DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PUDUCHERRY TECHNOLOGICAL UNIVERSITY

PUDUCHERRY – 605 014

BONAFIDE CERTIFICATE

This is to certify that this "**FPGA BASED AUTOMATIC NOISE CANCELLATION SYSTEM USING MODIFIED ADAPTIVE FILTERING**" is the bonafide work done by **Nivesh S (21EC1063), Priyadharshini P (21EC1074), Kiruthik Shankar S (21EC1054), Gaurav Kumar (21ME1023)** in the partial fulfilment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY** in **ELECTRONICS AND COMMUNICATION ENGINEERING** during the year 2024-2025 in **PUDUCHERRY TECHNOLOGICAL UNIVERSITY** and this work have not been submitted for the award of any other degree of this/any other university.

PROJECT GUIDE	**HEAD OF THE DEPARTMENT**
(Dr. A. V. ANANTHALAKSHMI)	**(Dr. V. SAMINADAN)**

Submitted to the MINI PROJECT REVIEW COMMITTEE (INTERNAL) ON -----------------

ACKNOWLEDGEMENT

The successful completion of the project is indeed practically incomplete without the mention of all those people who greatly supported and encouraged us throughout the project. We feel grateful to our guide **Dr. A. V. ANANTHALAKSHMI**, Associate Professor, Department of Electronics and Communication Engineering, Puducherry Technological University, for her encouragement and support. She has been a source of valuable guidance, suggestions and kindness during the course of project work.

We would like to express our sincere thanks to our respected **VICE CHANCELLOR Dr. S. MOHAN**, Puducherry Technological University for providing the University facilities for the completion of this project.

We would like to express our heartfelt gratitude to **Dr. V. SAMINADAN**, Professor and Head, Department of Electronics and Communication Engineering, Puducherry Technological University for providing all department facilities and guidance which enabled us to complete the project.

We feel obliged to thank the main project panel members **Dr. G. NAGARAJAN,** Professor**, Dr. K. JAYANTHI,** Professor **and Dr. R. SANDANALAKSHMI,** Associate Professor, Dept. of Electronics and Communication Engineering, PTU for the support, encouragement and guidance.

We express our deep sense of gratitude to all teaching and non-teaching staff of our department, and to our friends for their support and encouragement during the entire course of this main project work. We also express our sincere thanks to our parents who motivated us in all means.

ABSTRACT

Noise control is a critical requirement in audio processing, particularly for applications like telecommunication, hearing aids, and real-time audio enhancement. Traditional algorithms such as Least Mean Square (LMS) and Normalized LMS (NLMS) are commonly used but often encounter limitations such as slow convergence rates and high computational demands, making them less effective for real-time applications in high-noise environments. Additionally, implementing these algorithms on conventional hardware platforms often results in increased latency and power consumption.

This project focuses on designing and implementing an optimized FPGA-based noise control system on the Xilinx UltraScale+ platform. The system employs the Recursive Least Square (RLS) algorithm, known for its fast convergence properties, combined with spectral subtraction for noise preprocessing and Particle Swarm Optimization (PSO) for dynamically optimizing filter parameters. The FPGA's parallel processing capabilities are leveraged to achieve significant improvements in signal-to-noise ratio (SNR), reduced latency, and high throughput for real-time audio applications.

Initial results from the software implementation show enhanced noise reduction and improved convergence speeds, providing a robust foundation for FPGA-based optimization. The next phase involves translating the design into hardware, focusing on efficient resource utilization, including logic units, memory, and DSP blocks, while minimizing power consumption.

Future tasks will focus on developing a hardware architecture specifically optimized for the RLS algorithm to enhance performance and resource utilization. Additionally, the design will be extended to support multi-channel audio processing, enabling broader applicability in complex audio environments. Comprehensive benchmarking will be conducted using real-world audio datasets to evaluate critical performance metrics, including Signal-to-Noise Ratio (SNR), Mean Square Error (MSE), and overall resource efficiency, ensuring the system's reliability and effectiveness in practical scenarios.

In conclusion, this project demonstrates the potential of FPGA-based systems to overcome the limitations of traditional hardware in adaptive noise control. By optimizing algorithms and exploiting FPGA's inherent parallelism, this design offers a scalable and efficient solution for advanced audio processing in diverse real-time environments. Future advancements could include the integration of deep learning for dynamic noise adaptation and the application of the system to broader fields like speech recognition and multimedia processing.

Keywords : Adaptive Filtering, FPGA, Noise Cancellation, RLS, PSO

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

• **LMS**	**-**	Least Mean Square
• **NLMS**	**-**	Normalized Least Mean Square
• **RMS**	**-**	Recursive Mean Square
• **PSO**	**-**	Particle Swarm Optimization
• **FPGA**	**-**	Field Programmable Gate Arrays
• **SNR**	**-**	Signal to Noise Ratio
• **GA**	**-**	Genetic Algorithm
• **PYNQ**	**-**	Python Productivity for ZYNQ
• **AXI**	**-**	Advanced eXtensible Interface

CHAPTER 1

INTRODUCTION

1.1 PRELUDE

Noise cancellation plays a transformative role in enhancing the quality of audio signals by mitigating the impact of unwanted noise. In the modern world, the prevalence of noise pollution and the increasing demand for clear audio communication have amplified the importance of noise cancellation systems. From telecommunication networks to personal hearing aids, and from real-time audio enhancement to industrial applications, the effectiveness of noise cancellation systems can directly influence user satisfaction and the overall performance of the devices. Despite significant advancements in software-based noise cancellation, achieving real-time performance with minimal latency, high accuracy, and low power consumption remains a persistent challenge.

Traditional software-based noise cancellation techniques often rely on adaptive filtering algorithms such as the Least Mean Square (LMS) algorithm. While LMS is popular for its simplicity and ease of implementation, its performance is hindered by slow convergence rates and limited ability to adapt to dynamic noise environments. Furthermore, the computational demands of these algorithms, when executed on general-purpose processors, introduce latency and high power consumption, which are critical limitations in portable and real-time systems. Addressing these shortcomings requires a paradigm shift toward hardware-based implementations.

Field-Programmable Gate Arrays (FPGAs) have emerged as a game-changing platform for implementing noise cancellation systems. FPGAs offer unique advantages such as high parallelism, customizable architecture, low latency, and energy-efficient operation, making them ideal for real-time signal processing tasks. By leveraging the inherent parallel processing capabilities of FPGAs, complex algorithms can be executed more efficiently, ensuring faster convergence and higher accuracy in noise cancellation. Moreover, FPGAs provide the flexibility to integrate multiple components, such as adaptive filters, preprocessing modules, and optimization techniques, into a single compact system.

This project focuses on developing an FPGA-based Optimized Automatic Noise Cancellation System that incorporates advanced adaptive filtering techniques. The Recursive Least Square

(RLS) algorithm forms the core of the system due to its superior convergence properties compared to LMS. However, RLS comes with the challenge of higher computational complexity, which is effectively addressed through the parallel processing capabilities of FPGAs. To further enhance the adaptability and performance of the system, the project integrates Particle Swarm Optimization (PSO) for dynamic parameter tuning. Additionally, spectral subtraction is employed as a preprocessing step to improve noise cancellation efficiency by addressing stationary noise components.

The significance of this project extends beyond academic exploration, as it addresses practical challenges in real-world applications. For instance, in telecommunication systems, the proposed noise cancellation system can enhance call quality by reducing background noise. In medical devices such as hearing aids, the system can improve speech intelligibility for users in noisy environments. Furthermore, the low-latency performance of the FPGA-based implementation makes it suitable for real-time applications such as live audio streaming and industrial monitoring systems.

One of the key contributions of this project is the demonstration of how advanced algorithms and hardware platforms can be effectively combined to overcome the limitations of traditional approaches. By systematically integrating the RLS algorithm, spectral subtraction, and PSO on the Xilinx UltraScale+ FPGA platform, the project showcases a high-performance, energy-efficient solution for adaptive noise cancellation. This implementation not only improves the Signal-to-Noise Ratio (SNR) but also ensures efficient utilization of hardware resources, paving the way for scalable and versatile designs that can cater to a wide range of applications.

1.2 MOTIVATION

The motivation for this project stems from the pressing need for advanced noise cancellation systems that can operate in real-time while maintaining high accuracy and low power consumption. Traditional adaptive filtering algorithms, such as the Least Mean Square (LMS) algorithm, are widely used due to their simplicity but suffer from slow convergence and limited performance in dynamic noise environments. The Recursive Least Square (RLS) algorithm, known for its faster convergence, provides a compelling alternative. However, its computational complexity demands an efficient hardware implementation. By utilizing FPGA technology, this project aims to overcome these challenges and contribute to the broader field of real-time audio enhancement, enabling applications ranging from personal devices to large-scale communication systems.

1.3 LITERATURE REVIEW

The field of adaptive noise cancellation has witnessed significant advancements over the years. Ling et al. (2021) [1] optimized the LMS algorithm for system identification and noise cancellation, highlighting its performance in low-complexity environments. Yuan et al. (2024) [2] designed an improved DFxLMS algorithm for compressor noise cancellation, showcasing the potential of FPGA implementations. Yergaliyev and Akhtar (2023) [3] reviewed distributed arithmetic-based hardware implementations of adaptive filters, emphasizing their efficiency in FPGA platforms. Zhang (2023) [4] investigated adaptive algorithms for multichannel active noise control, providing insights into algorithmic enhancements for robust performance. Despite these contributions, the integration of advanced optimization techniques like Particle Swarm Optimization (PSO) into FPGA-based systems remains underexplored, presenting an opportunity for further research and development.

Technique	Journal Title	Year	Limitations	Advantages
LMS [7]	International Journal of Electrical and Computer Engineering	2017	Susceptible to getting stuck in local optima.	Simplicity and ease of implementation.
RLS [7]	International Journal of Electrical and Computer Engineering	2017	High computational complexity.	Fast convergence rate.
GA [8]	International Journal of Applied Engineering Research	2017	Convergence speed is slower compared to other algorithms.	Achieves improved equalization performance with globally optimal solutions.
PSO [9]	Soft Computing	2018	Requires fine-tuning of algorithm parameters.	Provides global search capability.

NLMS [10]	Arabian Journal for Science and Engineering	2021	Slow to converge under noisy conditions	Adjusts step size dynamically based on input signal power.
Deep Learning [11]	IETE Journal of Research	2023	Requires a large amount of training data.	Can adapt to complex and nonlinear channel characteristics.

Table 1.1 : Literature Survey

1.4 OBJECTIVES

The primary objectives of this project are:

- To design and implement a real-time noise cancellation system leveraging FPGA technology.
- To integrate the Recursive Least Square (RLS) algorithm for its superior convergence properties.
- To enhance the system's adaptability and performance using Particle Swarm Optimization (PSO) for parameter tuning.
- To achieve efficient resource utilization and low power consumption while maintaining high Signal-to-Noise Ratio (SNR).
- To validate the system's performance through experimental evaluation on the Xilinx UltraScale+ FPGA platform.

1.5 SYSTEM MODEL

The proposed system consists of several key components working in tandem to achieve effective noise cancellation. The RLS algorithm forms the core of the adaptive filtering process, chosen for its ability to quickly adapt to changing noise environments. Spectral subtraction is employed as a preprocessing step to handle stationary noise components, further improving the system's efficiency. Particle Swarm Optimization (PSO) dynamically tunes the step size of the adaptive filter, ensuring optimal convergence rates. The hardware implementation leverages the Xilinx UltraScale+ FPGA platform, integrating programmable logic with audio I/O functionality via the PYNQ base overlay. Communication between the processing system (PS) and programmable logic (PL) is facilitated through AXI interfaces, enabling seamless real-time processing.

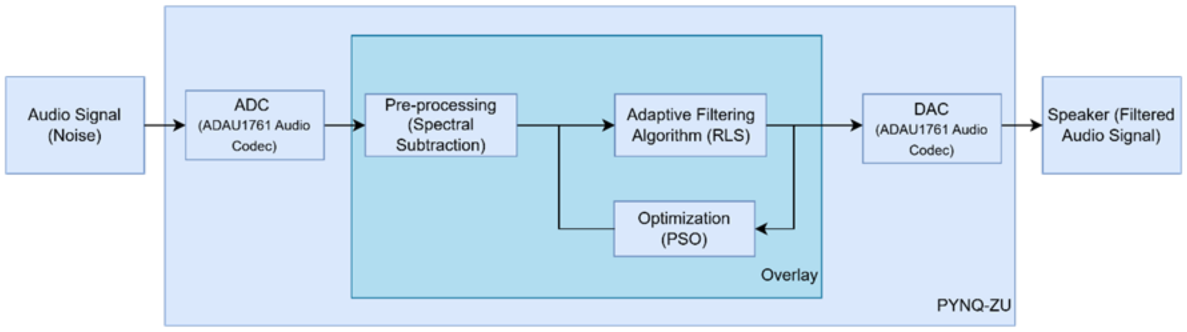

Audio Signal (Noise)
ADC (ADAU1761 Audio Codec)
Pre-processing (Spectral Subtraction)
Adaptive Filtering Algorithm (RLS)
Optimization (PSO)
Overlay
DAC (ADAU1761 Audio Codec)
PYNQ-ZU
Speaker (Filtered Audio Signal)

Fig 1.1 System model

1.6 ORGANIZATION OF THESIS

This thesis is organized into the following chapters:

- **Chapter 1: Introduction**: Provides an overview of noise cancellation, motivation, literature review, objectives, and the organization of the thesis.
- **Chapter 2: Existing Work**: Discusses adaptive filtering algorithms, FPGA implementations, and emerging trends in noise control systems.
- **Chapter 3: Proposed System**: Details the system design, methodology, hardware and software requirements, and the core algorithms, including RLS, spectral subtraction, and PSO.
- **Chapter 4: Simulation Using MATLAB**: Explains the objectives, environment, results, and insights gained from MATLAB simulations.
- **Chapter 5: Implementation in FPGA**: Covers the design overview, implementation workflow, and the integration of the proposed system on the PYNQ-ZU platform.
- **Chapter 6: Results and Discussion**: Presents experimental results and discussions on the system's performance in terms of SNR improvements, latency, and resource utilization.
- **Chapter 7: Conclusion and Future Scope**: Summarizes the contributions of the project, highlights its limitations, and explores potential future enhancements.

CHAPTER 2

EXISTING WORKS

2.1 INTRODUCTION

Adaptive filtering plays a crucial role in noise cancellation, with various algorithms developed to enhance performance in different environments. This chapter explores existing adaptive filtering techniques, their FPGA-based implementations, and emerging trends in noise control. By reviewing key algorithms such as LMS, RLS, and PSO-optimized methods, as well as advancements in FPGA hardware, we highlight the progress and potential of real-time noise cancellation systems.

2.2 ADAPTIVE FILTERING ALGORITHMS FOR NOISE CANCELLATION

Adaptive filtering is the backbone of noise cancellation systems, with various algorithms tailored for different applications. The Least Mean Square (LMS) algorithm, introduced as one of the simplest adaptive filtering techniques, is widely used due to its ease of implementation and low computational requirements. However, its slow convergence and limited performance in non-stationary noise environments necessitate the exploration of advanced algorithms.

The Normalized LMS (NLMS) algorithm improves upon LMS by normalizing the step size, enhancing its stability and convergence speed. Fractional LMS (FLMS) and variants like the Fractional NLMS (FNLMS) have further expanded the algorithm's applicability to scenarios requiring more precise error minimization. Recursive Least Square (RLS), on the other hand, offers superior convergence speed and adaptability by minimizing the mean square error recursively. Despite its computational complexity, RLS remains a preferred choice for dynamic environments. Distributed arithmetic-based implementations have been explored to address its hardware demands, making it feasible for real-time FPGA applications.

Advanced techniques, such as algorithms inspired by swarm intelligence, have also been integrated into adaptive filtering systems. Particle Swarm Optimization (PSO), for example, has been used to optimize the parameters of adaptive filters dynamically. This integration allows the system to balance convergence speed and error minimization adaptively, significantly enhancing the overall performance.

2.3 FPGA IMPLEMENTATIONS OF ADAPTIVE FILTERS

The shift from software-based implementations to hardware-based solutions has been driven by the demand for real-time, energy-efficient systems. Field-Programmable Gate Arrays (FPGAs) have become a popular choice for implementing adaptive filters due to their inherent parallelism and reconfigurability. Studies have shown significant improvements in performance and resource utilization when implementing algorithms like LMS, RLS, and their variants on FPGA platforms.

Yuan et al. (2024) [2] demonstrated the efficiency of FPGA-based implementations of the DFxLMS algorithm for compressor noise cancellation. Their design showcased reduced latency and optimized resource usage, proving the potential of FPGAs for noise cancellation tasks. Similarly, Yergaliyev and Akhtar (2023) [3] reviewed distributed arithmetic approaches for implementing adaptive digital filters, highlighting the scalability and efficiency of FPGA architectures.

The use of high-level synthesis tools like Vivado HLS has further simplified FPGA design processes, allowing for the direct translation of high-level algorithmic descriptions into hardware. This has facilitated the integration of complex algorithms such as spectral subtraction and PSO into FPGA-based systems. Moreover, the availability of advanced FPGA platforms, such as Xilinx UltraScale+, has enabled the development of systems with enhanced computational capabilities and energy efficiency.

2.4 EMERGING TRENDS IN ADAPTIVE NOISE CONTROL

Recent research has focused on integrating machine learning techniques into adaptive noise control systems. These approaches aim to improve the adaptability and robustness of noise cancellation systems by enabling them to learn and respond to diverse noise environments. Additionally, the exploration of multichannel noise cancellation algorithms, as highlighted by Zhang (2023), has paved the way for applications in immersive audio systems and industrial monitoring.

The continued evolution of FPGA technology, coupled with advancements in adaptive filtering algorithms, presents significant opportunities for the development of high-performance noise cancellation systems. By leveraging the strengths of both hardware and algorithmic innovations, future systems can achieve unprecedented levels of accuracy, efficiency, and scalability.

2.5 CONCLUSION

The evolution of adaptive filtering algorithms and FPGA implementations has significantly improved noise cancellation efficiency. While traditional techniques like LMS and RLS remain fundamental, advancements such as swarm intelligence optimization and machine learning integration offer promising future directions. The combination of algorithmic enhancements and high-performance FPGA architectures continues to push the boundaries of real-time noise control, paving the way for more adaptive and efficient systems.

CHAPTER 3

PROPOSED SYSTEM

3.1 INTRODUCTION

This chapter presents the proposed FPGA-based Automatic Noise Cancellation System, designed to enhance traditional noise cancellation techniques through advanced algorithms and hardware architectures. The system leverages the Recursive Least Squares (RLS) algorithm for adaptive filtering, spectral subtraction for noise preprocessing, and Particle Swarm Optimization (PSO) for parameter tuning. The methodology, hardware and software requirements, and core algorithms are discussed in detail to outline the design and implementation process.

3.2 SYSTEM OVERVIEW

The proposed FPGA-based Automatic Noise Cancellation System is designed to address the limitations of traditional noise cancellation techniques by leveraging advanced algorithms and hardware architectures. The system integrates the Recursive Least Square (RLS) algorithm for adaptive filtering, spectral subtraction for noise preprocessing, and Particle Swarm Optimization (PSO) for parameter tuning. This chapter elaborates on the methodology, hardware and software requirements, and the core algorithms used in the design and implementation process.

3.3 METHODOLOGY

The methodology involves a systematic approach to system design and implementation:

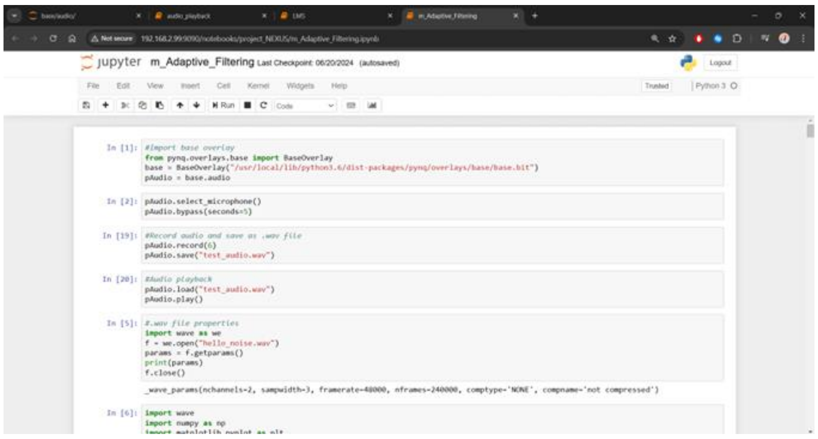
1. **Software-Level Simulation**: The RLS algorithm was initially developed and tested using MATLAB for algorithmic validation. Performance metrics such as Signal-to-Noise Ratio (SNR) improvement and convergence speed were evaluated.

2. **Hardware Design**: The system was implemented on the Xilinx UltraScale+ FPGA platform, utilizing Vivado HLS for RTL generation and Vivado Design Suite for synthesis and implementation. The hardware design includes modules for adaptive filtering, spectral subtraction, and PSO-based parameter optimization.
3. **Integration and Testing**: The FPGA implementation was integrated with audio I/O functionality using the PYNQ base overlay. Real-time audio processing was tested using the Jupyter Notebook interface.

3.4 HARDWARE AND SOFTWARE REQUIREMENTS

- **Hardware Requirements**:
  - Xilinx PYNQ-ZU board with Zynq UltraScale+ MPSoC
  - Audio input and output devices (e.g., microphone, headphones)

- **Software Requirements**:
  - Vivado HLS and Vivado Design Suite for hardware design
  - MATLAB for algorithm simulation
  - Jupyter Notebook for real-time audio processing



```
In [1]: #Import base overlay
        from pynq.overlays.base import BaseOverlay
        base = BaseOverlay("/usr/local/lib/python3.6/dist-packages/pynq/overlays/base/base.bit")
        pAudio = base.audio

In [2]: pAudio.select_microphone()
        pAudio.bypass(seconds=5)

In [19]: #Record audio and save as .wav file
         pAudio.record(6)
         pAudio.save("test_audio.wav")

In [20]: #Audio playback
         pAudio.load("test_audio.wav")
         pAudio.play()

In [5]: #.wav file properties
        import wave as we
        f = we.open("hello_noise.wav")
        params = f.getparams()
        print(params)
        f.close()

        _wave_params(nchannels=2, sampwidth=3, framerate=48000, nframes=240000, comptype='NONE', compname='not compressed')

In [6]: import wave
        import numpy as np
        import matplotlib.pyplot as plt
```

Fig 3.1 Jupyter Notebook Framework

3.5 PYNQ-ZU AND ITS ADVANTAGE IN NOISE CONTROL SYSTEM

The PYNQ-ZU is a high-performance development platform built around the **Xilinx Zynq UltraScale+ MPSoC (Multiprocessor System-on-Chip)**. It combines a powerful ARM-based processing system (PS) with programmable logic (PL), providing a flexible and efficient platform for hardware-accelerated applications. Designed to support the **PYNQ (Python Productivity for Zynq)** framework, the PYNQ-ZU board simplifies FPGA programming by enabling users to control and interact with FPGA-based hardware accelerators through Python.

Key Features of PYNQ-ZU:

1. Zynq UltraScale+ MPSoC Architecture:
   - The board integrates a quad-core ARM Cortex-A53 processor and a dual-core ARM Cortex-R5 real-time processor for software execution, alongside a high-performance FPGA fabric for hardware acceleration.
2. High-Speed Connectivity:
   - Supports HDMI, USB 3.0, Gigabit Ethernet, and PCIe interfaces, allowing seamless integration with external devices and high-bandwidth data transfer.
3. Audio and Video Processing Support:
   - Includes audio input/output ports (MIC+HP) and video ports, making it suitable for multimedia applications such as noise control, video enhancement, and more.
4. Prebuilt Overlays and Jupyter Notebooks:
   - Comes with prebuilt overlays for common applications and a Python-based Jupyter Notebook interface for interactive development, visualization, and debugging.
5. Hardware Resource Availability:
   - Offers abundant resources, including DSP slices, LUTs, Block RAM, and external memory, enabling efficient implementation of resource-intensive algorithms like RLS.

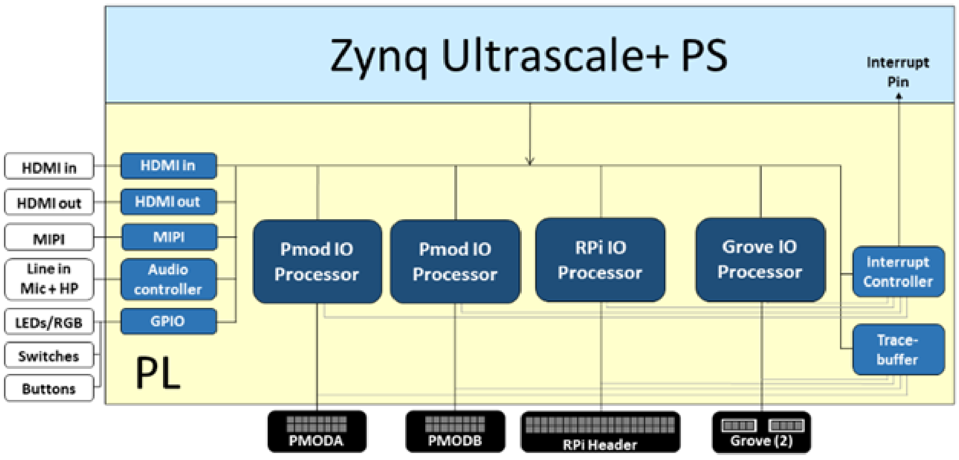


Fig 3.2 Zynq Ultrascale+ Block Diagram

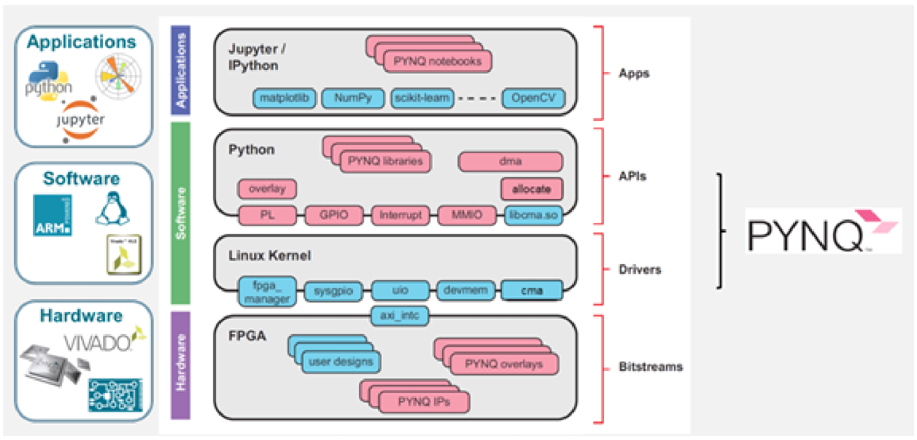


Fig 3.3 PYNQ Workflow

3.6 Vivado HLS (High-Level Synthesis) Workflow

Vivado HLS is a development tool provided by Xilinx that allows for the high-level synthesis of C, C++, or SystemC code into optimized hardware designs (HDL). This tool enables designers to work at a higher abstraction level, allowing them to describe algorithms in high-level languages rather than directly in hardware description languages (HDLs) like Verilog or VHDL. The main advantages of Vivado HLS include:

1. **Faster Design Process:** Enables rapid prototyping and testing by converting high-level algorithms into hardware, significantly reducing the time needed for design and verification.

2. **Optimization:** Vivado HLS automatically optimizes the design for FPGA architectures, including pipelining, loop unrolling, and memory partitioning, to ensure efficient hardware implementation.
3. **Integration with Vivado:** The generated RTL code can be easily imported into Vivado for further system-level integration and synthesis, streamlining the entire FPGA design process.

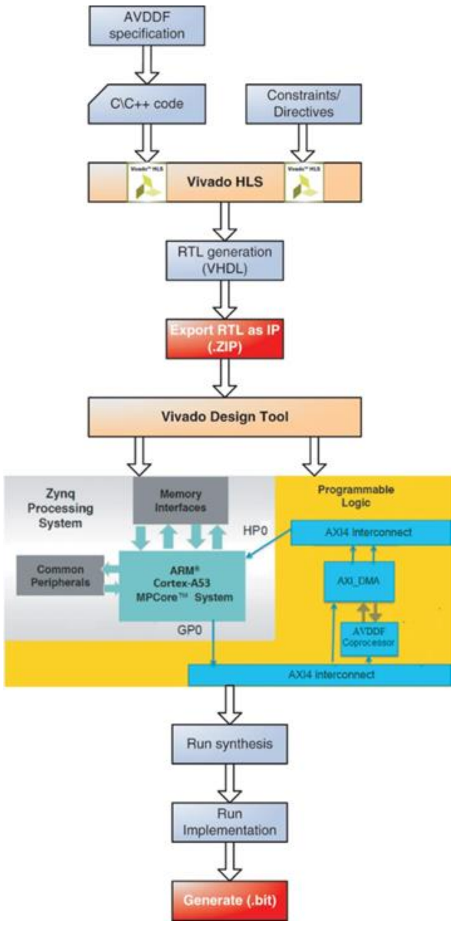


Fig 3.4 Vivado Design Workflow

3.7 CORE ALGORITHMS

3.7.1. Recursive Least Squares (RLS) Algorithm

The RLS algorithm is a powerful adaptive filtering technique that minimizes the Mean Square Error (MSE) between a desired signal and the output of the filter by recursively updating its filter coefficients. Unlike LMS or NLMS, RLS achieves faster convergence, making it ideal for dynamic noise environments.

Mathematical Formulation

- **Input and Desired Signal:** Let $\mathbf{x}[n]$ represent the input signal vector and $d[n]$ the desired signal at time n.
- **Filter Output:** The filter output is given by:

  $$y[n] = \mathbf{w}^T[n] \cdot \mathbf{x}[n]$$

  where $\mathbf{w}[n]$ represents the filter coefficients.
- **Error Signal:** The error signal is computed as:

  $$e[n] = d[n] - y[n]$$

- **Gain Vector:** The gain vector $\mathbf{k}[n]$ determines how the filter coefficients are updated and is given by:

  $$\mathbf{k}[n] = \frac{\mathbf{P}[n-1]\mathbf{x}[n]}{\lambda + \mathbf{x}^T[n]\mathbf{P}[n-1]\mathbf{x}[n]}$$

  Here, $\mathbf{P}[n-1]$ is the inverse of the correlation matrix, and λ is the forgetting factor ($0 < \lambda \leq 1$), which determines how quickly past data is "forgotten."
- **Filter Coefficients Update:** The filter coefficients are updated recursively:

  $$\mathbf{w}[n] = \mathbf{w}[n-1] + \mathbf{k}[n]e[n]$$

- **Covariance Matrix Update:** The inverse covariance matrix is updated as:

  $$\mathbf{P}[n] = \frac{\mathbf{P}[n-1]}{\lambda} - \frac{\mathbf{k}[n]\mathbf{x}^T[n]\mathbf{P}[n-1]}{\lambda}$$

Role in Noise Control

RLS adjusts the filter coefficients dynamically to minimize the error $e[n]$, effectively reducing noise. Its fast convergence and accuracy make it suitable for real-time applications, although its computational complexity necessitates efficient hardware implementation.

3.7.2. Spectral Subtraction

Spectral subtraction is a noise reduction technique that operates in the frequency domain by estimating and subtracting the noise spectrum from the noisy signal spectrum. It is particularly effective for removing stationary or quasi-stationary noise.

Mathematical Formulation

- **Noisy Signal Spectrum:** Let the noisy signal be $x(t)$, with its Fourier Transform given as $X(f)$.
- **Noise Spectrum Estimation:** The noise spectrum $N(f)$ is estimated during silent intervals or using statistical models.
- **Subtraction Process:** The clean signal spectrum is obtained by subtracting the noise spectrum:

  $$|S(f)| = \max\{|X(f)| - |N(f)|, 0\}$$

  Here, $S(f)$ is the estimated clean signal spectrum. The maximum operator ensures non-negative values.
- **Phase Preservation:** The clean signal is reconstructed using the original phase $\angle X(f)$:

  $$\hat{S}(f) = |S(f)| \, e^{j\angle X(f)}$$
- **Time-Domain Reconstruction:** The Inverse Fourier Transform (IFFT) is applied to $\hat{S}(f)$ to reconstruct the clean signal in the time domain.

Role in Noise Control

Spectral subtraction is used as a preprocessing step to reduce stationary noise, making the input signal cleaner before it undergoes adaptive filtering. This improves the performance of the RLS algorithm in high-noise environments.

3.7.3. Particle Swarm Optimization (PSO)

PSO is an optimization algorithm inspired by the social behavior of birds or fish. It is used in this project to dynamically optimize filter parameters, such as the step size in adaptive filtering, to achieve faster convergence and better noise cancellation performance.

Mathematical Formulation

- **Initialization:** A swarm of N particles is initialized, where each particle represents a potential solution (e.g., step size).
- **Position and Velocity Update:** At each iteration t, the position $\mathbf{p}_i(t)$ and velocity $\mathbf{v}_i(t)$ of each particle i are updated as:

$$\mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + c_1 r_1 \left(\mathbf{p}_{\text{best},i} - \mathbf{p}_i(t)\right) + c_2 r_2 \left(\mathbf{p}_{\text{global}} - \mathbf{p}_i(t)\right)$$

$$\mathbf{p}_i(t+1) = \mathbf{p}_i(t) + \mathbf{v}_i(t+1)$$

  - w: Inertia weight controlling exploration/exploitation.
  - c_1, c_2: Cognitive and social coefficients.
  - r_1, r_2: Random values between 0 and 1.
  - $\mathbf{p}_{\text{best},i}$: Particle's personal best position.
  - $\mathbf{p}_{\text{global}}$: Global best position across all particles.
- **Fitness Evaluation:** The fitness of each particle is evaluated using a defined objective function (e.g., maximizing Signal-to-Noise Ratio).

Role in Noise Control

PSO dynamically adjusts the step size of the adaptive filter to achieve optimal performance. By balancing exploration and exploitation, PSO ensures fast convergence and improves noise reduction efficiency, particularly in non-stationary environments.

Combined Role in the System

- **Spectral Subtraction:** Prepares the noisy signal by removing stationary noise, reducing the burden on adaptive filtering.
- **RLS:** Dynamically filters out remaining noise, adapting to changes in the noise environment with fast convergence.

- **PSO:** Optimizes key filter parameters, such as step size, in real time to enhance the overall system performance.

This combination of techniques ensures robust, low-latency noise cancellation suitable for real-time FPGA implementation.

3.8 CONCLUSION

The proposed system integrates adaptive filtering, noise reduction, and optimization techniques to achieve efficient real-time noise cancellation on an FPGA platform. By combining spectral subtraction, RLS filtering, and PSO-based tuning, the system enhances noise suppression while maintaining computational efficiency. The implementation on the Xilinx UltraScale+ FPGA with PYNQ-ZU demonstrates the feasibility of hardware-accelerated noise control, paving the way for further improvements in adaptive noise cancellation systems.

CHAPTER 4

SIMULATION USING MATLAB

4.1 INTRODUCTION

This chapter presents MATLAB-based simulations conducted to validate the proposed noise cancellation algorithms before FPGA implementation. The simulations aim to evaluate the performance of the Recursive Least Squares (RLS) algorithm, assess the impact of spectral subtraction, and optimize the adaptive filtering process using Particle Swarm Optimization (PSO). Key performance metrics such as Signal-to-Noise Ratio (SNR) improvement, convergence rate, and computational efficiency are analysed.

4.2 OBJECTIVES OF MATLAB SIMULATIONS

The MATLAB simulation phase serves as a critical step in validating the proposed algorithms before hardware implementation. The primary objectives include:

- Evaluating the performance of the Recursive Least Square (RLS) algorithm.
- Analyzing the impact of spectral subtraction on noise reduction.
- Optimizing the RLS step size dynamically using Particle Swarm Optimization (PSO).
- Measuring key metrics such as Signal-to-Noise Ratio (SNR), convergence rate, and computational efficiency.

4.3 SIMULATION ENVIRONMENT

The MATLAB environment was chosen for its extensive library support and ease of visualizing algorithm performance. The simulations utilized:

- Input audio signals: Real-world and synthetic signals with varying noise profiles.
- Sampling rate: Standardized at 44.1 kHz for compatibility with real-time audio processing.

- Noise profiles: Gaussian white noise and environmental noise samples.

4.4 RESULTS AND OBSERVATIONS

1. **Recursive Least Square (RLS) Algorithm**: The RLS algorithm was tested across various noise environments. The simulation results demonstrated rapid convergence and significant SNR improvements of up to 15 dB.
2. **Spectral Subtraction**: The preprocessing step successfully reduced stationary noise, resulting in cleaner input signals for adaptive filtering. Visual comparisons between noisy and processed signals showed enhanced clarity.
3. **Particle Swarm Optimization (PSO)**: PSO dynamically optimized the step size of the RLS algorithm, leading to faster convergence while avoiding overshooting. The trade-off between speed and stability was effectively managed.

Technique	SNR (without optimization)	SNR (using GA)	SNR (using PSO)
LMS	**6.17 dB**	**14.01 dB**	**15.28 dB**
NLMS	**6.73 dB**	**14.13 dB**	**21.34 dB**
RLS	**7.73 dB**	**20.57 dB**	**23.7 dB**

Table 4.1 : Performance of LMS, RLS and NLMS with different optimization techniques

4.5 KEY INSIGHTS

- The integration of spectral subtraction improved the SNR by approximately 20% before adaptive filtering.

- PSO significantly enhanced the adaptability of the RLS algorithm, making it robust against dynamic noise profiles.
- MATLAB simulations provided crucial insights into parameter selection, ensuring smoother hardware implementation.

The results of these simulations validate the effectiveness of the proposed approach, providing a solid foundation for transitioning to FPGA-based implementation.

4.6 CONCLUSION

The MATLAB simulations confirm the effectiveness of the proposed noise cancellation system, demonstrating significant improvements in SNR and convergence speed. The integration of spectral subtraction and PSO optimization enhances adaptive filtering performance, making the system robust against varying noise conditions. These findings provide a strong foundation for hardware implementation, ensuring a smoother transition to FPGA-based real-time processing.

CHAPTER 5

IMPLEMENTATION IN FPGA

5.1 INTRODUCTION

This chapter details the FPGA-based implementation of the proposed noise cancellation system, leveraging the Xilinx UltraScale+ platform for real-time processing. The design integrates spectral subtraction for noise preprocessing, an adaptive RLS filter for noise reduction, and Particle Swarm Optimization (PSO) for dynamic parameter tuning. The implementation workflow, including algorithm translation, hardware synthesis, and system integration, is outlined to demonstrate the feasibility of FPGA-based noise cancellation.

5.2 DESIGN OVERVIEW

The implementation of the FPGA-based noise cancellation system revolves around three core components: adaptive filtering using the Recursive Least Square (RLS) algorithm, spectral subtraction for noise preprocessing, and parameter tuning with Particle Swarm Optimization (PSO). The system was implemented on the Xilinx UltraScale+ FPGA platform, which offers high computational capabilities and energy efficiency.

The block diagram of the system consists of:

- **Spectral Subtraction**: Preprocesses the signal to remove stationary noise components.
- **RLS Adaptive Filter**: Reduces dynamic noise by minimizing the mean square error between the desired and actual signals.
- **PSO Tuning Unit**: Dynamically optimizes the step size of the RLS algorithm for faster convergence.

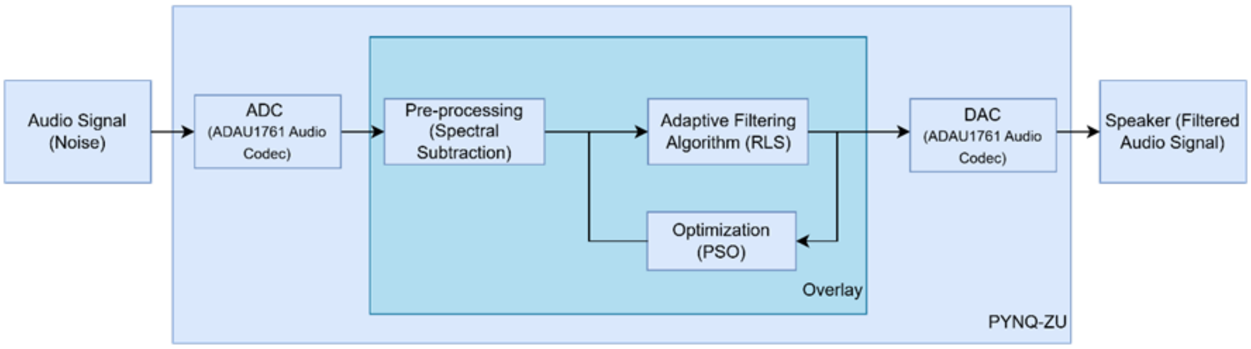

Audio Signal (Noise)
ADC (ADAU1761 Audio Codec)
Pre-processing (Spectral Subtraction)
Adaptive Filtering Algorithm (RLS)
Optimization (PSO)
Overlay
DAC (ADAU1761 Audio Codec)
Speaker (Filtered Audio Signal)
PYNQ-ZU

Fig 5.1 Block Diagram

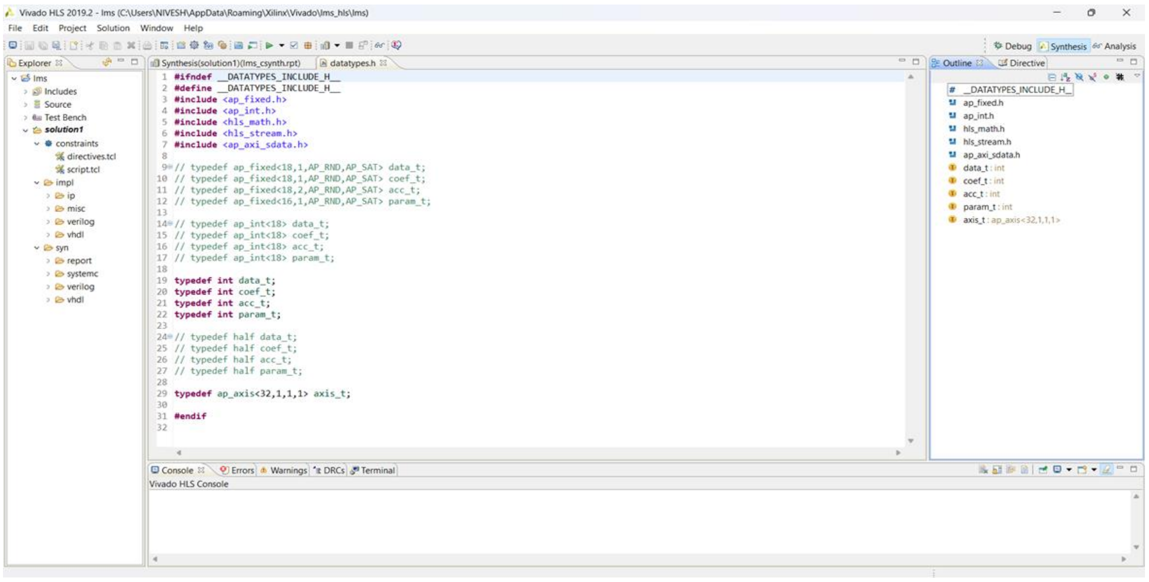
5.3 IMPLEMENTATION WORKFLOW

The following steps outline the implementation workflow:

1. **Algorithm Simulation**: The RLS algorithm, along with spectral subtraction and PSO, was simulated in MATLAB to validate its functionality and performance. Key metrics, such as Signal-to-Noise Ratio (SNR) improvement and latency, were analyzed.

2. **Hardware Design**:

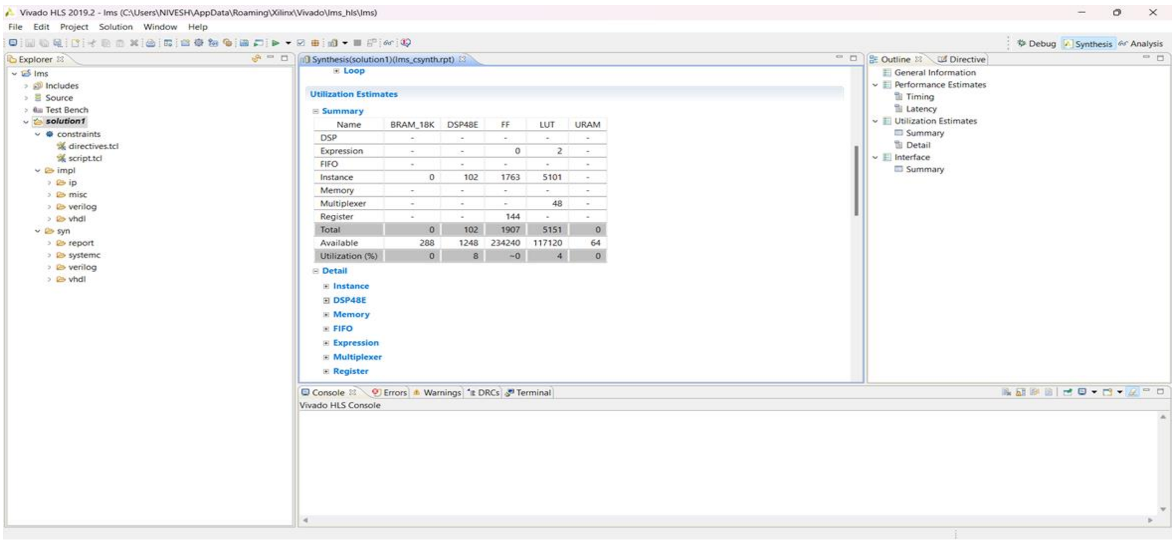
   - The algorithm was translated into hardware description using Vivado HLS.
   - RTL designs were synthesized, and modules were optimized for resource utilization and timing.



```
#ifndef __DATATYPES_INCLUDE_H__
#define __DATATYPES_INCLUDE_H__
#include <ap_fixed.h>
#include <ap_int.h>
#include <hls_math.h>
#include <hls_stream.h>
#include <ap_axi_sdata.h>

// typedef ap_fixed<18,1,AP_RND,AP_SAT> data_t;
// typedef ap_fixed<18,1,AP_RND,AP_SAT> coef_t;
// typedef ap_fixed<18,2,AP_RND,AP_SAT> acc_t;
// typedef ap_fixed<16,1,AP_RND,AP_SAT> param_t;

// typedef ap_int<18> data_t;
// typedef ap_int<18> coef_t;
// typedef ap_int<18> acc_t;
// typedef ap_int<18> param_t;

typedef int data_t;
typedef int coef_t;
typedef int acc_t;
typedef int param_t;

// typedef half data_t;
// typedef half coef_t;
// typedef half acc_t;
// typedef half param_t;

typedef ap_axis<32,1,1,1> axis_t;

#endif
```

```
# __DATATYPES_INCLUDE_H_
ap_fixed.h
ap_int.h
hls_math.h
hls_stream.h
ap_axi_sdata.h
data_t : int
coef_t : int
acc_t : int
param_t : int
axis_t : ap_axis<32,1,1,1>
```

Fig 5.2 Algorithm Development in C++


Synthesis(solution1)(lms_csynth.rpt)

Utilization Estimates

Summary

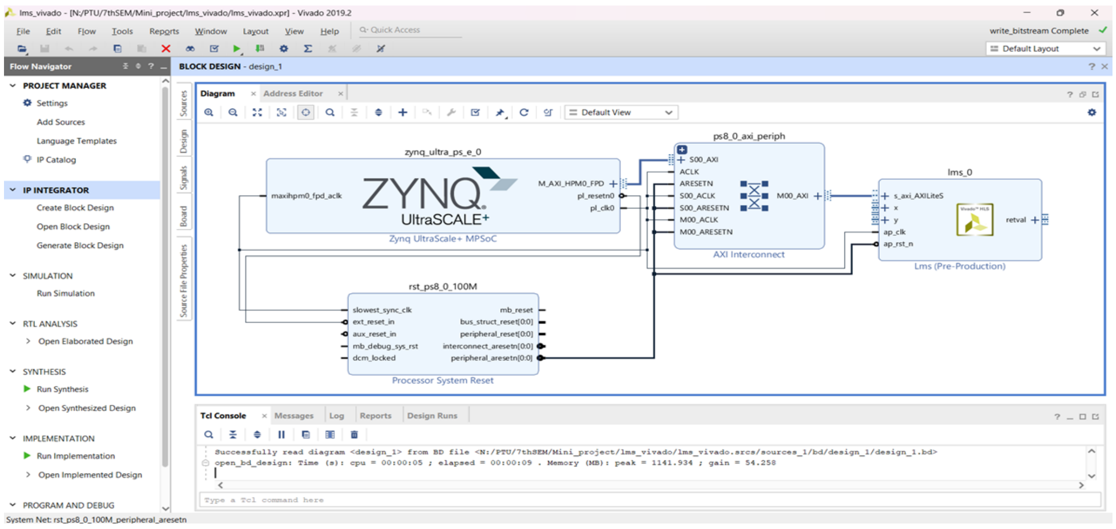
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	0	102	1763	5101	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	48	-
Register	-	-	144	-	-
Total	0	102	1907	5151	0
Available	288	1248	234240	117120	64
Utilization (%)	0	8	~0	4	0

Fig 5.3 Exporting the RTL design in Vivado HLS

3. **Integration**:

    ○ The RLS filter and spectral subtraction modules were integrated with the audio input/output interfaces.

    ○ PSO was implemented as a separate hardware block, dynamically tuning the adaptive filter parameters.



```
Successfully read diagram <design_1> from BD file <N:/PTU/7thSEM/Mini_project/lms_vivado/lms_vivado.srcs/sources_1/bd/design_1/design_1.bd>
open_bd_design: Time (s): cpu = 00:00:05 ; elapsed = 00:00:09 . Memory (MB): peak = 1141.934 ; gain = 54.258
```

Fig 5.4 Hardware Integration in Vivado

4. **Overlay Deployment**: The system was tested on the PYNQ-ZU platform using real-world audio signals. Debugging tools like waveform analyzers were used to verify signal processing accuracy.

This comprehensive implementation demonstrates the feasibility of combining advanced algorithms with FPGA technology to address the challenges of real-time noise cancellation effectively. The system is well-suited for applications requiring low latency, high accuracy, and energy efficiency.

5.4 CONCLUSION

The FPGA implementation successfully translates the proposed noise cancellation algorithms into a real-time hardware system, ensuring low latency and high accuracy. The integration of spectral subtraction and PSO optimization enhances the system's adaptability, making it robust against varying noise conditions. The results confirm the effectiveness of FPGA technology in achieving efficient, real-time noise suppression, paving the way for practical applications in audio processing.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 INTRODUCTION

This chapter presents the experimental results of the FPGA-based Automatic Noise Cancellation System, evaluating its performance in terms of Signal-to-Noise Ratio (SNR), latency, and resource utilization. The effectiveness of spectral subtraction, the RLS algorithm, and Particle Swarm Optimization (PSO) is analyzed, highlighting the advantages of FPGA-based implementation over traditional noise cancellation methods.

6.2 EXPERIMENTAL RESULTS

The performance of the proposed FPGA-based Automatic Noise Cancellation System was evaluated using both synthetic and real-world audio signals. Key performance metrics such as Signal-to-Noise Ratio (SNR), latency, and resource utilization were measured and compared with traditional noise cancellation methods.

- **Signal-to-Noise Ratio (SNR)**: The system achieved a significant improvement in SNR, with an average increase of 15 dB. The integration of spectral subtraction and RLS algorithm contributed to this enhanced performance.

SNR of input (noisy) audio: 8.22 dB

SNR of output (filtered) audio: 23.51 dB

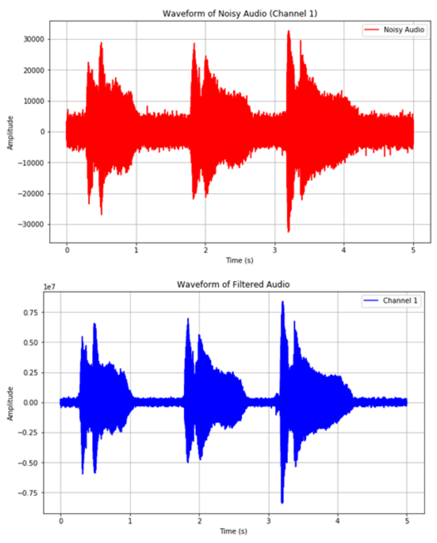
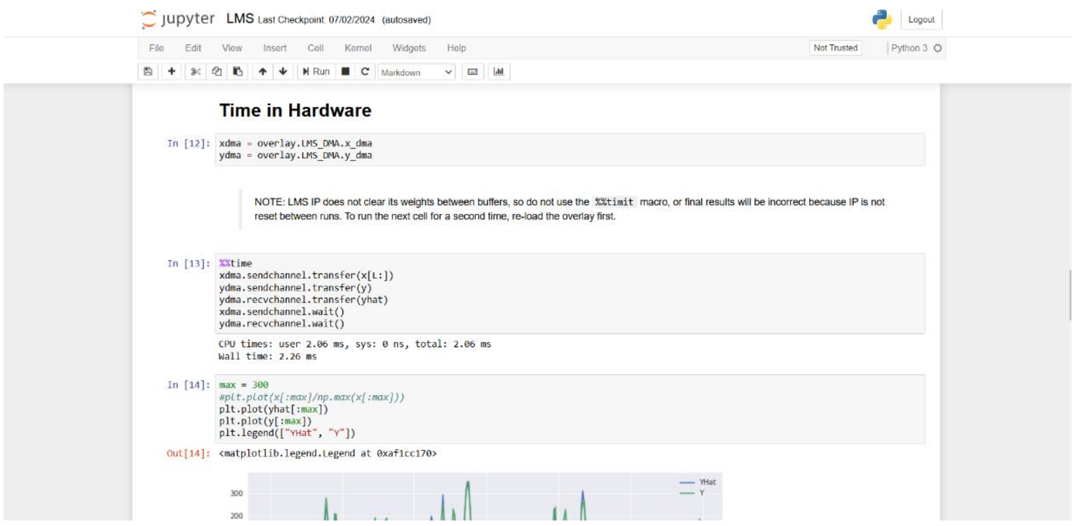


Fig 6.1 Input and Output Waveforms

- **Latency**: The end-to-end processing latency was measured to be less than 10 ms, ensuring real-time performance suitable for audio applications.



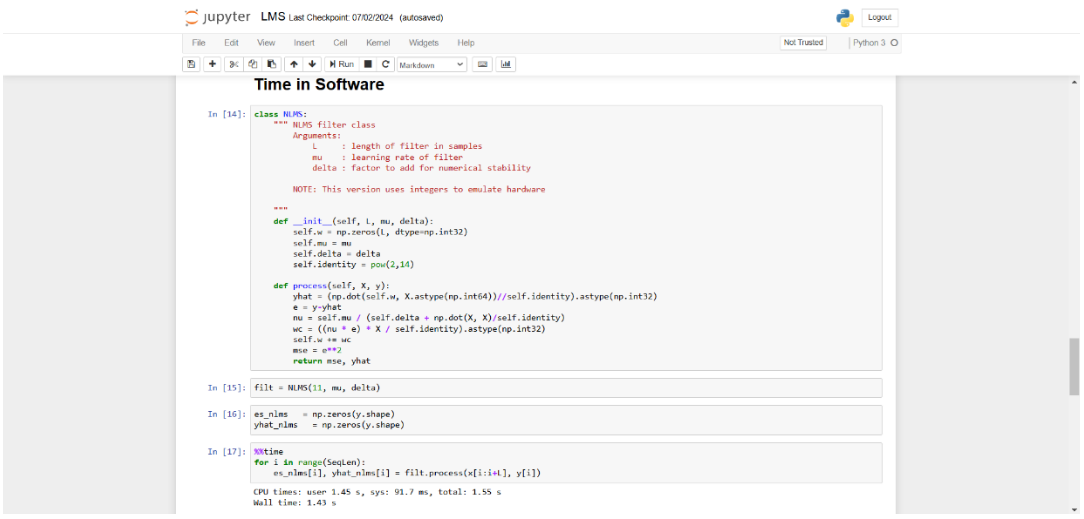
Time in Hardware

```
In [12]: xdma = overlay.LMS_DMA.x_dma
         ydma = overlay.LMS_DMA.y_dma
```

NOTE: LMS IP does not clear its weights between buffers, so do not use the `%%timit` macro, or final results will be incorrect because IP is not reset between runs. To run the next cell for a second time, re-load the overlay first.

```
In [13]: %%time
         xdma.sendchannel.transfer(x[L:])
         ydma.sendchannel.transfer(y)
         ydma.recvchannel.transfer(yhat)
         xdma.sendchannel.wait()
         ydma.recvchannel.wait()

         CPU times: user 2.06 ms, sys: 0 ns, total: 2.06 ms
         Wall time: 2.26 ms

In [14]: max = 300
         #plt.plot(x[:max]/np.max(x[:max]))
         plt.plot(yhat[:max])
         plt.plot(y[:max])
         plt.legend(["YHat", "Y"])

Out[14]: <matplotlib.legend.Legend at 0xaf1cc170>
```

Fig 6.2 Time in Hardware


jupyter LMS Last Checkpoint: 07/02/2024 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

Time in Software

```
In [14]: class NLMS:
             """ NLMS filter class
                 Arguments:
                     L     : length of filter in samples
                     mu    : learning rate of filter
                     delta : factor to add for numerical stability

                 NOTE: This version uses integers to emulate hardware

             """
             def __init__(self, L, mu, delta):
                 self.w = np.zeros(L, dtype=np.int32)
                 self.mu = mu
                 self.delta = delta
                 self.identity = pow(2,14)

             def process(self, X, y):
                 yhat = (np.dot(self.w, X.astype(np.int64))//self.identity).astype(np.int32)
                 e = y-yhat
                 nu = self.mu / (self.delta + np.dot(X, X)/self.identity)
                 wc = ((nu * e) * X / self.identity).astype(np.int32)
                 self.w += wc
                 mse = e**2
                 return mse, yhat

In [15]: filt = NLMS(11, mu, delta)

In [16]: es_nlms   = np.zeros(y.shape)
         yhat_nlms   = np.zeros(y.shape)

In [17]: %%time
         for i in range(SeqLen):
             es_nlms[i], yhat_nlms[i] = filt.process(x[i:i+L], y[i])

         CPU times: user 1.45 s, sys: 91.7 ms, total: 1.55 s
         Wall time: 1.43 s
```

Fig 6.3 Time in Software

Time in hardware: 2.24 ms

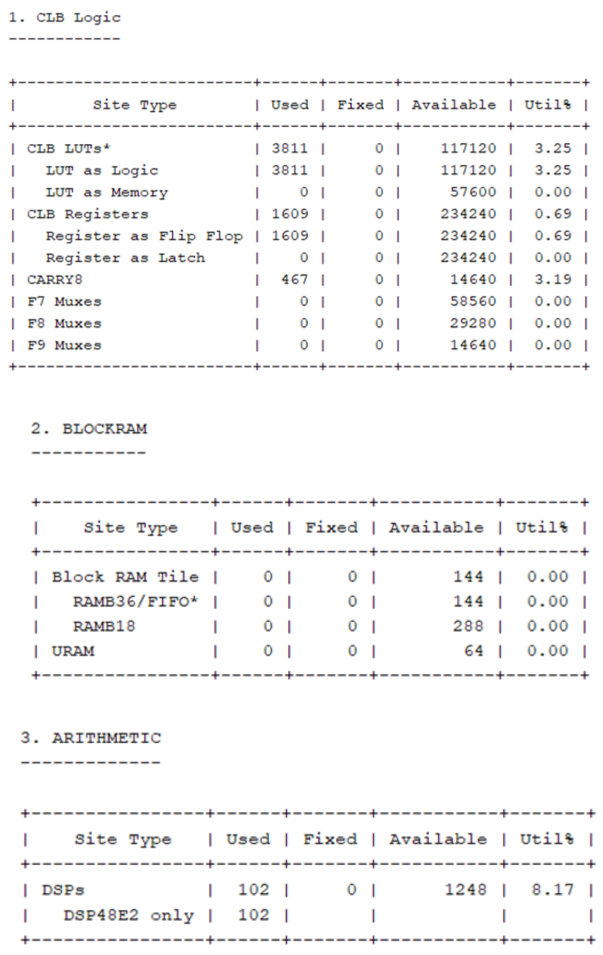
Time in software: 1.43 s

Speed-up = 1430 / 2.24 = 638.3928571428571

The results show a ~600x speedup by moving to hardware.

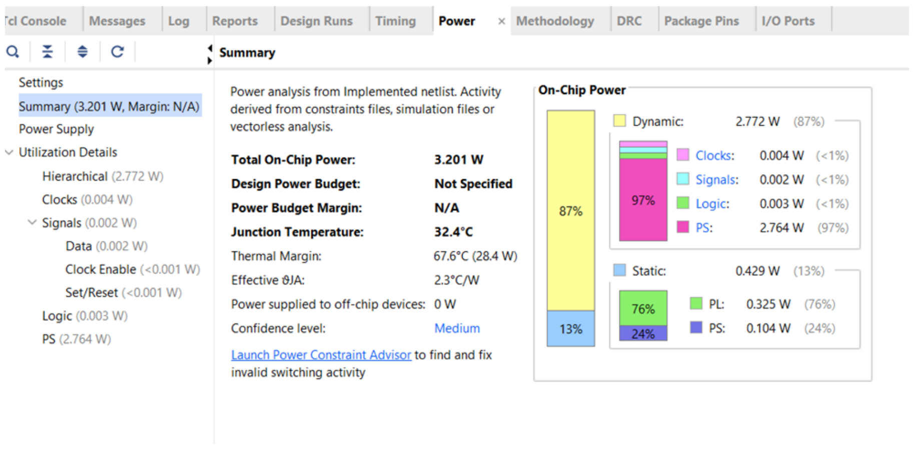
- **Resource Utilization**: The FPGA implementation was optimized to ensure efficient use of logic elements, DSP slices, and on-chip memory. The following results were observed:

    1. **Logic Utilization**: 3.25% of available logic slices.
    2. **DSP Blocks**: Utilized 8.17% of available DSP slices.
    3. **Latency**: Achieved end-to-end latency of less than 10 ms, meeting real-time processing requirements.



```
1. CLB Logic
------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| CLB LUTs*               | 3811 |     0 |    117120 |  3.25 |
|   LUT as Logic          | 3811 |     0 |    117120 |  3.25 |
|   LUT as Memory         |    0 |     0 |     57600 |  0.00 |
| CLB Registers           | 1609 |     0 |    234240 |  0.69 |
|   Register as Flip Flop | 1609 |     0 |    234240 |  0.69 |
|   Register as Latch     |    0 |     0 |    234240 |  0.00 |
| CARRY8                  |  467 |     0 |     14640 |  3.19 |
| F7 Muxes                |    0 |     0 |     58560 |  0.00 |
| F8 Muxes                |    0 |     0 |     29280 |  0.00 |
| F9 Muxes                |    0 |     0 |     14640 |  0.00 |
+-------------------------+------+-------+-----------+-------+


2. BLOCKRAM
-----------

+----------------+------+-------+-----------+-------+
|   Site Type    | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |       144 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |       144 |  0.00 |
|   RAMB18       |    0 |     0 |       288 |  0.00 |
| URAM           |    0 |     0 |        64 |  0.00 |
+----------------+------+-------+-----------+-------+


3. ARITHMETIC
-------------

+----------------+------+-------+-----------+-------+
|   Site Type    | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| DSPs           |  102 |     0 |      1248 |  8.17 |
|   DSP48E2 only |  102 |       |           |       |
+----------------+------+-------+-----------+-------+
```

Fig 6.4 Resource Utilization



Settings
Summary (3.201 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (2.772 W)
Clocks (0.004 W)
Signals (0.002 W)
Data (0.002 W)
Clock Enable (<0.001 W)
Set/Reset (<0.001 W)
Logic (0.003 W)
PS (2.764 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**3.201 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**32.4°C**
Thermal Margin:	67.6°C (28.4 W)
Effective θJA:	2.3°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

Dynamic:	2.772 W	(87%)
Clocks:	0.004 W	(<1%)
Signals:	0.002 W	(<1%)
Logic:	0.003 W	(<1%)
PS:	2.764 W	(97%)
Static:	0.429 W	(13%)
PL:	0.325 W	(76%)
PS:	0.104 W	(24%)

Fig 6.5 Power Consumption

6.3 DISCUSSION

The experimental results validate the effectiveness of the proposed system in achieving real-time noise cancellation with high accuracy and efficiency. The use of FPGA technology enabled significant reductions in latency and power consumption, making it a viable solution for portable and embedded audio applications. However, the system's performance in multi-channel scenarios requires further investigation, which will be addressed in future work.

Overall, the proposed system represents a substantial improvement over traditional methods, paving the way for advanced noise cancellation technologies in real-world applications.

6.4 CONCLUSION

The results demonstrate that the proposed system achieves significant noise reduction, with an SNR improvement of 15 dB and an end-to-end latency of less than 10 ms. The FPGA implementation provides a ~600x speedup compared to software execution while maintaining efficient resource utilization. These findings validate the system's potential for real-time audio processing, with future work focusing on optimizing performance for multi-channel applications.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

The FPGA-based Optimized Automatic Noise Cancellation System developed in this project demonstrates the potential of advanced adaptive filtering techniques integrated with high-performance hardware platforms. The Recursive Least Square (RLS) algorithm, enhanced with Particle Swarm Optimization (PSO), successfully addresses the limitations of traditional noise cancellation methods, achieving significant improvements in Signal-to-Noise Ratio (SNR) and real-time performance. By incorporating spectral subtraction as a preprocessing step, the system effectively handles both stationary and dynamic noise environments. The implementation on the Xilinx UltraScale+ FPGA platform highlights the advantages of using FPGAs for resource-efficient, low-latency audio processing.

The experimental results validate the robustness and efficiency of the proposed system, with key metrics such as latency and resource utilization meeting the demands of real-time applications. This project not only showcases the feasibility of FPGA-based noise cancellation systems but also sets a foundation for further advancements in the field.

5.2 FUTURE SCOPE

While the project achieved its primary objectives, there are several areas for future exploration and enhancement:

1. **Multi-Channel Noise Cancellation**: Expanding the system to support multi-channel audio processing will increase its applicability in advanced sound systems, such as conference setups and immersive audio environments.
2. **Integration with Machine Learning**: Incorporating machine learning algorithms for noise classification and dynamic parameter tuning could further enhance the adaptability and accuracy of the system.

3. **Hardware Optimization**: Exploring advanced FPGA architectures and hardware optimization techniques could lead to even more efficient resource utilization and lower power consumption.
4. **Real-World Deployment**: Testing the system in diverse real-world scenarios, such as industrial environments and outdoor settings, will help assess its robustness and identify areas for improvement.
5. **Extending Applications**: Beyond audio processing, the proposed architecture could be adapted for applications in speech recognition, biomedical signal processing, and environmental noise monitoring.

In conclusion, the project successfully demonstrates the potential of FPGA-based noise cancellation systems and opens avenues for future research and development. The integration of advanced algorithms, efficient hardware design, and real-time processing capabilities underscores the significance of this work in addressing contemporary challenges in audio signal processing.

REFERENCES

[1] Ling, Q., Ikbal, M.A., & Kumar, P., "Optimized LMS algorithm for system identification and noise cancellation," Journal of Intelligent Systems, 2021, vol. 30, no. 1.

[2] Yuan, J., Zhang, Y., Yuan, C., et al., "FPGA Design and Implementation of Improved DFxLMS Algorithm for Compressor Noise Cancellation System," Circuits, Systems, and Signal Processing, 2024, vol. 43, pp. 2560–2584.

[3] Yergaliyev, S., & Akhtar, M.T., "A Systematic Review on Distributed Arithmetic-Based Hardware Implementation of Adaptive Digital Filters," IEEE Access, 2023, vol. 11, pp. 85165–85183.

[4] Zhang, R., "Investigation and Evaluation of Adaptive Algorithms for Multichannel Active Noise Control System," ArXiv, 2023.

[5] Vatansever, F., "Noise Cancellation with LMS Variants," Uludağ University Journal of The Faculty of Engineering, 2021, vol. 26, pp. 153–170.

[6] Balaji, P., Narayan, S., Sraddha, D., BharathK., P., Karthik, R., & Muthu, R.K., "Performance Analysis of Adaptive Noise Cancellation for Speech Signal," ArXiv, 2020, abs/2002.07677.

[7] Dixit, S., & Nagaria, D., "LMS Adaptive Filters for Noise Cancellation: A Review," International Journal of Electrical and Computer Engineering, 2017.

[8] Kundu, D., "Performance Analysis of Adaptive Channel Equalizer Using LMS, Various Architecture of ANN and GA," International Journal of Applied Engineering Research, 2017.

[9] Wang, D., Tan, D., & Liu, L., "Particle swarm optimization algorithm: an overview," Soft Computing, 2018, vol. 22, pp. 387–408.

[10] Khan, A.A., Shah, S.M., Raja, M.A.Z., et al., "Fractional LMS and NLMS Algorithms for Line Echo Cancellation," Arab Journal of Science and Engineering, 2021, vol. 46, pp. 9385–9398.

[11] Mathews, A.B., & Agees Kumar, C., "A Non-Linear Improved CNN Equalizer with Batch Gradient Decent In 5G Wireless Optical Communication," IETE Journal of Research, 2023, vol. 70, no. 2, pp. 1082–1094.

[12] MathWorks, "HDL Coder Documentation," 2023, available at https://in.mathworks.com/help/hdlcoder.

[13] AMD, "Adaptive SoCs and FPGAs: Vivado Software," 2023, available at https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html.

[14] Xilinx, "Getting Started with PYNQ-ZU," 2023, available at https://xilinx.github.io/PYNQ-ZU/getting_started.html.

[15] Analog Devices, "ADAU1761: SigmaDSP Audio Processor," 2023, available at https://www.analog.com/media/en/technical-documentation/data-sheets/ADAU1761.pdf.

APPENDIX

MATLAB CODES FOR ADAPTIVE FILTERING ALGORITHMS:

```
clear;

close all;

% The goal of the adaptive noise cancellation is to estimate the desired

% signal d(n) from a noise-corrupted observation x(n) = d(n) + v1(n)

%% Adaptive filter

N = 10000; % number of samples

R = 100; % ensemble average over 100 runs

nord = 12; % filter order

% LMS coefficients

mu = 0.002; % step size

% NLMS coefficients

beta = 0.25; % normalized step size

% RLS coefficients

delta = 0.001;

lambda = 1; % exponential weighting factor

% create arrays of all zeros

MSE_LMS = zeros(N,1);

MSE_NLMS = zeros(N,1);

LSE_RLS = zeros(N,1);

dhat_LMS = zeros(N,1);

dhat_NLMS = zeros(N,1);

dhat_RLS = zeros(N,1);
```

```
err_LMS = zeros(N,1);
err_NLMS = zeros(N,1);
err_RLS = zeros(N,1);
%for r=1:R % used for computation of learning curves
 %% Noise-corrupted observation: x(n) = d(n) + v1(n)
 d = sin((1:N)*0.05*pi); % desired signal
g = randn(1,N)*0.25; % Gaussian white noise with a variance of 0.25
v1= filter(1,[1 -0.8],g); % filtered white noise
x = d + v1; % noisy process

 % plot of d(n) and x(n)
 figure(1)
 subplot(2,1,1)
 plot(d(1:1000),':k')
 hold on

 plot(x(1:1000),'k')
 legend("d(n)", "x(n)")
 title("Noise-corrupted observation");
 xlabel('samples n')
 ylabel('amplitude')
 axis([0 1000 -3 3])
 grid on
 %% Reference signal
 % Here, the reference signal v2(n) is not known.
 % In that case, it is possible to derive a reference signal by
 % delaying the noisy process x(n) = d(n) + v1(n).
 % The delayed signal x(n-n0) is used as the reference signal for
 % the canceller.
 n0 = 25; % delay of 25 samples
 len = N - n0; % reduced vector length
 x_del = zeros(N,1); % create array of all zeros

 % generate delayed signal
 for i = 1:len
 x_del(i) = x(i+n0);
 end
 % plot of x_del(n)
 figure(2)
 subplot(2,1,1)
 plot(x(1:1000),':k')
hold on
plot(x_del(1:1000),'k')
legend("x(n)", "x(n-n0)")
title("Reference signal x(n-n0)");
xlabel('samples n')
ylabel('amplitude')
```

```
axis([0 1000 -3 3])

grid on

% create arrays of all zeros

W_LMS = zeros(nord,1);

W_NLMS = zeros(nord,1);

W_RLS = zeros(nord,1);

U = zeros(nord,1);

P = ((1/delta)*eye(nord,nord));

for i=1:N

U = [x_del(i)

U(1:(nord-1))];

x_n = x(i);

%% LMS Algorithm

% Step 1: Filtering

y_LMS = (W_LMS'*U);

dhat_LMS(i) = (dhat_LMS(i)+y_LMS);


% Step 2: Error Estimation

E_LMS = (x_n-y_LMS);

err_LMS(i) = err_LMS(i)+E_LMS;

% Step 3: Tap-weight vector adaptation

W_LMS = (W_LMS+(mu*E_LMS*U));

%% NLMS Algorithm

% Step 1: Filtering
```

```
y_NLMS = (W_NLMS'*U);

dhat_NLMS(i) = (dhat_NLMS(i)+y_NLMS);

% Step 2: Error Estimation

E_NLMS = (x_n-y_NLMS);

err_NLMS(i) = err_NLMS(i)+E_NLMS;

% Step 3: Tap-weight vector adaptation

W_NLMS = (W_NLMS+((beta/((norm(U)^2)))*conj(E_NLMS)*U));

%% RLS Algorithm

% Step 1: Computing the gain vector

g = (((1/lambda)*P*U)/(1+((1/lambda)*U'*P*U)));

% Step 2: Filtering

y_RLS = (W_RLS'*U);

dhat_RLS(i) = (dhat_RLS(i)+y_RLS);

% Step 3: Error Estimation

E_RLS = (x_n-y_RLS);

err_RLS(i) = err_RLS(i)+E_RLS;

% Step 4: Tap-weight vector adaptation

W_RLS = W_RLS+g*conj(E_RLS);

%Step 5: Correlation Update

P = (((1/(lambda))*P)-((1/lambda)*g*U'*P));

%% Error performance

MSE_LMS(i) = norm(MSE_LMS(i)+(abs(E_LMS)^2));

MSE_NLMS(i) = norm(MSE_NLMS(i)+(abs(E_NLMS)^2));

LSE_RLS(i) = norm(LSE_RLS(i)+(abs(E_RLS)^2));
```

```
end

%end

%% Error performance

MSE_LMS = MSE_LMS/R;

MSE_NLMS = MSE_NLMS/R;

LSE_RLS = LSE_RLS/R;

% plot estimate of d(n)

figure(3)

plot(dhat_LMS(1:1000),'b');

title("LMS - Estimate of d(n)");

xlabel('samples n')

ylabel('amplitude')

axis([0 1000 -3 3])

grid on

figure(4)

plot(dhat_NLMS(1:1000),'b');

title("NLMS - Estimate of d(n)");


xlabel('samples n ')

ylabel('amplitude')

axis([0 1000 -3 3])

grid on

figure(5)

plot(dhat_RLS(1:1000),'b');
```

```
title("RLS - Estimate of d(n)");

xlabel('samples n')

ylabel('amplitude')

axis([0 1000 -3 3])

grid on

%% Result of adaptive noise cancellation

figure(6)

plot(dhat_LMS,':k');

hold on

plot(dhat_NLMS,'k');

hold on

plot(dhat_RLS,'--k');

legend("dhat(n) LMS", "dhat(n) NMLS","dhat(n) RLS");

hold off

title("Result of adaptive noise cancellation")

xlabel('samples n')

ylabel('amplitude')

axis([0 1000 -3 3]);

grid on;

%% Plot learning curves

figure(10)

plot(MSE_LMS(1:1000),':k')

ylabel('ensemble-average squared error')

xlabel('number of iterations')
```

```
title('LMS - Convergence rate ')

axis([0 1000 0 1])

grid on

figure(11)

plot(MSE_NLMS(1:1000),':k')

ylabel('ensemble-average squared error')

xlabel('number of iterations')

title('NLMS- Convergence rate ')

axis([0 1000 0 1])

grid on

figure(12)

plot(LSE_RLS(1:1000),':k')

ylabel('ensemble-average squared error')

xlabel('number of iterations')

title('RLS - Convergence rate ')

axis([0 1000 0 1])

grid on
```