FPGA-Based LMS Adaptive Filter Design Report

VLSI Design

Table of Content

I. INTRODUCTION

1. Scope and Goals

This report presents a complete development flow for an adaptive filtering system, beginning with algorithm-level exploration and ending with hardware demonstration on an FPGA platform. The main objective is to construct a functional LMS-based noise-reduction system that can operate in real time while maintaining a clear correspondence between theory, simulation, RTL design, and physical implementation. The design effort focuses on translating algorithmic blocks into synthesizable architectures, validating correctness through structured verification, and confirming behaviour on actual hardware interfaces.

2. Overview

The work begins with the LMS adaptive filter, including signal modelling and MATLAB simulation to observe convergence and noise-suppression behaviour. The study then transitions to RTL design, detailing the FIR computation path and LMS update logic. Verification is performed using a dedicated testbench to ensure functional correctness before synthesis. The hardware phase targets an FPGA platform and incorporates several system-level elements: integration with the XADC wizard for on-board analog-to-digital conversion, creation of a custom IP core, IP-level verification, and output of filtered data through an SPI interface. Final oscilloscope measurements provide external confirmation of system performance and demonstrate the end-to-end operation of the adaptive filter in a real environment.

II. FILTER DESIGN

1. LMS Adaptive Filter

The LMS adaptive filter is a common algorithm used to iteratively adjust the coefficients of a linear filter to minimize a cost function, typically the mean square of the error signal. This algorithm is favored due to its relative simplicity and robustness in real-world applications.

It modifies the standard Finite Impulse Response (FIR) filter by introducing a mechanism for weight updates. A standard FIR filter operates with fixed, predetermined coefficients, w(n), where the output y(n) is the convolution of the input signal x(n) and the filter coefficients:

$$y[n] = w[n] * x[n]$$

The LMS filter retains this basic structure but uses an iterative process to update the weight vector w(n) at each time step n. The update rule is based on the instantaneous error e(n), which is the difference between the desired signal d(n) and the filter output y(n):

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

The weight vector is adjusted using the steepest descent method, which simplifies to the following expression:

$$w[n+1] = w[n] + \mu \cdot x[n] \cdot e[n]$$

In this equation, w(n) is the current coefficient vector, x(n) is the input signal vector, and μ is the step-size parameter. The role of the step-size μ is to control the stability and convergence rate of the adaptation.

2. LMS Filter in Active Noise Canceling (ANC)

In active noise canceling we aim to remove the noise by producing a reversed version of it. However since we cannot perfectly sample the noise, we need to use a filter that can estimate the real noise from correlated samples. There are 2 inputs to the system:

d: mixed input signal of the clean signal (**s**) and the noise (**n**): d = s + n

x(n'); sampled signal from the noise source which is correlated but often not equal to the original noise **n** in the input signal.

Assume that signals **s**, **n**(primary noise), **n'**(reference/auxiliary noise) and filter output **y** are zero-mean, wide-sense stationary; **s** is uncorrelated with **n** and **n'**. Under these conditions the ANC objective can be expressed as minimization of the output power measured by an error sensor.

The error (output) can be defined as:

$$e = d - y = s + n - y$$

Squaring and taking expectations gives:

$$E[e^2] = E[s^2] + E[(n - y)^2] + E[s(n - y)^2]$$

Because **s** is uncorrelated with **n** and with **y** (the latter holds when **y** is generated from the noise reference **n'**), the cross term vanishes and

$$E[e^2] = E[s^2] + E[(n - y)^2]$$

Therefore, minimizing $E[e^2]$ (which is the goal of LMS algorithm) becomes minimizing $E[(n-y)^2]$ ($E[s^2]$ is constant).

At perfect estimation, the residual contains only the desired signal **s** and the primary noise has been completely canceled:

$$e \approx s$$

If the primary noise **n** is correlated with the reference **n'**, the adaptive filter can then exploit that correlation to form **n**.

The system is shown in Figure 1.

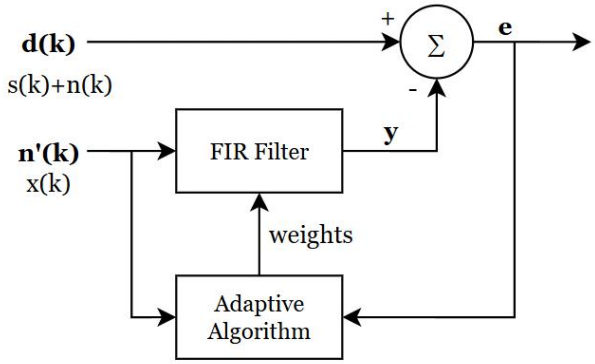

Figure 1: Adaptive Noise Filter Diagram

3. MATLAB Modeling

(1) Generate Signals

The desired signal s(n) is chosen as a sinusoid, which provides a steady narrow-band component unaffected by the adaptive process. The primary noise n(n) is generated as Gaussian white noise. A correlated reference noise n'(n) is produced by filtering an independent noise sequence through a short FIR system:

b = [0.5, 0.3, 0.2];

n_p = filter(b, 1, noise);

This construction ensures that the reference noise is correlated to the primary noise.

(2) Implement Filter

The adaptive filtering process can be modeled by the following code:

```
for k = 1:N
    x = [noise(k); x(1:end-1)]; % Update reference signal buffer
    y(k) = w.' * x; % Filter output
    e(k) = d(k) - y(k); % Error (desired - output)
    w = w + step_size * e(k) * x; % LMS weight update
end
```

Code 1: Iteration Part of the LMS Adaptive Filter in MATLAB

(3) Results

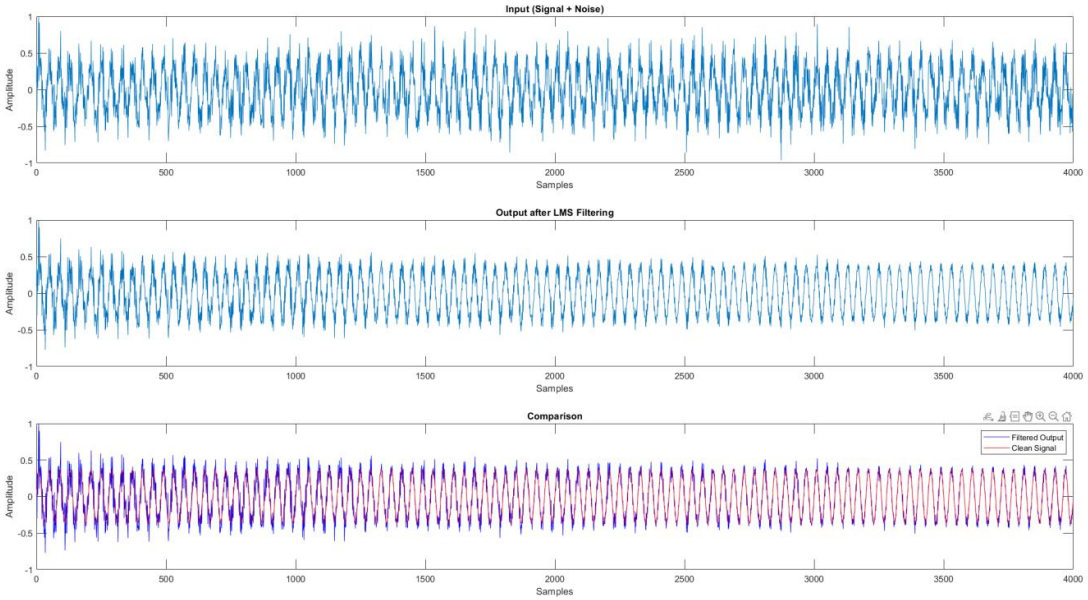

Figure 2. MATLAB Result of Noise Filter

As shown in Figure 2, The adaptive filter gradually shapes its impulse response so that y(k) approximates the primary noise n(k)n(k)n(k). As a result, the output signal gets closer to the clean signal.

III. RTL DESIGN

1. FIR Filter

This section outlines the RTL structure of the FIR filter block and describes how different architectural

choices influence timing, hardware cost, and implementation flexibility.

(1) Standard FIR Filter

The basic direct-form FIR filter places all multipliers and adders in a single combinational chain as shown in Figure 3. The critical path for a N-tap filter is:

$$T = T_{mul} + (N-1)T_{add}$$

When the number of taps becomes large, the adder chain can grow long. The critical path then limits the maximum clock frequency and creates timing closure issues in large filters. As a result, this structure only suit moderate tap counts or relaxed timing margins.

(2) Pipelined Filter

Pipeline registers are inserted between arithmetic stages, reducing the effective combinational span. If a register is inserted for every tap as shown in Figure 4, the critical path becomes

$$T = T_{mul} + T_{add}$$

Throughput increases because the pipeline breaks long paths, but latency rises and extra registers are needed.This form is preferred when strict timing closure is necessary and throughput must match high-rate input streams.

(3) MAC Filter

The multiply-accumulate(MAC) architecture reuses a single MAC unit over several cycles to compute the tap sum as shown in Figure 5.

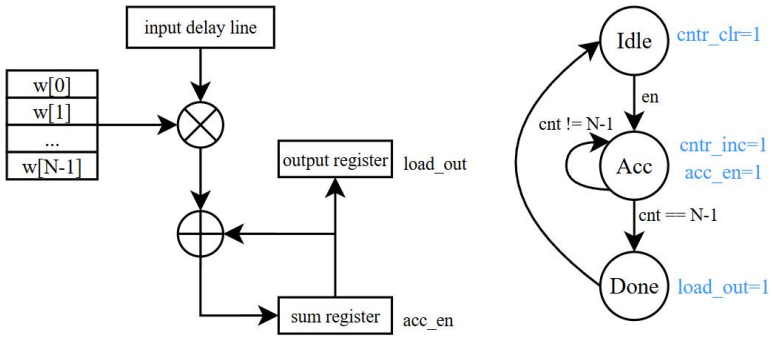

Figure 5: Datapath and FSM of MAC-based FIR Filter

The critical path is also:

$$T = T_{mul} + T_{add}$$

This approach is feasible when the sample rate is significantly lower than the system clock. Several clock cycles are available between input samples, hence allowing sequential accumulation of tap products instead of full parallel processing. The structure minimizes multiplier/adder count and saves area, but cannot keep up when the sampling period is too short. Consequently, this choice is applicable only when a generous cycle budget exists for each incoming sample.

2. LMS Algorithm

Each coefficient update follows

$$w_k[n+1] = w_k[n] + \mu e[n]x[n-k]$$

The Datapath Diagram is shown in Figure 6.

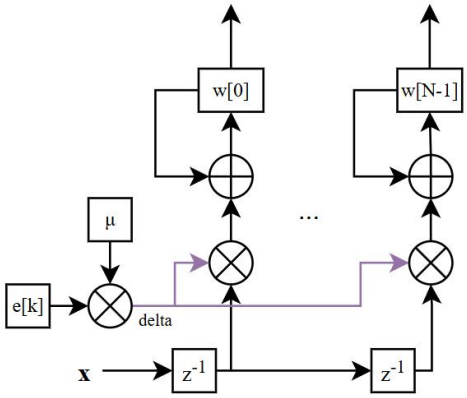

Figure 6: Datapath for LMS Update Block

IV. VERIFICATION

1. Hardware Behavioral Simulation in MATLAB

To verify the functionality of our RTL design and generate test vectors, a MATLAB script is created to simulate how hardware performs LMS algorithm. The script modifies the code in Code 1 but using Q1.15 fixed-point instead of float-point. The filter output part is also changed to multiply-accumulate structure.

```
for k = 1:N
    e(k) = e_new;
    x_in = n_q15(k);
    d_in = d_q15(k);
    % FIR Filter
    acc(1) = q15_mult(w(1), x_in);
    for i = 2:N_TAPS
        acc(i) = acc(i-1) + q15_mult(w(i), x(i-1));
    end
    y(k) = acc(N_TAPS);
    % LMS Update
    e_new = d_in - y(k);
    delta(k) = q15_mult(step_size_q15, e(k));
    w = w + q15_mult(delta(k),  x(1:end-1));
    x = [x_in; x(1:end-1)];
end
```

Code 2: MATLAB Script to Simulate Hardware Orientated LMS Filter Behaviour

q15_mult is a function that performs multiplication between 2 Q1.15 fixed-point scalars and/or vectors. The core code is shown in Code 3.

```
% Clip inputs to Q1.15 range
a_clipped = min(max(int32(a), Q15_MIN), Q15_MAX);
b_clipped = min(max(int32(b), Q15_MIN), Q15_MAX);

prod = a_clipped .* b_clipped;
y32 = bitshift(prod, -15);
% Saturate result to Q1.15 range
y32 = min(max(y32, Q15_MIN), Q15_MAX);
% Convert back to int16
y = int16(y32);
```

Code 3: <q15_mult> Function That Performs Multiplication Between Q1.15 Scalars and/or Vectors

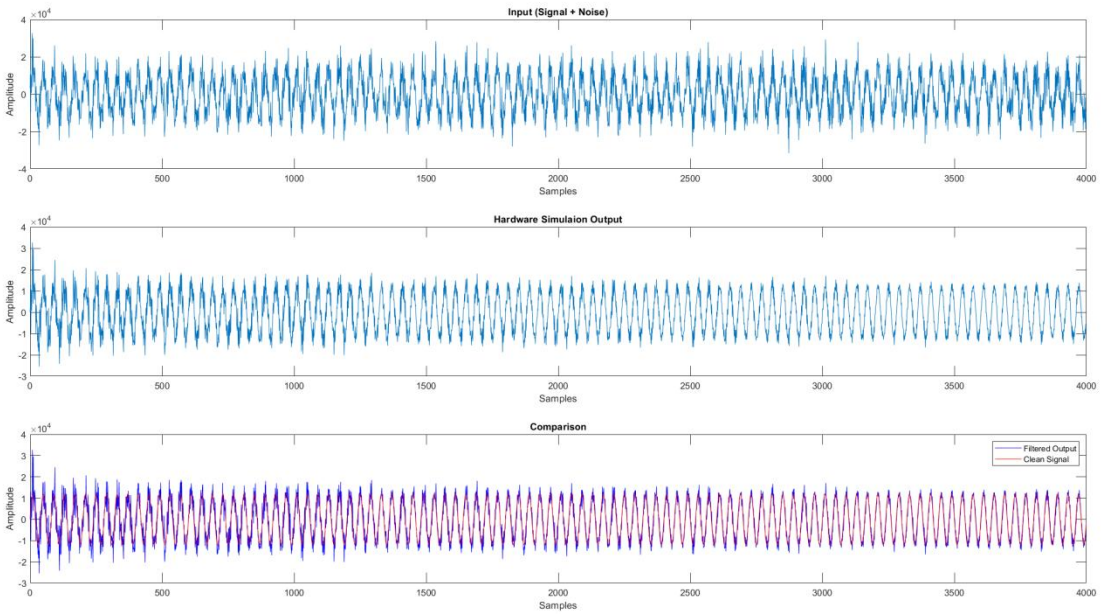

Figure 7. MATLAB Simulation Result of Hardware Noise Filter

The result matches the prototype discussed in Section 2-3, showing our hardware structure functions as expected. We now can output the variables and use them in testbench.

2. Testbench

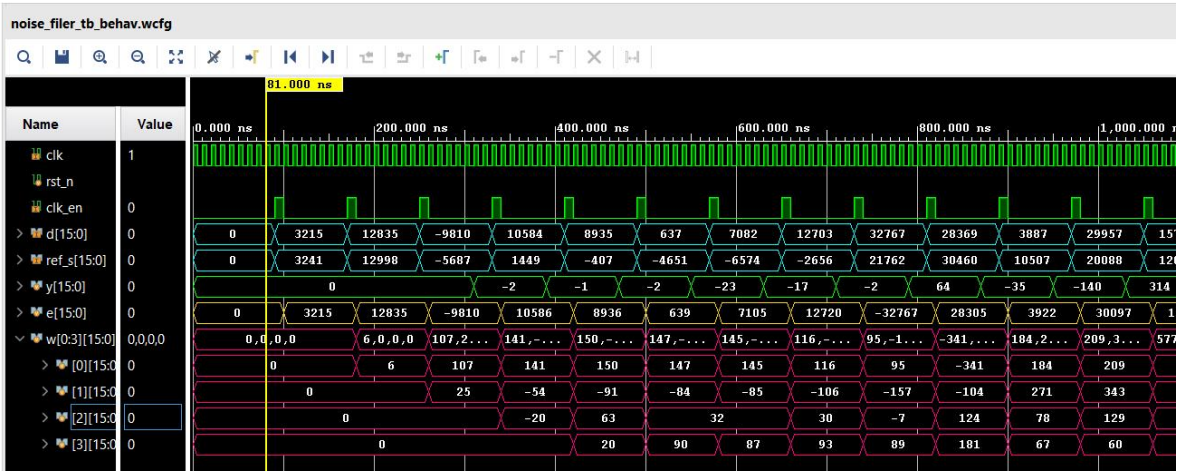

Figure 8: Simulation Waveform

V. IMPLEMENTATION

1. Integration with ADC

In this section we will deploy the filter on the ZYBO Z7 board using block design. We will create a custom IP to encolse the noise filter and preprocessing system shown in Figure 9.

To ulitize the on-board ADC module, we can use the XADC wizard. The wizard communicates with external modules via AXI4 interface or DRP interface. Since the implementation only needs to read the conversion from the wizard, DRP interface is preferred for its simplicity.

The ADC module on the board has a resolution of 12-bit but the filter takes Q1.15 inputs, so a 12-bit unsigned to Q1.15 fixed-point is needed.

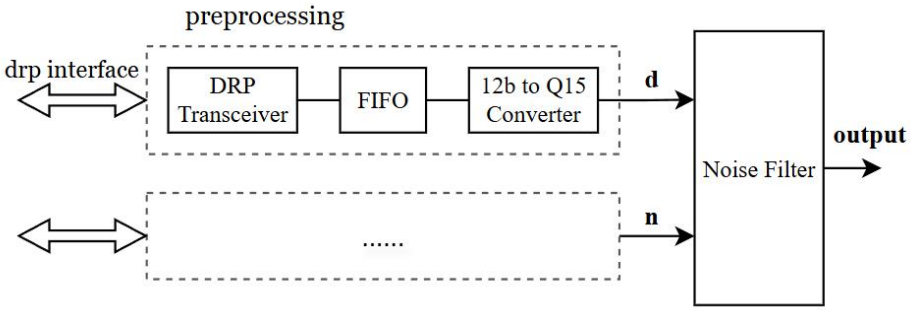

Figure 9. Noise Filter IP Diagram with 12-bit Inputs From XADC

2. IP Verification

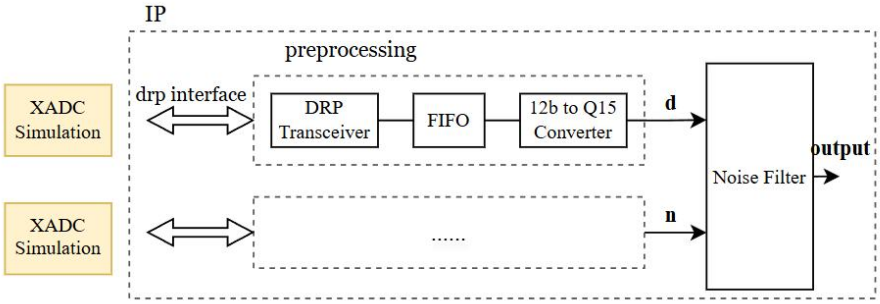

Figure 10. IP Testbench Setup

A XADC Simulator is developed to mimic XADC Wizard Behaviour. The Simulator will continously generate a 12-bit unsigned reading and transmit it through DRP interface when requested.

Test vectors are generated using the same method described in Section 4-1 with input signals being converted to 12-bit unsigned integers.

```
% Q15 = int16(ADC - 2048) * 16
% ADC = Q15 / 16 + 2048
d_u12 = bitshift(d_q15, -4, 'int16') + 2048;
n_u12 = bitshift(n_q15, -4, 'int16') + 2048;
% signals to MATLAB filter
d_q15_12 = bitshift((d_u12-2048), 4);
n_q15_12 = bitshift((n_u12-2048), 4);
```

Code 4: 12-bit unsigned to Q1.15 Conversion

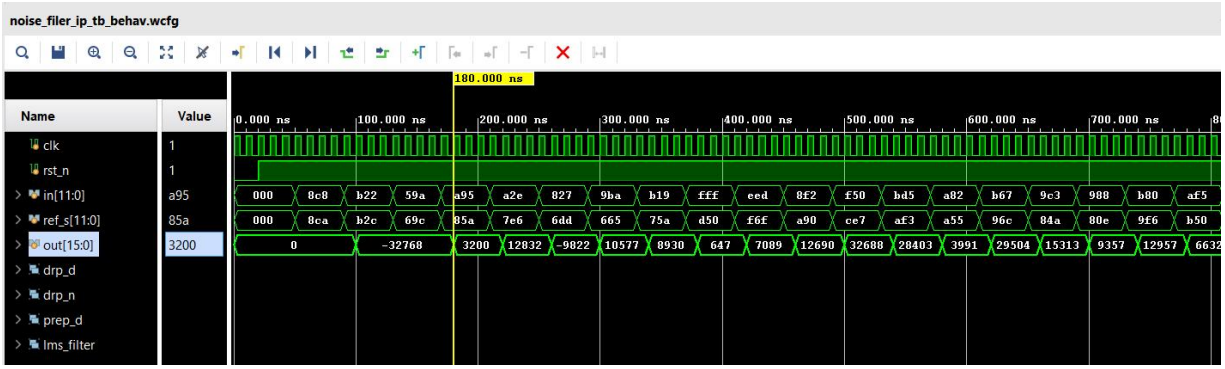

Figure 11. Noise Filter IP Simulation Waveform

3. Block Design

Since the ZYBO board does not have a on-board DAC module, we need to send the signal to an external DAC module via SPI protocol.

Vivado provides a SPI IP using AXI4 interface. Therefore we need a driver that communicate with the IP and send filter output through AXI-Lite interface.

The complete block design is shown in Figure 11.

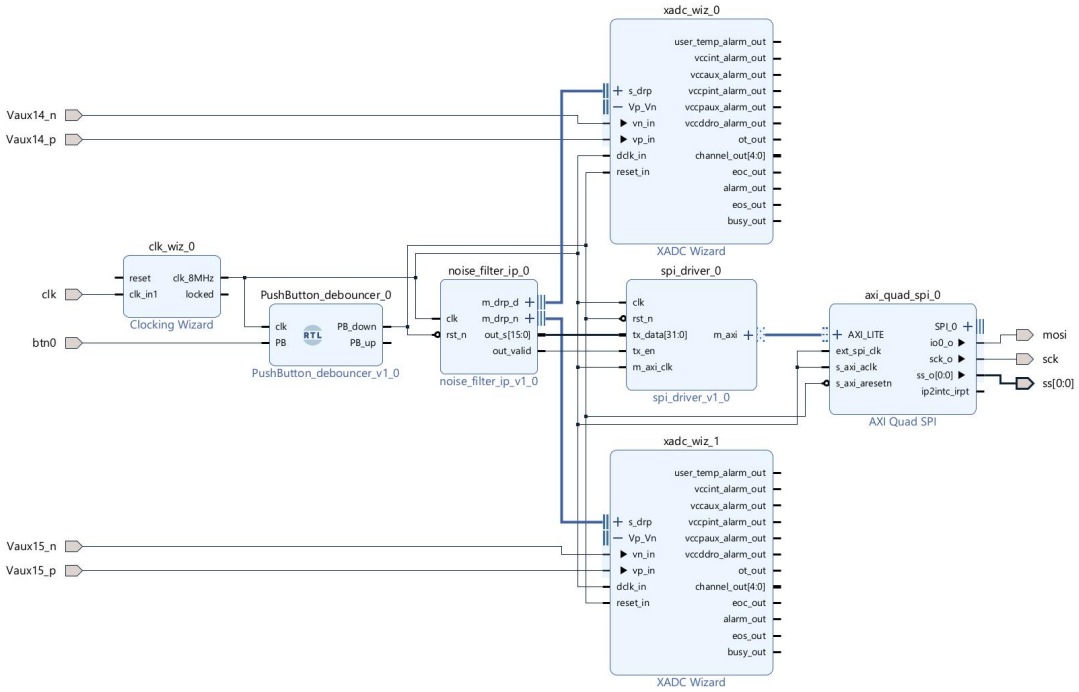

Figure 12. Final Block Diagram

4. Hardware Setup

Device List:

(1) FPGA Board: Zybo Z7-10.

(2) STM32F767: Used as a signal collector

(3) ADALM2000: Oscilloscope and Signal Generator

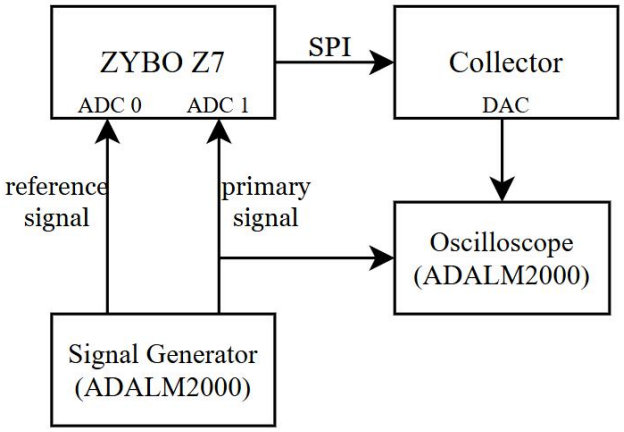

Figure 13: Hardware Setup Diagram

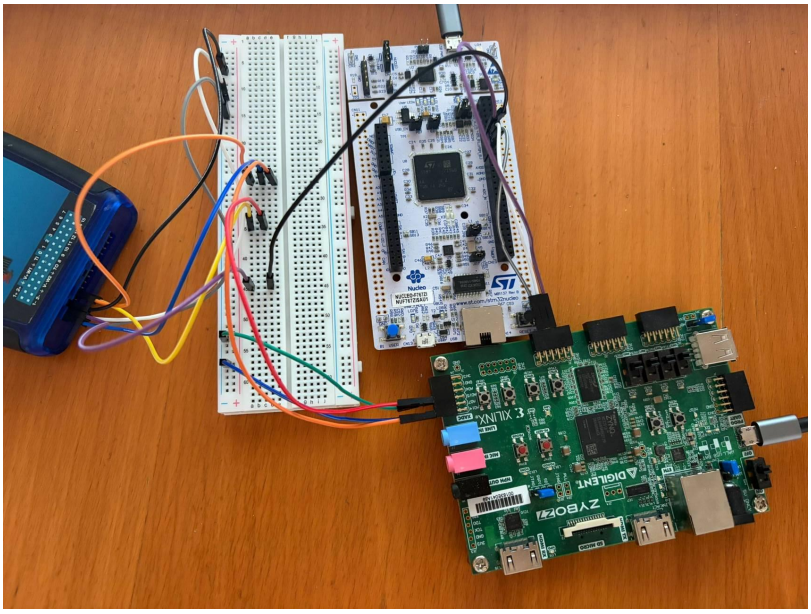

Figure 14: Hardware Setup Photo

5. Results

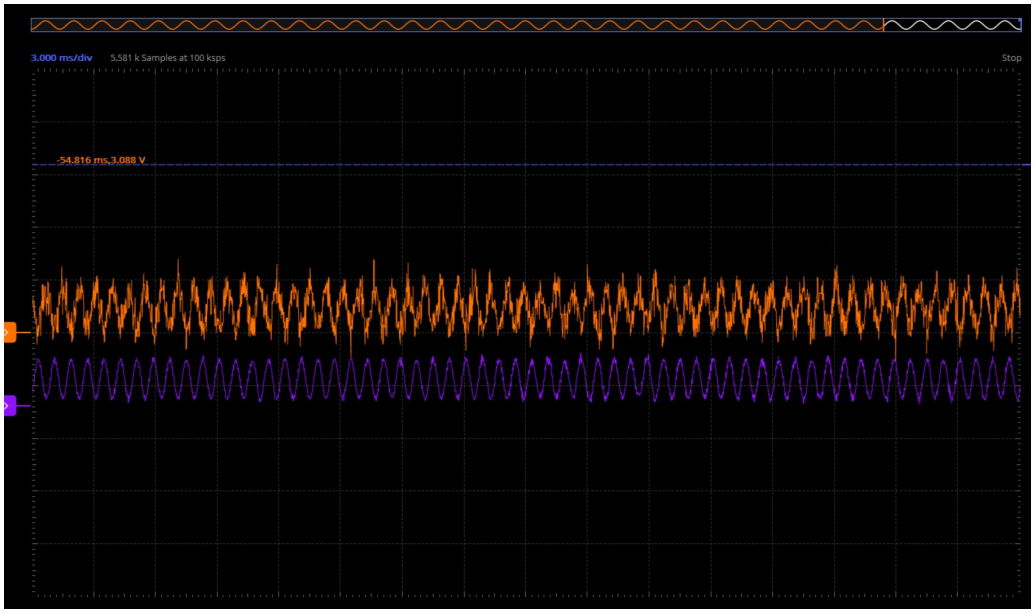

Figure 15: Oscilloscope Image. Organe(ch0): Primary Input Signal, Purple(ch1): Filter Output

VI. SUMMARY

This project presents an adaptive filtering system developed from algorithm to hardware. The study begins with the LMS method and its behaviour evaluated through MATLAB simulation. The design is translated into RTL, covering FIR structures and coefficient-update logic. A structured verification process confirms correctness before synthesis. The implementation targets an FPGA and incorporates data acquisition through the XADC interface, a custom IP core, and an SPI output path for observing

filtered signals. Final oscilloscope measurements demonstrate real-time operation of the adaptive filter and validate the overall design flow.