STUDY ON FPGA BASED COMPUTATION UNITS FOR OCULAR ARTIFACT REMOVAL FILTER ALGORITHM FOR MULTI-CHANNEL ELECTROENCEPHALOGRPHY (EEG)

by

Vishal Reddy Reddy

A thesis submitted to the Department of Electrical and Computer Engineering,
Cullen College of Engineering
in partial fulfillment of the requirements for the degree of

Master of Science
in Electrical and Computer Engineering

Chair of Committee: Dr. Jose L. Contreras-Vidal
Committee Member: Dr. Yuhua Chen
Committee Member: Dr. Xin Fu

University of Houston
May 2024

ABSTRACT

Introduction: The goal of this research to design and validate detailed computational units required for the implementation of an adaptive noise canceling (ANC) algorithm for ocular artifact removal on Field Programmable Gate Arrays (FPGA) for multi-channel electroencephalography (EEG) signals. The primary objective is to analyze the mathematical representation of the ocular artifact removal algorithm and convert it to hardware implementable computational modules. Input is taken from an existing scalp EEG recording dataset from EEG signal acquisition research at the University of Houston.

Methods: Considering the non-stationary nature of ocular artifacts and their adverse effect on EEG signal integrity, the dynamic ANC technique was selected. This technique is based on the H-infinity algorithm, developed by Kilicarslan, Grossman and Contreras-Vidal (2016), renowned for its robust ocular artifact removal capabilities. It also eliminates signal amplitude bias and local/global drifts. The H-infinity algorithm was successfully implemented in hardware using Verilog, incorporating custom divide, 3×3 matrix inversion, and the core H-infinity module. Operating in a 64-bit fixed-point format, the design produced expected outcomes and demonstrated a high correlation with MATLAB-simulated reference results.

Discussion: The H-infinity algorithm encompasses multiple computation stages, generating sizable sub-products that can lead to data overflow due to fixed-point number range constraints. Overcoming this limitation involved experimentation to strike the optimal balance between the integer and fractional parts. Through strategic design techniques, implementing this intricate ANC algorithm on FPGA became feasible, leveraging FPGA's parallel processing capabilities and accelerating the H-infinity algorithm on hardware.

Significance: This research serves as a foundation for FPGA-based implementation of real-time ANC algorithms for mobile brain-body imaging and brain-computer interface applications. This work not only contributes to the technical domain of EEG signal processing and adaptive noise cancelling, but also has the potential to enhance the quality of life for individuals relying on BCI technologies for communication and control, marking a significant step forward in the intersection of neuroscience and technology.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

1.1. Introduction to Electroencephalography in BCI Technology

Electroencephalography (EEG) stands at the forefront of Brain-Computer Interface (BCI) technology, serving as a crucial element in its development and progress. These EEG signals, which represent the electrical manifestations of brain activity, can be obtained non-invasively from the scalp. They play a pivotal role in BCIs by providing a direct pathway for interpreting and translating neural activity into commands that can control external devices or computer systems, thereby bridging the gap between human cognition and machine operations.

One of the key advantages of EEG-based BCI systems is their ability to utilize the brain's natural electrical activity for communication and control, eliminating the need for physical movement. This feature holds significant implications, particularly for individuals with severe motor impairments, offering them a new means of interaction with their environment. Additionally, the non-invasive nature of EEG makes it an attractive choice for BCI applications, minimizing risks and discomfort for users.

1.2. Challenges in EEG Signal Processing for BCI Systems

EEG signals come with their own set of challenges. EEG is a method used to measure the brain's electrical activity through small metal electrodes attached to the scalp, typically organized within an EEG cap (as shown in Fig 2.1.) The accuracy of signal interpretation directly impacts system precision. A standard EEG signal exhibits voltages ranging from 10−100 μV and operates within a frequency range of 0.1−100 Hz. The low amplitude characteristics of EEG signals necessitate advanced signal processing techniques to differentiate relevant brain activity from various artifacts that can interfere with useful EEG data.

From [4], major noise sources contributing to signal contamination include:

- Ocular artifacts stem from eye movements or blinks,
- Cardiac artifacts result from heartbeat volume conduction,
- Muscle artifacts originate from facial expressions, and
- Motion artifacts arise from head and body movements.

Additionally, challenges to signal quality arise from:

- Line noise from AC current sources,
- Environmental changes like white noise or wire movements,
- Variations in electrode skin impedance due to sweat,
- Local/global drifts, and
- Signal amplitude biases from EEG sources.

This research primarily addresses signal amplitude biases, local/global drifts, and ocular artifacts, which significantly impact EEG signal fidelity due to their proximity to EEG electrodes.

1.3. Importance of Artifact Removal and Signal Processing

[4] highlights the overlap in amplitude and frequency ranges between ocular, muscle, and cardiac artifacts (depicted in Fig 1.1) and EEG signals. This overlap leads to signal contamination and misinterpretation of data.

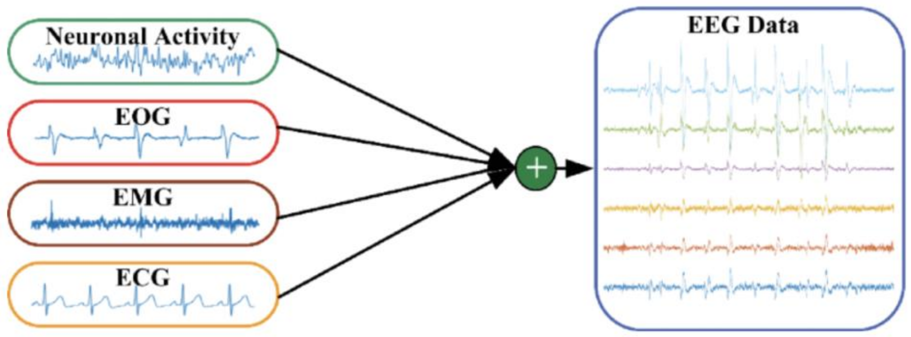


Fig 1.1. Artifacts contaminating EEG signals, from [4].

In another study from [5], researchers observed a time-varying difference in the impedance of EEG electrodes throughout an experiment, displaying predictable behavior. This variance, as illustrated in Figure 1.2, can be analyzed, and mitigated using various artifact removal techniques outlined in detail in Literature Study.

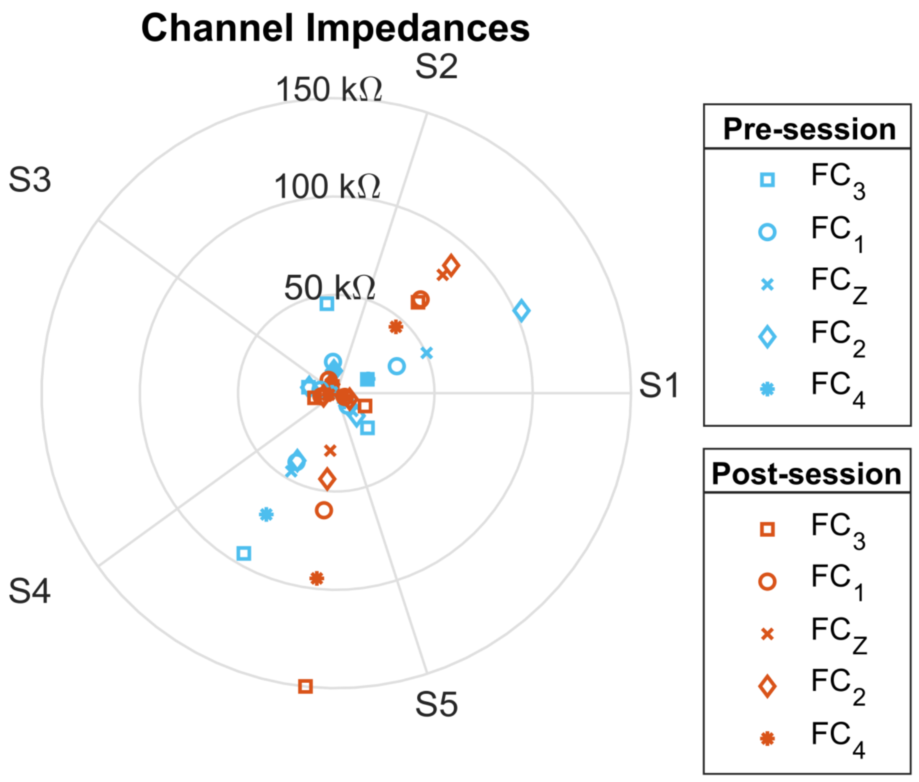


Fig 1.2. From [5], Channel Impedance: Impedance values from the open loop sessions for 5 participants. The values were taken before (blue) and after (orange) the session. The values are in kΩ.

The findings from these references underscore that the non-stationary time varying frequency characteristics of EEG signals requires an adaptive noise cancellation filter which is robust to such vast variety of artifacts in EEG-based BCI systems. These techniques are crucial for enhancing signal fidelity and improving system performance due to the complex nature of brain's electrical signals and the presence of potential contaminants.

1.4. Potential Applications and Impact of EEG-Based BCIs

The potential applications of EEG-based BCIs are vast and varied, ranging from medical rehabilitation and assistive technologies to gaming and virtual reality. In the medical field, BCIs can provide alternative communication methods for patients with conditions like amyotrophic lateral sclerosis (ALS) or severe strokes, empowering them to interact with their surroundings and express their needs. Beyond clinical applications, BCIs are also exploring new frontiers in enhancing human-computer interaction, offering novel ways to engage with digital environments.

This thesis focuses on leveraging advanced ocular artifact removal techniques and understanding how they can be implemented on hardware to enhance the performance and accessibility of EEG-based BCI systems. By addressing the inherent challenges in EEG signal analysis and classification, the research aims to pave the way for more responsive, efficient, and user-friendly BCI applications. In doing so, it contributes to the ongoing effort to realize the full potential of BCIs as a transformative technology for both individuals and society at large.

CHAPTER-2

LITERATURE STUDY

2.1. Overview of Conventional BCI Systems

In a conventional Brain-Computer Interface (BCI) system, the initial stages of signal acquisition and amplification occur within hardware, while subsequent preprocessing, feature extraction, and classification are conducted using software on a PC for real-time analysis or on a cluster for offline analysis. Typically, the training phase of machine learning models is carried out offline on a cluster using test data, and the pre-trained model is then loaded onto a PC for real-time classification during BCI trials. However, this setup comes with challenges related to bulk setup, wired connections and data transfer.

2.2. Neuro Rex Lower Limb Exoskeleton BCI Control System

As shown in Figure 2.1, [2] presents a real-time BCI system where a subject operates a lower limb exoskeleton, Neuro Rex, via an EEG cap with multiple hardware components and tangled wires. This setup utilizes a total of 64 EEG electrode channels, with 60 dedicated to EEG signals and 4 serving as Electrooculography (EOG) reference channels. In [2], the experiment incorporates a pre-processing pipeline with Adaptive noise cancellation H-infinity artifact removal filter [1] running on a nearby PC to process the EEG data. Figure 2.1.b provides a visual representation of the channel locations of the EEG electrodes utilized in the experiment, showcasing the intricate setup involved in real-time EEG-based BCI systems.

Neuro-Rex, similar to other BCI systems, faces challenges such as bulk setup, tangled wires, environmental white noise, ocular artifacts, signal amplitude biases, and local/global drifts. This research inspired me to select the adaptive noise cancellation H-infinity artifact removal filter algorithm and study the feasibility of implementing the algorithm on hardware. Decomposing the algorithm and building computational modules with detailed study on the precision limitations is discussed in Results and Discussions.

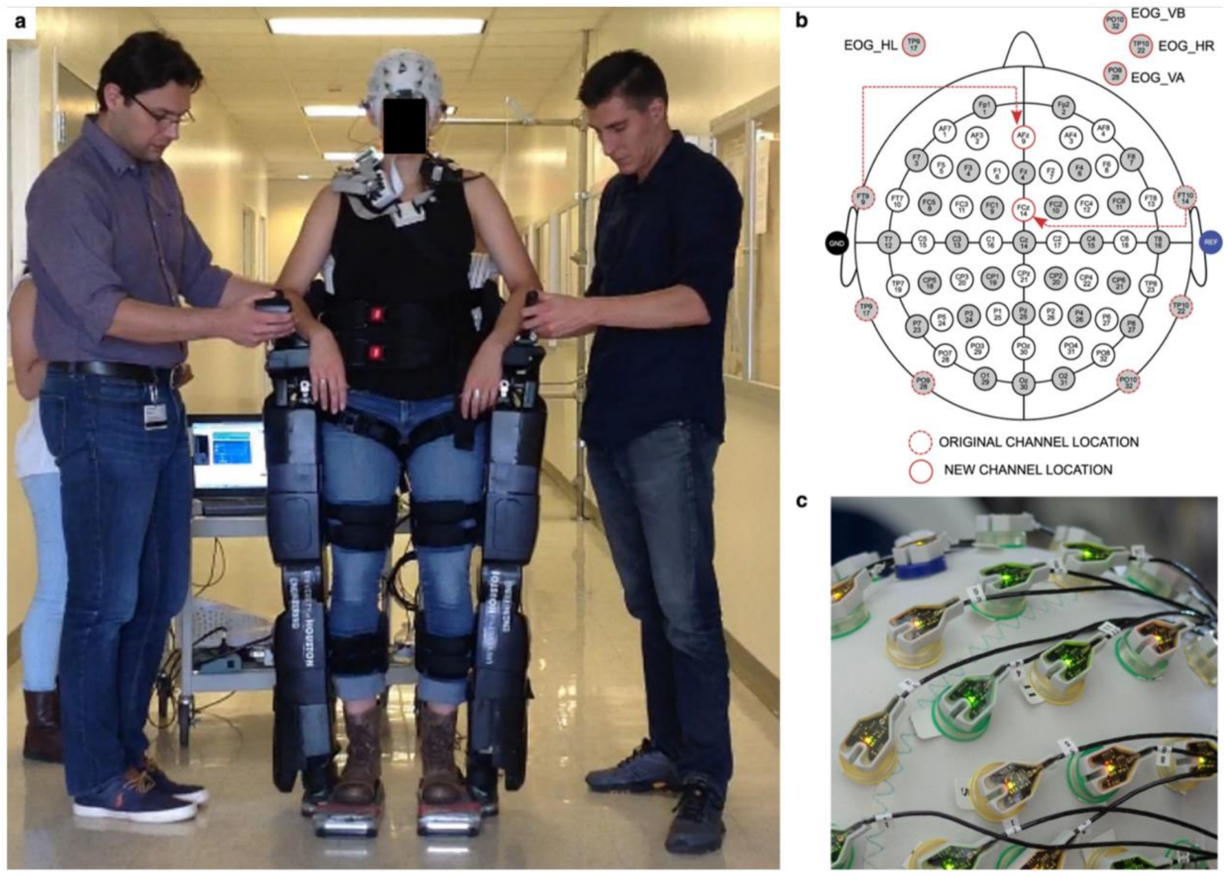

[redacted]

Fig 2.1. (a) Subject in the REX exoskeleton wearing an EEG cap. (b) EEG channel montage and EOG channel locations. (c) Active EEG electrodes. Adapted from [2].

2.3. ANC scheme with H-infinity Filter Algorithm

[1] implies that there are several offline artifact removal algorithms like Independent Component Analysis (ICA), Principal Component Analysis (PCA), Common Average Referencing (CAR), and Artifact Subspace Reconstruction (ASR) for pre-processing raw EEG data. While offline algorithms are effective, they are not suitable for real-time processing in most of the BCI applications. In contrast, online pre-processing algorithms employing adaptive noise cancellation are more appropriate for real-time EEG signal filtering in closed-loop BCI applications.

There are a few online pre-processing algorithms which uses adaptive noise cancellation approach, but they are proved to remove useful EEG signals along with the noise. EEG signals are affected by volume conduction and EOG signals that are close to EEG cap can affect each EEG electrode in a different manner. So, each EEG channel is considered as a separate subsystem and computed independently while applying the pre-processing filter. Non-stationary time varying frequency characteristics of EEG signals requires an adaptive noise cancellation filter which is robust to such vast variety of artifacts. From [1], H-infinity filter stands out among real-time pre-processing artifact removal algorithms, providing great correlation with input raw EEG data.

As depicted in Fig 2.2 from [1], the comparison of results obtained from offline Independent Component Analysis (ICA) and Artifact Subspace Reconstruction (ASR) methods highlights that certain local quantities are managed more effectively by the sample-adaptive real-time framework. The uppermost plot in the panel's raster plot represents the measured reference noise sources, including the horizontal and vertical ocular artifact components. Subsequently, the plot displays the time course of raw EEG data, offline ICA-processed data, offline ASR-processed data, and real-time H-infinity filtered EEG data, along with correlation values between the raw data and the H-infinity filtered data. The black circular marks on green line signify the correlation values, scaled to a maximum of 100%, and marked for the leading 200ms windows. The figure illustrates that contamination characteristics such as amplitude, shape, and polarity can vary significantly based on the electrode location, attributed to volume conduction effects.

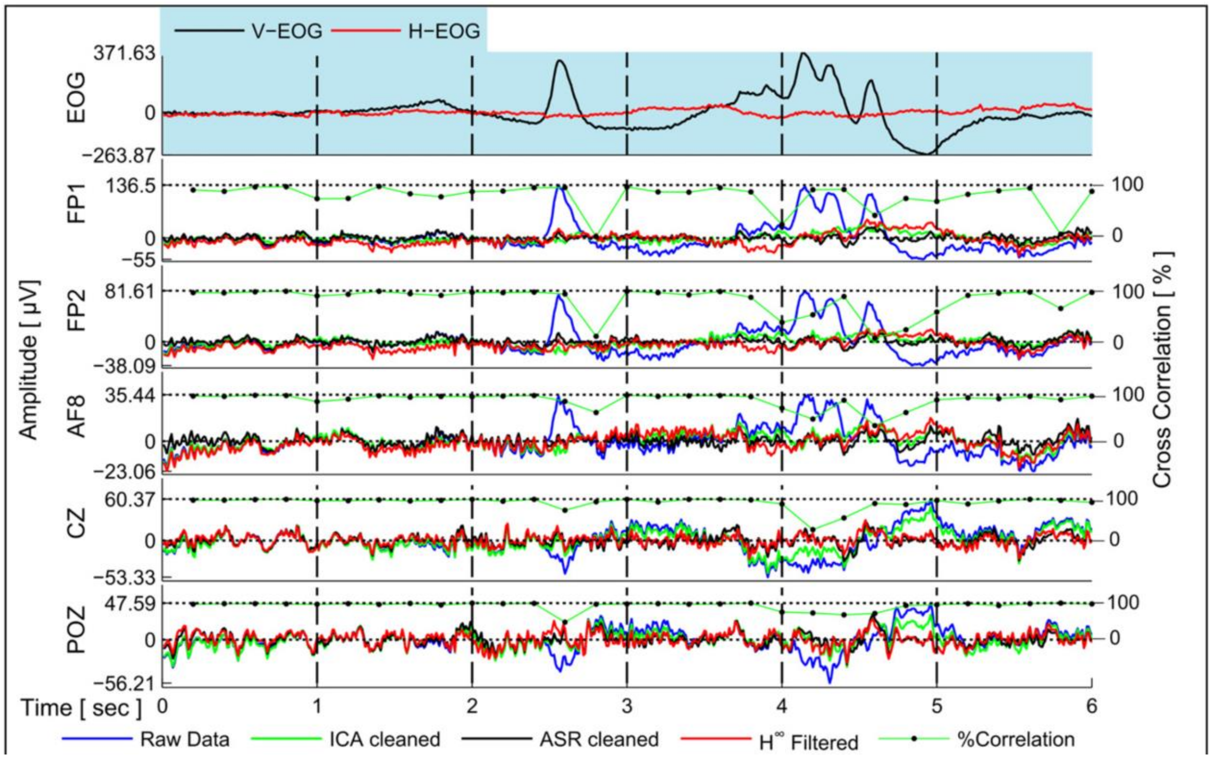


Fig 2.2. From [1], time courses of raw, offline ICA processed, offline ASR processed, and real-time H-infinity filtered EEG show the measured ocular artifacts, followed by their contamination on select electrode locations with varying amplitude and polarity (reproduced with permission).

The H-infinity filter employs a sample-by-sample adaptation scheme to update filter weights, effectively eliminating major artifacts like EOG, amplitude drifts, and recording impedance biases. Adaptive noise cancellation scheme of H-infinity filter is shown in Fig 2.3. The sample-by-sample adaptive scheme with a robust H-infinity algorithm updates weights with each input sample, learning from real-time data. This feature maintains a high correlation between raw EEG and processed EEG data, ranging from 95% to 99.9% in artifact-free regions and between 40% to 70% when ocular artifacts are prevalent, as evidenced by experimental results from [1].

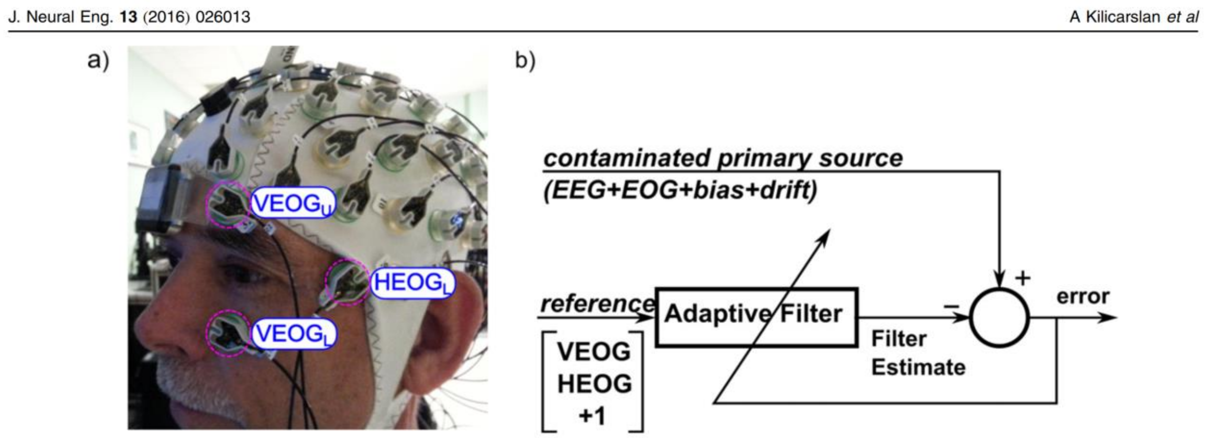


Fig 2.3. ANC scheme with H-infinity filter algorithm from [1].

From [5], the incorporation of dedicated EOG signal references, such as VEOG and HEOG (as shown in Fig 2.3), significantly enhances the removal of ocular artifacts. VEOG represents the difference between vertical upper and lower EOG electrodes, providing a reference for eye blinks, while HEOG reflects the difference between horizontal right and left EOG electrodes, offering a strong reference for horizontal eye movements. An external noise reference signal with a constant amplitude of +1 was introduced to the system, to mitigate the signal amplitude biases. Additionally, an extra weight was adjusted to minimize the overall gain per channel. This adjustment aimed to eliminate any bias and drift present in the recorded EEG signal.

2.4. Input Parameters and Mathematical Representation of the Filter

Ocular artifacts like eye blinks and eye movements stem from electric potential changes near the eyes, posing significant contamination challenges in EEG recordings. These artifacts can dominate EEG channel modulation and spread across the scalp due to volume conduction, also influenced by factors like electrode impedance shifts, cable movements, or variations in eye blink and eye motion intensities. To enhance artifact removal, as recommended by the author from [1], paired signal subtraction (VEOG and HEOG) is used to minimize low-amplitude components, yielding two strong noise reference measurements as shown in the below equations (1) and (2):

$$\mathrm{VEOG}(t) = \mathrm{VEOG}_{U}(t) - \mathrm{VEOG}_{L}(t) \tag{1}$$

and

$$\mathrm{HEOG}(t) = \mathrm{HEOG}_{R}(t) - \mathrm{HEOG}_{L}(t). \tag{2}$$

As previously noted, several adaptation methods are available for an ANC system. Utilizing the H-infinity method for sample-by-sample filter weight adaptation ensures robustness against modeling errors and unknown external effects, preventing significant estimation errors caused by minor perturbations. While assuming fixed weights per EEG channel with linear or exponential convergence is common, the dynamic nature of EEG measurements, including artifact amplitude fluctuations due to volume conduction, limits the robustness of fixed weight formulations. Nonetheless, sample-adaptive formulations can still achieve a satisfactory level of filtering. A more robust approach involves assuming time-varying weights per channel, providing greater adaptability and robustness. From [1], the H-infinity filter mathematical representation is expressed as

$$\hat{w}_{i+1} = \hat{w}_i + \frac{P_i r_i}{1+ r_i^T P_i r_i} y_i, \tag{3}$$

for which

$$y_i = s_i - r_i^T \hat{w}_i \,, \tag{4}$$

$$P_i^{-1} = \widetilde{P}_i^{-1} - \gamma^{-2} r_i r_i^T \,, \tag{5}$$

where

$$\widetilde{P}_{i+1} = \left[\widetilde{P}_i^{-1} - (1 - \gamma^{-2}) r_i r_i^T\right]^{-1} + qI. \tag{6}$$

Here is a breakdown of each parameter:

- 's_i' represents the measured primary input signal.
- 'y_i' denotes the output signal produced by the filter.
- 'w_{i+1}' signifies the estimated weight per channel for sample i+1, which undergoes update with every input sample.
- 'r_i' stands for the vector of reference measurements.
- '$\widetilde{P}_i$' is the noise covariance matrix, defining how different components of noise are correlated with each other.
- 'γ' represents the bound on the H-infinity filter gain from input noise to output estimation error. This parameter sets the tolerance levels for disturbances that

the H-infinity filter can handle. As 'γ' approaches 1, the filter's behavior approaches optimal performance.

- 'q' reflects the rate at which the weights vary over time.

Input parameters like γ, $\widetilde{P}_0$ = μI and q play a crucial role in guiding the behavior of the filter in handling contamination levels and adapting quickly in real-time scenarios. The values of ' γ ' and 'q' are constants that are typically determined through iterative experimentation to identify the most effective values for achieving optimal filter performance. The initial parameters are determined through trial and error to identify the optimal settings that yield improved results. Following the guidance of the author [1], the reference initial parameters selected are $\gamma = 1.15$, $q_0 = 1E - 8$, and $P_0 = 5$.

In conclusion, the H-infinity algorithm, coupled with adaptive noise cancellation, presents a robust and efficient method for real-time EEG signal processing. This approach holds significant promise for enhancing the reliability, responsiveness, and overall performance of Brain-Computer Interface technology.

CHAPTER-3

PROBLEM STATEMENT

3.1. Background

The field of EEG-based Brain-Computer Interface (BCI) systems faces a critical bottleneck in its signal processing pipeline due to sequential execution in software. While hardware components effectively handle signal acquisition, amplification, and Analog-to-Digital conversion, the subsequent stages of preprocessing, denoising, feature extraction, and classification heavily rely on software implementations typically deployed on personal computers (PCs) or clusters. This sequential software execution process is a significant bottleneck in system performance, leading to delays in real-time data processing and limiting the scalability of BCI systems.

3.2. Specific Challenge

One particular aspect exacerbating this bottleneck is the preprocessing phase of EEG data analysis. EEG data is inherently multidimensional, comprising multiple channels of data that require extensive preprocessing before meaningful features can be extracted. Each channel of EEG data can be treated as an independent entity, ripe for parallel computation. Although utilizing GPU-based implementations could potentially expedite computations due to their parallel processing capabilities, the substantial infrastructure required renders this approach impractical for many real-world BCI applications. Therefore, the primary focus lies in mitigating challenges related to scalability, processing speed, and system compactness while maintaining real-time processing capabilities.

3.3. Significance of Robust Adaptive Noise Cancellation Algorithm

Ocular artifacts represent a significant challenge in EEG signal processing due to their proximity to EEG electrodes, resulting in contamination of EEG signals. The characteristics of Electrooculography (EOG) signals, including frequency and voltage, closely resemble those of EEG signals, making their removal a complex task requiring continuous real-time adaptation. An effective approach involves utilizing an adaptive filter that continuously updates its weights based on a primary signal (contaminated EEG) and a secondary signal (artifacts) to accurately separate the noise component from the signal of interest (uncontaminated EEG).

The H-infinity algorithm, renowned for its robust noise cancellation capabilities, offers a promising solution to enhance real-time processing capabilities in Brain-Computer Interfaces (BCIs). Specifically, in EEG data preprocessing, the H-infinity algorithm can effectively eliminate signal amplitude biases, local/global drifts, and ocular artifacts from raw EEG signals. This process significantly improves the quality of data for subsequent stages such as feature extraction and classification, leading to more accurate and reliable brain signal analysis. This study focuses on integrating the H-infinity algorithm into FPGA-based hardware accelerators, enabling BCI systems to achieve real-time noise reduction and signal enhancement. By leveraging FPGA technology, BCI systems can process EEG signals with high efficiency, and compactness facilitating more accurate brain signal analysis and advancing the field of neurotechnology.

3.4. Research Objective

The primary objective of this research is to investigate FPGA-based computational modules designed for adaptive noise cancellation H-infinity artifact removal algorithm, specifically focusing on processing multi-channel EEG data efficiently on hardware. The study encompasses the comprehensive design, verification, and optimization of key computational modules, including custom divide modules, the 3×3 matrix inversion module, and the H-infinity filter module. Equations (5) and (6) involve multiple instances of 3×3 matrix inversions, and within each matrix inversion, divide operations are necessary due to the chosen method of 3×3 matrix inversion implementation. By achieving these objectives, this research aims to contribute to the advancement of FPGA-based computational modules for EEG data processing, addressing key challenges and optimizing system functionality and performance in removing artifacts from EEG signal data.

CHAPTER-4

PROPOSED SOLUTION

The proposed solution involves decomposing the ocular artifact removal algorithm into FPGA implementable computational modules to check the feasibility of using fixed-point notation in representing the data and address the potential difficulties that arise due to the fixed-point precision range. This study focus on implementing the Adaptive Noise Cancellation H-infinity Algorithm on FPGAs to offer customizable and parallel processing capabilities that can be tailored to specific BCI applications. By offloading preprocessing tasks to FPGA-based hardware accelerators, the overall system performance can be significantly improved in terms of processing speed, scalability, and system compactness. Furthermore, FPGA-based solutions can operate independently of traditional computing platforms, offering a standalone and portable preprocessing solution for BCI systems.

4.1. System Architecture:

The system architecture revolves around the development of three key modules: the Custom H-infinity filter module, 3×3 Matrix Inversion module, and Custom Divide module, which are explained in detail in further sections. These modules are integrated to create a hardware implementable solution for the H-infinity filter algorithm. The architecture diagram, as depicted in Fig. 4.1., outlines the data flow starting from the Receiver First In First Out (FIFO) to the H-infinity filter module. Within this flow, the data undergoes processing through a 16-state Finite State Machine (FSM) model, utilizing the custom divide and 3×3 matrix inversion modules. The processed clean EEG data is then stored in the Transmitter FIFO.

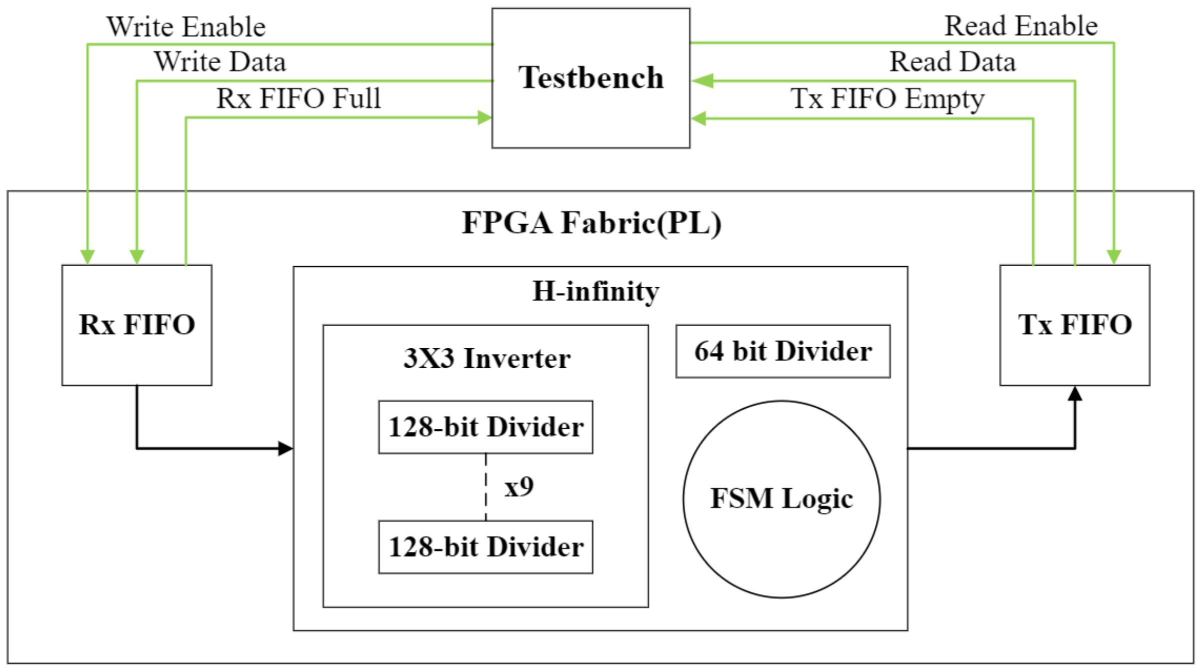


Fig 4.1. Architecture diagram for FPGA-based implementation of ANC algorithm.

The testbench refers to a simulation environment or tool used to validate and verify the functionality of a hardware design. It's a software component that simulates the behavior of the actual hardware under different conditions and inputs. The testbench allows for comprehensive testing and debugging of the hardware design before it is implemented on actual hardware, ensuring its correctness and performance.

During testing, the testbench sends data of “N” EEG channels and “M” EOG reference channels per sample of input, which are stored in the Rx FIFO. Subsequently, in “N+M” consecutive clock cycles, the data is retrieved from the Rx FIFO and saved in “N+M” internal fixed-point registers. To address the resource-intensive nature of designing a filter with a Floating-point Database on FPGA, a strategic decision was made to convert the input data to a Fixed-point data type before inputting it into the Programmable Logic (PL). This approach optimizes resource utilization and computational efficiency. The system leverages parallel computations for the “N” EEG channels after storing the inputs in internal registers. This utilization of FPGA's parallel processing architecture ensures scalability, low-power operation, standalone functionality, and cost-effectiveness. In this study, “N” is taken as 5 and “M” is taken as 3, corresponding to 5 EEG channels and 3 EOG channels.

4.2. FIFO IP Integration: Enhancing Data Flow Efficiency

4.2.1. Standard FIFO Configuration

The system utilizes 5 EEG input channels and 3 EOG reference channels, each channel featuring a 64-bit width. This configuration necessitates a total of 512 input pins. Additionally, the design stipulates eight output channels, also requiring 512 pins, alongside 6 additional pins for control signals. This extensive requirement for I/O pins renders direct implementation on FPGA platforms impractical and resource intensive. To address this challenge, implementing First-In, First-Out (FIFO) buffers is proposed. FIFO buffers can effectively manage data flow, significantly reducing the number of required I/O pins by storing input and output data temporarily. This strategy not only optimizes the use of FPGA resources but also enhances the scalability and feasibility of the algorithm's implementation on hardware platforms.

A standard FIFO configuration is employed, featuring a 64-bit data width and a depth of 512 entries. This FIFO serves as a crucial component in storing input/output data, allowing for the sequential processing of data samples, channel by channel, as they are passed to/from the H-infinity filter module within the FPGA architecture.

Input data for this example is set at the rate of 100 Hz (typically used in closed-loop BCI systems), whereas the FPGA boasts a significantly higher operational speed of 100 MHz. Consequently, the rate of data read from the Received FIFO is comparatively slower than the rate of data writing into the FIFO. To optimize resource utilization and minimize latency, an optimal FIFO depth of 512 is utilized, ensuring seamless data flow without bottlenecks.

4.2.2. IP Block Diagram

Vivado Design Suite is a comprehensive software package offered by AMD for FPGA design, synthesis, implementation, and verification. It provides a platform for designing and programming FPGA and SoC (System on Chip) devices, enabling engineers and designers to create custom hardware solutions for various applications. Within Vivado, numerous open-source Intellectual Properties (IPs) are available, among which is the FIFO IP. The FIFO IP block diagram from Vivado provides a visual representation of the input and output ports of FIFO IP. It has 32-bit Data in, Read enable, Write enable as inputs and FIFO Full, FIFO Empty, 32-bit Data out as outputs to the IP module.

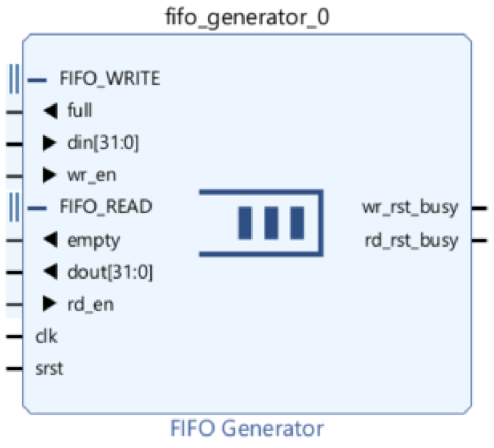


Fig 4.2. FIFO block diagram from Vivado.

4.2.3. Addressing Data Reading/Writing Challenges

To manage the potential risks associated with the disparity in data reading and writing rates within FIFO systems, it is crucial to implement robust control mechanisms. One significant risk is the possibility of reading invalid or junk data from an empty receive FIFO (Rx FIFO) or losing data while attempting to write to a full transmit FIFO (Tx FIFO). To counter these risks, the deployment of specific control signals, such as "Rx FIFO empty" and "Tx FIFO full", is essential.

The integration of the "Rx FIFO empty" signal ensures that the Read enable operation is only activated when data is available in the FIFO, thereby preventing the system from processing incorrect or junk data. Similarly, the "Tx FIFO full" signal inhibits the Write enable operation when the FIFO has reached its capacity, effectively preventing any data loss during write operations.

Employing these control signals not only enhances the reliability of the data transfer process but also optimizes the overall functionality of the FIFO system, ensuring that the data integrity is maintained under various operational scenarios. This approach is particularly beneficial in FPGA implementations where precise control over data flow is critical to the performance and efficiency of the system.

4.3. Custom Divide Module

4.3.1. Design Rationale

The development of a custom 64-bit and 128-bit divide modules was necessitated by the specific requirements outlined in the thesis. Despite the availability of standard Divide IPs, a custom design was preferred to leverage the low input data rate effectively and optimize resource utilization. With an input data flow operating at 100Hz and the Programmable Logic (PL) functioning at 100MHz, there exists ample processing time for computations to conclude before the next input. Custom modules are tailored for maximum resource optimization, aligning perfectly with the precision and performance goals of this design. In the design, nine 128-bit custom divide modules are instantiated inside 3×3 matrix inversion sub-module and one 64-bit custom divide module is instantiated directly at top level H-infinity module.

4.3.2. Implementation Strategy

The custom Divide module employs the sequential shift and subtract method, akin to long division in decimal arithmetic. This method, characterized by a finite number of iterations and straightforward arithmetic operations such as shift, subtraction, and addition, is well-suited for hardware implementation, as it does not have any multiplications involved, saving the DSP resources. Its selection is driven by the need for minimal resource utilization and optimal performance, in line with the design's requirements.

4.3.3. Module Description

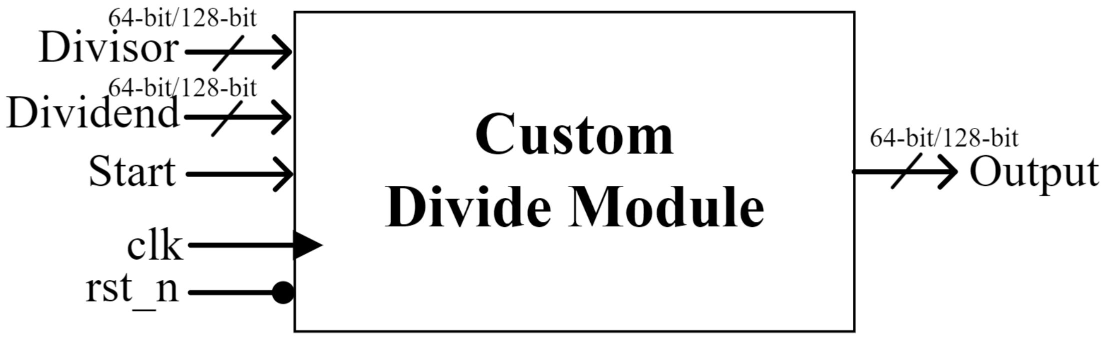


Fig 4.3. Block diagram of Custom Divide Module.

- **Inputs:**

  Divisor and **Dividend**: Signed 64-bit inputs/128-bit inputs

  Start: A control signal to initiate the division operation

  clk: Clock

  rst_n: Active low reset signal

- **Output:**

  Output: The signed 64-bit/128-bit quotient of the division.

4.3.4. Algorithm Breakdown and FSM Flow

In the FPGA-based implementation of the algorithm, a Finite State Machine (FSM) plays a pivotal role in managing data flow and operational logic. The FSM is designed to ensure efficient processing and synchronization of tasks related to data handling and algorithm execution. Below is a detailed description of the custom divide module FSM functionality and its operational flow:

- **INIT 1 State:** Initializes the division operation upon assertion of the Start signal. Determines the sign of the quotient by XORing the sign bit of the divisor and dividend.
- **INIT 2 State:** Initializes the accumulator and quotient registers, setting the stage for division.
- **CAL State:** The division operation is executed iteratively utilizing the shift and subtract method. For the 64-bit implementation, the division process completes after 108 iterations, while for the 128-bit implementation, it concludes after 196 iterations. Following this, the FSM transitions to the next phase of operation.
- **RND State:** Conducts rounding of the quotient based on the remainder and carry in the last iteration.
- **RESULT State:** Computes the final quotient based on the sign calculated in INIT 1 state. Resets the FSM to the Initialization state for subsequent operations.

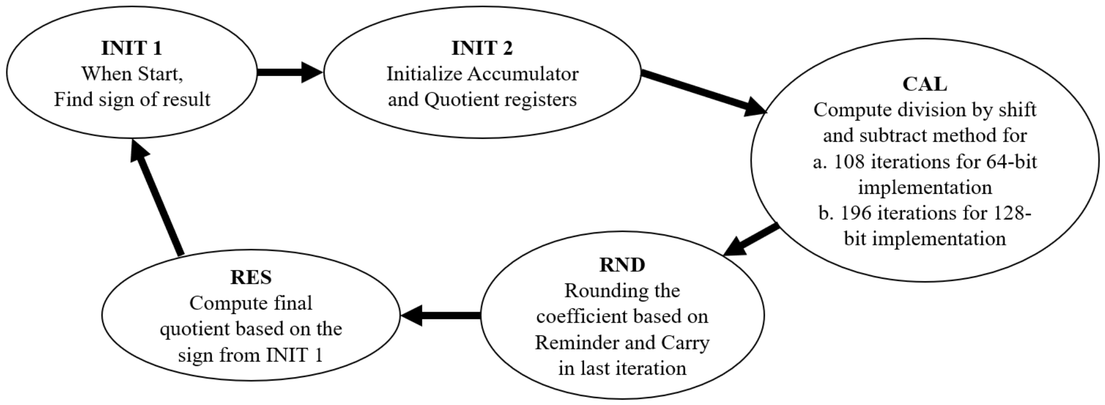


Fig 4.4. FSM flow chart for Custom Divide Module.

4.4. 3×3 Matrix Inversion Module

4.4.1. Method Selection Rationale

In selecting the method for computing the inverse of a 3×3 matrix, various sophisticated techniques like QR decomposition, Gaussian Elimination, and LU Decomposition were considered. However, for the specific scope of handling a 3×3 matrix within the FPGA architecture, the classical method of matrix inversion was deemed most suitable for several key reasons, as mentioned below:

- **Simplicity:** The classical method involves straightforward calculations such as finding the determinant, calculating the cofactor matrix, and obtaining the adjoint matrix. These calculations are relatively simple and can be efficiently implemented in hardware with minimal resource usage.
- **Fewer Operations:** Due to the smaller size of a 3×3 matrix, the number of operations required for calculating the determinant, cofactor matrix, and division

by the determinant is significantly lower compared to larger matrices, leading to improved computational efficiency.

- **No Complex Decomposition:** Unlike more complex decomposition methods, such as QR decomposition and LU decomposition, the classical method does not involve intricate decomposition and back substitution steps, resulting in reduced hardware complexity.
- **Direct Computation:** Classical methods directly compute the inverse matrix using well-defined formulas, eliminating the need for iterative or recursive algorithms commonly found in other techniques.

4.4.2. Implementation Strategy

To develop the 3×3 Matrix Inversion module, a strategic optimization methodology was employed to harmonize resource allocation with computational efficiency. Two primary implementation strategies were considered for the matrix inversion module: the first approach focuses on maximizing performance through parallel computation of each matrix element, while the second approach involves serializing portions of the computational process to minimize resource consumption.

1. **Performance Optimization:** Once the Adjoint matrix and determinant are computed, matrix inversion is performed by dividing each element of the Adjoint matrix by the non-zero determinant. In Verilog, the implementation of the division operation consumes significant resources and requires multiple clock cycles. To enhance performance, this method utilizes nine instantiated divide modules operating in parallel to compute all nine elements of the matrix

simultaneously. This approach maximizes performance but at the expense of increased resource utilization.

2. **Resource Optimization:** Instead of employing nine separate division modules, this strategy utilizes three divide modules. The three modules are serialized in their operation, effectively utilizing the same resources across different computation cycles. This method triples the computation time but reduces resource consumption by half compared to the performance-optimized implementation.

These implementation strategies underscore a deliberate trade-off between maximizing computational speed and minimizing resource usage, tailored to meet different operational priorities and constraints. The decision-making process regarding the selection of the appropriate implementation method for the 3×3 Matrix Inversion module is comprehensively discussed in the Results section.

4.4.3. Module Description

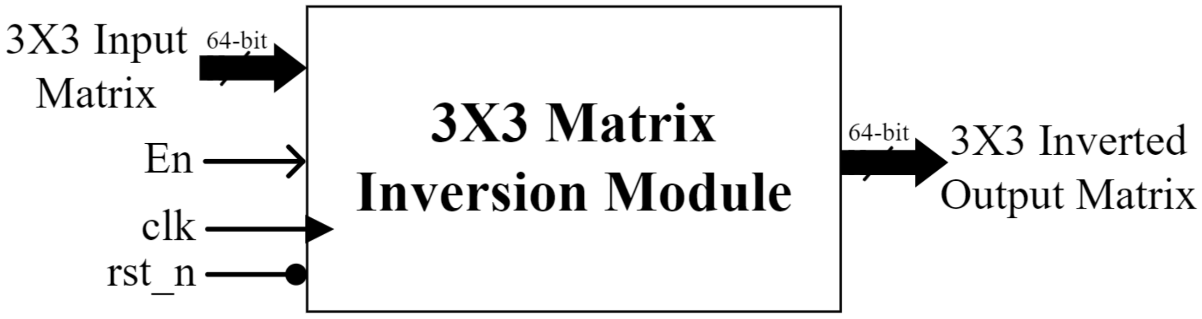


Fig 4.5. Block diagram of 3×3 Matrix Inversion Module.

- **Inputs:**

  3×3 Input Matrix: Signed 64-bit inputs

En: Enable signal to start matrix inversion computation

clk: Clock

rst_n: Active low reset signal

- **Outputs:**

  3×3 Inverted Output Matrix: signed 64-bit outputs after inversion

4.4.4. Algorithm Breakdown and FSM Flow

The implementation of the 3×3 Matrix Inversion module is governed by two distinct Finite State Machine (FSM) flows, designed to optimize either performance or resource utilization. Each approach is structured to meet specific system requirements and constraints.

Performance Optimized FSM Flow

The performance optimization strategy utilizes an FSM designed to maximize computational speed by parallel processing:

- **Initialization (INIT) State:** Upon receipt of the Enable signal, this state initializes the computation. The adjoint matrix is calculated, with each element stored as a 128-bit wide variable to ensure precision and manage overflow. Combinatorial logic is employed within this state to address hold time violations, enhancing reliability and performance, as detailed further in the Results section.
- **Determinant Calculation (Det) State:** This state handles the calculation of the determinant, producing a 128-bit result. The determinant is fed into the division modules to facilitate the inversion calculation. The transfer of numerator values

to the division modules is managed within a combinational logic block, optimizing data flow and processing efficiency.

- **Inverse Computation (Inv) State:** The final state of the FSM activates all nine division modules simultaneously (with a non-zero determinant) to compute the inverse matrix elements. Once the computations are complete, the FSM transitions back to the INIT state, ready to process the next matrix.

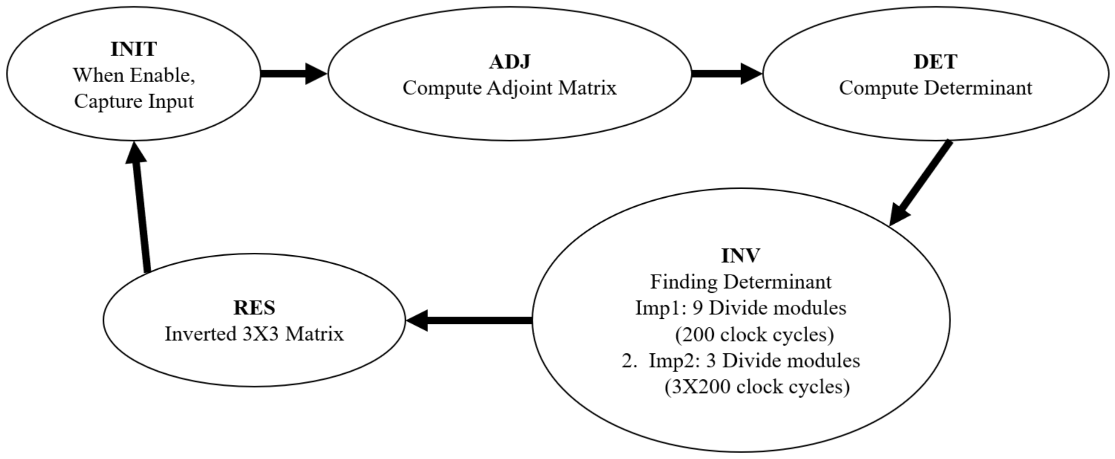


Fig 4.6. FSM flow chart for 3×3 Matrix Inversion Module.

Resource Optimized FSM Flow

The resource optimization strategy employs a more complex FSM flow, consisting of multiple states that serialize the division process to conserve hardware resources:

- **Initialization (INIT) State:** This state commences with the activation of the Enable signal and immediately transitions to the Adjoint (Adj) State.
- **Adjoint (Adj) State:** Here, the adjoint of the matrix is computed with each element initially handled as a 128-bit variable. This stage ensures adequate handling of data during the computation to prevent overflow.

- **Determinant Calculation (Det 1 and Det 2 States):** In Det 1 state, elements of the adjoint matrix are converted to 64-bit, followed by the calculation of the 128-bit determinant in Det 2 state.
- **Inverse Computation (Inv 1, Inv 2, and Inv 3 States):** This segment of the FSM divides each element of the adjoint matrix by the determinant. The division process is serialized across these three states, each handling a portion of the matrix elements. This serialization enables the use of fewer division modules, effectively managing resource allocation and synchronization of clock cycles.
- **Result Capture (Res) State:** This final state captures the inverted matrix outputs after the completion of the necessary clock cycles, resetting the FSM to handle subsequent operations efficiently.

4.5. H-Infinity Filter Module

4.5.1. Design Rationale

The H-infinity filter module serves as the central component for adaptive noise cancellation, dynamically adjusting weight parameters based on input sample data. It operates by computing output predictions using existing weights, based on input variations, and actively canceling noise for each input sample.

Noise Covariance Matrix ($\widetilde{P}_i$) is initialized to a constant Identity matrix, $\widetilde{P}_0$ = μI, to facilitate noise estimation and cancellation. According to [1], as the sample frequency of input data is 100Hz, μ is taken as 0.1. Weight Matrix ($\hat{w}_{i+1}$) is the estimated weight per channel per sample. The weight matrix, sized 3X5, is initialized to zeros and updated iteratively to optimize noise cancellation.

4.5.2. Implementation Strategy

The design strategy balances performance with resource optimization by strategically serializing elements of the design flow wherever beneficial. In this framework, division operations are serialized to achieve an optimal balance between computational complexity and hardware resource efficiency. Similarly, the 3×3 Matrix inversion module is instantiated only once and reused multiple times, which significantly conserves hardware resources.

For input handling, the system employs signed 64-bit fixed-point data, characterized by 1 sign bit, 28 integer bits, and 35 fractional bits, instead of the standard fixed-point format. This approach allows all intermediate results to be computed with this expanded integer bit width, thereby minimizing precision errors when processing a large volume of samples. This structured approach to the design of the H-infinity filter module critically supports the performance, resource efficiency, and precision of the EEG signal processing system.

The management of the Finite State Machine (FSM) within this module requires precise bit manipulations and rigorous clock cycle management to meet stringent timing requirements and ensure computational accuracy. Given the complexity associated with the hundreds of multiplications involved, careful attention is needed to manage bit-expansion. This includes bit manipulations in multiple stages of FSM, which are strictly planned to minimize precision loss.

4.5.3. Module Description

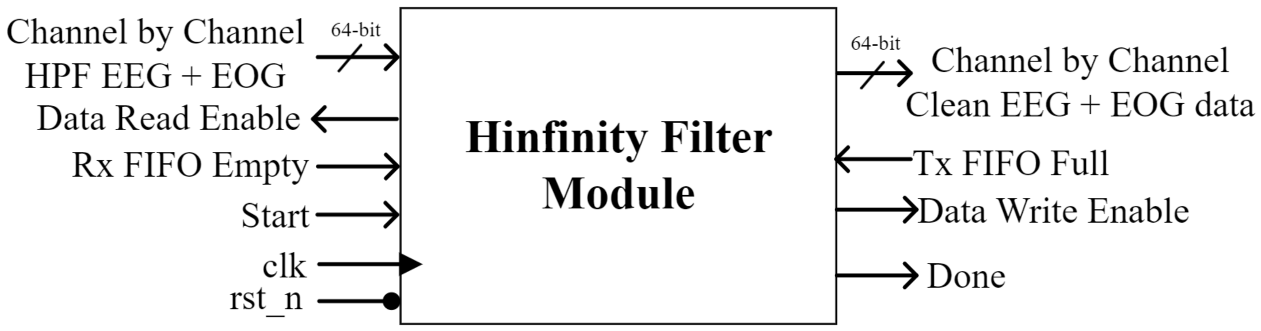


Fig 4.7. Block diagram of H-infinity Filter Module.

- **Inputs:**

  HPF EEG + EOG data: Channel by channel 64-bit signed High Pass Filter output

  Start: Start signal to trigger the processing of input data

  Rx FIFO Empty: A status signal that indicates Receiver FIFO is empty

  Tx FIFO Full: A status signal that indicates Transmitter FIFO is full

  clk: Clock

  rst_n: Active-low reset signal

- **Outputs:**

  Clean EEG + EOG data: Channel by channel 64-bit signed output

  Data Read Enable: Allows Reading the input data from Receiver FIFO

  Data Write Enable: Allows writing the processed data to Transmitter FIFO

  Done: A status signal that indicates computation is done

4.5.4. Algorithm Breakdown and FSM Flow

This module takes input data stream from the Rx FIFO and runs through a lot of computations passing through several FSM states. The FSM in this design comprises 16 distinct states, each tailored to handle specific aspects of the computational sequence efficiently and systematically. The flow of the FSM is structured as follows:

- **INIT State:** Upon receiving the start signal, it loads channel by channel input sample data to 8 internal registers in consecutive 8 clock cycles and goes to Step 1.
- **Step 1:** Computes the EOG matrix $\boldsymbol{r_i}$ using the 3 EOG signals, leveraging specialized formulas based on [1]. The calculation of the $\boldsymbol{r_i}$ matrix varies based on the number of EOG channels. In this case, we have 3 EOG channels, so VEOG and HEOG of Ri matrix is computed using equations 1 and 2.
- **Step 2:** Computes the product Rit = $r_i * r_i^T$, Yti = $r_i^T * w_i$, and starts computing $\widetilde{\boldsymbol{P}}_{\boldsymbol{i}}$ inverse. As it takes 120 clock cycles to calculate the inverse of a 3×3 Matrix.
- **Step 3:** After waiting for 120 clock cycles, $\widetilde{\boldsymbol{P}}_{\boldsymbol{i}}$ inverse is saved in temporary registers for using in subsequent FSM stages. Clean EEG output is calculated using equation 4, converting 128-bit Yt to 64-bit using bit manipulation techniques. Gain constant gt1 = γ^{-2} is calculated, and the sub-product Ritg = $\gamma^{-2} * Rit$ is computed, resulting in a 96-bit variable Ritg.

- **Step 4:** Initializes the second inverse of a sub-function, as shown in equation 5, to find the $\boldsymbol{P_i}$ matrix after converting 96-bit Ritg to 64-bit, which is further consumed to update the $\boldsymbol{w_i}$ matrix.
- **Step 5:** After waiting for 120 clock cycles, the inverse output $\boldsymbol{P_i}$ is multiplied with the $\boldsymbol{r_i}$ matrix and saved in a 96-bit temporary variable PR.
- **Step 6:** Calculates the product of $\boldsymbol{r_i^T}$ and PR incremented by a unit constant, saved in a 128-bit variable PRt as shown in equation 3.
- **Step 7-10:** To update $\boldsymbol{w_i}$ matrix as shown in equation 3, PR/PRt is computed in subsequent 4 states, as each division takes few clock cycles. Output of the division is saved in 64-bit temporary registers Qi.
- **Step 11:** Product PRYij is computed by multiplying Qi with output Yi computed in step 3 and saved in 64-bit temporary register PRYij.
- **Step 12:** Each element of PRYij is added to previous $\boldsymbol{w_i}$ values to get the latest weight matrix values according to equation 3. Also, Ritg2 = $(1-\gamma^{-2})r_i r_i^T$, is calculated before going to Step 14.
- **Step 13:** In this state, $\boldsymbol{\widetilde{P}_i^{-1}}$ from Step 3 is added to the Ritg2 register values and passed through 3×3 Matrix inversion module.
- **Step 14:** After 120 clock cycles, the outputs of the inverse function module are added to a constant identity matrix (qI), and new noise covariance parameters are calculated as per equation 6.
- **Step 15:** Transforms the 5 clean EEG output channel data from 64-bit to 32-bit and saves it in the Transmitter FIFO along with the 3 EOG channels, in subsequent 8 clock cycles.

Upon completing the computation cycle, the FSM restarts to the INIT state, awaiting the next input sample from Rx FIFO.

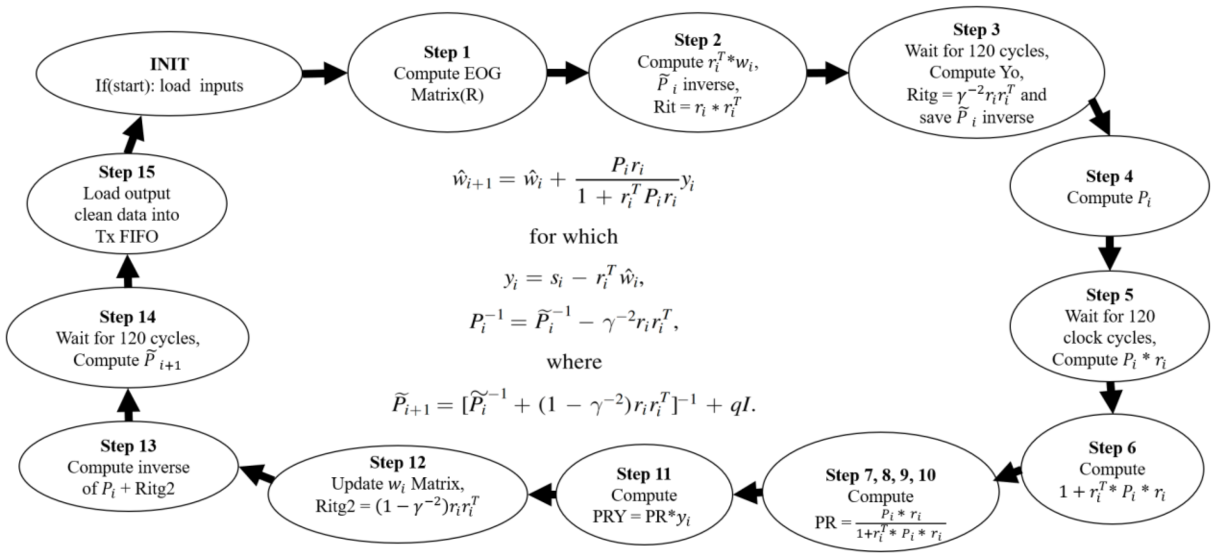

INIT
If(start): load inputs
Step 1
Compute EOG Matrix(R)
Step 2
Compute $r_i^T * w_i$, $\widetilde{P}_i$ inverse, Rit = $r_i * r_i^T$
Step 3
Wait for 120 cycles, Compute Yo, Ritg = $\gamma^{-2} r_i r_i^T$ and save $\widetilde{P}_i$ inverse
Step 4
Compute P_i
Step 5
Wait for 120 clock cycles, Compute $P_i * r_i$
Step 6
Compute $1 + r_i^T * P_i * r_i$
Step 7, 8, 9, 10
Compute PR = $\frac{P_i * r_i}{1 + r_i^T * P_i * r_i}$
Step 11
Compute PRY = PR*y_i
Step 12
Update w_i Matrix, Ritg2 = $(1-\gamma^{-2}) r_i r_i^T$
Step 13
Compute inverse of P_i + Ritg2
Step 14
Wait for 120 cycles, Compute $\widetilde{P}_{i+1}$
Step 15
Load output clean data into Tx FIFO
$\hat{w}_{i+1} = \hat{w}_i + \frac{P_i r_i}{1 + r_i^T P_i r_i} y_i$
for which
$y_i = s_i - r_i^T \hat{w}_i$,
$P_i^{-1} = \widetilde{P}_i^{-1} - \gamma^{-2} r_i r_i^T$,
where
$\widetilde{P}_{i+1} = [\widetilde{P}_i^{-1} + (1-\gamma^{-2}) r_i r_i^T]^{-1} + qI$.

Fig 4.8. FSM flow chart of H-infinity Filter Algorithm.

CHAPTER-5

RESULTS AND DISCUSSION

5.1. Custom Divide Module

5.1.1. Validation Results

The development of custom 64-bit and 128-bit divide modules was engineered to optimize resource utilization while maintaining the requisite speed for the designated design objectives. This evaluation validates the module's functionality, performance metrics, and outcomes, substantiating its efficacy and reliability.

A comprehensive testing regimen encompassed the utilization of varied input vectors to thoroughly test the module's functionality and resilience. Performance evaluation was conducted by measuring the number of clock cycles required to compute results, with a clock-period of 10 nanoseconds (ns). Across a spectrum of inputs, including both positive and negative real numbers, the computation consistently completed within 115 clock cycles for the 64-bit version and approximately 200 clock cycles for the 128-bit version. The output waveforms depicted in Figures 5.1 and 5.2, generated via the Vivado tool, delineate the input, control signal, and output of the divide module, offering a visual representation of its operational dynamics.

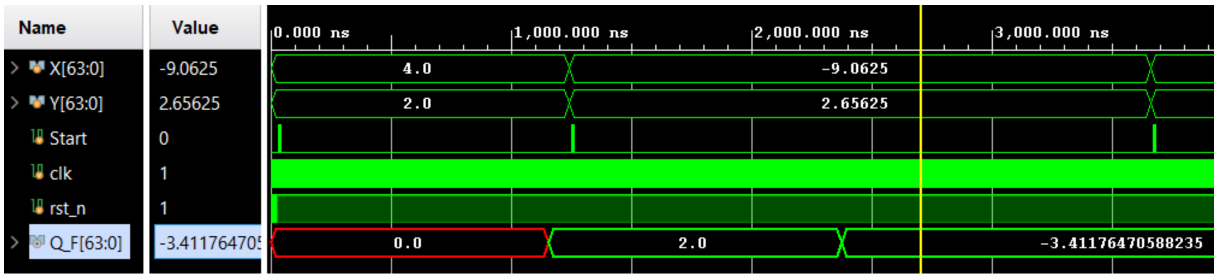


Fig 5.1. Output waveform of 64-bit Divide module.

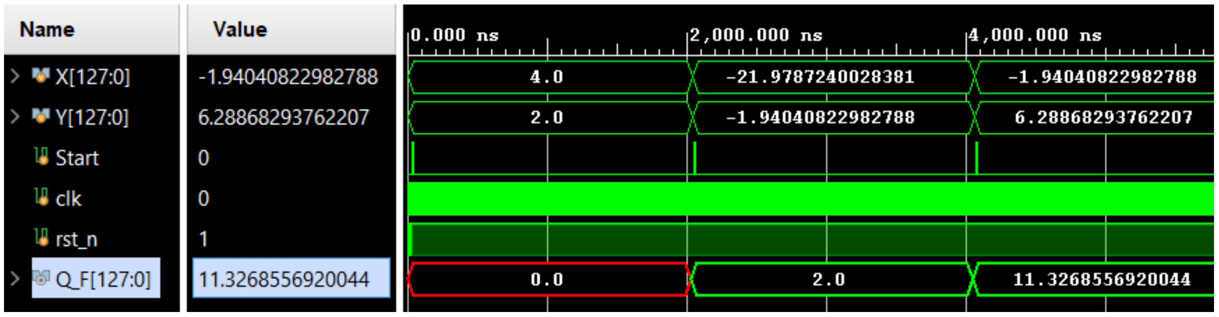


Fig 5.2. Output waveform of 128-bit Divide module.

A detailed comparison was conducted between computed outputs and expected outputs for diverse input scenarios, including extreme corner cases. The validation process, detailed in Tables 5.1 and 5.2, verified the module's correct operation and ability to yield accurate results across a different input combination, demonstrating negligible error margins.

Table 5.1. Expected vs computed outputs for 64-bit Divide module.

Inputs		**Expected Output**	**Computed Output**	**% error**
Numerator	**Denominator**			
4	2	2	2	0
-9.0625	2.65625	-3.411764706	-3.411764706	0.00000000

Table 5.2. Expected vs computed outputs for 128-bit Divide module.

Inputs		**Expected Output**	**Computed Output**	**% error**
Numerator	**Denominator**			
4	2	2	2	0
-21.978724	-1.94040823	11.32685569	11.32685569	0.00000000
-1.94040823	6.288682938	-0.308555583	-0.308555583	0.00000000

5.1.2. Synthesis, Timing and Resource Utilization Results

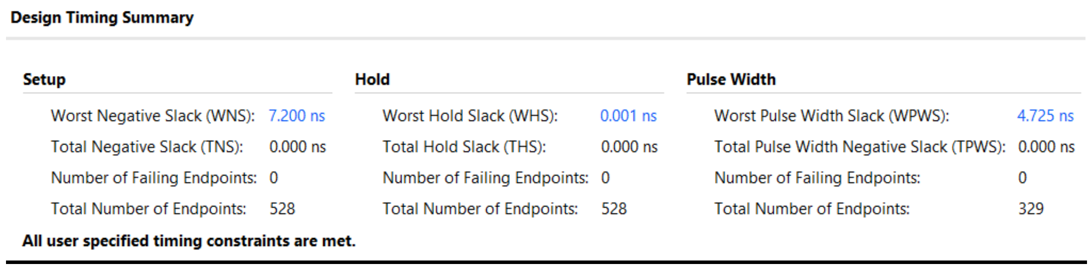
The analysis presented in Table 5.3. highlights a significant disparity in resource consumption between the default 64-bit IP solution and the custom 64-bit Divide module, with the latter consuming less than half the resources. This substantial difference emphasizes the crucial importance of optimizing performance while managing resource allocation in alignment with project specifications. However, employing the 64-bit Divide module necessitated trimming inputs to fit within 64-bit fixed-point registers, leading to data loss and erroneous results.

An in-depth examination of input ranges directed the development of a custom 128-bit Divide module, specifically engineered to mitigate data loss during bit manipulation of sub-products. Although the upgraded module incurs increased resource utilization compared to the custom 64-bit version, it remains more resource-efficient than the default 64-bit IP solution, as evidenced in Table 5.3. The design and implementation of the custom Divide module sought to strike an optimal balance between performance efficiency and resource conservation, culminating in a tailored solution tailored to the project's unique requirements.

Table 5.3. Resources comparison between default and Custom Divide Modules.

Resource	**Available**	**Default 64-bit IP**	**64-bit Divide Module**	**128-bit Divide Module**
		Used	**Used**	**Used**
LUT	117120	1244	542	1041
FF	234240	2565	328	650
BRAM	144	0.5	0	0
DSPs	1248	21	0	0

A comprehensive timing analysis was conducted to identify and rectify potential setup or hold time violations. Figure 5.3. illustrates the results of this analysis, showcasing the module's adherence to timing constraints without encountering violations. This robust adherence ensures dependable and consistent performance across operational scenarios.


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.200 ns	Worst Hold Slack (WHS):	0.001 ns	Worst Pulse Width Slack (WPWS):	4.725 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:	528	Total Number of Endpoints:	528	Total Number of Endpoints:	329

All user specified timing constraints are met.

Fig 5.3. Post-synthesis timing summary for Custom Divide Module.

5.2. 3×3 Matrix Inversion Module

5.2.1. Validation Results

In the development of the Matrix Inversion module, two distinct approaches were employed to balance performance and resource utilization, each with its trade-offs and benefits. This section presents a comparative analysis of these methodologies and their integration into the H-infinity filter module.

As discussed previously, the performance-optimized implementation computes the inverse of all 9 elements of the matrix in parallel using 9 distinct Divide blocks, while the resource-efficient implementation utilizes only 3 divide blocks, serializing the data flow at the cost of performance for resource optimization. Both design flows were validated, and their results were thoroughly verified. Table 5.4. provides a detailed comparison of the expected and computed results from both methodologies, demonstrating an average percentage error of 2.86165E-07. This analysis underscores the accuracy and reliability of both approaches.

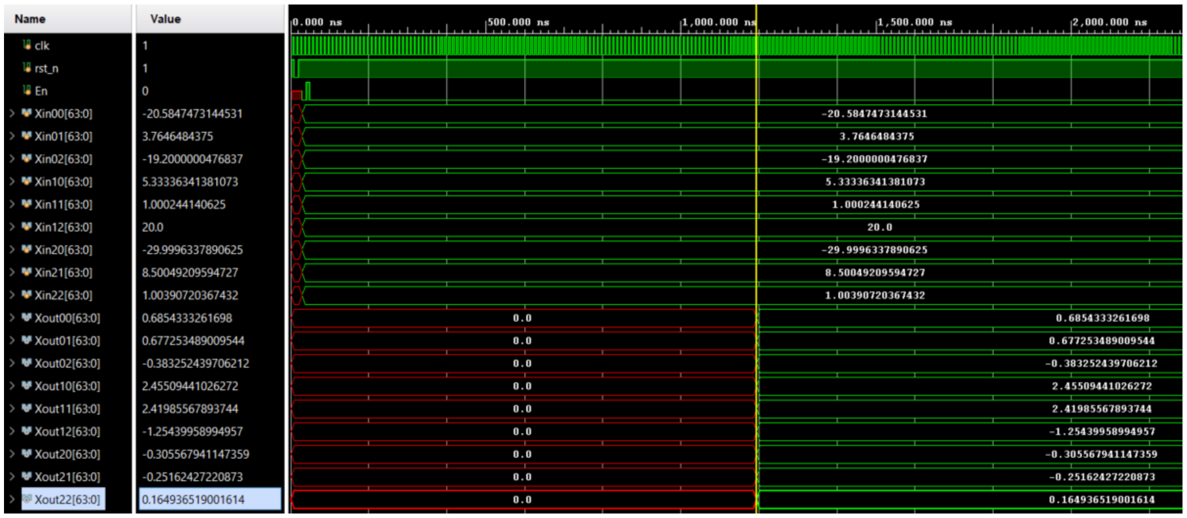
Table 5.4. Expected vs computed results comparison for 3×3 Matrix Inversion Module.

Coordinate	Inputs	Expected Output	Computed Output	% error
(0,0)	432.34000	0.00039	0.00038513721665	4.28776E-06
(0,1)	-34.34000	-0.00077	-0.00077475325088	-1.5588E-08
(0,2)	-343.54000	0.01515	0.01514874110580	1.90706E-07
(1,0)	8.40000	-0.00405	-0.00405148285790	-2.5778E-08
(1,1)	95.75000	0.00715	0.00715252873488	1.78273E-07
(1,2)	-190.33000	0.03051	0.03050562410499	1.37565E-08
(2,0)	55.45000	-0.00202	-0.00202119653113	-1.2958E-06
(2,1)	5.77000	-0.00169	-0.00168997689616	-9.0939E-07
(2,2)	-1.00000	0.01602	0.01601514700451	1.51493E-07
			Average Error	2.86165E-07

Opting for the performance-optimized version of the 3×3 matrix inversion module, considering its reusability and seamless integration within the H-infinity filter module, was a strategic decision. This choice ensures efficient utilization of resources while maintaining a high level of performance, crucial for the overall functionality and effectiveness of the H-infinity filter module. By leveraging the performance-optimized approach, where the inverse of all 9 elements of the matrix is computed in parallel using multiple Divide blocks, the matrix inversion module can efficiently handle computations within the H-infinity filter module. This optimized design allows for faster processing and reduced latency, essential for real-time applications and complex algorithms such as those employed in the H-infinity filter module.

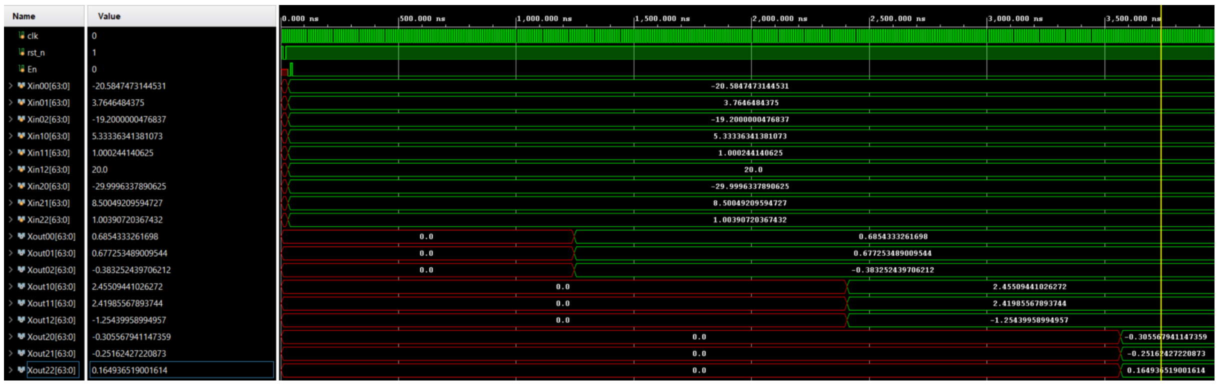
Moreover, the reusability aspect is significant as the 3×3 matrix inversion module is instantiated once but utilized multiple times within the H-infinity filter module. This streamlined approach not only conserves resources but also promotes consistency and reliability across the entire system.

Figure 5.4. depicts the output waveform for the performance-optimized implementation with a 64-bit Divide module, showcasing the parallel computation of the inverse of all 9 elements. In contrast, Figure 5.5. illustrates the output waveform for the resource-optimized implementation with a 64-bit Divide module, demonstrating serialized data flow where each row of elements is computed in sequence. This results in outputs appearing in 3 steps for every 115 clock cycles.



Name	Value
clk	1
rst_n	1
En	0
Xin00[63:0]	-20.5847473144531
Xin01[63:0]	3.7646484375
Xin02[63:0]	-19.2000000476837
Xin10[63:0]	5.33336341381073
Xin11[63:0]	1.000244140625
Xin12[63:0]	20.0
Xin20[63:0]	-29.9996337890625
Xin21[63:0]	8.50049209594727
Xin22[63:0]	1.00390720367432
Xout00[63:0]	0.6854333261698
Xout01[63:0]	0.677253489009544
Xout02[63:0]	-0.383252439706212
Xout10[63:0]	2.45509441026272
Xout11[63:0]	2.41985567893744
Xout12[63:0]	-1.25439958994957
Xout20[63:0]	-0.305567941147359
Xout21[63:0]	-0.25162427220873
Xout22[63:0]	0.164936519001614

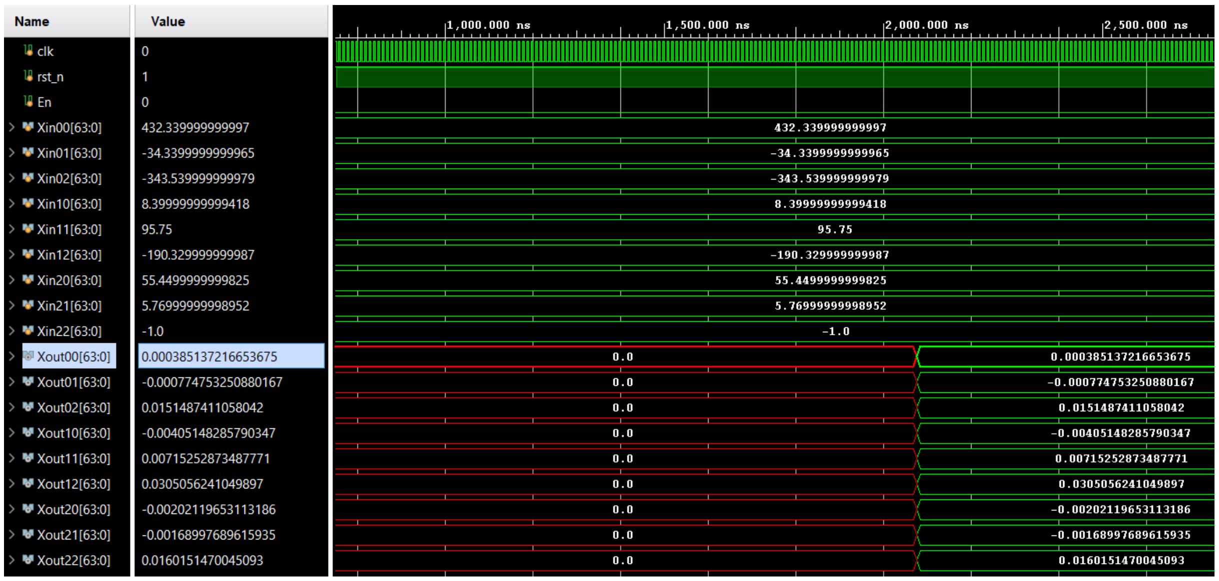
Fig 5.4. Output waveform for performance optimized implementation with 64-bit Divide module.



Name	Value
clk	0
rst_n	1
En	0
Xin00[63:0]	-20.5847473144531
Xin01[63:0]	3.7646484375
Xin02[63:0]	-19.2000000476837
Xin10[63:0]	5.33336341381073
Xin11[63:0]	1.000244140625
Xin12[63:0]	20.0
Xin20[63:0]	-29.9996337890625
Xin21[63:0]	8.50049209594727
Xin22[63:0]	1.00390720367432
Xout00[63:0]	0.6854333261698
Xout01[63:0]	0.677253489009544
Xout02[63:0]	-0.383252439706212
Xout10[63:0]	2.45509441026272
Xout11[63:0]	2.41985567893744
Xout12[63:0]	-1.25439958994957
Xout20[63:0]	-0.305567941147359
Xout21[63:0]	-0.25162427220873
Xout22[63:0]	0.164936519001614

Fig 5.5. Output waveform for resource optimized implementation.

To address precision errors arising from large sub-products during the computation of Adjoint and Determinant of the Matrix, the performance-optimized matrix inversion module was upgraded with a 128-bit divide module. This enhancement minimizes precision errors and is elaborated upon in the functional verification section of the H-infinity filter module. Figure 5.6. showcases the output waveform for the performance-optimized implementation with the 128-bit Divide module, illustrating the parallel computation of the inverse of all 9 elements. This configuration ensures that all 9 outputs are observed simultaneously after 200 clock cycles, maintaining performance while enhancing precision and accuracy.



Name	Value
clk	0
rst_n	1
En	0
Xin00[63:0]	432.339999999997
Xin01[63:0]	-34.3399999999965
Xin02[63:0]	-343.539999999979
Xin10[63:0]	8.39999999999418
Xin11[63:0]	95.75
Xin12[63:0]	-190.329999999987
Xin20[63:0]	55.4499999999825
Xin21[63:0]	5.76999999998952
Xin22[63:0]	-1.0
Xout00[63:0]	0.000385137216653675
Xout01[63:0]	-0.000774753250880167
Xout02[63:0]	0.0151487411058042
Xout10[63:0]	-0.00405148285790347
Xout11[63:0]	0.00715252873487771
Xout12[63:0]	0.0305056241049897
Xout20[63:0]	-0.00202119653113186
Xout21[63:0]	-0.00168997689615935
Xout22[63:0]	0.0160151470045093

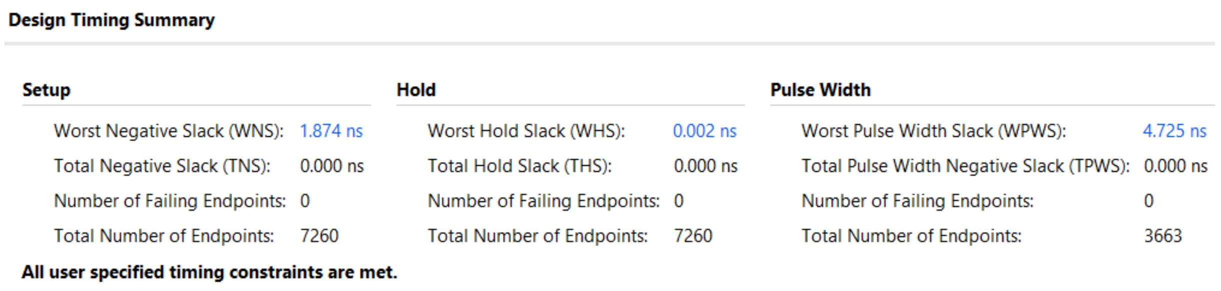
Fig 5.6. Output waveform for performance optimized implementation with 128-bit Divide module.

5.2.2. Synthesis, Timing and Resource utilization Results

Detailed timing analysis is done, as depicted in Fig 5.7., ensuring no setup or hold time violations. The comparison between the number of resources utilized for both the methodologies, along with integrated 128-bit custom divide module is shown in Table 5.5. Based on the improved performance over the percentage of extra resources utilized, I decided to integrate performance optimized design in H-infinity filter module.

Table 5.5. Resource utilization summary for 3×3 Matrix Inversion Module.

Resource	Available	Impl 1(9-Div)	Impl 2(3-Div)	Impl 3(128-bit Div)
		Used	Used	Used
LUT	117120	9649	6304	14032
FF	234240	3662	3112	6566
BRAM	144	0	0	0
DSPs	1248	336	336	336


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.874 ns	Worst Hold Slack (WHS):	0.002 ns	Worst Pulse Width Slack (WPWS):	4.725 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:	7260	Total Number of Endpoints:	7260	Total Number of Endpoints:	3663

All user specified timing constraints are met.

Fig 5.7. Post-synthesis timing summary for 3×3 Matrix Inversion Module.

5.3. H-infinity Filter Module

The H-infinity Filter module plays a crucial role within the system architecture, incorporating a 16-state Finite State Machine (FSM) implementation alongside complex matrix inversions, numerous multiplications, and divisions. Its primary function is to filter noise from various artifacts in the Raw EEG dataset. This section outlines the verification process, challenges faced, and corrective measures taken during functional verification. The testbench code was intricately designed to facilitate the transmission of High Pass filtered data from MATLAB into the H-infinity Filter module for direct comparison with MATLAB-generated outputs. To validate the functionality, High pass filtered EEG data from MATLAB was converted from floating point to signed 64-bit fixed point data and then fed into the H-infinity module.

The development of the testbench setup involved a series of precise data conversions, transitioning between floating point and fixed-point representations. Following the recommended approach by the Author [1], the raw EEG data underwent high pass filtration to eliminate noise spectra below 0.5Hz, using a 4th order Butterworth IIR high pass filter in MATLAB. The resulting filtered data was then converted to a 64-bit signed fixed-point notation with 1 sign bit, 28 integer bits, and 35 fractional bits through a customized Python script. The processed data was directed into the H-infinity filter module, stored in a 64-bit internal registers.

Custom 64-bit fixed-point implementation has resulted in achieving a Pearson correlation index of 0.9993 for all 5 EEG channels when compared with MATLAB-computed results across the entire dataset of 17187 samples, with a resource utilization of less than 33% of the available LUT resources post implementation. This signifies a high level of accuracy and alignment between the hardware-implemented H-infinity filter module and the MATLAB simulation results. A thorough examination of value ranges within internal registers prompted the standardization of all internal registers to a 64-bit fixed-point format, comprising 1 sign bit, 28 integer bits, and 35 fractional bits.

The subsequent sections delve into the functional validation results, insights gained, synthesis reports, timing and resource utilization analyses, implementation results, and the challenges encountered throughout the process, offering a comprehensive understanding of the H-infinity Filter module's functionality and performance.

5.3.1. Validation Results

Implementing H-infinity filter algorithm with a complex FSM design necessitates a deep understanding of the range of internal register values to avoid overflow or data loss. Several challenges were encountered during the design process, highlighting the importance of adopting various debugging techniques. These techniques were pivotal in identifying and resolving issues, ultimately leading to the successful implementation of a complex adaptive noise cancellation H-infinity filter on an FPGA platform.

The subsequent subsections provide a comprehensive examination of precision challenges, instances of data loss, and the constraints associated with fixed-point ranges. This detailed analysis shows the intricacies of the design process and underscores the critical considerations essential for ensuring accurate and robust functionality. Furthermore, effective debugging strategies are outlined to address these limitations and optimize the performance of the H-infinity filter module. These strategies are tailored to enhance the reliability and efficiency of the system, ensuring that it operates within specified parameters and delivers consistent results across various scenarios.

5.3.1.1. Loss of data during bit manipulation

The H-infinity filter module, renowned for its complex matrix inversions and multiplications, poses significant challenges when processing a large volume of samples. A notable issue stems from the intermediate products directed into the inversion module, leading to substantial sub-products within the 3×3 Matrix inversion computation and resulting in data loss problems. Tables 5.6. and 5.7. showcase Pt inverse and Pi inverse intermediate results for the initial few samples, with a range of 4 decimal integral digits. However, as the sample count increases, these sub-products grow exponentially, reaching up to 8 decimal integer digits. To manage such extensive data processing, internal registers necessitate a minimum of 25 integer bits, while sub-products within the matrix inversion module demand at least 50 integer bits for accurate processing.

Table 5.6. Expected vs computed results for Pt inverse.

P_t^{-1}	Sample 1		Sample 2		Sample 3	
	Expected	Computed	Expected	Computed	Expected	Computed
(0,0)	0.200000	0.200000	2391.198744	2391.198763	4792.296220	4792.296281
(0,1)	0.000000	0.000000	1569.897624	1569.897621	3052.695881	3052.695890
(0,2)	0.000000	0.000000	24.146635	24.146635	48.344198	48.344198
(1,0)	0.000000	0.000000	1569.897624	1569.897621	3052.695881	3052.695890
(1,1)	0.200000	0.200000	1030.973670	1030.973658	1946.676048	1946.676034
(1,2)	0.000000	0.000000	15.854356	15.854356	30.797566	30.797566
(2,0)	0.000000	0.000000	24.146635	24.146635	48.344198	48.344198
(2,1)	0.000000	0.000000	15.854356	15.854356	30.797566	30.797566
(2,2)	0.200000	0.200000	0.443856	0.443856	0.687712	0.687712
Avg Error	0		1.9907E-07		7.38851E-06	

Table 5.7. Expected vs computed results for Pi inverse.

P_i^{-1}	Sample 1		Sample 2		Sample 3	
	Expected	Computed	Expected	Computed	Expected	Computed
(0,0)	-7413.75213	-7413.75214	-5054.07476	-5054.07474	-2228.88276	-2228.88272
(0,1)	-4867.90128	-4867.90124	-3027.93281	-3027.93276	-1657.40264	-1657.40259
(0,2)	-74.87331	-74.87331	-50.88467	-50.88467	-24.51882	-24.51882
(1,0)	-4867.90128	-4867.90124	-3027.93281	-3027.93276	-1657.40264	-1657.40259
(1,1)	-3195.99852	-3195.99846	-1808.41761	-1808.41757	-1213.05369	-1213.05364
(1,2)	-49.16081	-49.16081	-30.48124	-30.48124	-18.08198	-18.08198
(2,0)	-74.87331	-74.87331	-50.88467	-50.88467	-24.51882	-24.51882
(2,1)	-49.16081	-49.16081	-30.48124	-30.48124	-18.08198	-18.08198
(2,2)	-0.55614	-0.55614	-0.31229	-0.31229	-0.06843	-0.06843
Avg Error	1.43753E-05		1.68556E-05		2.18333E-05	

Initially, employing a 64-bit fixed-point notation with 1 sign bit, 24 integer bits, and 39 fractional bits yielded precise results for the first few hundred samples. However, the error escalated with increasing samples, resulting in invalid data. Figure 5.8. visually illustrates the cumulative average error per sample for intermediate sub-products Pi inverse and Pt inverse. The trend indicates a progressively increasing error, notably prominent after processing several hundred samples.

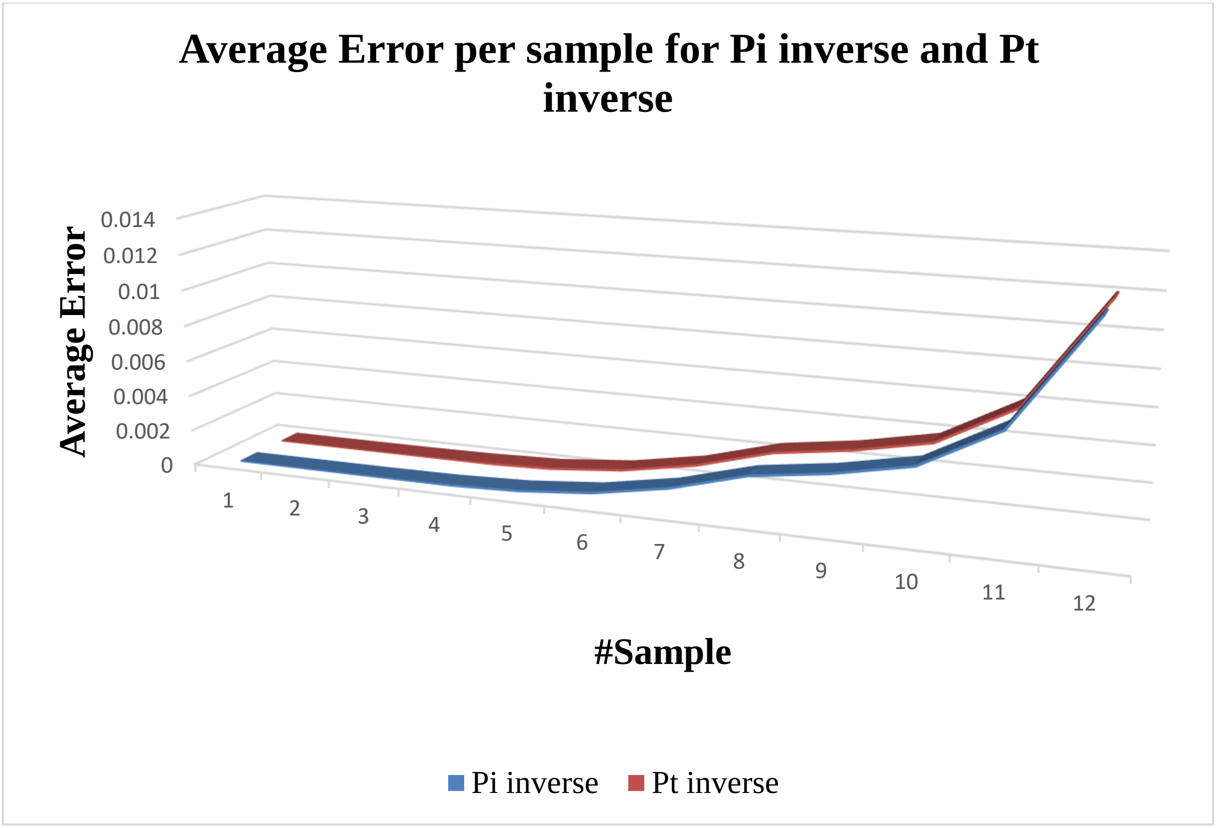
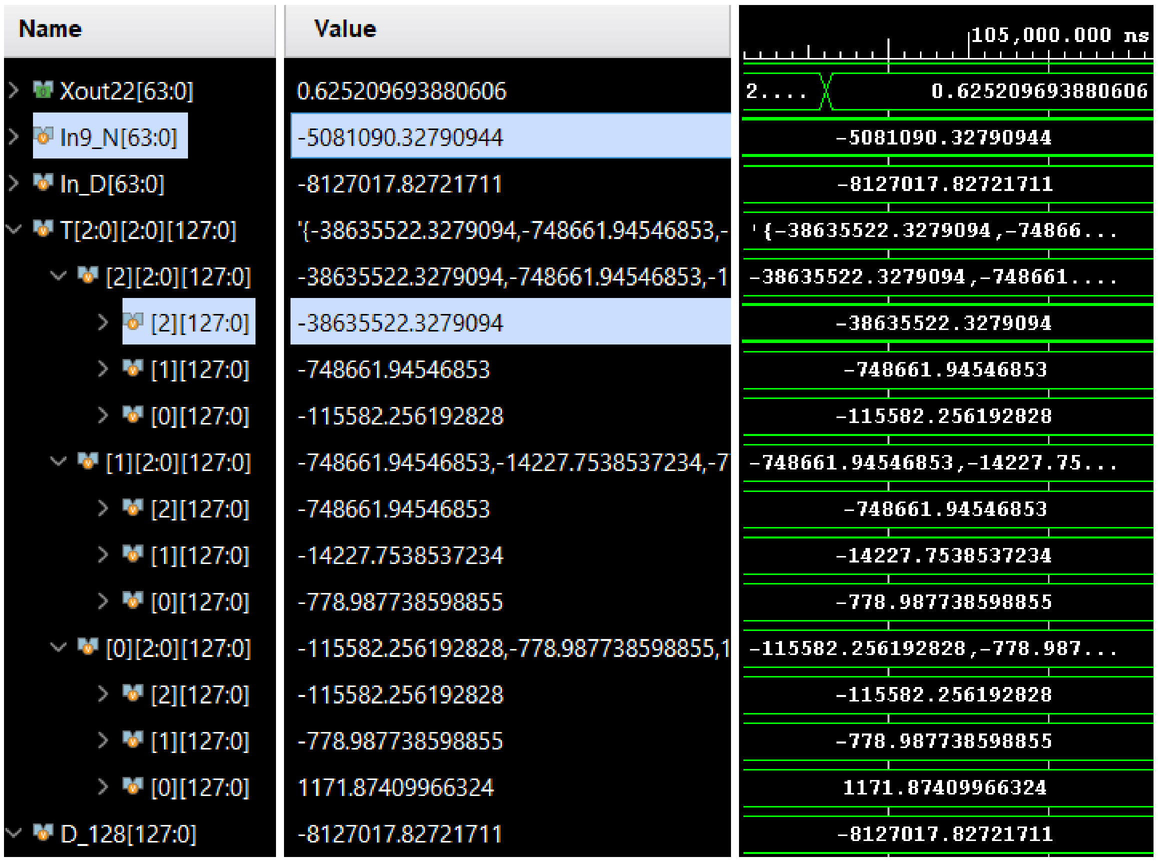


Fig 5.8. Accumulating precision error curve of intermediate sub-products for 64-bit (1,24,39) implementation.

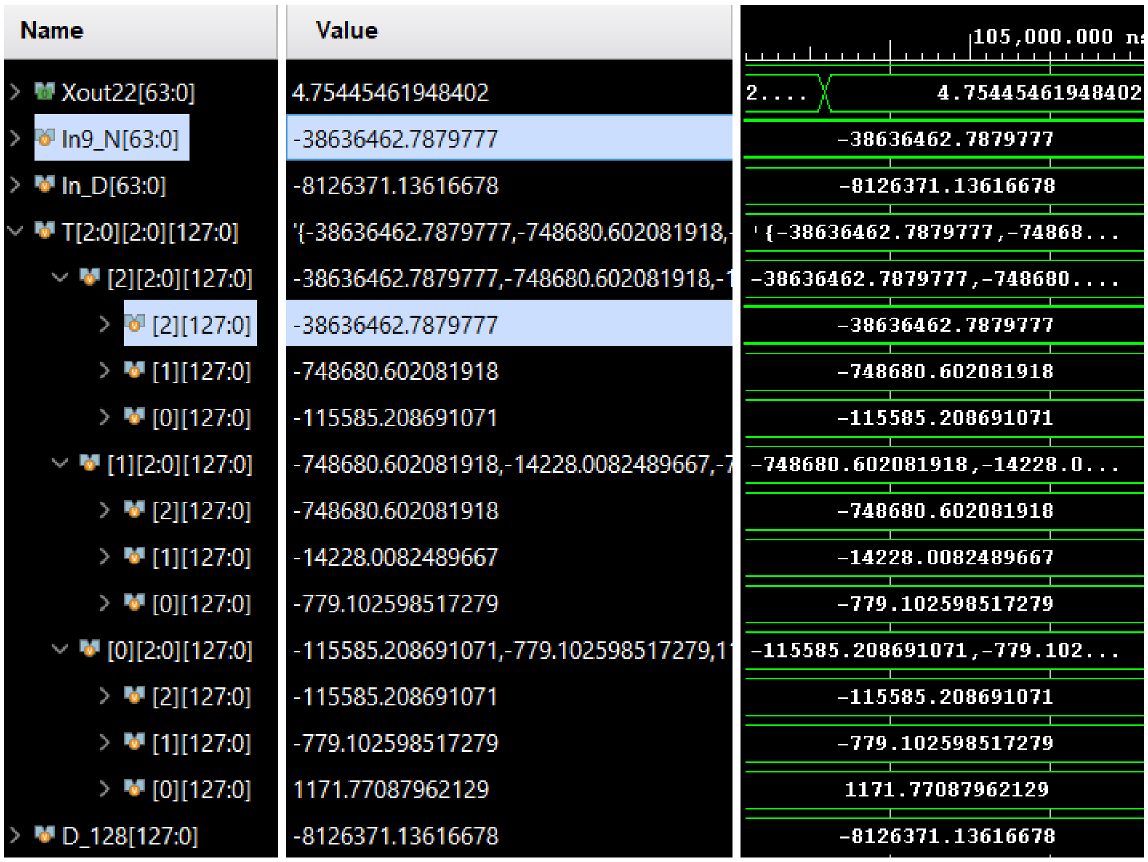
The exponential growth of internal register values led to data loss during bit manipulation, notably affecting the In9_N register within the 3×3 Matrix inversion module, causing the module to malfunction. Figure 5.9. depicts T[2][2], representing the 128-bit internal register, subsequently converted to the 64-bit register In9_N using a (1, 24, 39) fixed-point notation. It becomes evident that the 64-bit register lacks the capacity to accommodate such a large value, resulting in data loss during the conversion process from 128-bit to 64-bit. The affected values are highlighted in blue in Figure 5.9.



Name	Value	105,000.000 ns
Xout22[63:0]	0.625209693880606	2.... 0.625209693880606
In9_N[63:0]	-5081090.32790944	-5081090.32790944
In_D[63:0]	-8127017.82721711	-8127017.82721711
T[2:0][2:0][127:0]	'{-38635522.3279094,-748661.94546853,-	'{-38635522.3279094,-74866...
[2][2:0][127:0]	-38635522.3279094,-748661.94546853,-1	-38635522.3279094,-748661....
[2][127:0]	-38635522.3279094	-38635522.3279094
[1][127:0]	-748661.94546853	-748661.94546853
[0][127:0]	-115582.256192828	-115582.256192828
[1][2:0][127:0]	-748661.94546853,-14227.7538537234,-7	-748661.94546853,-14227.75...
[2][127:0]	-748661.94546853	-748661.94546853
[1][127:0]	-14227.7538537234	-14227.7538537234
[0][127:0]	-778.987738598855	-778.987738598855
[0][2:0][127:0]	-115582.256192828,-778.987738598855,1	-115582.256192828,-778.987...
[2][127:0]	-115582.256192828	-115582.256192828
[1][127:0]	-778.987738598855	-778.987738598855
[0][127:0]	1171.87409966324	1171.87409966324
D_128[127:0]	-8127017.82721711	-8127017.82721711

Fig 5.9. Loss of data from 128-bit to 64-bit manipulation with (1, 24, 39) format.

To address the necessity for handling a broader range of integer values, the entire system was revamped to utilize 64-bit registers with a format of 1 sign bit, 28 integer bits, and 35 fractional bits. As depicted in Figure 5.10., expanding the integral range within the 64-bit internal registers enabled In9_N to retain the complete value of T[2][2]. This strategic approach to design methodology effectively resolved all data loss issues associated with these internal register values within the Matrix inversion module for the initial few samples. However, during the execution of thousands of samples, it became apparent that these intermediate registers with a 28-bit integer part are also insufficient to hold all the data.

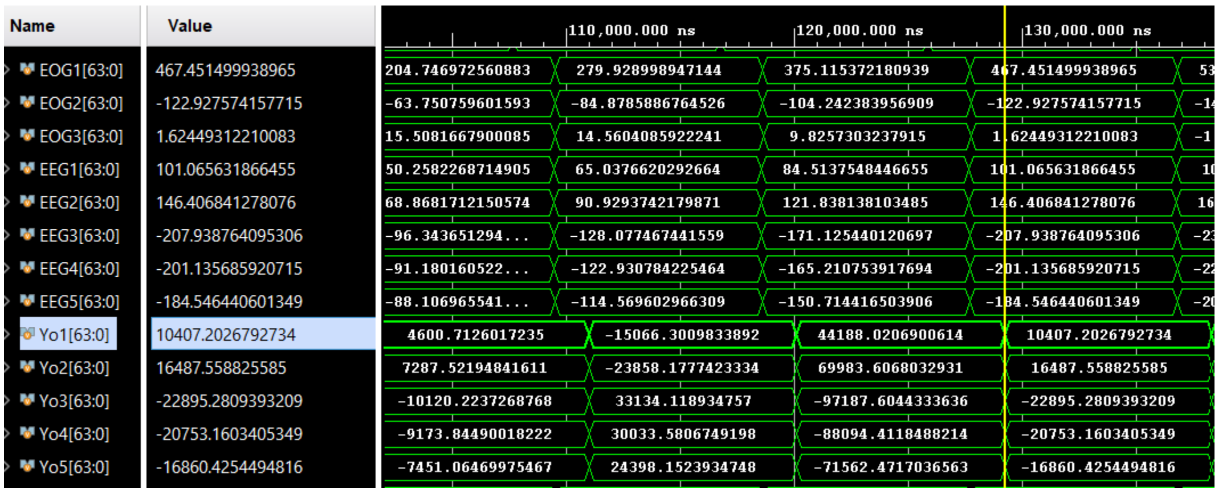


Name	Value	105,000.000 n
Xout22[63:0]	4.75445461948402	2.... 4.75445461948402
In9_N[63:0]	-38636462.7879777	-38636462.7879777
In_D[63:0]	-8126371.13616678	-8126371.13616678
T[2:0][2:0][127:0]	'{-38636462.7879777,-748680.602081918,	'{-38636462.7879777,-74868...
[2][2:0][127:0]	-38636462.7879777,-748680.602081918,-1	-38636462.7879777,-748680....
[2][127:0]	-38636462.7879777	-38636462.7879777
[1][127:0]	-748680.602081918	-748680.602081918
[0][127:0]	-115585.208691071	-115585.208691071
[1][2:0][127:0]	-748680.602081918,-14228.0082489667,-7	-748680.602081918,-14228.0...
[2][127:0]	-748680.602081918	-748680.602081918
[1][127:0]	-14228.0082489667	-14228.0082489667
[0][127:0]	-779.102598517279	-779.102598517279
[0][2:0][127:0]	-115585.208691071,-779.102598517279,1	-115585.208691071,-779.102...
[2][127:0]	-115585.208691071	-115585.208691071
[1][127:0]	-779.102598517279	-779.102598517279
[0][127:0]	1171.77087962129	1171.77087962129
D_128[127:0]	-8126371.13616678	-8126371.13616678

Fig 5.10. Valid 128-bit to 64-bit manipulation with (1, 28, 35) format.

5.3.1.2. Need for 128-bit Custom Divide Module

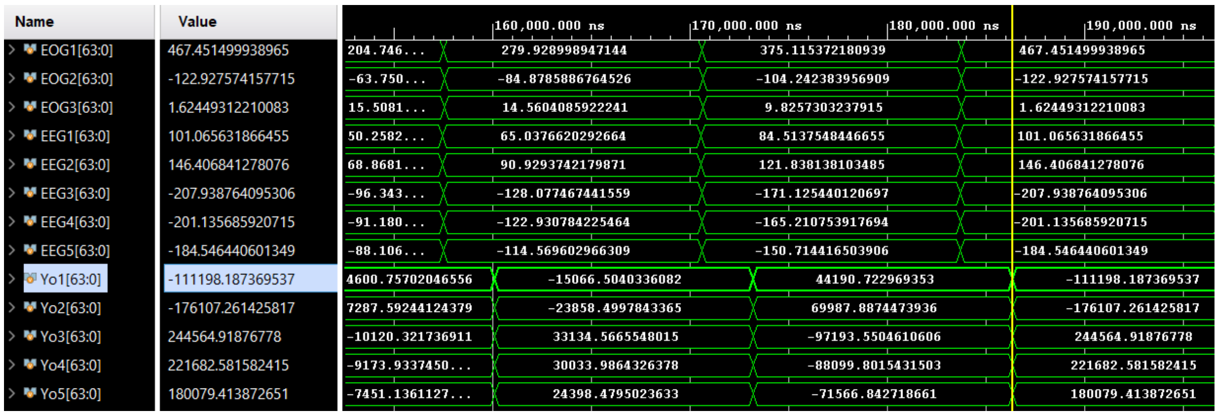
To address the data loss during bit manipulation, I delved deeper into the root cause instead of solely expanding the range of integral bits. Through analysis, it was identified that converting 128-bit values to 64-bit values before feeding them into the divide sub-module within the 3×3 Matrix inversion module was the underlying problem. Consequently, a custom 128-bit divide module was developed and was directly fed the 128-bit inputs to the divide module, effectively bypassing any data loss during bit manipulation.



```
Name          Value               |110,000.000 ns           |120,000.000 ns           |130,000.000 ns
EOG1[63:0]    467.451499938965    204.746972560883   279.928998947144    375.115372180939    467.451499938965    53
EOG2[63:0]    -122.927574157715   -63.750759601593   -84.8785886764526   -104.242383956909   -122.927574157715   -1
EOG3[63:0]    1.62449312210083    15.5081667900085   14.5604085922241    9.8257303237915     1.62449312210083    -1
EEG1[63:0]    101.065631866455    50.2582268714905   65.0376620292664    84.5137548446655    101.065631866455    1
EEG2[63:0]    146.406841278076    68.8681712150574   90.9293742179871    121.838138103485    146.406841278076    16
EEG3[63:0]    -207.938764095306   -96.343651294...   -128.077467441559   -171.125440120697   -207.938764095306   -2
EEG4[63:0]    -201.135685920715   -91.180160522...   -122.930784225464   -165.210753917694   -201.135685920715   -2
EEG5[63:0]    -184.546440601349   -88.106965541...   -114.569602966309   -150.714416503906   -184.546440601349   -2
Yo1[63:0]     10407.2026792734    4600.7126017235    -15066.3009833892   44188.0206900614    10407.2026792734
Yo2[63:0]     16487.558825585     7287.52194841611   -23858.1777423334   69983.6068032931    16487.558825585
Yo3[63:0]     -22895.2809393209   -10120.2237268768  33134.118934757     -97187.6044333636   -22895.2809393209
Yo4[63:0]     -20753.1603405349   -9173.84490018222  30033.5806749198    -88094.4118488214   -20753.1603405349
Yo5[63:0]     -16860.4254494816   -7451.06469975467  24398.1523934748    -71562.4717036563   -16860.4254494816
```

Fig 5.11. Partial passing scenario with 64-bit(1, 28, 35) fixed-point format.

In Figure 5.11., the output of EEG channel 1, denoted as Yo1 in a 64-bit (1, 28, 35) fixed-point format, shows valid results up to its 12th sample output (4600.72) compared with MATLAB results. However, from the 13th sample onward, the outputs started deviating from expected results. When custom 128-divide module is built and integrated, this solution significantly mitigated the issue, ensuring valid outputs during the processing of subsequent samples until reaching a few thousand samples. As depicted in Fig 5.12., subsequent samples from the 13th sample onward began exhibiting accurate results.



```
Name          Value                            |160,000.000 ns      |170,000.000 ns       |180,000.000 ns        |190,000.000 ns
EOG1[63:0]    467.451499938965    204.746...         279.928998947144       375.115372180939        467.451499938965
EOG2[63:0]    -122.927574157715   -63.750...         -84.8785886764526      -104.242383956909       -122.927574157715
EOG3[63:0]    1.62449312210083    15.5081...         14.5604085922241       9.8257303237915         1.62449312210083
EEG1[63:0]    101.065631866455    50.2582...         65.0376620292664       84.5137548446655        101.065631866455
EEG2[63:0]    146.406841278076    68.8681...         90.9293742179871       121.838138103485        146.406841278076
EEG3[63:0]    -207.938764095306   -96.343...         -128.077467441559      -171.125440120697       -207.938764095306
EEG4[63:0]    -201.135685920715   -91.180...         -122.930784225464      -165.210753917694       -201.135685920715
EEG5[63:0]    -184.546440601349   -88.106...         -114.569602966309      -150.714416503906       -184.546440601349
Yo1[63:0]     -111198.187369537   4600.75702046556   -15066.5040336082      44190.722969353         -111198.187369537
Yo2[63:0]     -176107.261425817   7287.59244124379   -23858.4997843365      69987.8874473936        -176107.261425817
Yo3[63:0]     244564.91876778     -10120.321736911   33134.5665548015       -97193.5504610606       244564.91876778
Yo4[63:0]     221682.581582415    -9173.9337450...   30033.9864326378       -88099.8015431503       221682.581582415
Yo5[63:0]     180079.413872651    -7451.1361127...   24398.4795023633       -71566.842718661        180079.413872651
```

Fig 5.12. Passing scenario with 128-bit Divide module.

5.3.1.3. Fine Tuning the Initial Parameters

The integration of the 128-bit custom divide module into the H-infinity module has significantly improved its processing capabilities, enabling the accurate handling of thousands of samples. However, after approximately 8000 samples, a substantial error emerged, resulting in corrupted outputs for all subsequent samples, as depicted in Figure 5.13. The results remained correct until 34.7 seconds, after which the error escalated, leading to data corruption. Extensive debugging efforts and experimentation with various initial parameters, including γ, P0, and q, were conducted. After stabilizing the model with the following parameter set, $q = 10^{-8}$, $\gamma = 1.15$, and $P_0 = 5$, the expected results were achieved, as shown in Fig 5.17.

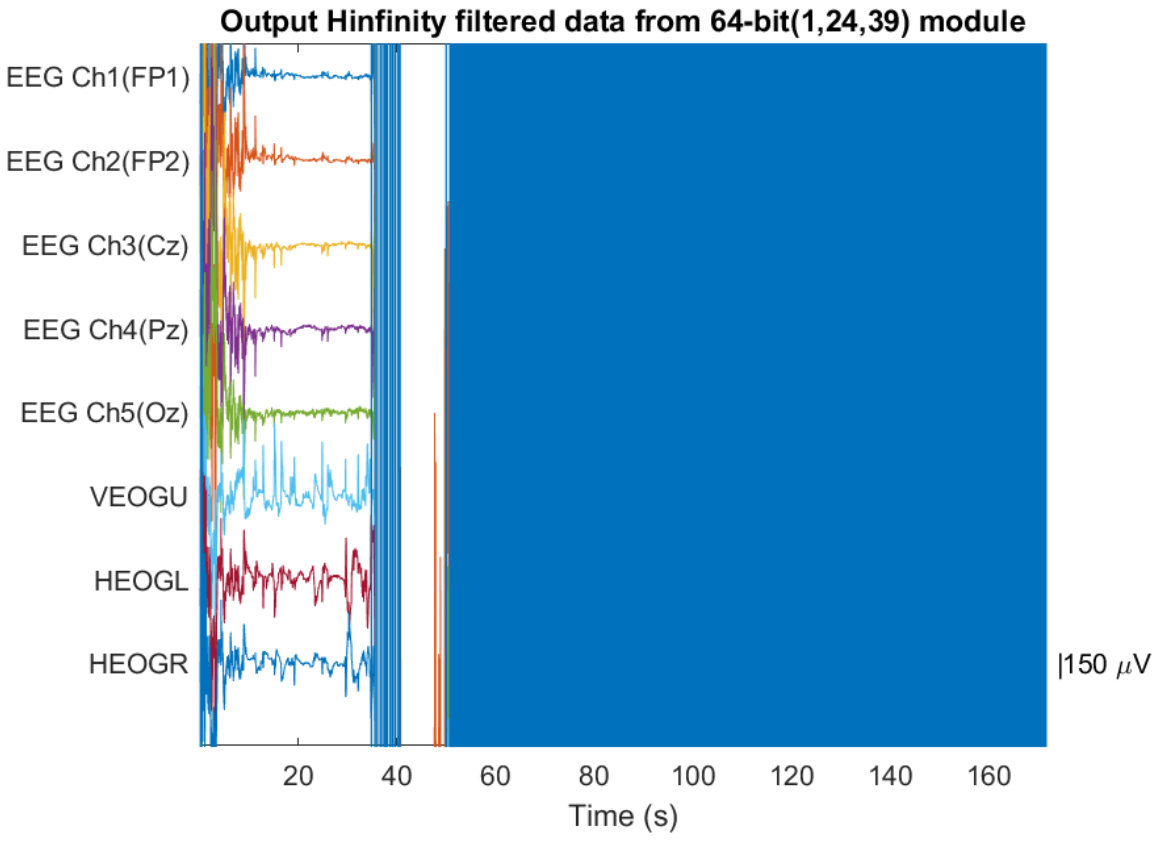


Fig 5.13. H-infinity filtered EEG data plot from 64-bit(1,24,39) Module.

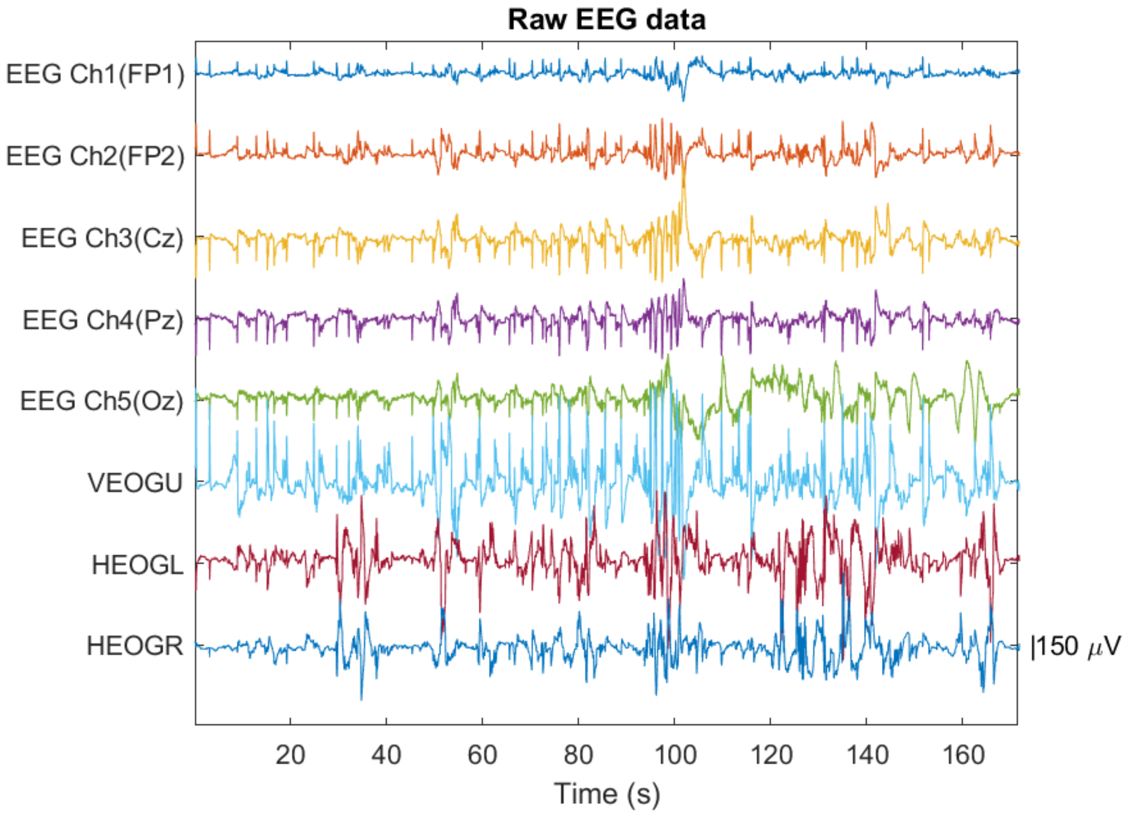


Fig 5.14. Raw EEG data plot from MATLAB.

Figure 5.14 illustrates a MATLAB plot of Raw EEG data, displaying the voltages of 5 EEG channels and 3 Reference EOG channels on the Y-axis in microvolts (µV) against time on the X-axis in seconds. Figure 5.15 depicts the MATLAB plot of High pass filtered EEG data, where a 4th order Butterworth high pass filter with a sampling frequency of 500 Hz and a cutoff frequency of 0.5 Hz was applied to eliminate noise spectrum below 0.5 Hz as recommended by the Author [1]. This filtered EEG data, converted to 64-bit fixed-point data, is then fed into the H-infinity module.

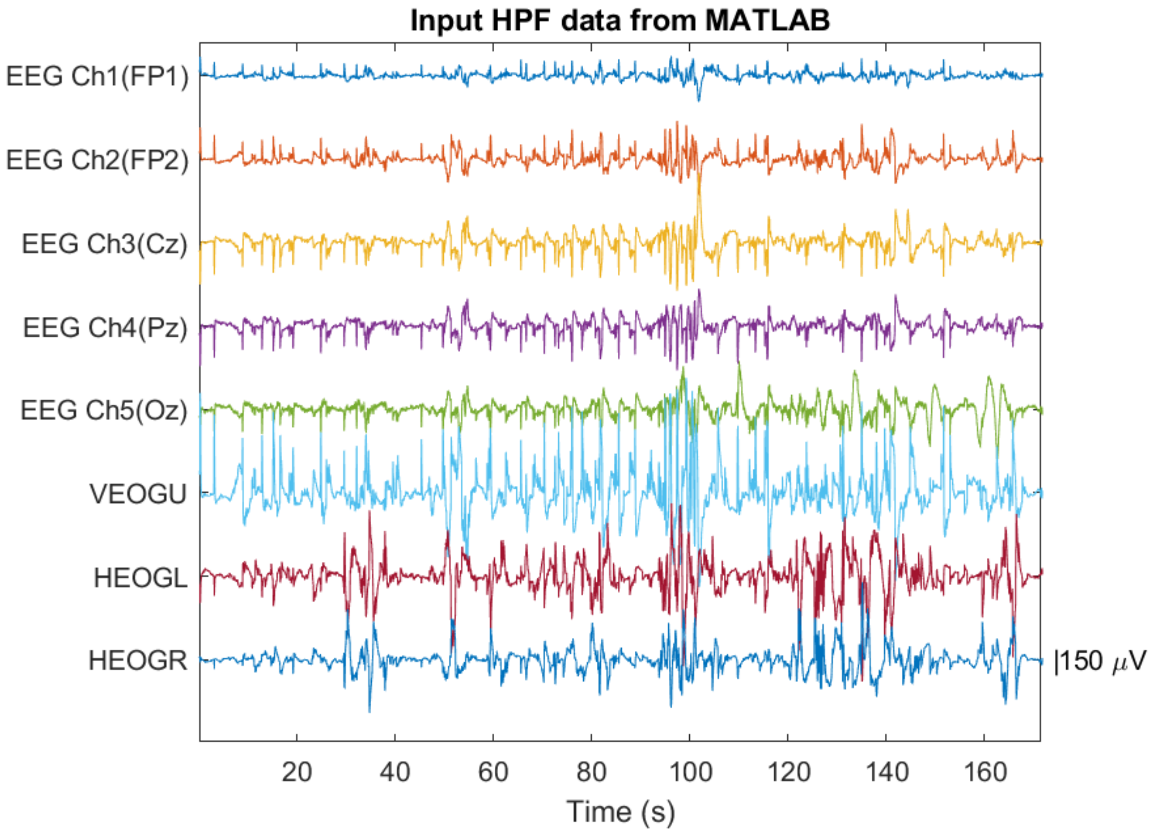


Fig 5.15. High-pass filtered EEG data plot from MATLAB.

Subsequently, Figure 5.16 shows the clean EEG data after processing through the High pass filter and H-infinity filter plotted in MATLAB. This data, representing the reference for the hardware-implemented H-infinity filter model, demonstrates the effectiveness of the filtering process.

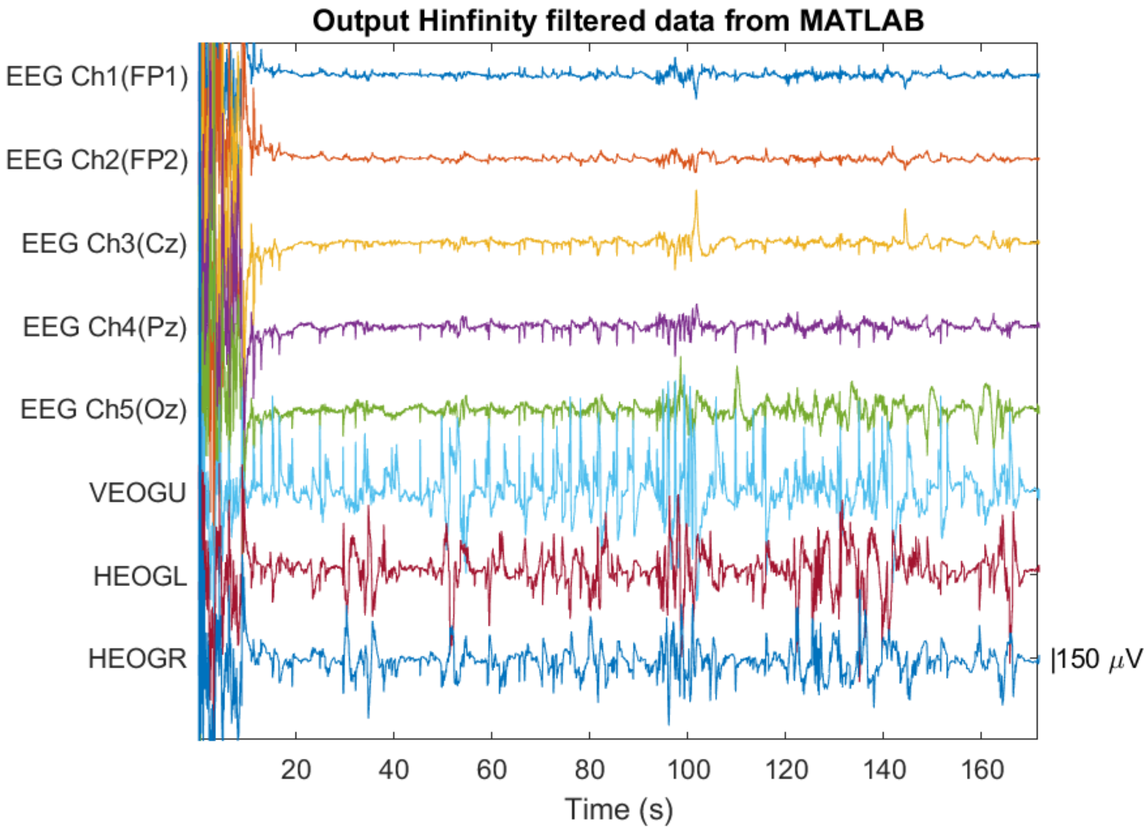


Fig 5.16. H-infinity filtered EEG data plot from MATLAB.

Finally, Figure 5.17 presents the clean EEG data after passing through the H-infinity filter module on hardware, plotted in MATLAB. The voltages of 5 EEG channels and 3 Reference EOG channels on the Y-axis in microvolts (µV) against time on the X-axis in seconds exhibit a significant correlation with the MATLAB simulated results, thereby validating the accuracy and performance of the hardware implementation. The results obtained from the hardware implementation were compared with MATLAB simulated outputs, revealing a Pearson correlation index of 0.9993. This high correlation index indicates excellent agreement and strong similarity between the hardware-generated results and the MATLAB simulations. It underscores the accuracy and reliability of the hardware implementation of the H-infinity filter module, validating its effectiveness in processing EEG signals and removing artifacts.

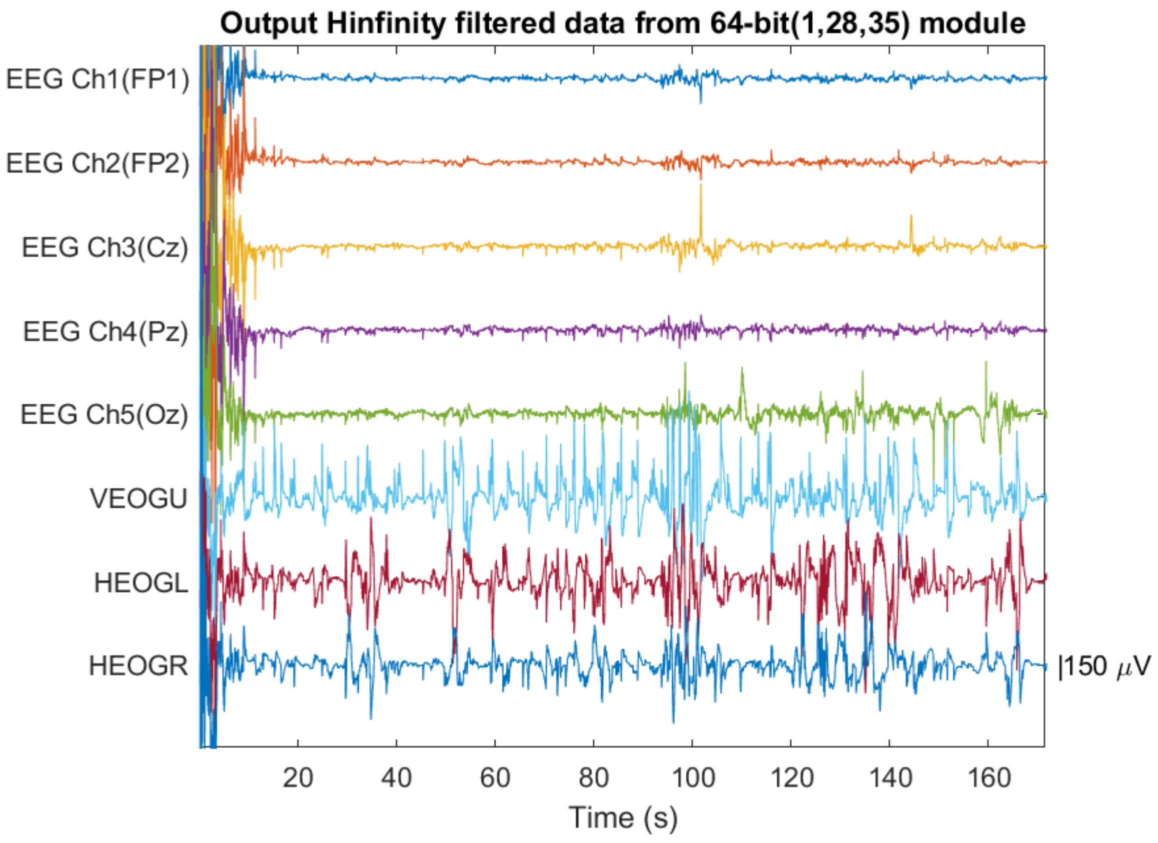


Fig 5.17. H-infinity filtered EEG data plot from 64-bit(1,28,35) Module.

5.3.2. Synthesis and Implementation

The H-infinity module's 64-bit implementation has been effectively synthesized, with a post-synthesis resource utilization report provided in Table 5.8. This table outlines the percentage of utility across various resources. Subsequent to successful placement and routing, the final resource utilization summary is presented in Table 5.9.

Table 5.8. Resource utilization comparison post synthesis for 64-bit implementation.

Resource	**Available**	**H-infinity Module with 128-bit Divide module**	
		Used	**%Utility**
LUT	117120	44413	37.92
FF	234240	14577	6.22
BRAM	144	0	0.00
DSPs	1248	984	78.85

Table 5.9. Resource utilization comparison post implementation for 64-bit implementation.

Resource	**Available**	**H-infinity Module with 128-bit Divide module**	
		Used	**%Utility**
LUT	117120	37979	32.43
FF	234240	14674	6.26
BRAM	144	0	0.00
DSPs	1248	984	78.85

Figure 5.18 visually presents post-implementation metrics including LUTs, DSPs, IOBs, and other parameters, all aligning with the resource constraints predefined for the Kria KR260 Zynq Ultrascale+ MPSoC FPGA.

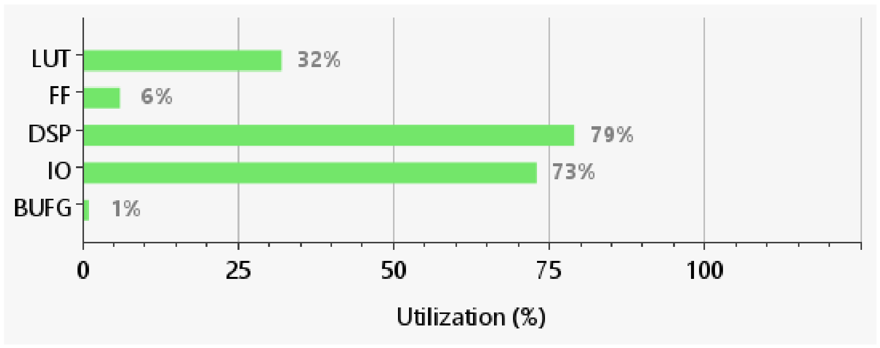
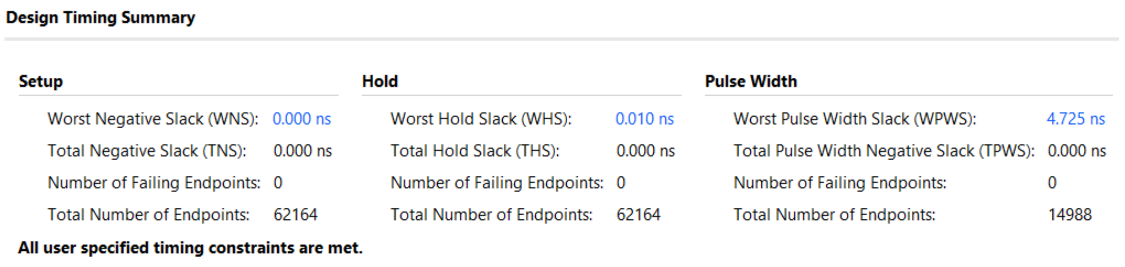


Fig 5.18. Percentage utilization post implementation snippet from Vivado.

The 16-state FSM logic and instantiation of Divide and Matrix inversion sub-modules resulted in a notable number of hold time violations. Specifically, the absence of combinational logic between consecutive flops in the FSM flow, particularly where intermediate computed results are stored in internal registers, caused hold time issues. Additionally, capturing output from sub-modules within different stages of the FSM contributed to hold time violations.

To rectify hold time violations, strategic buffer insertions were made between the Divide and Matrix Inversion modules and the receiving flops. This resolved a portion of the hold time violations. Advanced implementation techniques from Vivado tool were employed to address the remaining hold time violations. Figure 5.19. illustrates that the design complies with all setup and hold time constraints, with a setup time slack of 0.188ns within a clock period of 10ns.


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.000 ns	Worst Hold Slack (WHS):	0.010 ns	Worst Pulse Width Slack (WPWS):	4.725 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:	62164	Total Number of Endpoints:	62164	Total Number of Endpoints:	14988

All user specified timing constraints are met.

Fig 5.19. Post-implementation timing summary.

In summary, the resolution of hold time violations in the H-infinity Filter module highlights the intricacies involved in implementing complex algorithms within a system architecture. Through strategic buffer insertion, clock adjustments, and integration techniques, timing constraints were successfully resolved, meeting both setup and hold time requirements.

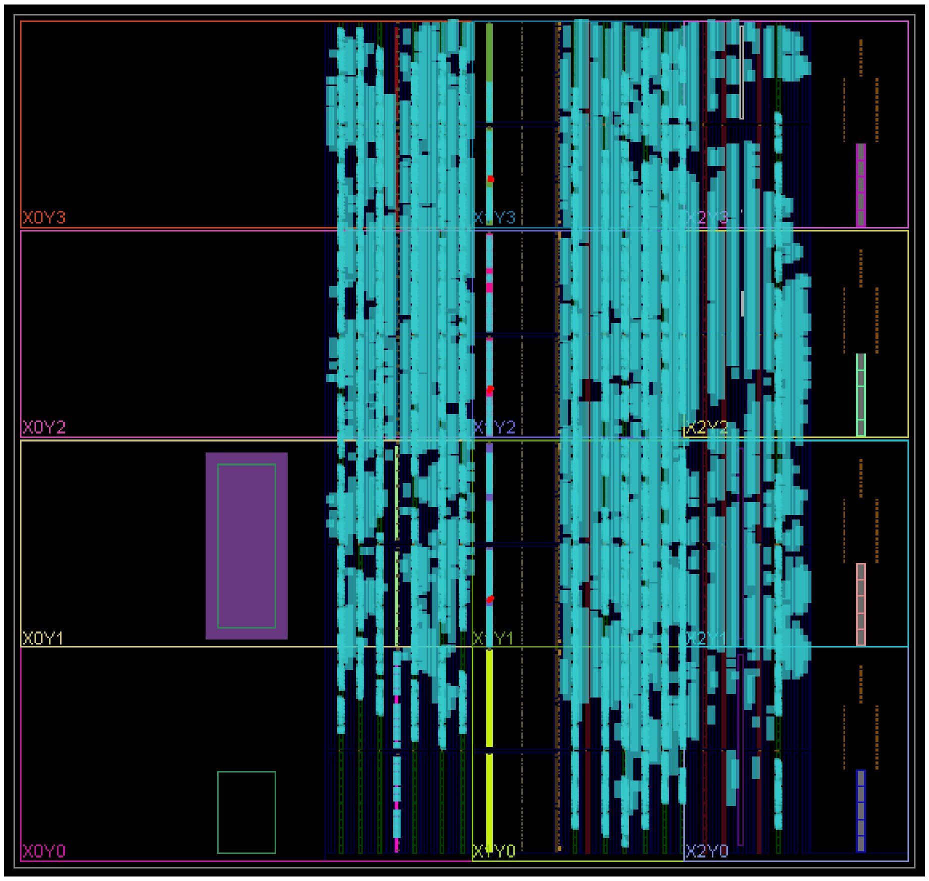


Fig 5.20. Resource utilization snippet after place and route from Vivado.

CHAPTER-6

SUMMARY AND CONCLUSIONS

Aim of the research is to study FPGA based computational modules for ocular artifact removal algorithm to process multi-channel EEG data on hardware. The research delves into the comprehensive design, verification, and optimization process of key computational modules, specifically focusing on custom divide modules, 3×3 matrix inversion module, and the H-infinity Filter module. Detailed functional verification, performance analysis, and resource utilization assessments were conducted throughout the development cycle to ensure robustness, accuracy, and efficient use of FPGA resources.

- **Custom Divide Modules:** The custom 64-bit and 128-bit divide modules were designed and validated for optimal resource utilization and performance. Extensive testing with various input scenarios demonstrated accurate computation and negligible errors.
- **3×3 Matrix Inversion Module:** Two distinct methodologies were explored for the 3×3 Matrix Inversion module, balancing performance optimization and resource efficiency. The performance-optimized approach, integrating 9 distinct divide blocks, was selected based on thorough validation and timing analysis, aligning with project requirements.
- **H-infinity Filter Module:** The H-infinity Filter module serves as a critical component within our system architecture, aimed at filtering noise from Raw EEG data through intricate matrix inversions, multiplications, and fixed-point

data handling. Throughout the development and optimization process, several key aspects were addressed to enhance its functionality and performance.

The design process encountered several challenges, which were addressed through key strategies as mentioned below:

- **Data Loss during bit manipulation:** The integer part of internal 64-bit registers was increased, allowing for the retention of larger values without data loss. This adjustment was crucial in maintaining accuracy throughout the processing of multiple samples.
- **Custom 128-bit Divide Module:** A custom 128-bit divide module was designed to handle the increased range requirements of sub-product values. By directly feeding 128-bit inputs into the divide module, data loss during bit manipulation was prevented, ensuring valid outputs, and improving overall performance.
- **Fine tuning the initial parameters:** The H-infinity filter's initial parameters, such as γ, P0, and q values, were finalized through trial and error, testing various combinations to stabilize the model and ensure accurate processing of thousands of samples.

Transitioning to the 128-bit custom divide sub-module inside 3×3 matrix inversion module and expanding the integer part of 64-bit internal registers to 28 bits marked a significant improvement in accuracy, achieving a high Pearson correlation index (0.9993) with MATLAB results across all EEG channels. Functional validation results, comparative analyses, and resource utilization summaries provide a comprehensive understanding of the complexities involved in implementing high-precision algorithms like the H-infinity Filter module on FPGA platforms. Despite challenges, strategic design decisions and effective debugging techniques were instrumental in achieving accurate and reliable performance.

In conclusion, the development and validation of the H-infinity Filter module have been a journey of addressing precision challenges, data loss issues, and fixed-point range limitations while striving for high accuracy and correlation with MATLAB results. Key takeaways include the importance of systematic design approaches, the necessity for custom modules to handle precision errors, and the delicate balance between accuracy, performance, and resource utilization.

In the forthcoming phases, ongoing research and development endeavors will concentrate on seamlessly integrating the H-infinity filter module with Signal Acquisition infrastructure, aiming to construct an efficient solution for preprocessing EEG signals. The objective is to deliver pristine EEG signals to Brain-Computer Interface (BCI) systems, facilitating the extraction of meaningful data for the implementation of diverse BCI applications. This research initiative serves as a cornerstone for future strides in FPGA-based implementations of sophisticated adaptive noise cancellation filters, thereby enriching the domain of signal processing and neurotechnology.

REFERENCES

[1] Atilla Kilicarslan, Robert G Grossman, and Jose Luis Contreras-Vidal, “A robust adaptive denoising framework for real-time artifact removal in scalp EEG measurements,” *Journal of Neural Engineering,* 2016 J. Neural Eng. 13 026013, DOI 10.1088/1741-2560/13/2/026013, Available: https://iopscience.iop.org/article/10.1088/1741-2560/13/2/026013/meta

[2] Kevin Nathan, Atilla Kilicarslan, Robert Grossman, and Jose Luis Contreras-Vidal, “Evaluating a longitudinal brain-machine interface training paradigm with a lower-limb exoskeleton,” Department of Electrical & Computer Engineering, IUCRC BRAIN, University of Houston, Houston, TX, USA.

[3] Juan José González-España, Alexander Craik, Carolina Ramirez, Ayman Alamir, and Jose Luis Contreras-Vidal, “Optimization of electrode configuration for the removal of eye artifacts with adaptive noise cancellation,” 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC), DOI:10.1109/SMC53992. 2023.10394375, Available: https://ieeexplore-ieee-org.ezproxy.lib.uh.edu/document /10394375

[4] Malik Muhammad Naeem Mannan, Muhammad Ahmad Kamran, Myung Yung Jeong, “Identification and removal of physiological artifacts from electroencephalogram signals: A review,” IEEE Access, Volume: 6, Digital Object Identifier 10.1109 /ACCESS.2018.2842082, Available: https://ieeexplore.ieee.org/document/8369420

[5] Alexander Craik, Juan José González-España, Ayman Alamir, David Edquilang, Sarah Wong, Lianne Sánchez Rodríguez, Jeff Feng, Gerard E. Francisco, and

Jose L. Contreras-Vidal, “Design and validation of a low-cost mobile EEG-based Brain–Computer Interface,” *Sensors* 2023, *23*(13), 5930; https://doi.org/10.3390/s23135930

[6] Atilla Kilicarslan and Jose Luis Contreras-Vidal, “Characterization and real-time removal of motion artifacts from EEG signals,” *Journal of Neural Engineering*, 2019 J. Neural Eng. 16 056027, DOI 10.1088/1741-2552/ab2b61, Available: https://pubmed.ncbi.nlm.nih.gov/31220818/

[7] Yu Xie, Tamás Majoros, and Stefan Oniga, “FPGA-based hardware accelerator on portable equipment for EEG signal patterns recognition,” *Electronics* 2022, *11*(15), 2410; https://doi.org/10.3390/electronics11152410