FPGA Implementation of Smoothing Filters

Tinashe Timba *TMBTIN004*
Murray Inglis *INGMUR002*
Dylan Kuming *KMNDYL001*
Dan Rom *RMXDAN002*

***Abstract*—This report evaluates the implementation of two distinct smoothing filters, namely the moving average filter and the Savitzky-Golay filter, on FPGA platforms. The moving average filter is straightforward, averaging data points within a sliding window, and is known for its simplicity and effectiveness in reducing high-frequency noise. In contrast, the Savitzky-Golay filter employs polynomial fitting to better preserve the signal's essential features, making it suitable for applications requiring high fidelity. This work establishes a golden standard through MATLAB implementations and assesses filter performance using Verilog implementations on FPGA. The comparative analysis aims to delineate the strengths and limitations of each filter, providing insights crucial for selecting the appropriate smoothing technique according to the application needs.**

I. Introduction

Smoothing filters play a crucial role in signal processing by reducing noise and enhancing the signal quality. In various fields such as biomedical engineering, telecommunications, and finance, the accurate extraction of meaningful information from noisy signals is important. This report focuses on two types of smoothing filters: the moving average filter(Section 1) and the Savitzky-Golay filter(Section 2). These filters were selected due to their distinct approaches to smoothing, offering a comparison of their effectiveness across different applications.

The moving average filter is a fundamental technique in signal processing and it operates by averaging a set of adjacent data points within a sliding window. This method smooths out high-frequency noise while preserving the overall trend of the signal. However, its performance may be limited in scenarios where rapid changes in the signal occur or when dealing with non-linear trends.

In contrast, the Savitzky-Golay filter employs polynomial fitting within a moving window to estimate the value of each data point. By fitting a polynomial to the data, this filter not only reduces noise but also preserves important features and shapes within the signal. Its adaptability to different signal characteristics makes it particularly useful in applications where maintaining the integrity of the underlying data is essential.

The implementation of these filters was carried out using MATLAB to establish a golden standard in software, providing a benchmark for comparison. Furthermore, to demonstrate their practical application and performance in real-world scenarios, the filters were also deployed on hardware platforms using Verilog for FPGA (Field-Programmable Gate Array) deployment.

Through this comparative analysis and practical demonstration, this report aims to provide insights into the strengths and limitations of the moving average filter and the Savitzky-Golay filter, enabling informed decisions in selecting the most suitable smoothing technique for specific signal processing tasks.

II. Background

A. Moving Average Filter

The moving average filter is one of the simplest and most commonly used smoothing filters in digital signal processing. It operates by averaging a set of adjacent data points within a sliding window, effectively reducing high-frequency noise while preserving the overall trend of the signal.

1) Types of Moving Average Filters: Moving average filters can be categorized into simple, cumulative, and exponential types. The simple moving average (SMA) calculates the average of data within a sliding window. The cumulative moving average (CMA) averages all data up to the current point, and the exponential moving average (EMA) applies more weight to recent data, making it more responsive to new changes.

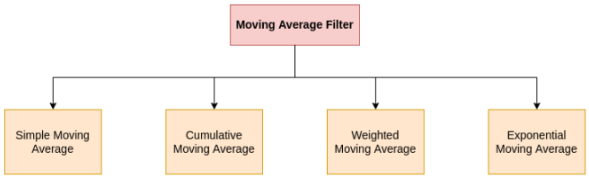


Fig. 1. Types of Moving average filters [1]

2) Mathematical Description: The mathematical implementation of the moving average filter is as follows:

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j] \tag{1}$$

where M is the number of points in the sliding window, and $x[i+j]$ are the input signal values.

3) Basic Implementation: In a basic implementation, the moving average filter slides over the data point sequence, computing averages repeatedly, which smoothens the signal effectively. This can be efficiently implemented using a circular buffer to store the recent M values of the input signal.

4) Limitations: While moving average filters are effective for reducing random noise, they tend to blur sharp edges, reduce signal resolution, and introduce a lag in signal processing.

5) Applications: Moving average filters are widely used in finance for trend analysis in stock prices, in digital signal processing for noise reduction, and in real-time systems for sensor data smoothing.The figure below shows the moving average in trend analysis.

Fig. 2. Trend Analysis [1]

B. Savitzky-Golay Filter

1) Mathematical Description: The Savitzky-Golay filter enhances the moving average by fitting successive subsets of adjacent data points with a low-degree polynomial by the method of linear least squares. The general form of the filter's output is expressed as:

$$y[n] = \sum_{k=-m}^{m} c_k x[n+k] \tag{2}$$

where c_k are the coefficients derived from the polynomial fit, providing optimal trade-off between smoothing and fitting.

2) Basic Implementation: The basic implementation involves calculating the convolution coefficients c_k that are used to multiply with the input data points in the window. This process involves solving a set of linear equations derived from polynomial least squares fitting.

3) Limitations: The Savitzky-Golay filter preserves higher moments better than a simple moving average filter, but it is more computationally intensive and can introduce artifacts when the window size does not match the signal characteristics.

4) Applications: Due to its ability to preserve features of the signal such as relative maxima, minima, and width, the Savitzky-Golay filter is extensively used in spectroscopy, chromatography, and in any application where peak-preserving smoothing is crucial.Below is a figure showing the filter working on a some gene data.

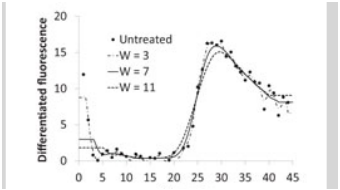


Fig. 3. Savitzky Golay polynomial fitting [2]

C. Gradient Descent

A feature worth exploring is that of gradient descent primarily for the purpose of increasing polynomial order.Gradient descent is an optimization method used to adjust polynomial coefficients by iteratively minimizing errors between the fitted polynomial curve and the given data points. In general, it works by computing the gradient of the loss function with respect to the coefficients and updating them in the direction that reduces the error. This process continues until convergence, resulting in a polynomial model that closely fits the data. It's particularly effective for applications like curve fitting, where minimizing the fitting error is essential. [3] Gradient

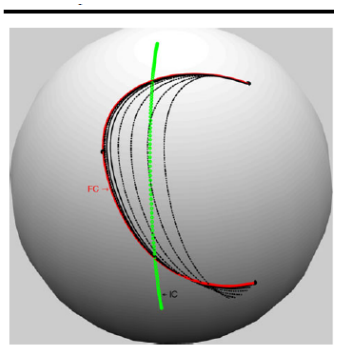


Fig. 4. Curve Fitting [3]

descent in the Savitzky-Golay filter can be used to fine-tune the polynomial coefficients and improve the fitting accuracy of the data smoothing. This would involve setting up an objective function representing the fitting error, and then iteratively adjusting coefficients to minimize that error.

III. Methodology

This section outlines the methodology used for this project, utilizing the to run the final versions of the designed filters. Due to time constraints, the project was split into two main sections: the first focuses on the moving average filter, and the second on the more complex Savitzky-Golay filter. Starting with the simpler moving average filter allows for a smoother transition to the complexities of the Savitzky-Golay filter, ensuring a streamlined development approach.

A. Section 1:Moving Average

Development of the moving average filter began with a review of existing models, particularly MATLAB's built-in moving average filter, which served as a benchmark for initial testing. In Verilog, we started by instantiating a module named SF (Smoothing Filter) to implement the moving average algorithm. This involved designing a simple data path that averages a window of input values to smooth out fluctuations.After setting up the algorithm, a comprehensive test bench was developed to simulate the filter with various window sizes and noise levels to assess its robustness and efficiency. Upon successful simulation, the final step was to deploy and test the filter on the FPGA, ensuring it operates correctly under real-world conditions.

B. Section 2: Savitzky Golay

The implementation of the Savitzky-Golay filter expanded on the foundation laid by the moving average filter but introduced additional complexity. This filter requires supplementary modules for polynomial fitting and coefficient calculations. In Verilog, implementing these functions involved developing advanced mathematical models that could handle polynomial approximations efficiently. After constructing the main processing modules, we developed a test environment similar to that of the moving average filter, allowing us to test and refine the filter under various scenarios. This rigorous testing ensured that the Savitzky-Golay filter met the required performance criteria before its final deployment on the FPGA.

C. Hardware

Design can not occur without nting the hardware on which the smoothing filters are to be implemented.The device used to run the Matlab implementation and the Verilog is a laptop with the specifications listed below: **Insert laptop specs** The FPGA used to run the implementation is the Insert laptop specs

D. Software

The selection of software tools was a pivotal aspect of our methodology, ensuring efficiency, accuracy, and user-friendliness throughout the development of the smoothing filters. MATLAB was chosen for its robust mathematical and simulation capabilities, setting a benchmark for the performance expected from the Verilog implementation. Verilog was selected due to its compatibility with FPGA environments.

1) Matlab: **Purpose:** MATLAB was utilized primarily for its powerful mathematical and simulation capabilities. It served as the platform for developing and validating the golden standard models of the Moving Average and Savitzky-Golay filters.

Features Used:

Signal Processing Toolbox: This toolbox offers comprehensive resources for analyzing, designing, and simulating signal processing systems. It was instrumental in designing and testing the smoothing filters, allowing us to simulate performance and verify algorithm correctness before hardware implementation. By using MATLAB, we were able to create a "golden standard" — a benchmark model that the Verilog implementation needed to meet or exceed in performance.

2) Verilog: **Purpose**: Chosen for its direct compatibility with FPGA technology, Verilog enabled the practical implementation of the designed filters onto hardware.

Features Used: Simulation and Synthesis Tools: Verilog's simulation tools were used to model the digital circuits that would implement the smoothing filters, ensuring that each filter's design was optimized for performance and resource utilization. Synthesis tools were then used to translate the Verilog code into a design that could be physically implemented on an FPGA, allowing for rigorous testing and validation of the functional correctness and performance of the filters.

By leveraging the strengths of both MATLAB and Verilog, we ensured a robust development process that combined thorough simulation testing with practical hardware implementation, resulting in efficient and effective smoothing filters.

3) Vivado 2023.2: **Purpose**: Vivado 2023.2 is the design suite from Xilinx Studios that is used to generate a bitstream from Verilog code, as well as write this bitstream to the Nexys 4 DDR FPGA. This allows the project to move from a simulated environment to a physical environment.

Features Used: Throughout the process of snythesizing the filter Verilog module onto the FPGA, the following Vivado features were used: Synthesis Design, Implementation Design, Bitstream Generator, as well as the Hardware Manager to write the generated bitstream to the actual FPGA.

E. Proposed Perfomance Metrics

To validate the performance of the smoothing filters implemented in Verilog , the following metrics will be recorded and analyzed:

Signal-to-Noise Ratio (SNR): This metric will measure the quality of the filter output by comparing the level of the desired signal to the level of background noise. A higher SNR indicates a cleaner extraction of the signal from noise, which is critical for the effective performance of smoothing filters. **Mean Squared Error (MSE)**: MSE will be used to quantify the difference between the values predicted by the filter and the actual values of the input signal. This metric is crucial for assessing the accuracy of the filter in replicating the desired output.

F. Expected Results

Based on the performance metrics outlined, the following results are anticipated:

Conformance to the Golden Standard: The Verilog implementations of the Moving Average and Savitzky-Golay filters are expected to meet or exceed the performance benchmarks established by the MATLAB simulations in terms of SNR and MSE. This would validate the effectiveness of the Verilog design relative to the Matlab model.

Low SNR: The signals are expected to have a low SNR if the filters are working as expected

IV. Design

A. Section 1

1) Moving Average Module: In Verilog, the filter is designed to process input from a CSV file and generate filtered output based on a specified window size.

Design Parameters:

1) Filter Size (M): Defines the window size for averaging data points.
2) Input Data Length (len): Specifies the total number of input samples to be processed.

Internal Structure:

Initialising: The module begins by reading data from a CSV file named data.csv. Using $fopen to open the file and $fscanf to read each sample, data is stored sequentially into a data array until the end of the file is reached.

Reset Logic: The rst signal is used to reset the internal state and control registers, such as sum, idx, and others, ensuring that the module is ready for the next filtering operation.

Filtering Logic:

1) The filtering starts when the start signal is high and works on the positive edge of the clk.
2) The started flag is set to indicate the beginning of filtering.
3) The main filter loop processes each data point sequentially by summing up M consecutive samples starting from the current index (i) and calculates the average by dividing by M.
4) The averaged value is stored in the filtered_data array.
5) To prevent accessing invalid data indices, the loop only iterates up to len - M.
6) These end values are padded with the value at the index of len - M, so that they are not left floating

Output:

1) The filtered results are concatenated into a string format.
2) The output is saved to a new CSV file named output.csv using $fwrite.

2) Moving Average Testbench: The moving average testbench module facilitates the simulation of the moving average module by providing it with test vectors. The testbench instantiates the moving average module with the required reset, start and clock inputs. The reset line is set high and after 10 time units it is set low again. The same is done with the start line. The testbench then ends the simulation after a sufficient amount of time. Some of the structures used in the moving average testbench are only applicable in simulation and will need to be changed for synthesis. For example, the loop used for filtering the data happens in one clock cycle in simulation.

3) FPGA Implementation: While the above design provides a proof of concept for the moving average smoothing filter a few changes are made to the Verilog code to ensure the module can be deployed to the Nexys 4 DDR FPGA.

Input Data Acquisition:
The FPGA is not able to read directly from a CSV file for data acquisition. Therefore the way in which the data is acquired must be altered. Whereas in the initial version, the input signal is read directly from a CSV file into the smoothing filter module, the input is now first read into RAM, through a Memory Control Unit (MCU). Additionally, the data being written to RAM is transmitted via UART from an external device, to the FPGA. The address of RAM to which the input is written, is determined by the *src* input which is to be provided as a input. This is done by representing the desired source address in binary using the first 8 switches on the Nexys 4 (SW 15 to SW 8).

Filtering Process:
Once the input signal has been written to RAM in the specified address, a LED is turned on to indicate the process is done and the filtering can begin. An input is to be provided by the user, in the form of the push of a push-button, to begin the filtering.

The filtering process itself is also slightly adjusted. In the first design, the filtering took place in a simple for-loop over the span of a single clock pulse. However, in the synthesizable version, the filtering takes place over as many clock cycles as there are data points in the input. In each clock pulse, a value of the signal is read from RAM and added to a FIFO buffer. This buffer holds M values, where M is the window size of the filter. The value of the filtered signal at any point is then the average of the elements inside the buffer.

Data Storage:
After the entire input signal has been processed, the filter module raises a *done* flag, at which point the smoothed signal is written to RAM, starting from address *dest* - which is provided as input by the user. This is provided by representing the desired destination address of the filtered signal in binary using the last 8 switches available on the Nexys 4 (SW 7 to SW 0). Once again, an LED is turned on to indicate the filtering is complete.

At this stage, the user can signal via the push of a push-button, for the smoothed signal to be transmitted via UART back to the external device from where it came. This allows users to visualise the result of the smoothing filter if they choose.

B. Section 2

1) Savitzky Golay Module: Like the moving average module, this module reads input from a CSV file and generates smoothed data based on a specific window size. **Design Parameters:**

1) Window size (M): Defines the window size (number of data points) that a curve will be fitted. This is typically an odd number due to the midpoint being used in the curve fitting.
2) Input data length (len): Specifies the total number of input samples to be processed

Internal Structure: Initialising: Again, like the moving average module, this module begins by reading data in from a CSV file named data.csv. This code is copied over and works in the exact same manner. Pre-calculated coefficients for the window-size are read in from a stored Savitzky-Golay coefficients table. **Reset Logic:** When the rst line is set high, the internal registers are reset to 0 values to ensure that the module is ready for the next filtering operation. **Filtering Logic:**

1) The filtering starts on the rising edge of the clock signal when the start signal is high
2) The started flag is set to indicate the beginning of filtering
3) The main filter loop processes each window size M, fitting a curve to the points in M and using the point at the centre of that curve as the new value for the point that has its position at the centre of the window.

4) Since the midpoint is used, the first M/2 and last M/2 + 1 values are padded as the first and last points of the output data
5) The smoothed data is stored in the data_out array

Output:

1) The smoothed output data is concatenated into a string with new line characters for each row
2) The output is saved to a CSV file named output.csv

C. Savitsky-Golay Testbench

This module works in exactly the same way as the moving average testbench module. It instantiates the Savitsky-Golay module with its required clock, reset and start wires. It then resets the module and starts it.

D. FPGA IMPLEMENTATION

Although the Savitzky-Golay filter has not been synthesized on to an FPGA, the process of doing so would follow a very similar, if not identical, procedure to the moving average filter.

V. Proposed Development Strategy

The proposed system has potential application as an an FPGA-based commercial product providing flexible and optimized filtering solutions using Moving Average (MA) and Savitzky-Golay (SG) filters.The focus would be industries requiring noise reduction and signal enhancement such as the telecommunications industry or the medical industry.

Development Approach: The product could provide an interface for end users to specify window size, input data length and other filter parameter which could be achieved by Graphic User Interfaces.Not only could they select filter parameters but users would be able to select or combine the effects of both filters.

Supporting Tools and Framework We could provide development kits with pre-configured FPGA designs, hardware, and comprehensive documentation for ease of integration into client applications. Furthermore a GUI to set filter parameters and visualize real-time data with supporting APIs for integrating the filtering system into existing workflows.Like other commercial products in the market a comprehensive SDK (Software Development Kit) to accelerate application development could be offered. This SDK would include sample projects demonstrating common filtering applications, prebuilt libraries to handle common data formats and protocols and Tutorials and API documentation for customizing the filters.

VI. Planned Experimentation

To systematically evaluate the performance of the implemented smoothing filters, a detailed experimental setup has been designed. The experimentation primarily revolves around the application of the golden standard, established through MATLAB simulations, against the FPGA implementations coded in Verilog.

A. Experimental Setup

The experimental process involves deploying both the moving average and Savitzky-Golay filters on FPGA platforms. Initial validation is performed through MATLAB to ensure the correctness of the algorithms and to establish baseline performance metrics such as Signal-to-Noise Ratio (SNR), Mean Squared Error (MSE) and execution time.

B. Implementation of Experiments

For each filter, the following steps are executed:

1) **MATLAB Simulation:** Generate the output using MATLAB for predefined input signals to establish the golden standard.
2) **Verilog Deployment:** Implement the filters in Verilog and deploy them on an FPGA. The inputs identical to those used in MATLAB simulations are fed into the FPGA.
3) **Data Collection:** Capture the output from the FPGA for analysis.
4) **Command Execution:** Use specific commands in the FPGA's control software to initiate and monitor the filtering process, ensuring each command aligns with the intended filter operation.

C. Use of Golden Measure

The golden measure, or the results from the MATLAB simulations, serves as a benchmark for comparing the accuracy and efficiency of the FPGA implementations. This comparison is crucial for validating the fidelity of the hardware deployment. Differences between the MATLAB output and the FPGA output are carefully analyzed to assess any discrepancies and potential improvements.

By methodically following this planned experimentation, the study aims to rigorously assess the practical utility and efficiency of the smoothing filters under real-world operational conditions.

VII. Results

This section discusses the outcomes obtained from both the MATLAB simulations (golden measure) and the FPGA implementations. The analysis includes comparisons of Signal-to-Noise Ratio (SNR), Mean Squared Error (MSE), and execution time, with considerations of their implications for practical applications.

A. MATLAB Implementation

The MATLAB simulation served as the golden standard for benchmarking the FPGA implementation. The following results were achieved:

TABLE I
Performance metrics for MATLAB implementations.

Filter	Execution Time (s)	SNR (dB)	MSE
Moving Average	0.005790	19.0258	0.006251
Savitzky-Golay	0.002773	15.8647	0.012944

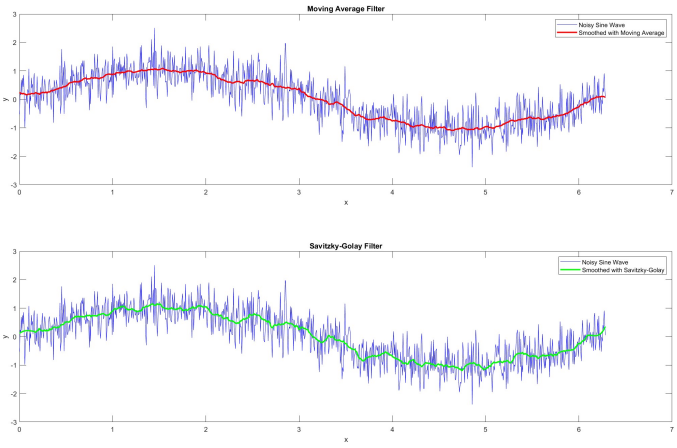


Fig. 5. Effects of Moving Average (top) and Savitzky-Golay (bottom) filters on a noisy sine wave, demonstrating their smoothing capabilities.

B. Verilog Simulation Results

The following graphs show the output of the moving average module and the Savitsky-Golay module from simulation. It

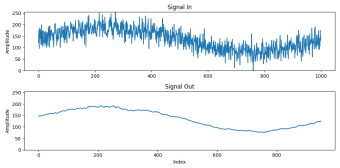


Fig. 6. Output from simulated moving average filter

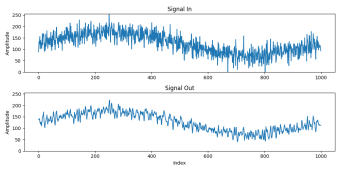


Fig. 7. Output from simulated Savitsky-Golay filter with a window size of 7

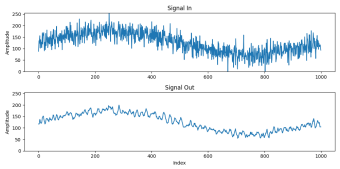


Fig. 8. Output from simulated Savitsky-Golay filter with a window size of 15

is important to note that the signals are scaled to values from 0 to 255 for the verilog simulation, this results in a higher relative MSE. From the results of the simulation, it can clearly be seen that the moving average filter and Savitzky-Golay filter improve the signal quality. Noise has been removed and there is an adequate signal-to-noise ratio. This confirms the functioning of the algorithms that can now be modified for synthesis on the FPGA.

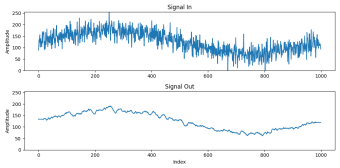


Fig. 9. Output from simulated Savitsky-Golay filter with a window size of 31

TABLE II
PERFORMANCE METRICS FOR SIMULATION.

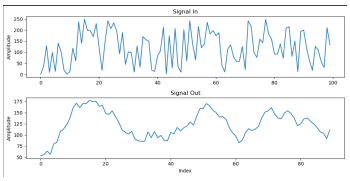
Filter	Execution Time (s)	SNR (dB)	MSE
Moving Average	0.084546	8.9054	3.224
S-G (Window size 7)	0.063010	8.843	3.182
S-G (Window size 15)	0.054671	9.0133	3.059
S-G (Window size 31)	0.169658	9.094	3.003

C. FPGA Implementation

The results from the FPGA implementation are crucial for validating the effectiveness of the hardware application. The exact metrics will be provided here

TABLE III
PERFORMANCE METRICS FOR FPGA IMPLEMENTATIONS.

Filter	Execution Time (s)	SNR (dB)	MSE
Moving Average	*10.32*	*8.10912*	*4.135*



D. Discussion

The comparison between MATLAB and FPGA results highlights the translation from simulation to real-world application. The analysis of execution times, SNR, and MSE values provides insight into the scalability and efficiency of hardware implementations. It is clear that the FPGA implementation takes a significant longer amount of time to run compared to the simulated version. This is due to the extra time added because of the UART transmission between computer and FPGA. When measured, the total amount of execution time in which the FPGA was receiving or sending data via UART was 10.1 seconds. Therefore, the actual time it took to apply the filter is 0.22 seconds - only a slight drop from the simulated version. This drop is due to the more rigidly clocked nature of the synthesized version.

VIII. ACCEPTANCE TESTING PROCESS (ALPHA TESTING)

A. Introduction

Alpha testing is conducted to verify the implementation of smoothing filters on the Nexys 4 DDR FPGA board. This phase focuses on testing the software and hardware integration

in a controlled lab environment to simulate expected real-world operations.

B. Testing Objectives

The main objective of alpha testing is to ensure the FPGA implementation of the smoothing filters meets the design specifications and performs effectively under simulated conditions. Specific goals include:

- Verifying the correct implementation of the moving average filter
- Testing that the FPGA board is capable of receiving data from, as well as sending data to, an external device using UART
- Ensuring the FPGA board handles data processing within expected time frames.

C. Test Setup

- **Hardware:** Nexys 4 DDR FPGA board.
- **Software:** Custom test scripts written in Verilog to implement and test the filters.
- **Data Sets:** Synthetic signals generated to simulate typical input scenarios. These signals are generated using a Python script, and are specifically designed to be noisy.

D. Execution of Test Cases

In order to execute each one of the test cases outlined above, the following procedures are followed:

Verifying the correct implementation of the moving average filter simulation:
As described in an earlier section, a testbench is developed to verify the simulations of both the moving average filter and savitzky-golay filter function as expected when supplied with noisy signals.

Testing that the FPGA board is capable of receiving data from, as well as sending data to, an external device using UART:
Vivado is used to generate a bitstream and upload this to the FPGA. A python script is used, with pyserial library, to implement the UART communication of a noisy signal from the computer to the FPGA. The filter is then applied to this, and sent back to the computer vvia UART. The signal is then plotted and compared to the golden measure equivalent to measure correctness and verify the filter functions correctly. The original, and returned, signal used for testing can be seen above in Section VII.C.

E. Results and Documentation

- Detailed results of each test case are documented, including performance metrics and any anomalies observed.
- Documentation includes test logs, issue resolution logs, and final assessment reports.

F. Conclusion

The alpha testing phase aims to validate the functional and performance aspects of the FPGA-based smoothing filters, ensuring they are ready for further beta testing with potential end-users. This phase is crucial for identifying any critical issues before the system is deployed in a real-world scenario.

IX. Conclusion

This project set out to assess the implementation of moving average and Savitzky-Golay smoothing filters on FPGA platforms. Our objectives were to demonstrate these filters' capabilities in noise reduction and signal integrity preservation, which are critical in various signal processing applications. Throughout this project, we established a golden standard through MATLAB simulations and proceeded to implement both these standards in simulation using Verilog. Furthermore, the moving average filter was physically realised by implementing it on to a Nexys 4 DDR FPGA board.

The results from both the MATLAB and FPGA implementations have shown that the moving average filter efficiently reduces high-frequency noise while preserving the signal's overall trend. However, it is somewhat limited in handling rapid signal changes. Conversely, the Savitzky-Golay filter, while more computationally intensive, excels in maintaining essential signal features, making it highly suitable for applications that require high fidelity.

While the objectives outlined in the design phase have largely been met, there are areas for improvement. For future iterations of this project, the following enhancements are recommended:

- Optimization of the Verilog implementation to improve computational efficiency and reduce resource utilization.
- Extended testing under varied operational conditions to ensure robustness and reliability of the filters.
- Exploration of additional smoothing techniques that could be integrated to enhance filter performance further.

In conclusion, the project has successfully demonstrated the practical application and effectiveness of these smoothing filters on FPGA platforms. The insights gained from this study will guide the selection and implementation of appropriate filtering techniques for specific signal processing tasks, pushing the boundaries of what is possible with FPGA-based implementations.

References

[1] A. C. Salian. (2022) Moving average filter: Towards signal noise reduction. [Online; accessed 08-May-2024]. [Online]. Available: https://codemonk.io/blog/moving-average-filter/

[2] C. Gaudreault, J. Salvas, and J. Sirois, "Savitzky–golay smoothing and differentiation for polymerase chain reaction quantification," *Biochemistry and cell biology*, vol. 96, no. 3, pp. 380–389, 2018.

[3] C. Samir, P.-A. Absil, A. Srivastava, and E. Klassen, "A gradient-descent method for curve fitting on riemannian manifolds," *Foundations of computational mathematics*, vol. 12, no. 1, pp. 49–73, 2012.