

Project Title: AUDIO PRE-PROCESSOR

1- Project Scope

This project implements an **Audio Preprocessor** capable of enhancing audio quality using four main filters, having general filter values:

1. **Bass (Low-Pass Filter):** Enhances low frequencies below ~250 Hz.
2. **Treble (High-Pass Filter):** Emphasizes high frequencies above ~4 kHz.
3. **Band-Pass Filter:** Allows frequencies within 500 Hz–2 kHz range.
4. **Echo Effect:** Adds a delayed signal to simulate natural reverberation.

Functional Block Diagram:

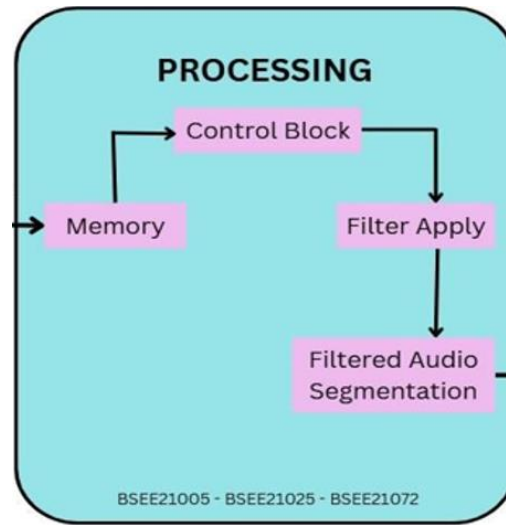


Fig 1: shows Processing Block functional block level diagram

2- Micro-Architecture

Block-Level Diagram:

- Include blocks for filter modules (Bass, Treble, Band-Pass, Echo).
- Show interconnections with input, clock, reset, and control signals.

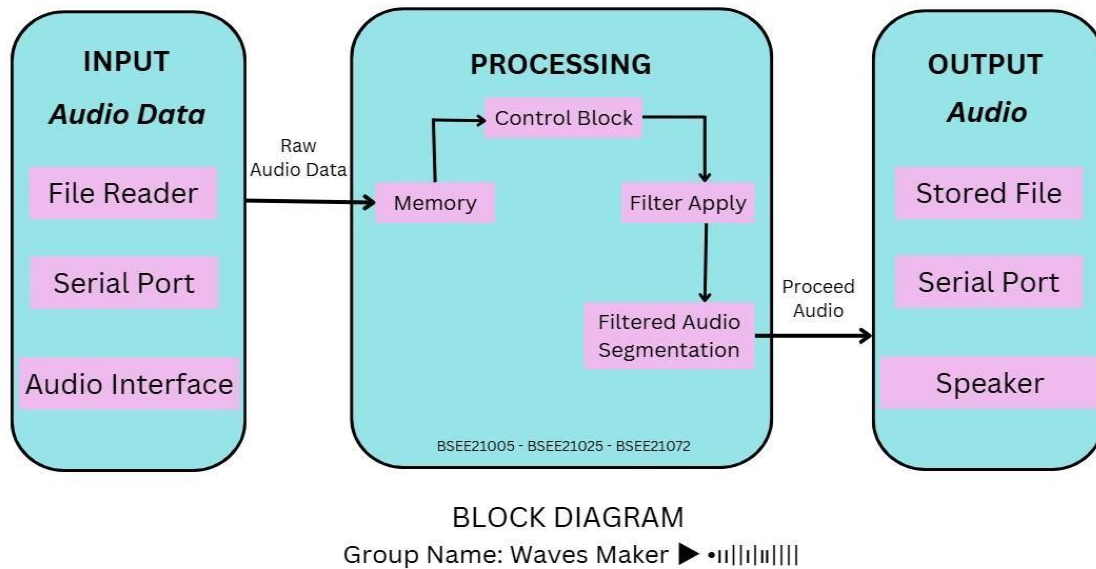


Fig II: shows Complete Block level Diagram

Detailed RTL Diagram:

- Display how filters interact with delay lines, coefficients, and the computation engine.

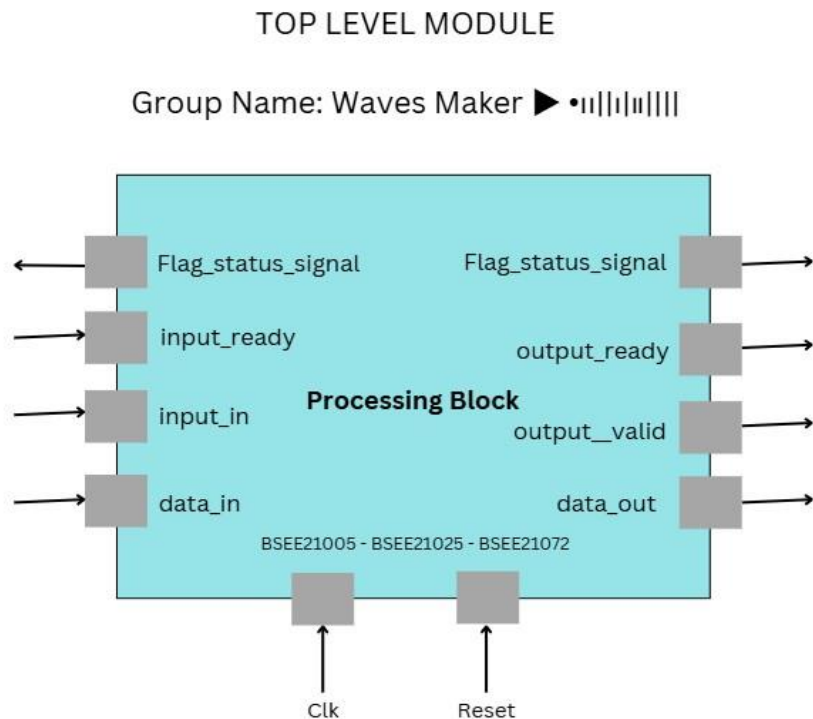


Fig III: shows Top level Diagram showing Control and Datapath signal

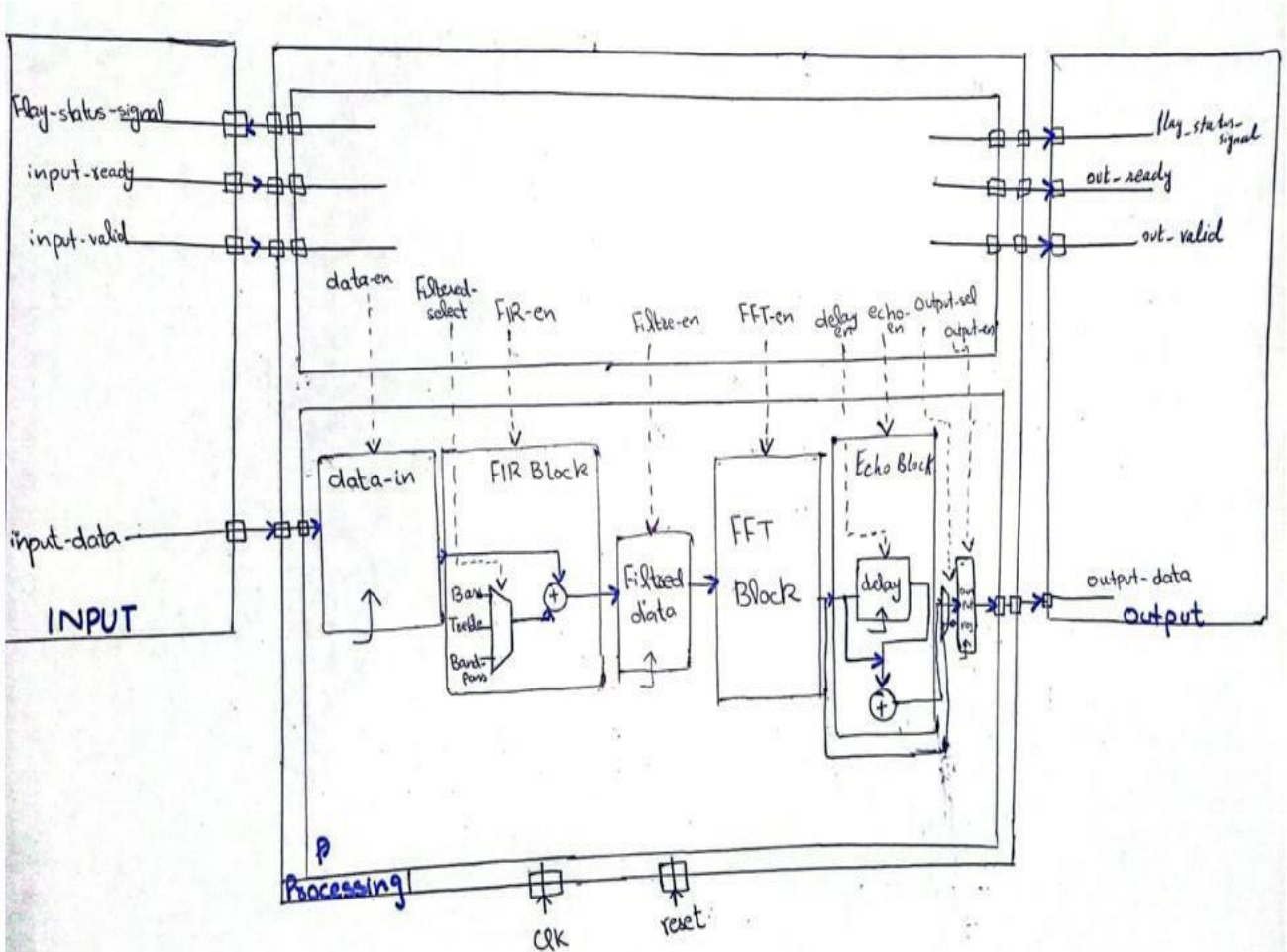


Fig IV: shows Datapath and control path RTL design

FSM Details:

- Two states:
 1. **Processing State:** Computes the output based on filter selection.
 2. **Idle State:** Waits for the next input signal.

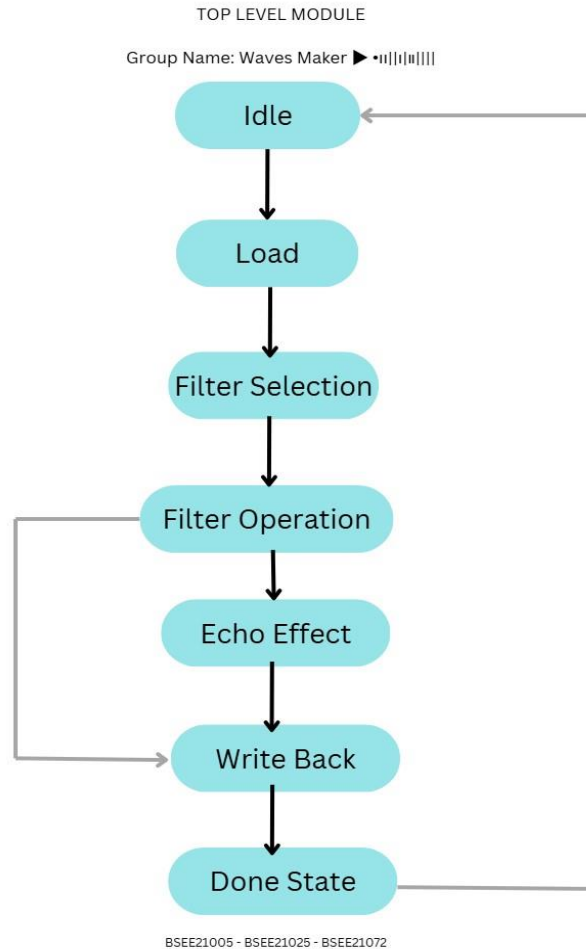


Fig V: shows FSM of Audio Preprocessor

3- Design Goals & Optimization Strategies

1. Design Goals:

- **Primary Goal:**
 - **Latency Optimization:** Focused on minimizing processing delays to enable **real-time audio filtering**.
- **Secondary Goal:**
 - **Bandwidth Efficiency:** Ensuring sufficient data transfer rates for high-fidelity audio processing.
 - **Timing Constraints:** Maintaining strict adherence to **FPGA clock cycles** to avoid buffer overflows.

Reason for Goals:

- Latency is critical for **real-time audio applications** such as filtering and echo effects.
- Timing precision is required for **serial data communication** and ensuring accurate data transfer from the microphone to FPGA and back to the speaker.

2. Optimization Strategies:

1. Data Representation:

- Convert **MP3 to .mem format** in MATLAB for easy Verilog input/output handling.
- Optimize **memory initialization** to pre-load data efficiently into FPGA registers.

2. Pipelining:

- Break down the filter operation into **pipeline stages** to improve throughput.
- Use **separate registers** for each stage to reduce processing delays.

3. Compression Trees:

- Implement **parallel data processing** for computations, especially in echo effects, where delays are added.

4. Memory Optimization:

- Use **Block RAM (BRAM)** for storing intermediate values to minimize external memory accesses.

5. Filter Design Strategy:

- Implement **IIR and FIR filters** in Verilog with coefficient values generated using MATLAB tools.
- Optimize coefficients for minimal hardware utilization and precision trade-off.

MATLAB Workflow: To get the audio file and see outputs

```
% Read MP3 file
[input, Fs] = audioread('D:\Software\Codes\Matlab\input.mp3');

% Scale to 16-bit integer format
scaled_input = int16(input * 32767);
```

```
% Write to memory file
fid = fopen('input_audio.mem', 'w');
fprintf(fid, '%d\n', scaled_input);
fclose(fid);

Step 2: Process Output File from FPGA (.mem):

matlab
Copy code

% Read FPGA output
fid = fopen('out_audio.mem', 'r');
output = fscanf(fid, '%d');
fclose(fid);

% Convert back to floating point
output = double(output) / 32767;

% Play processed output
sound(output, Fs);






% Save to WAV for reference
audiowrite('D:\Software\Codes\Matlab\processed_audio.wav', output,
Fs);

% Input Signal Analysis
subplot(2,2,1);
plot(input);
title('Input Signal - Time Domain');
xlabel('Samples'); ylabel('Amplitude');
```

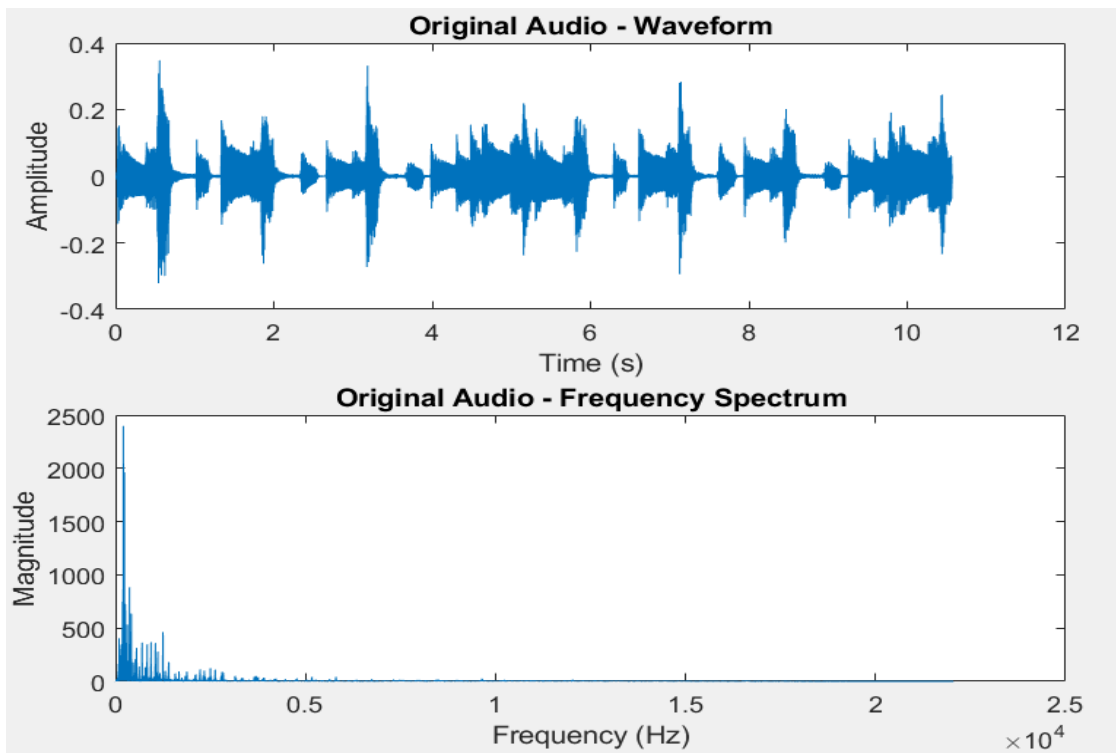
```
subplot(2,2,2);  
pwelch(input,[],[],[],Fs);  
title('Input Signal - Frequency Spectrum');  
  
% Output Signal Analysis  
subplot(2,2,3);  
plot(output);  
title('Output Signal - Time Domain');  
xlabel('Samples'); ylabel('Amplitude');  
  
subplot(2,2,4);  
pwelch(output,[],[],[],Fs);  
title('Output Signal - Frequency Spectrum');
```

Results:

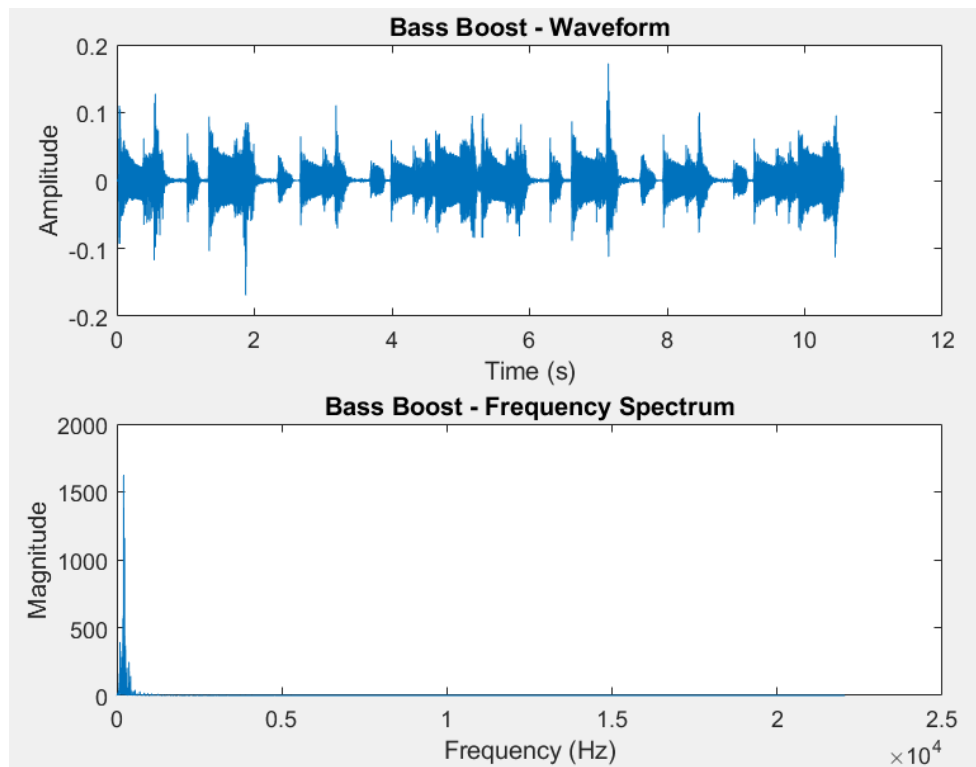
Files make are:

Current Folder	
	Name ▾
	processed_audio.wav
	out_audio.mem
	input_audio.mem
	input.mp3

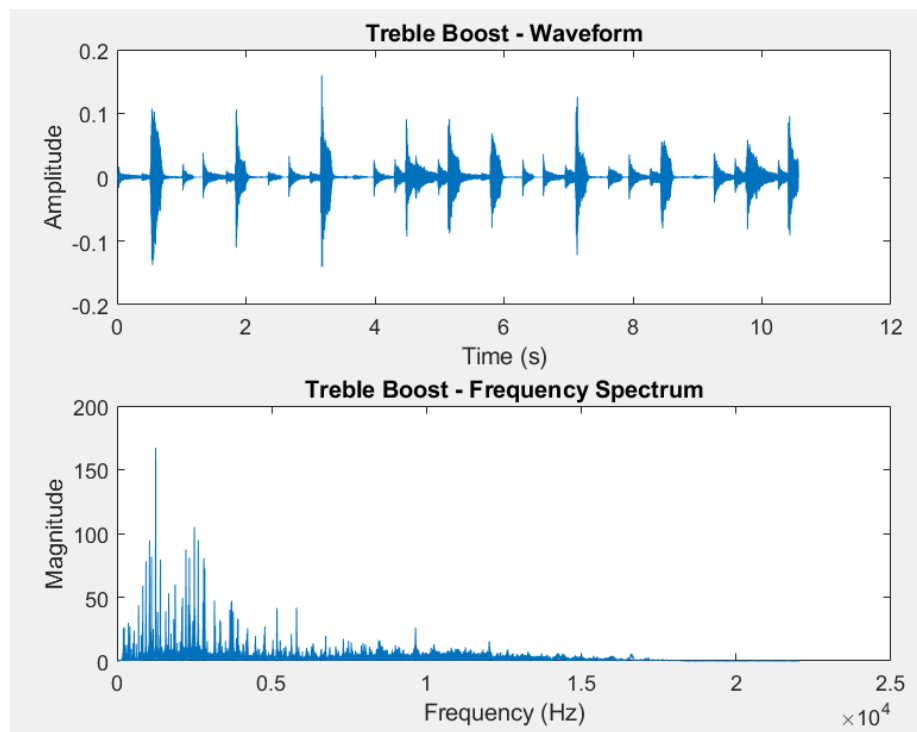
Original Audio



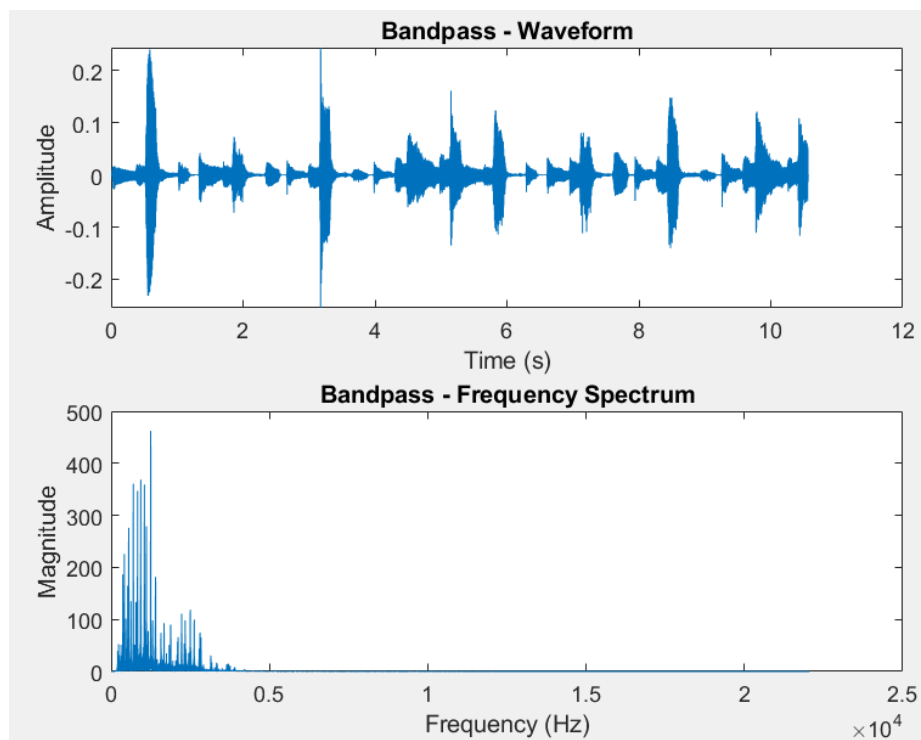
Bass filter



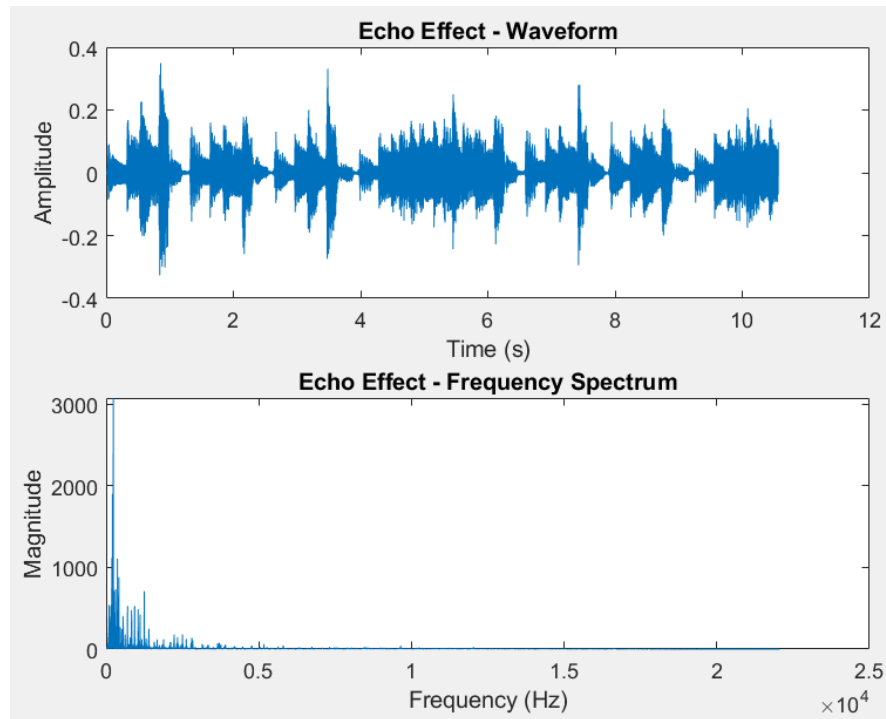
Treble filter



Bandpass filter



Echo



These .mem file read by;

Read .mem file:

```
$readmemb("input_audio.mem", D:\Software\Codes\Matlab\input_audio.mem);
```

Write .mem file:

```
$fdisplay(file, "%b", memory_array[i]);
```

Serial Data Communication (FPGA Integration):

- **Input Data Handling:**
 - Microphone data is captured via **I2S or PDM microphones** and sent serially to the FPGA.
 - **UART or SPI protocol** is used for data transfer into FPGA registers.
 - Data is deserialized and stored in memory blocks for processing.

- **Output Data Handling:**
 - Processed data is serialized and sent back through UART or SPI to a **speaker module**.
 - Timing synchronization ensures real-time output without glitches.

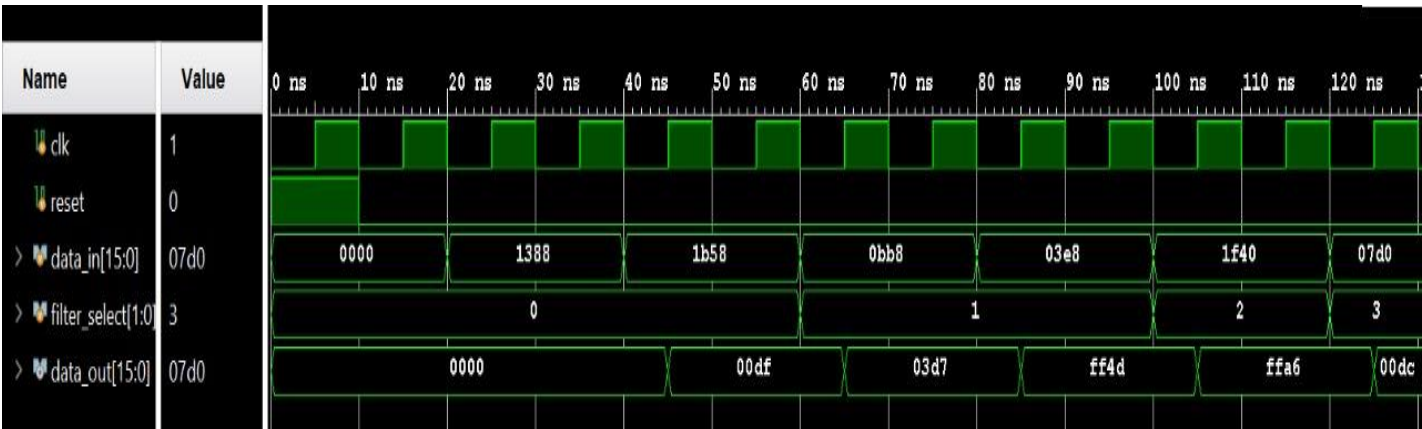
Filter and Echo Processing (Verilog Implementation):

- **Filter Design:** Implemented low-pass and high-pass FIR filters in Verilog. Coefficients are pre-calculated in MATLAB.
- **Echo Effect:** Delay-line buffer implemented in Verilog to simulate echo by adding delayed versions of signals.
- **Verification:** MATLAB simulation confirms the accuracy of filtering and echo effects before deploying to FPGA.

4- Simulation Results

Using testbench stimuli, the following waveforms demonstrate the processed outputs for different filter settings:

1. **Low-Pass Filter:** Smooth output enhancing bass frequencies.
2. **High-Pass Filter:** Highlights treble elements.
3. **Band-Pass Filter:** Retains mid-range audio.
4. **Echo Effect:** Shows delayed repetitions in the signal.



5- Performance Results

1. **Latency:**
 - Processing time for each filter: ~1 cycles.
2. **Bandwidth:**
 - Suitable for audio sampling at 44.1 kHz.
3. **Timing Analysis:**
 - Maximum clock frequency: 100 MHz.

6- Level of Completeness

Completed:

- All four filters are functional and verified through simulations.
- Modular testbench ensures scalability.

Remaining:

- Hardware implementation for real-world testing.
- Optimization for power and area efficiency.

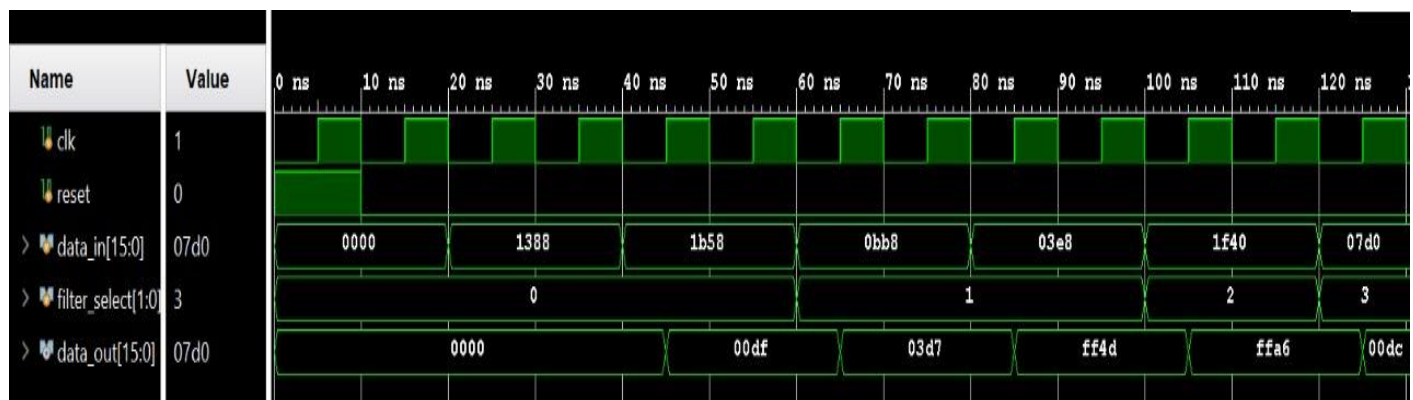
7- CEP Attributes

Attribute	Description	Elaborate and justify the level of Attribute achieved in your project. Relate with various portions of the report.
WP1: Depth of knowledge	The project shall involve in-depth engineering knowledge related to the area of Digital Design. [WK-4, Engineering Specialization].	The project integrates concepts of DSP, Verilog, and audio signal processing, showcasing engineering depth.
WP2: Range of conflicting requirements	The project has multiple conflicting requirements in terms of optimal usage of resources and performance.	Achieving real-time processing while managing resources like memory (for echo delay line) and computational units.

WP5 Extent of applicable codes	The projects expose the students to broadly defined problems which require development of codes that may be partially outside those encompassed by well documented standards.	Custom Verilog code written for DSP-specific filters extends beyond standard libraries.
WP7 Interdependence	The projects shall have multiple components at hardware level and software level for verification and analysis	Includes hardware (delay lines, multipliers) and software (testbench verification).

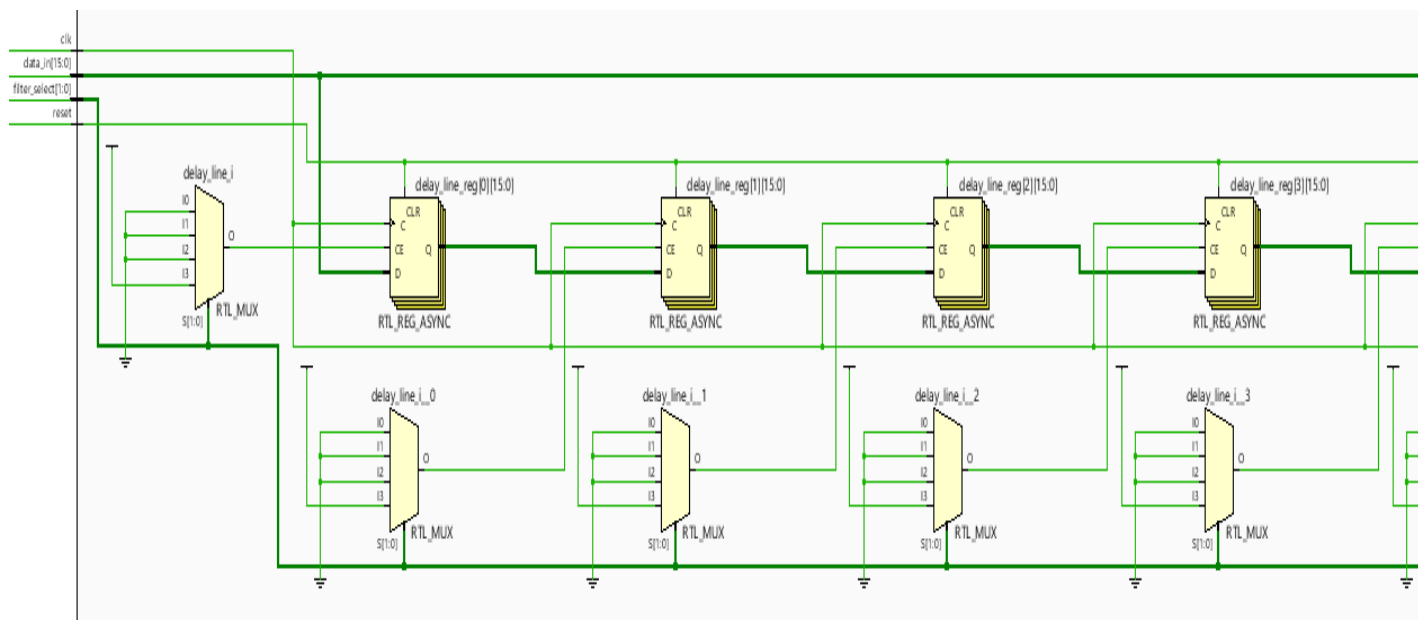
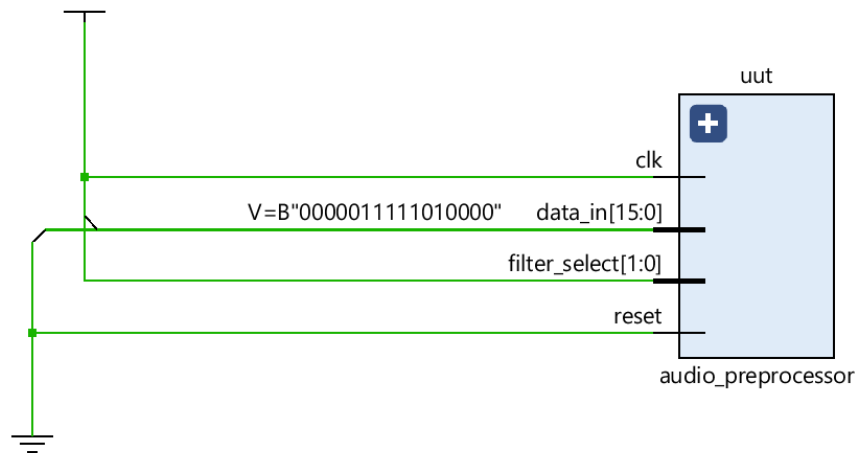
Annexure-A (Testbench Snapshots)

- Simulation waveform shows the correct functionality:
 - Input samples (data_in) are processed by the selected filter, producing corresponding outputs (data_out).
 - Transition between filters is seamless, with results reflecting the expected behavior of each filter.
 - Echo effect (filter_select = 11 (in decimal 3)) introduces a delayed version of the signal.



RTL Analysis

← → 🔍 🔍 🔄 🔄 ⌂ ⌂ + - ↺ 1 Cell 2 Nets



Power utilization

Settings

Summary (0.084 W, Margin)

Power Supply

✓ Utilization Details

Hierarchical (0 W)

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power: 0.084 W
Design Power Budget: Not Specified
Power Budget Margin: N/A
Junction Temperature: 25.4°C

On-Chip Power



Dynamic: 0.000 W (0%)
Device Static: 0.084 W (100%)

```

Module audio_preprocessor
Detailed RTL Component Info :
+---Registers :
                32 Bit    Registers := 1
                16 Bit    Registers := 4414
                1 Bit     Registers := 1

+---Muxes :
    4 Input      32 Bit    Muxes := 1
    4 Input      14 Bit    Muxes := 1
    4 Input      13 Bit    Muxes := 2
    4 Input       1 Bit    Muxes := 2
    2 Input       1 Bit    Muxes := 1

```

Annexure-B (Code)

Code Listing

Audio Preprocessor

```

module audio_preprocessor (
    input clk,
    input reset,
    input signed [15:0] data_in,      // Input audio sample
    input [1:0] filter_select,        // 00: Bass, 01: Treble, 10:
Band-Pass, 11: Echo
    output reg signed [15:0] data_out // Processed output
);

    // Filter coefficients (scaled for Q15 format)
    localparam signed [15:0] LPF_COEFF_0 = 16'd2929;
    localparam signed [15:0] LPF_COEFF_1 = 16'd5858;
    localparam signed [15:0] LPF_COEFF_2 = 16'd2929;

    localparam signed [15:0] HPF_COEFF_0 = 16'd2929;
    localparam signed [15:0] HPF_COEFF_1 = -16'd5858;
    localparam signed [15:0] HPF_COEFF_2 = 16'd2929;

    localparam signed [15:0] BPF_COEFF_0 = 16'd2066;
    localparam signed [15:0] BPF_COEFF_1 = 16'd0;
    localparam signed [15:0] BPF_COEFF_2 = -16'd2066;

```

```

// Echo parameters
parameter DELAY = 4410; // 100ms delay at 44.1kHz
parameter signed [15:0] ECHO_GAIN = 16'd16384; // 0.5 in Q15
format

// Delay lines for filtering and echo
reg signed [15:0] x [0:2]; // For filter processing
reg signed [15:0] delay_line [0:DELAY-1]; // Echo buffer
reg signed [31:0] y; // Extended precision for
calculation
reg processing_done; // Signal to indicate
computation completion

integer i;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        // Reset delay lines and outputs
        for (i = 0; i < 3; i = i + 1) x[i] <= 16'd0;
        for (i = 0; i < DELAY; i = i + 1) delay_line[i] <=
16'd0;

        data_out <= 16'd0;
        processing_done <= 0;
    end else begin
        // Update filter delay line
        x[2] <= x[1];
        x[1] <= x[0];
        x[0] <= data_in;

        case (filter_select)
            2'b00: begin
                // Low-pass filter
                y <= LPF_COEFF_0 * x[0] + LPF_COEFF_1 * x[1] +
LPF_COEFF_2 * x[2];
                processing_done <= 1;
            end
            2'b01: begin
                // High-pass filter
                y <= HPF_COEFF_0 * x[0] + HPF_COEFF_1 * x[1] +
HPF_COEFF_2 * x[2];
                processing_done <= 1;
            end
            2'b10: begin

```



```

        // Band-pass filter
        y <= BPF_COEFF_0 * x[0] + BPF_COEFF_1 * x[1] +
BPF_COEFF_2 * x[2];
        processing_done <= 1;
    end
    2'b11: begin
        // Echo effect
        data_out <= data_in + (delay_line[DELAY-1] *
ECHO_GAIN >> 15);
        for (i = DELAY-1; i > 0; i = i - 1)
            delay_line[i] <= delay_line[i-1];
        delay_line[0] <= data_in;
        processing_done <= 1;
    end
    default: begin
        data_out <= data_in; // Pass-through
        processing_done <= 1;
    end
endcase

// Output is updated only after computation
if (processing_done) begin
    data_out <= y[31:16];
    processing_done <= 0; // Reset processing flag
end
end
end
endmodule

```

Testbench for simulation check

```

module audio_preprocessor_tb;
    reg clk;
    reg reset;
    reg signed [15:0] data_in;
    reg [1:0] filter_select;
    wire signed [15:0] data_out;

    // Instantiate the DUT
    audio_preprocessor uut (
        .clk(clk),
        .reset(reset),
        .data_in(data_in),

```

```

        .filter_select(filter_select),
        .data_out(data_out)
    );

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 100MHz clock
    end

    // Test sequence
    initial begin
        // Initialize inputs
        reset = 1;
        filter_select = 2'b00; // Start with Low-pass
        data_in = 16'd0;

        #10 reset = 0;

        // Apply test stimuli
        #10 data_in = 16'd5000; filter_select = 2'b00; // Low-pass
        #20 data_in = 16'd7000;
        #20 data_in = 16'd3000; filter_select = 2'b01; // High-pass
        #20 data_in = 16'd1000;
        #20 data_in = 16'd8000; filter_select = 2'b10; // Band-pass
        #20 data_in = 16'd2000; filter_select = 2'b11; // Echo

        #100 $finish;
    end
end
endmodule

```

Testbench for .mem audio file in\out

```

module audio_preprocessor_tb;
    reg clk;
    reg reset;
    reg signed [15:0] data_in;
    reg [1:0] filter_select;
    wire signed [15:0] data_out;

    // Instantiate the DUT
    audio_preprocessor uut (
        .clk(clk),

```

```

        .reset(reset),
        .data_in(data_in),
        .filter_select(filter_select),
        .data_out(data_out)
    );

    // Parameters
    parameter NUM_SAMPLES = 256; // Adjust according to your input
file size

    // Memory for input and output data
    reg signed [15:0] input_audio [0:NUM_SAMPLES-1];
    reg signed [15:0] output_audio [0:NUM_SAMPLES-1];
    integer i, out_file;

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 100MHz clock
    end

    // Test sequence
    initial begin
        // Initialize inputs
        reset = 1;
        filter_select = 2'b00; // Start with Low-pass
        data_in = 16'd0;

        // Load input data from file
        $readmemb("D:\Software\Codes\Matlab\input_audio.mem ",
input_audio);

        #10 reset = 0;
        // Apply test stimuli
        for (i = 0; i < NUM_SAMPLES; i = i + 1) begin
            #10;
            data_in = input_audio[i];

            // Cycle through filter selections for testing
            case (i % 4)
                0: filter_select = 2'b00; // Low-pass
                1: filter_select = 2'b01; // High-pass
                2: filter_select = 2'b10; // Band-pass
            endcase
        end
    end

```

```
        3: filter_select = 2'b11; // Echo
    endcase
end

// Save output data to file
out_file = $fopen("D:\Software\Codes\Matlab\out_audio.mem",
"w");
for (i = 0; i < NUM_SAMPLES; i = i + 1) begin
    #10 output_audio[i] = data_out;
    $fwrite(out_file, "%d\n", output_audio[i]);
end
$fclose(out_file);

#100 $finish;
end
endmodule
```