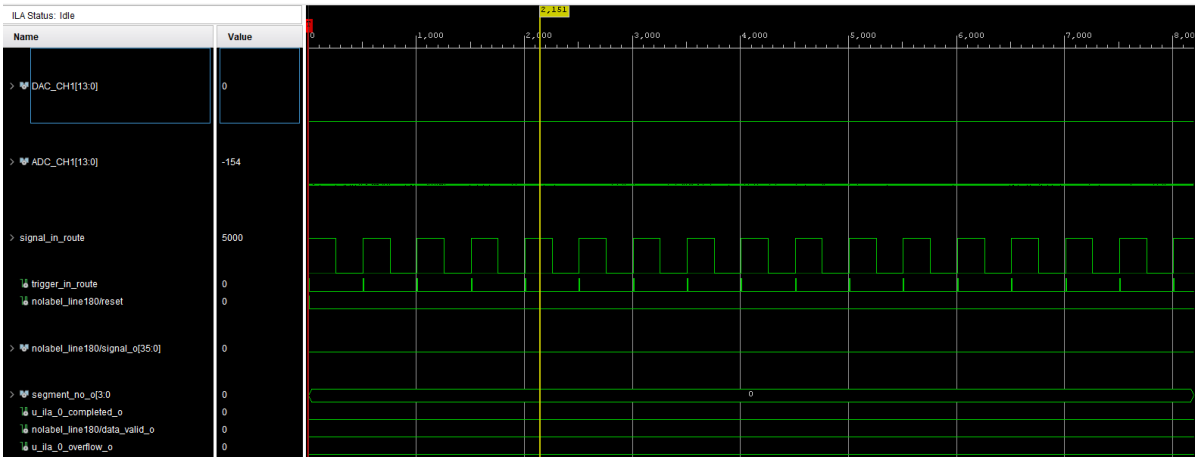



Figure E.2.1.4: Signal Generation Module Output Signal

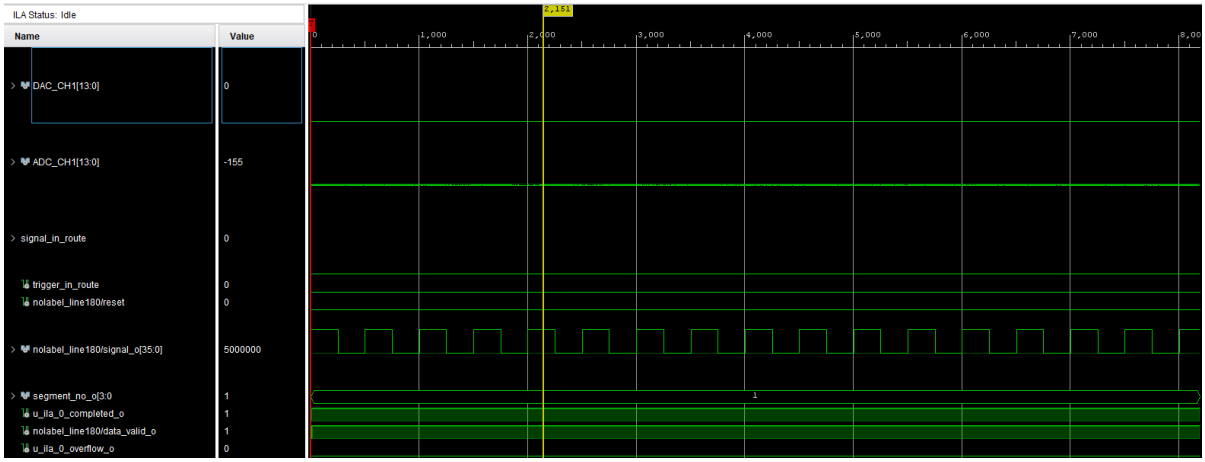



Figure E.2.1.4: Accumulator Module Output Signal

It can easily be observed that the accumulator module added the signal for 1000 times. This indicates a perfect match with the expected result.

## E.2.2. Using ADC/DAC Channels

The same Signal Generation module is routed using external connections. Signal and trigger outputs of the module are routed to DAC Channel 1 and an external I/O pin respectively. An active high reset signal is generated from outside the FPGA thorugh external I/O pins. Start signal is generated automatically after the reset period is over through a simple clock counter.

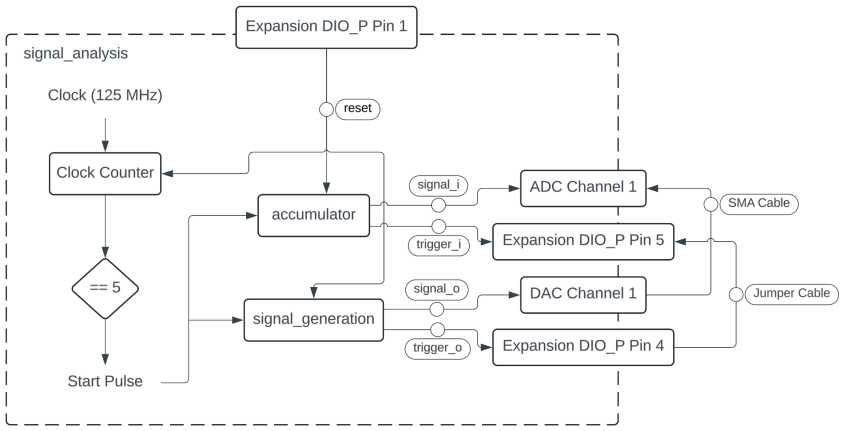



Figure E.2.2.1: Module Connections and Signal Generation Module Architecure

Resource utilization is provided in table E.2.2.1.

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 3818 | 17600 | 21.69 |
| LUTRAM | 444 | 6000 | 7.4 |
| FF | 5473 | 35200 | 15.55 |
| BRAM | 45 | 60 | 75 |
| DSP | 1 | 80 | 1.25 |
| IO | 79 | 100 | 79 |
| BUFG | 5 | 32 | 15.63 |
| MMCM | 1 | 2 | 50 |

Table E.2.2.1: Post-Implementation Resource Utilization

Other than Block RAM and I/O, resource utilization is observed to be frugal.

I/O is utilized at 79%. As all the inputs required for this implementation are declared, the design does not require any reconsiderations.

75% of the available Block RAM is utilized. At this point of the project this does not require any reconsiderations.

Timing report is provided in Figure E.2.1.2.

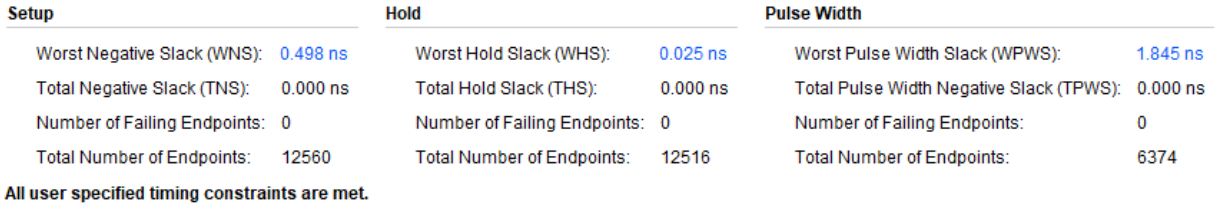



Figure E.2.2.2: Timing Report

A positive WNS indicates that a signal may arrive earlier than expected, which infers no problems.

Results obtained from the ILA are provided in figures E.2.2.3 and E.2.2.4.

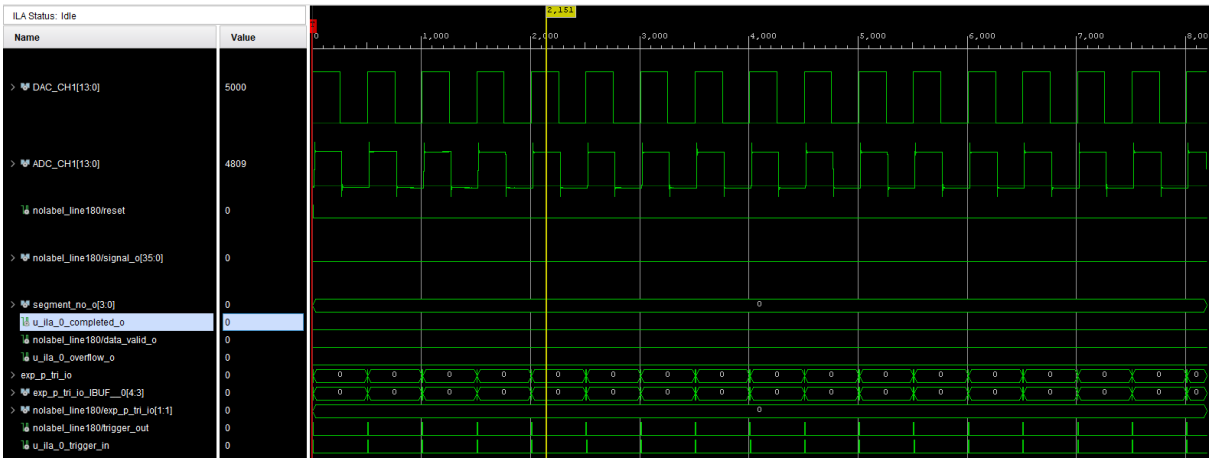



Figure E.2.2.3: Signal Generation Module Output Signal

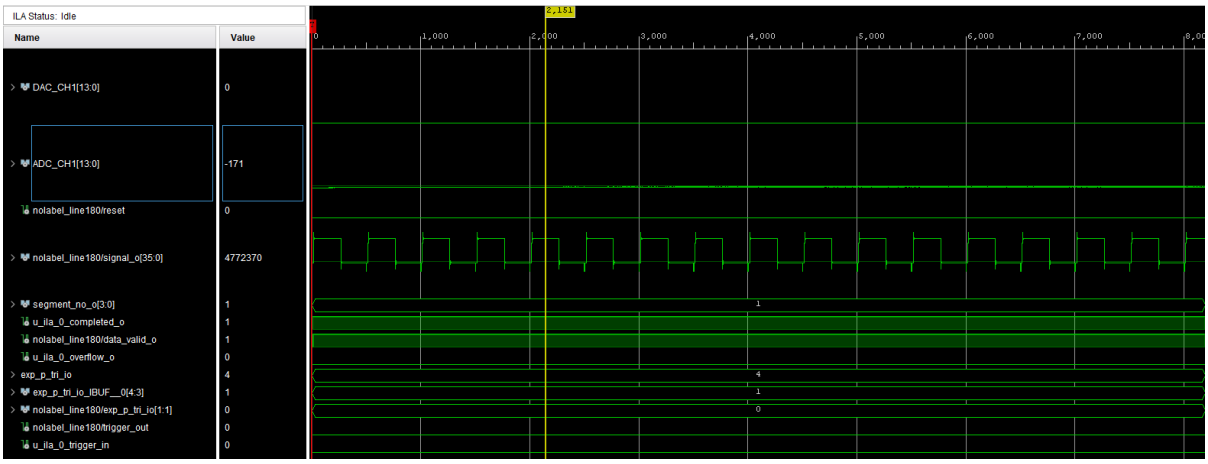



Figure E.2.2.4: Accumulator Module Output Signal

Some disturbance effects were observed on the output signal. Firstly it can be seen that the ADC input has a -20 mV (200 per magnitude resolution) offset. Another issue is that even with a 50Ω impedance mathcer, still some reflections are present. These issues are both non-firmware related and can be compensated with better components and calibration.

The accumulator module output signal was observed to have an amplitude 1000 times the ADC incoming signal. Hence, the measurement can be deemed a good match with the expected results.

# F. Verification Using Multiple Segments of Various Rectangular Waveforms

## F.1. Introduction

In this part of the project, multiple various rectangular forms will be used to validate the design. This will assess the design validity under different conditions. The routing will be done with external connections.

## F.2. Implementation

Signal Generation Multiple module that generates a three segment rectangular waveform of 1000 repetitions is implemented. Each segment is 4µs (500 samples per a 125 MHz clock). The segment are sent with every trigger and the trigger period is determined to match the signal period at 4µs. Simple counters are used to implement this functionality. Figure F.2.1 provides a diagram for the architecure of this module.

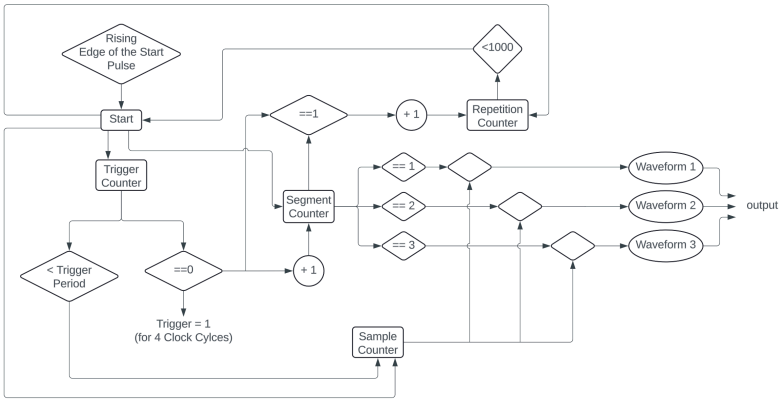



Figure F.2.1: Architecture of the Multiple Signal Generation Module

Module connections follow the same layout provided in Figure E.2.2.1.

Resource utilization is provided in table F.2.1.

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 3834 | 17600 | 21.78 |
| LUTRAM | 444 | 6000 | 7.40 |
| FF | 5486 | 35200 | 15.59 |
| BRAM | 45 | 60 | 75 |
| DSP | 1 | 80 | 1.25 |
| IO | 79 | 100 | 79 |
| BUFG | 5 | 32 | 15.63 |
| MMCM | 1 | 2 | 50 |

Table F.2.1: Post-Implementation Resource Utilization

Other than Block RAM and I/O, resource utilization is observed to be frugal.

I/O is utilized at 79%. As all the inputs required for this implementation are declared, the design does not require any reconsiderations.

75% of the available Block RAM is utilized. At this point of the project this does not require any reconsiderations.

Timing report is provided in Figure F.2.2.

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 0.746 ns | Worst Hold Slack (WHS): | 0.007 ns | Worst Pulse Width Slack (WPWS): | 1.845 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 5073 | Total Number of Endpoints: | 5045 | Total Number of Endpoints: | 2352 |

**All user specified timing constraints are met.**

Figure F.2.2: Timing Report

Positive WNS indicates that a signal may arrive earlier than expected, which infers no problems.

Results obtained from the ILA are provided in figures F.2.3 and F.2.4.

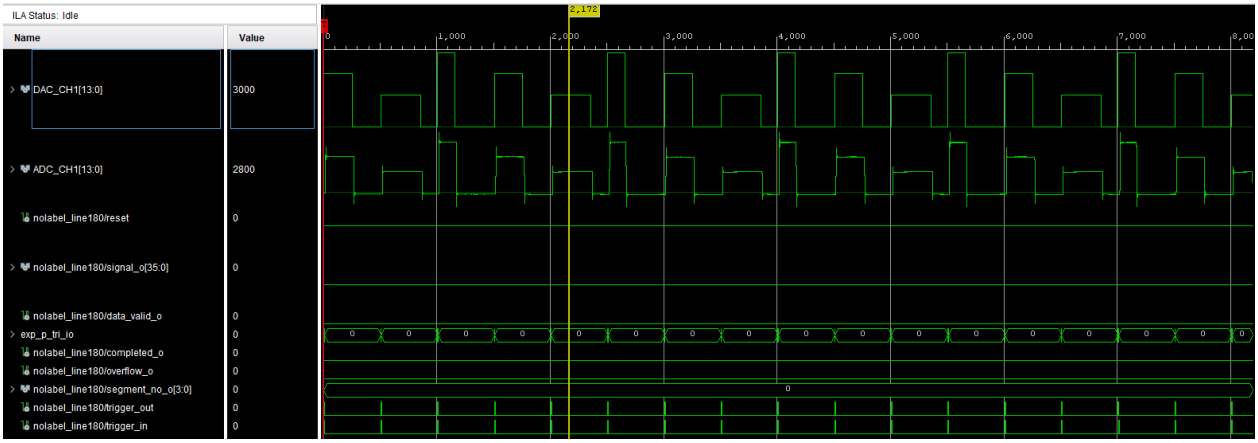



Figure F.2.3: Signal Generation Module Output Signal

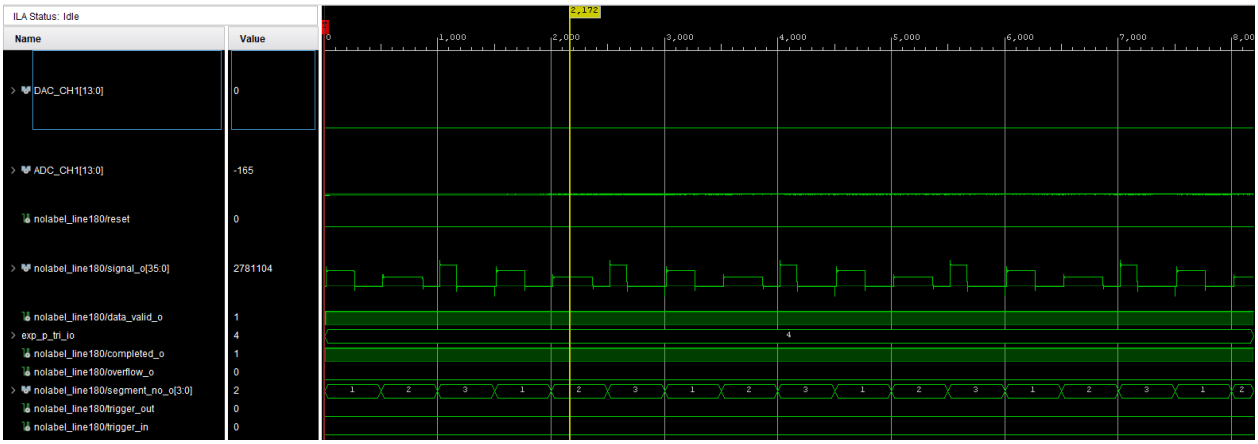



Figure F.2.4: Accumulator Module Output Signal

Some disturbance effects were observed on the output signal. Firstly it can be seen that the ADC input has a -20 mV (200 per magnitude resolution) offset. Another issue is that even with a 50Ω impedance mathcer, still some reflections are present. These issues are both non-firmware related and can be compensated with better components and calibration.

The accumulator module output signal was observed to have an amplitude 1000 times the ADC incoming signal. Hence, the measurement can be deemed a good match with the expected results.

## G. Verification Using a Pseudo-Randomly Generated Noisy Rectangular Waveform

### G.1. Introduction

In this part of the project, some noise will be added to the test signals generated in the previous signals. The noise will be pseudo-randomly generated on the FPGA and added to the signal. This will assess the design validity under real-life conditions.

Additionaly, a user interface through UART will be implemented. This user interface will configure the experiment parameters, and send the start and module reset signals.

### G.2. Implementation

#### G.2.1. Noise Generation through a Pseudo-Random Number Generator

It is impossible to achieve true randomness generated only by a deterministic machine. But the randomness can be improved through reducing the times a pattern occurs in a series of elements. A psuedo random number generator is implemented using a 168-bit Linear Feedback Shift Register. A 168-bit LFSR repeats itself only after a $3.741 \times 10^{50}$ rotations. Compared to a configuration like in this project where maximum number of samples is $2.048 \times 10^{10}$, this amount of randomness is sufficient enough.

***Linear Feedback Shift Register*** has a simple operating principle where some bits in the register are XOR'ed and placed in the least significant bit. The remaining bits are shifted one times to the left. The XOR'ed bits are named tapping points. The selection of these bits are optimized[1] to achieve the least possibility of encountering a pattern. For a 168-bit LFSR, the most optimized polynomial was discovered to be $x^{168} + x^{167} + x^{153} + x^{151} + 1$ where the $167^{\text{th}}$, $166^{\text{th}}$, $153^{\text{rd}}$ and the $151^{\text{st}}$ bits are XOR'ed.

The last 9-bits are of the implemented LFSR are assigned to a 9-bit signed "noise" register. Later, the generated noise of amplitude 256 (0.03125 V) is added on to the signal.

[1] https://docs.amd.com/v/u/en-US/xapp052 - Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators

Noisy signal generation architecture and the experiment architecture including processing system is provided in Figure G.2.1.1.

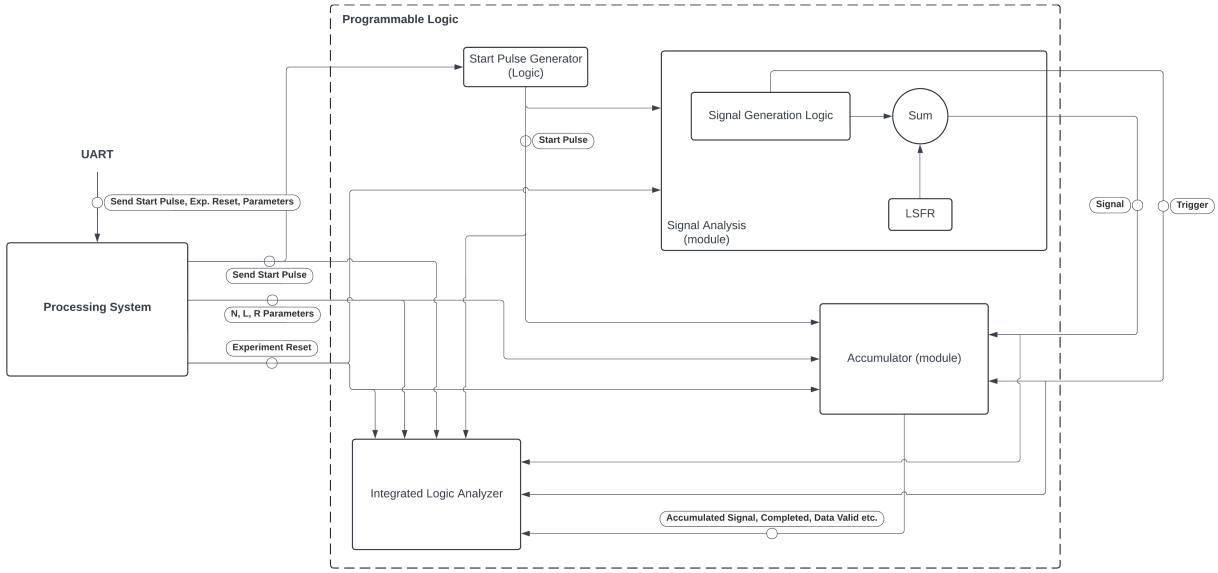



Figure G.2.1.1: Experiment Architecture

Resource utilization is provided in table G.2.1.1.

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 4045 | 17600 | 22.98 |
| LUTRAM | 496 | 6000 | 8.27 |
| FF | 5728 | 35200 | 16.27 |
| BRAM | 50 | 60 | 83.33 |
| DSP | 1 | 80 | 1.25 |
| IO | 59 | 100 | 59 |
| BUFG | 5 | 32 | 15.63 |
| MMCM | 1 | 2 | 50 |

Table G.2.1.1: Post-Implementation Resource Utilization

Only Block RAM usage was found to be concerning. Since the architecture relies heavily on the usage of debug tools which utilize the Block RAMs, this amount of cannot be ommited. An approach to reduce the Block RAM usage can be utilizing the FPGA I/O and using external devices to observe the results.

Timing report is provided in Figure G.2.1.2.

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 0.473 ns | Worst Hold Slack (WHS): | 0.036 ns | Worst Pulse Width Slack (WPWS): | 1.845 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 13534 | Total Number of Endpoints: | 13504 | Total Number of Endpoints: | 6729 |

**All user specified timing constraints are met.**

Figure G.2.1.2: Timing Report

Positive WNS indicates that a signal may arrive earlier than expected, which infers no problems.

Generated noisy signal and accumulation results are provided in figures G.2.1.3 and G.2.1.4.

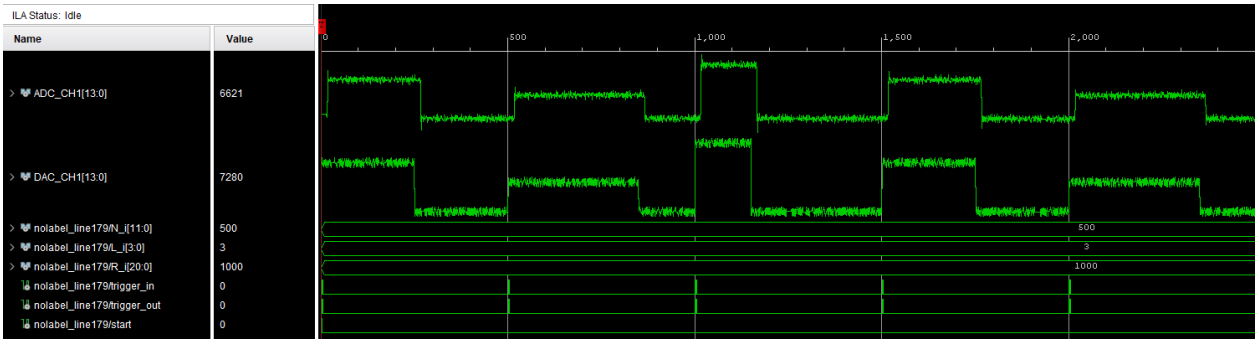



Figure G.2.1.3: Noisy Signal Generated from the Signal Generation Module

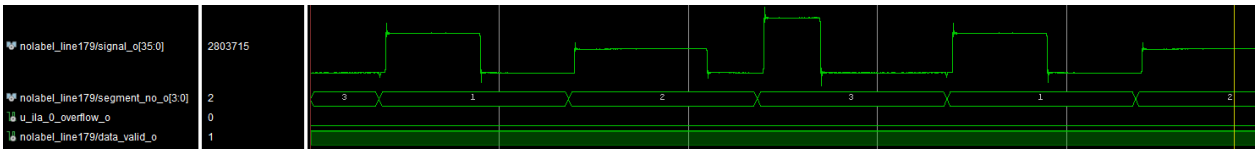



Figure G.2.1.4: Accumulated Noisy Signal

To ensure the accumulator works correctly with the different user defined parameters, R = 1 was sent from the UART. Obtained result is provided in Figure G.2.1.5.

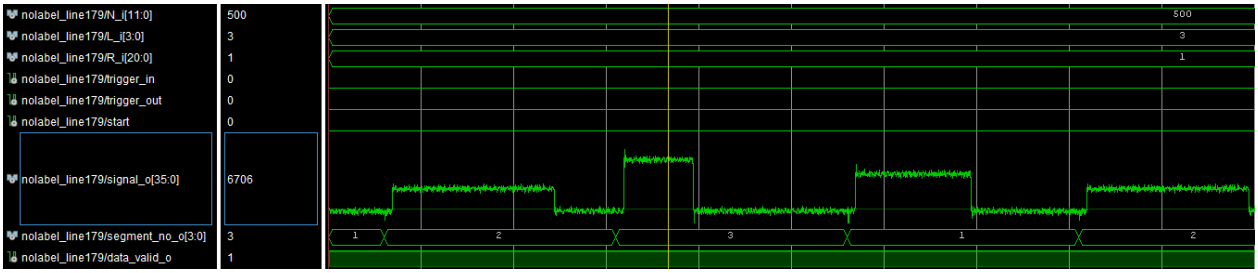



Figure G.2.1.5: Accumulator Output Signal with R = 1

It was observed that the accumulator is functioning as expected with different user defined paramters at runtime.

### G.2.2. UART-Based User Interface for Experiment Control

A UART based user interface is created for experiment control. This system handles the module resets, control of the start pulse and user defined parameters. System starts the accumulator,and signal generation modules and the “start pulse generation” logic in a reset state. This reset signal differs from the peripheral reset signal generated from the processing system. Peripheral reset resets each module in the FPGA which causes UART connection to drop. Hence, another “module reset” signal is used to reset only experiment modules.

This user interface can detect invalid inputs for each stage of interaction and prompt user with a warning to enter a valid input.

Experiment control flow is as follows:

1. User is prompted the current experiment setup and asked if they want to define a new setup.
   1.1. If the user selects yes, they enter the parameter they want to change in the next stage. The parameters are updated accordingly and the program will move on with the next stage.
   1.2. If the user selects no, the program will move on with the stage after the parameter definition stage.
2. The user is prompted the current updated experiment setup and asked whether they want to start an experiment with the current setup.
   2.1. If the user selects yes, the parameters are sent to the programmable logic unit, module reset is deasserted and the experiment start signal is sent to the programmable logic unit with the “send start pulse” signal.
   2.2. If the user selects no, the parameters are not updated in the programmable logic and the program will move on with the next stage.
3. The user is prompted if they want to continue with another experiment. In this stage, module reset is asserted without depending on the users choice. If the user selects yes, the modules will be ready for a new experiment. Otherwise, modules are resetted once again before the peripheral reset is asserted. If the user selects no at this stage, the program is ended.

An image of the user interface is provided in Figure G.2.2.1.

```
The current experiment setup is:
Sample Count: 2048
Segment Count: 10
Sequence Count: 1000000
Do you want to define a new experiment setup? (y/n): y
Change sample count (N), segment count (L) or sequence count (R). Enter (x) to stop defining parameters: N
Enter your desired sample count (max 2048): 500
The new sample count is 500 samples per segment
Change sample count (N), segment count (L) or sequence count (R). Enter (x) to stop defining parameters: L
Enter your desired segment count (max 10): 3
The new segment count is 3 samples per sequence
Change sample count (N), segment count (L) or sequence count (R). Enter (x) to stop defining parameters: R
Enter your desired sequence repetition count (max 1000000): 1000
The new sequence repetition count is 1000 sequences per waveform
Change sample count (N), segment count (L) or sequence count (R). Enter (x) to stop defining parameters: x
The current experiment setup is:
Sample Count: 500
Segment Count: 3
Sequence Count: 1000
Do you want to start the experiment with the selected parameters? (y/n): y
Do you want to set up another experiment?  (y/n): n
```

Figure G.2.2.1: UART and Terminal Based User Interface

# H. Verification Using Quantum Experiment Signals

## H.1. Introduction

In this part of the project, real quantum experiment signals will be fed to the modules and validation will be done using these signals. In a real world scenario, these signals would be sent through the ethernet connection. However in this project, these signals will be sent using on-board memoru. A key issue to consider here is that waveforms of these size require a large memory. Memory on the programmable logic unit is not enough for data of this size. On-board DDR RAM has a sufficient size but control is tedious for such memory units.

To keep the design as simple as possible, programmable logic block memory is used to store the signals. To fit the memory size constraints, the signals will be averaged using MATLAB and a generated noise will be added on the FPGA. While this is not a hundred-percent accurate representation of how this module would perform in a real-world experiment, this setup is sufficient for a proof-of-concept.

## H.2. Implementation

Provided quantum experiment signals were averaged using the mean command. Averaged signals are provided in Figure H.2.1.1.

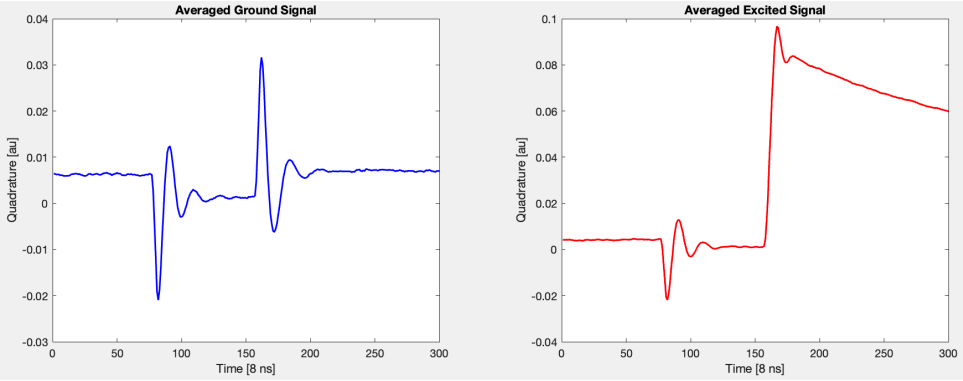
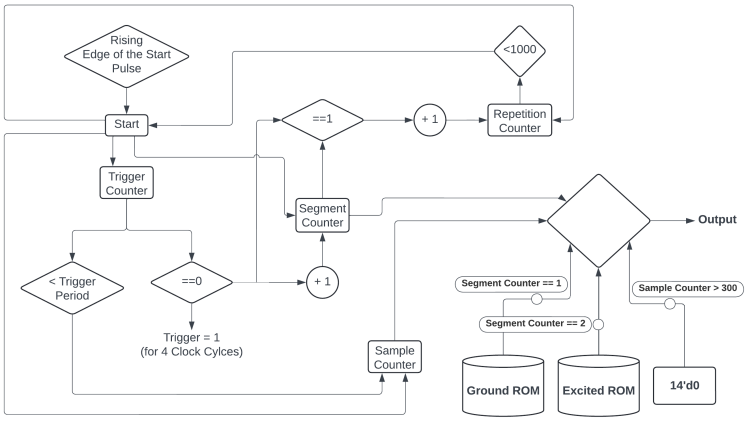



Figure H.2.1: Averaged Quantum Experiment Signals

Averaged quantum experiment signals were exported to a memory coefficient file with the correct initialization parameters. The generated memory coefficient files were used to initialize two ROMs of size 300.

Architecture of the signal generation module for generating noisy quantum signals is provided in Figure H.2.2.




H.2.2: Signal Generation Logic for Quantum Experiment Signals

Provided signal generation logic is a substitute for the “Signal Generation Logic” block inside the Signal Generation module in Figure G.2.1.1.

Resource utilization is provided in table H.2.1.

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 4050 | 17600 | 23.01 |
| LUTRAM | 496 | 6000 | 8.27 |
| FF | 5752 | 35200 | 16.34 |
| BRAM | 51 | 60 | 85 |
| DSP | 1 | 80 | 1.25 |
| IO | 59 | 100 | 59 |
| BUFG | 5 | 32 | 15.63 |
| MMCM | 1 | 2 | 50 |

Table H.2.1: Post-Implementation Resource Utilization

Only Block RAM usage was found to be concerning. Since the architecture relies heavily on the usage of debug tools which utilize the Block RAMs, this amount of cannot be ommited. An approach to reduce the Block RAM usage can be utilizing the FPGA I/O and using external devices to observe the results.

Timing report is provided in Figure H.2.3.

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 0.411 ns | Worst Hold Slack (WHS): | 0.030 ns | Worst Pulse Width Slack (WPWS): | 1.845 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 13654 | Total Number of Endpoints: | 13624 | Total Number of Endpoints: | 6757 |

**All user specified timing constraints are met.**

Figure H.2.3: Timing Report

Positive WNS indicates that a signal may arrive earlier than expected, which infers no problems.

Generated noisy signal and accumulation results are provided in figures H.2.4 and H.2.5.

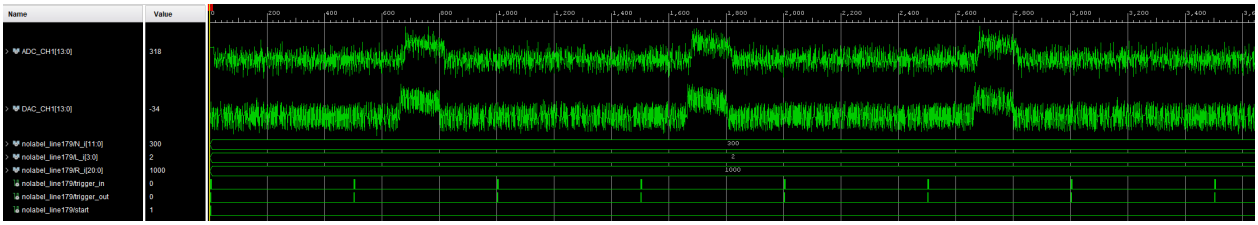



Figure H.2.4: Averaged Quantum Experiment Signals with Added Noise, Sent Throguh DAC

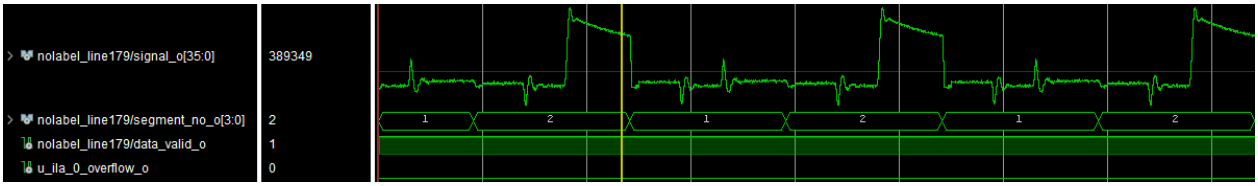



Figure H.2.5: Accumulated Quantum Experiment Signals

Acquired data was imported to MATLAB for a comparison. Signals sent from the DAC and received from the ADC are provided in Figure H.2.6.

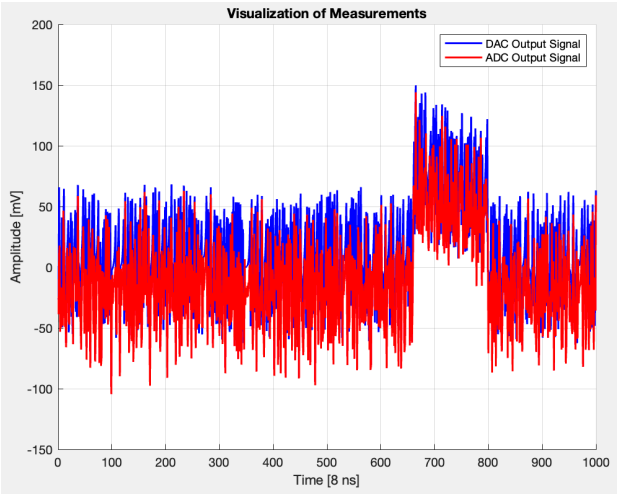



Figure H.2.6: DAC and ADC signals

Aforementioned ~20 mV negative voltage offset can be observed from the signals. When comparing the accumulated signal to the MATLAB averaged signal, this offset was compensated. Visual comparison of the signals is provided in Figure H.2.7.

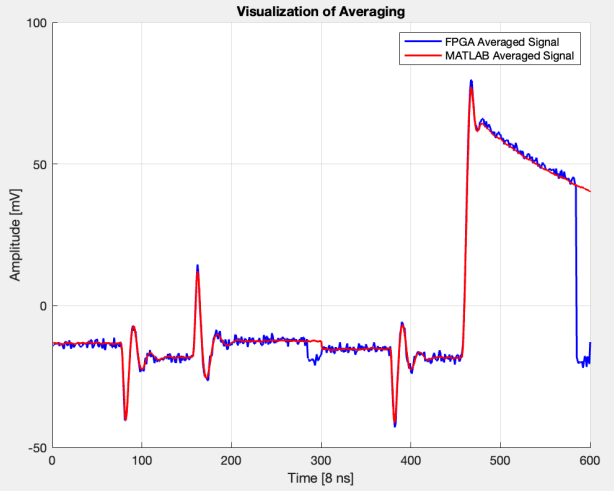



Figure H.2.7: Averaged Signals

The maximum error was found to be 4.05 mV. Compared to signal max $V_{pp}$, this is a 4.2% error rate. An error rate this low indicates a good match with the expected result and the conclusion that the system is working as expected can be made.

It was observed that the signals dropped to zero 26 samples earlier. This may be caused by the improper transmission of triggers and can be calibrated.