Fachhochschule Dortmund

SS2020

Microelectronics & HW/SW-Co-Design
VHDL and FPGA

AUDIO MIXER

Team Members List	Vignesh Somasundaram	7206904
	Mehul Kumar Uttamchand	7207071
	Tejas Hosur Upendra	7206911

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Hardware Description language

Hardware description language (HDL) is a specialized computer language used to describe the structure and behavior of electronic circuits, and most commonly, digital logic circuits.

It looks much like a programming language such as C. It is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicitly include the notion of time. Another difference is statements in HDL code involve parallel operation, whereas programming languages represent sequential operation.

HDL code is used for describing real hardware of the system.

1.2 Field Programmable Gate Arrays

A Field Programmable Gate Array(FPGA), is an array of logic gates. It's an array of carefully designed and interconnected digital subcircuits that efficiently implement common functions while also offering very high levels of flexibility.

The digital subcircuits are called configurable logic blocks (CLBs), and they form the core of the FPGA's programmable-logic capabilities:

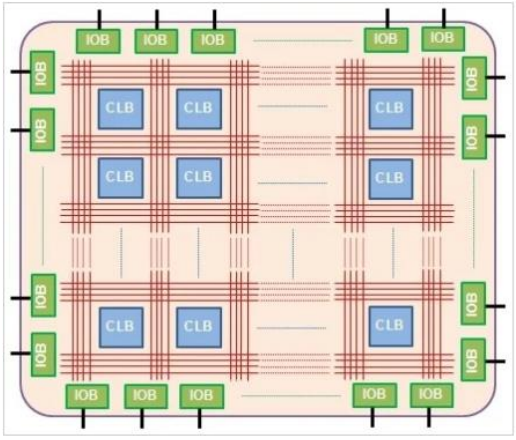

Figure 1 FPGA

The CLBs need to interact with one another and with external circuitry. For these purposes, the FPGA uses a matrix of programmable interconnects and input/output (IOB) blocks.

1.3 RTL view

RTL is a higher level abstraction for your digital hardware design and comes somewhere between strictly behavioral modeling on one end and purely gate-level structural modeling

on other ends. RTL can also be thought of as analogous to the term "pseudo-code" used in software programming. In RTL design a circuit is described as a set of registers and a set of transfer functions describing the flow of data between the registers.

1.4 Audio Mixer

A mixer is an electronic device which is often used for changing the quality and the levels of audio signals. It is also known as a mixing console, an audio mixer, or a soundboard. Using a mixer is the most convenient way to route or combine various audio signals and even change the timbre and dynamics of the sound.

2. AUDIO MIXER

2.1 Working principle

2.1.1 Functional Block Diagram

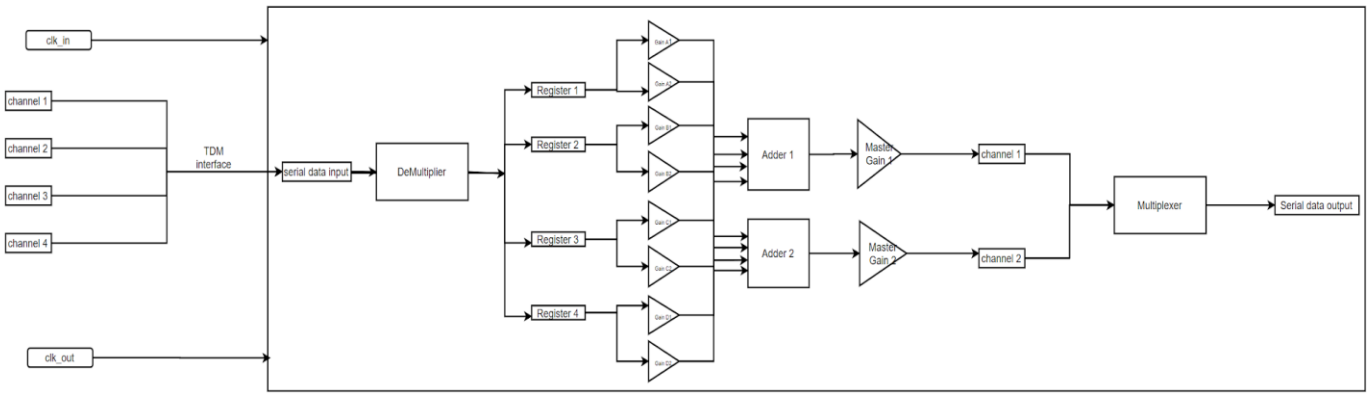

Figure 2:functional block diagram

2.1.2 Proof of Concept

The Audio mixer device does the function of mixing audio samples with the given gains.As per the requirement, there are 10 different gains for 4 different channels of the input. The input is a serial bit stream which is multiplexed by an TDM interface at a certain clock period. The data loss and precision of this function is not in the scope of this assignment.

After the serial bit stream of the 4 channel data is available, it is sent as an input to the Audio Mixer.The audio mixer separates the data stream into 4 channels into 4 registers which carry 16 bit data based on the clock cycle and the computation is performed on this data. After the computation the data is serially packed and sent via the out port of the device.

2.2 Components of the Device

2.2.1 Introduction to TDM and interface

2.2.1.1 Channel Block

This TDM interface consists of a control clock, serial clock (clk_in), and the serial audio data line (sr_in). Each channel block is composed of the audio data word of 16 bit data . Total number of channels is 4 which adds to 64 bit frame rate.The below figure shows 64-bit channel block with 16-bit audio data. The audio word is typically transmitted with the Most Significant Bit (MSB) first. The current industry standard for representing Pulse-Coded-Modulation (PCM) audio data is a 16 to 32 bit word (16- and 24-bit are the most common) coded in a two's-complement method.

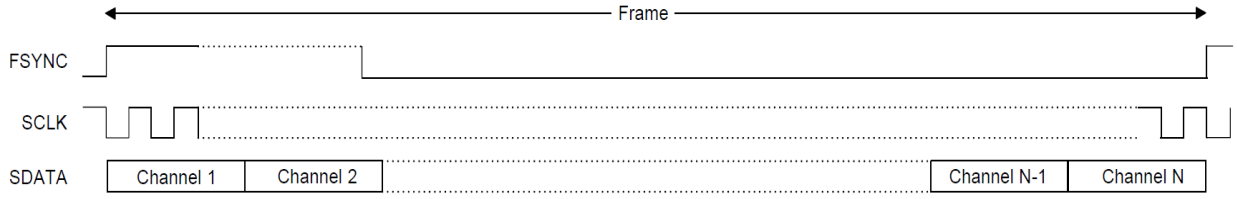

Figure 3

2.2.1.2 Serial Clock (input and output clock)

The sole purpose of the serial clock is to shift the audio data into or out of the serial audio ports. The required frequency for the serial clock is directly proportional to the system audio sample rate, the number of channel blocks within a frame and the bit-width of each channel block.So the input clock in this case with 4 channels and 16 bit channel blocks operating at 48 kHz requires a 3.072 MHz serial clock. So the period is 325 ns.Similarly the output clock being a 24 bit 2 channel serial output data at 48 kHz has to have a 2.304 MHz serial clock or a period of 434 ns.

2.2.2 Demultiplexers

An input signal is switched to one of several outputs. For control, the demultiplexer has control inputs, which are necessary for switching its switches. This switchover can be static or periodic or cyclical, but then always requires a time coordination between the switch position of the multiplexer and the demultiplexer. Ensuring the temporal connection requires special synchronization facilities. In this project we have used only one Demux.

2.2.3 Shift Registers

In digital circuits, a shift register is a cascade of flip flops, sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, "shifting in" the data present at its input and 'shifting out' the last bit in the

array, at each transition of the clock input. More generally, a shift register may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays; this is implemented simply by running several shift registers of the same bit-length in parallel.Shift registers can have both parallel and serial inputs and outputs. In this project we have taken 4 shift registers for serial to parallel data conversion. These registers contain the channel bit stream value at any point in time.

2.2.4 Adders

For adding two or more signals and use it for further usage and analysis. Two adder circuits have been used both being to sum of the amplified signals taken four at a time.

2.2.5 Multipliers

The following describes the single entity for the entire project for all ports associated with the device. The multiplication function is utilized through the modular concept of functions and data encapsulation, where in a function if defined which accepts two input variables and after multiplication truncation is performed to be consistent with the data format and ensure the same is followed for the other processes as well.As per the synthesis tool we two multipliers have been used.

```
    Number Of Mapped DSP Components:
--------------------------------
MULT36X36B            0
MULT18X18B            2
MULT18X18MACB         0
MULT18X18ADDSUBB      0
MULT18X18ADDSUBSUMB   0
MULT9X9B              0
MULT9X9ADDSUBB        0
MULT9X9ADDSUBSUMB     0
--------------------------------
```

Figure 4 Snippet of design summary

2.2.6 Gains

The input values given by users which act as amplification factors for the input signals. Gains get multiplied with the input signals with the help of multipliers. There are 8 individual channel gains and two master gains.

```
entity AudioMixer is
port (
    clk_in : in std_logic;                          -- clock variable
    clk_out : in std_logic;
    enable : in std_logic;                         -- start of serial transmission
    sr_in : in std_logic;
    sr_ou : out std_logic;                         -- serial bit data
    gain_A1 : in unsigned(9 downto 0);       -- gain 1 (10 bt data) for multiplication with seril data 1
    gain_A2 : in unsigned(9 downto 0);       -- gain 2 (10 bt data) for multiplication with seril data 1
    gain_B1 : in unsigned(9 downto 0);       -- gain 3 (10 bt data) for multiplication with seril data 2
    gain_B2 : in unsigned(9 downto 0);       -- gain 4 (10 bt data) for multiplication with seril data 2
    gain_C1 : in unsigned(9 downto 0);       -- gain 5 (10 bt data) for multiplication with seril data 2
    gain_C2 : in unsigned(9 downto 0);       -- gain 6 (10 bt data) for multiplication with seril data 2
    gain_D1 : in unsigned(9 downto 0);       -- gain 7 (10 bt data) for multiplication with seril data 2
    gain_D2 : in unsigned(9 downto 0);       -- gain 8 (10 bt data) for multiplication with seril data 2
    gain_1 : in unsigned(9 downto 0);        -- gain 9 (10 bt data) for multiplication with seril data 2
    gain_2 : in unsigned(9 downto 0));       -- gain 10(10 bt data) for multiplication with seril data 2
```

Figure 5 Gains

2.2.7 Multiplexers

Also known as a **data selector**, is a device that selects between several analog or digital input signals and forwards it to a single output line. One multiplexer has been used just for the purpose of serialization of the parallel data which comes out of the individual adder circuits.

2.2.8 Serialization

With the help of the multiplier the output is serialized and sent to a wav generator for reconstruction of the amplified tone.

2.3 Software components of the Audio Mixer

2.3.1 Entity Declaration and description

1) clk_in : This represents the clock input for the FPGA device. As per the requirement and the project it will be tied to a 4*48kHz frame rate for 64 bits which corresponds to the clock period of 325 ns.
2) clk_out : This represents the output clock frequency for the FPGA device. The frame rate of the output is 48 kHz for 48 bits (2* 24 bits). As per the requirement it will be tied to a clock period of 434 ns.
3) sr_in : This represents the serial input to the device. This input is typically tied to an ADC device which sends the digital values based on the external global clock.
4) sr_ou : This represents the serial output from the device from which the data is usually transmitted to a processor to construct the digital values into an audio file. Can also be linked to an audio generator.
5) gain_A1, gain_A2, gain_B1, gain_B2, gain_C1, gain_C2, gain_D1, gain_D2, gain_1 and gain_2 : Represents the respective gains which is set by an external device.

Similarly the other input ports representing the gains set by external devices.

2.3.2 Architecture declaration description

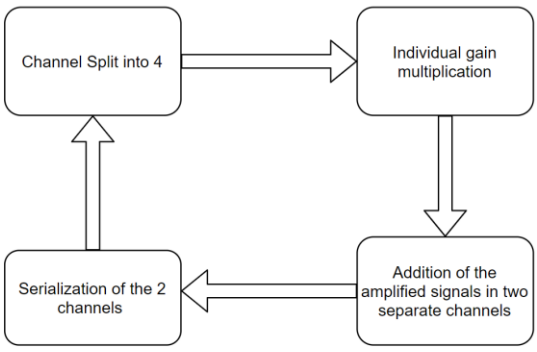

Figure 6 Architechture

The channel split up is performed according to the clock cycle in an asynchronous reset process. Individual gain multiplication is performed using a process with a sensitivity list decided on the b counter and the channel counter, which spawns and the multiplication is performed using defined functions. Addition of the amplified signals into separate channels is performed in a process whose sensitivity list is the amplified individual signals. Finally the serialization takes place according to the output clock set by the user through a synchronous process.

2.3.3 Gain Conversion

The amplification factor, also called gain, is the extent to which an active device boosts the strength of a signal. The damping factor, also called loss, is the extent to which a passive device reduces the strength of a signal. Amplification means voltage ratio $V_2 / V_1 = V_{out} / V_{in}$, and voltage gain in dB = 20 × log (V_2 / V_1).

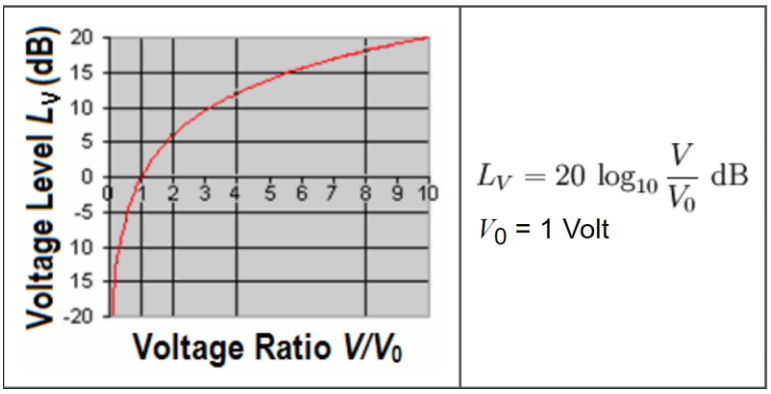

Figure 7 Gain ratio

For the testbench suitable gainw are given in the range of the below table with possible binary values in the permissible boundary and supplied and verified. The same range holds good for all the 10 gains in the audio mixer.

Amplification level *L* voltage level dB	**Gain factor** $v = V_2 / V_1$ (ratio) for **field** sizes, e.g. voltage	**Binary value (5.5 format)**
30	31.622776601683793	11111.10011
-30	0.03162277660168379	00000.00001

2.3.4 Saturation arithmetic in vhdl

Operations such as addition and multiplication are limited to a fixed range between a minimum and maximum value.If the result of an operation is greater than the maximum, it is set ("clamped") to the maximum; if it is below the minimum, it is clamped to the minimum. The name comes from how the value becomes "saturated" once it reaches the extreme values; further additions to a maximum or subtractions from a minimum will not change the result.

- (60 + 43) − (75 + 75) → 0. (*not* the expected −47.) (100 − 100 → 0.)
- 10 × 11 → 100. (*not* the expected 110.)

Signed and unsigned types exist in the **numeric_std** package, which is part of the ieee library. It should be noted that there is another package file that is used frequently to perform mathematical operations: **std_logic_arith**. However, std_logic_arith is not an official ieee supported package file and it is not recommended for use in digital designs.

In this project, due to channel bit stream restrictions the following operations have been performed to limit the bit stream to two 24 bit channel considering without loss of generality,

- All unsigned gains (5.5 fixed point values) are converted into signed values as multiplication is possible only with signed values, by concatenating with '0' as all the gain values are all assumed to be positive.
- After multiplication with the individual bit stream (values stored in the registers) the LSBs are clipped to maintain uniformity in code style in the multiplication function.
- This is carried out with all the individual channels with 8 different gains (2 gains for each channel)
- Before adding the first set of gains to a temporary signal sufficient numbers of zeros are padded to ensure overflow does not happen.
- After the addition the master gain is also added to the additive sum of the first set of 4 channels using the same function.
- Following the same principles the next set of multiplication is also performed and the bit stream is trimmed to ensure the requirements are fulfilled.

2.3.5. Data Extraction and conversions involved

2.3.5.1 Conversion into signed variables

The unsigned gain values are converted into signed temporary signals with zero padding to detect overflow of data during addition and multiplication. To prevent overflow concatenation of leading '0' to each operand

2.3.5.2 Ways to detect overflow:

In case **sum** has an extra bit (compared to the largest of the operands), the following can be done:

1. If the system is unsigned:
   sum <= ('0' & operand1) + ('0' & operand2);
2. If the system is signed (requires sign extension):
   sum <= (operand1(N-1) & operand1) + (operand2(N-1) & operand2);

In this projects we have used the technique of zero padding to preserve the MSBs in the amplified signals.

3. Synthesis

3.1 RTL synthesis

Logic synthesis is a process by which an abstract specification of desired circuit behavior, typically at register transfer level (RTL), is turned into a design implementation in terms of logic gates, typically by a computer program called a synthesis tool. Common examples of this process include synthesis of designs specified in

hardware description languages, including VHDL and Verilog. Some synthesis tools generate bitstreams for programmable logic devices such as PALs or FPGAs, while others target the creation of ASICs. Logic synthesis is one aspect of electronic design automation. For this project we are using Lattice Diamond, a leading edge design software for Lattice FPGA families.

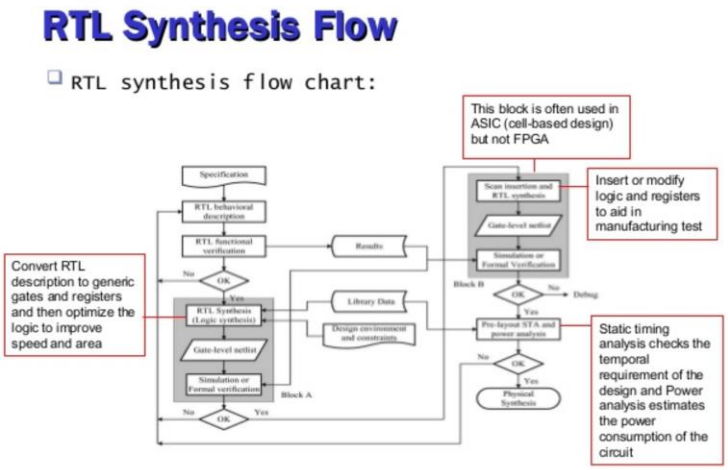

Figure 8 RTLFlowchart (source:Google)

3.2 RTL view

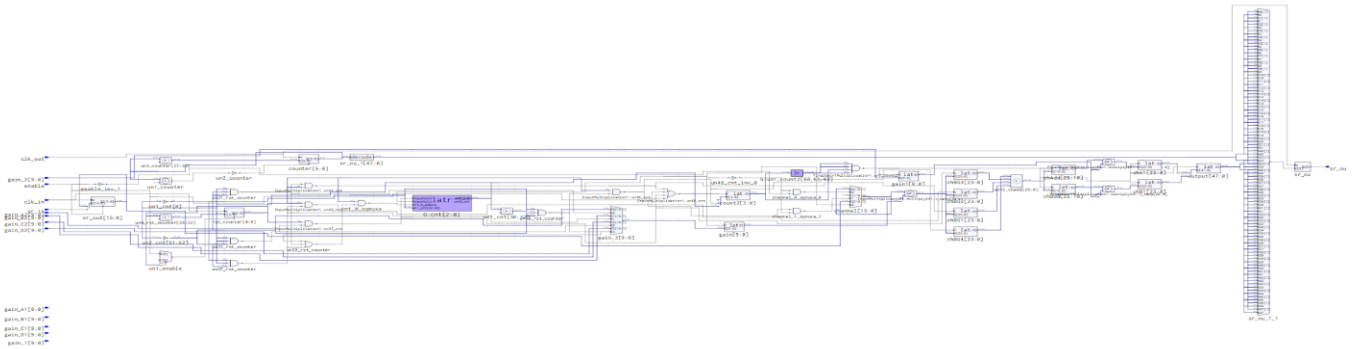

Figure 9 RTL View

3.3 Processes incurred in Synthesis and final report generation

3.3.1 Design Synthesis

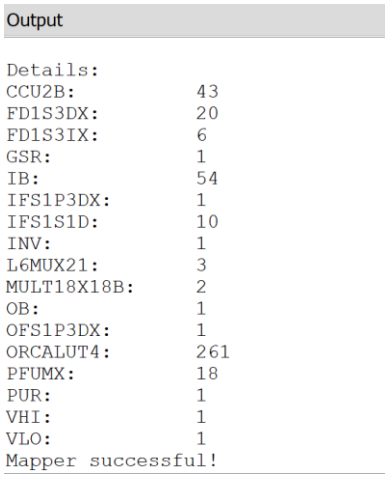

Figure 10: Synthesis output

During the process of design synthesis generation a series and set of log output is generated. Errors, anomalies and issues are reported to the designer in the form of errors and warnings. Incase of errors, all the issues need to be addressed before mapping and the actual synthesis, else the tool does not synthesize. Warnings can be studied for the effects based on which decision can be taken.
A sample design synthesis log report comprises of the following

- All circuit optimizations which the tools observe are reported by the designer.
- Path information along with path number and timing analysis.
- Errors and warnings related to the ports, registers and other circuit related connections.
- Clock Load Summary and preamp clock optimization report
- Constraint checking, slacks and other worst cases tests

All the warnings and errors related to logical netlist are displayed.

3.4 Map design and Place and route design

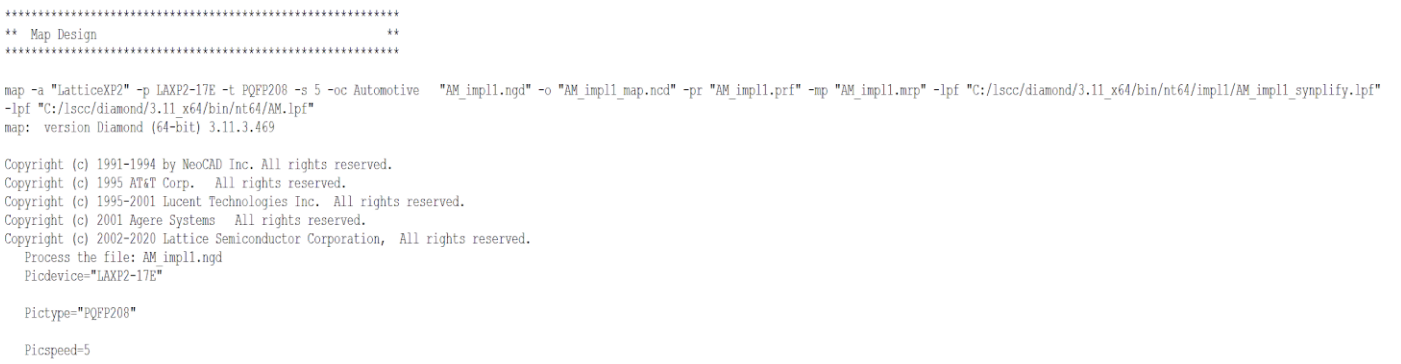

Figure 111: Map Report

Maps the ports and routes it to the real hardware. Also gives a full report on clocks, signals and routes taken by signal along with the cost report summary.

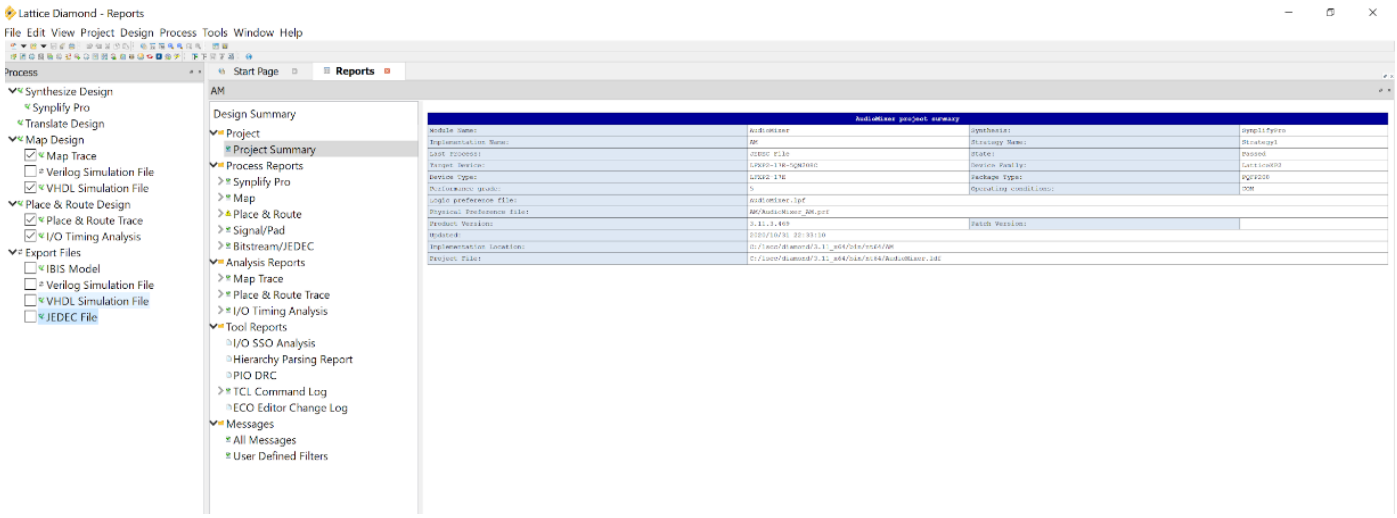

Figure 12:Full report

3.5 Data Flow view

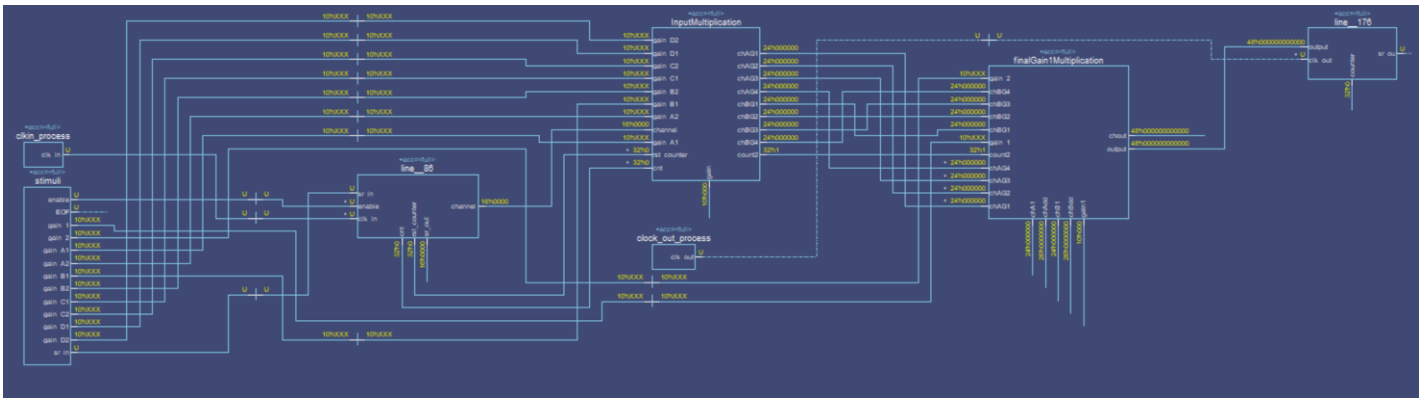

Figure 12

3.6 Floor plan view

The floor plan view and physical view generated after the synthesis are respectively shown below.The view and the actual circuit depends on the device and instrument selected for synthesis.

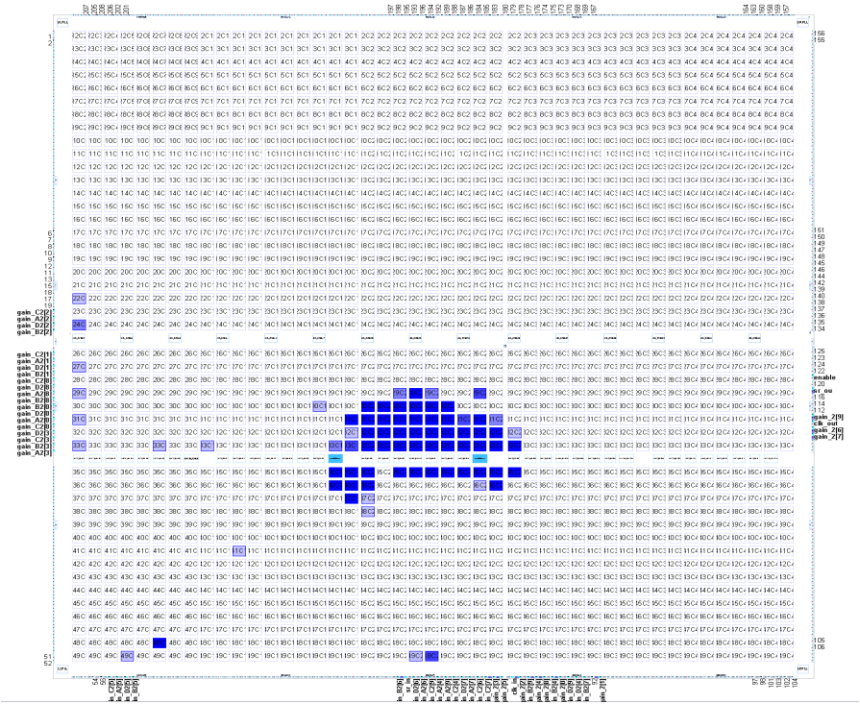

Figure 13: Floor plan view

3.7 Physical view

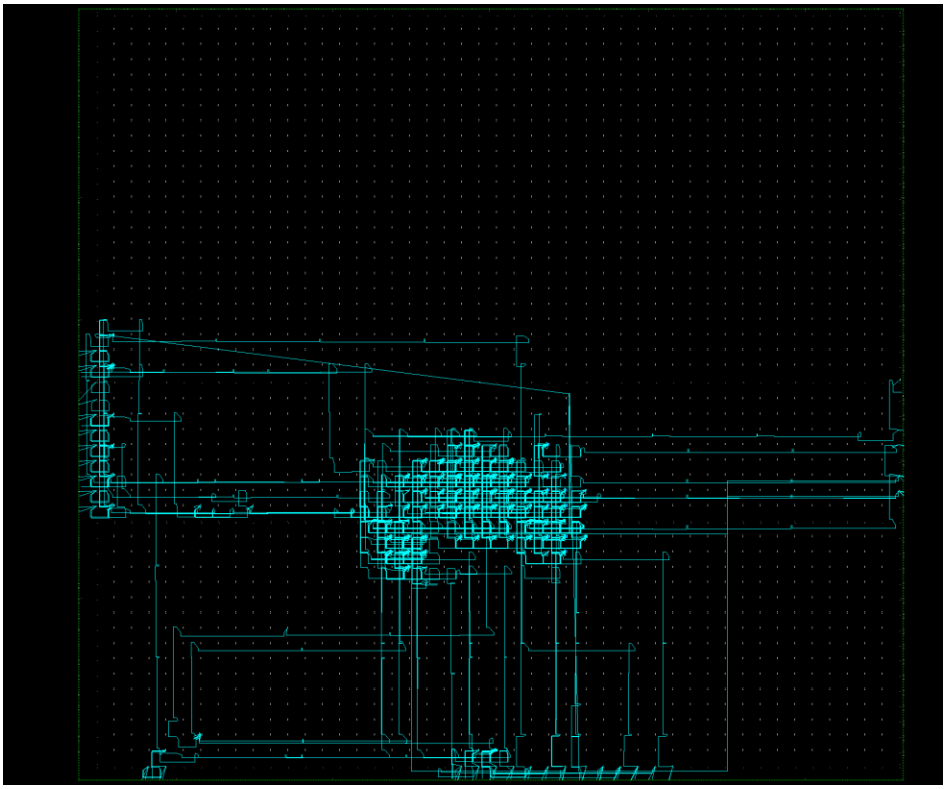

Figure 14: Physical view

4. Verification and validation

4.1 Testbench

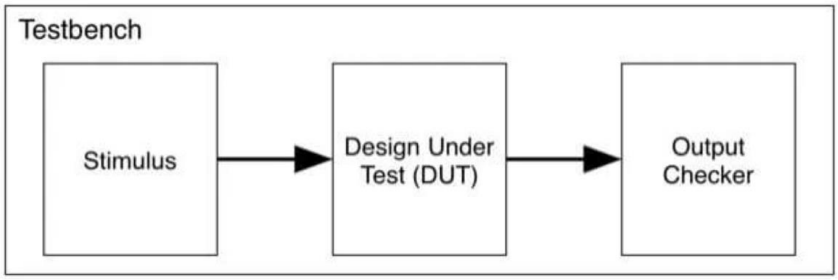

Figure 15: Testbench flowchart

The stimulus block generates the inputs to the FPGA design and a separate block checks the outputs. The stimulus and output checker will be in separate files. Used to stimulate the code and ensure that the functionality is correct. The testbench for this project imports a text file (Input.txt) which has the data in serial bit format and port maps the components in the design and generates the required output in the format of another text file (output.txt).

The serial data is converted into a 2 channel 24 bit stream using a simple python code. The format of this output file is of text and this text file is imported into matlab for reconstruction into a wav file for verification.

```
def outputfile_print(filename, toFilename):
    binary = ""
    i = 0
    count = 0
    channel = 0
    try:
        with open (filename) as infile:
            for charac in infile:
                if count < 24:
                    #if count == 23:
                    #    binary += " "
                    #    count += 1
                    #else:
                    binary += charac.strip()
                    count += 1
                else:
                    count = 0
                    channel += 1
                    print(binary)
                    try:
                        with open(toFilename, 'a+') as outfile:
                            if channel == 1:
                                outfile.write(binary)
                                i += 1
                                outfile.write('\n')
                            else:
                                channel = 0
                    except IOError:
                        print('Oops, no file by that name')
                    binary = ""
                    print(i)

        infile.close()
        outfile.close()

    except KeyboardInterrupt:
        print('You cancelled the operation.')

    except:
        print('An error occurred.')
        print('No file exists')

outputfile_print(r'output_serial.txt', r'output24.txt')
```

After the conversion to parallel data bit data the data is converted to a wav format file with the help of Matlab tool box and plotted using the available functions.

```
fs = 48000;
deltaT = 1/fs;
T = 0.5;
t = (0:deltaT:T-deltaT);
f1 = 20;
f2 = 80;
f3 = 140;
f4 = 200;
t1 = 0.2*sin(2*pi*f1*t);
t2 = 0.1*sin(2*pi*f2*t);
t3 = 0.3*sin(2*pi*f3*t);
t4 = 0.4*sin(2*pi*f4*t);
additiveSum = t1 + t2 + t3 + t4;
plot(t,additiveSum);
title('Summation')
figure;
quan = quantizer([16,15]);
bin1=num2bin(quan,t1);
bin2=num2bin(quan,t2);
bin3=num2bin(quan,t3);
bin4=num2bin(quan,t4);

% need to add the file and array and store it into c %
quan2 = quantizer([24,23]);
output = bin2num(quan2, output1);
channelStream = transpose(cell2mat(output));
plot(t,channelStream);
title('Gain multiplied summation')
audiowrite('output.wav',channelStream, fs);
```

4.2 Matlab code description:

Performs 4 different sinusoidal tone generation and plots the normal summation of the sinusoidal signals. With the output file generated from our test bench, the data is converted into bit stream with python and then converted to wav format using matlab and again imported into the same matlab file with the plot.

4.3 Comparison

Sr_in represents the serial data input and sr_ou represents the serial data output obtained from the device.

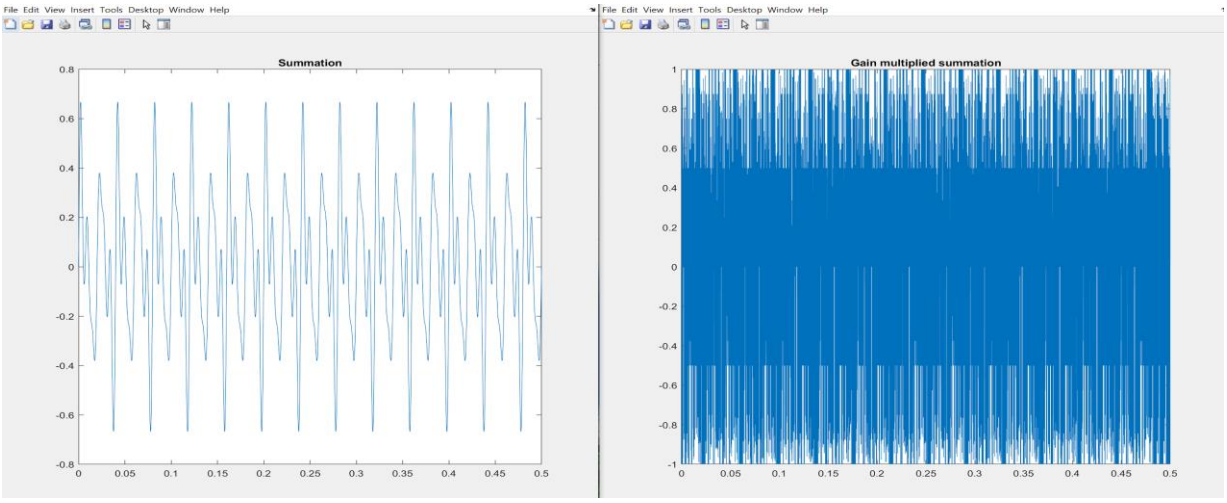

Figure 16: Comparison of input and output

4.4 Waveform generation in Modelsim:

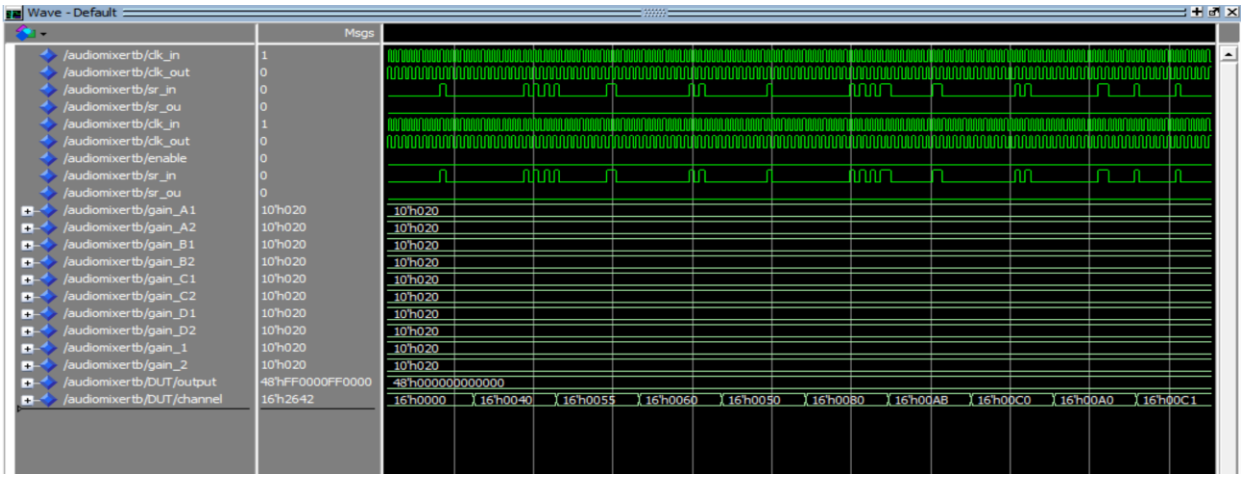

Figure 18: Sample waveform timestamp 1

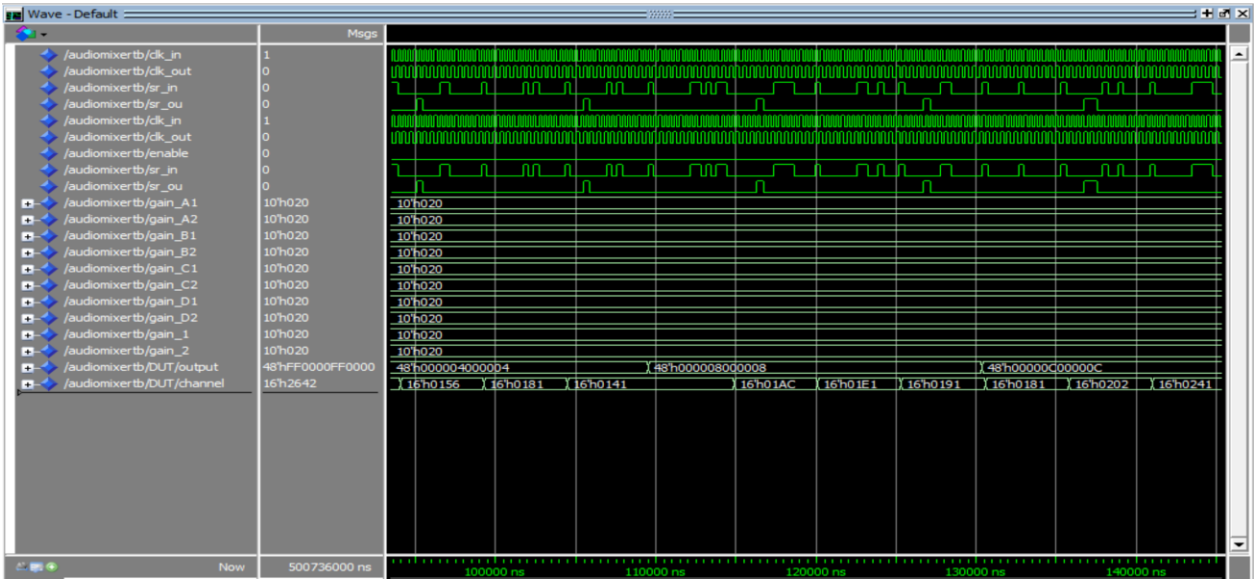

Figure 17 Sample waveform timestamp 2

5. Conclusion

Learnings and outcomes

The project deals with the design and synthesis of Audio Mixer device in FPGA technology Lattice XP2, and device: XP2 - 17 PQFP 208.

Tools used :

- ModelSim for design and testbench generation.
- Lattice Diamond for design synthesis.
- Anaconda spyder for serial data to parallel data conversion.
- Matlab for data reconstruction and wav file generation.
- Synplify pro for RTL view generation.

Main challenges faced:

- Generation of serial data input from wav files.
- Initial inception of the idea of the device.
- Testbench verification using text files.
- Reconstruction of wav file using matlab file.
- Understanding the concept of TDM.

Ideas implemented

Serialization of the bit stream using python. The file being a text file is a binary string data generated in matlab. Serial data output is converted into bit stream. Finally reconstruction of the tone from the bit stream using the UIimport command.

Advantages and effectiveness of the tools.

ModelSim is a multi-language environment by Mentor Graphics, for simulation of hardware description languages such as VHDL, Verilog and SystemC, and includes a built-in C debugger and this is very much versatile in hardware designs using HDLs.

Is it easy to use Lattice Diamond FPGA design and verification software environment (version 3.10 Service Pack 3) is certified as compliant with the IEC 61508 and ISO 26262 functional safety standards. These standards are widely used by developers in automotive and industrial applications, as OEMs require mission-critical systems used in their industrial control equipment and vehicles to deliver highly reliable performance with minimal system failures.Lattice Diamond FPGA Development Tool Receives Key Industrial and Automotive Functional Safety.

References :

https://www.geeksforgeeks.org/overflow-in-arithmetic-addition-in-binary-number-system/

https://chortle.ccsu.edu/AssemblyTutorial/Chapter-08/ass08_6.html

https://electronics.stackexchange.com/questions/143693/signed-addition-of-two-std-logic-vectors-while-looking-for-overflow-and-carry

https://stackoverflow.com/questions/20223869/vhdl-adding-two-8-bit-vectors-into-a-9-bit-vector

https://d3uzseaevmutz1.cloudfront.net/pubs/appNote/AN301REV1.pdf

http://www.sengpielaudio.com/calculator-gainloss.htm

https://en.wikipedia.org/wiki/Saturation_arithmetic

https://www.nandland.com/vhdl/examples/example-signed-unsigned.html

https://startingelectronics.org/software/VHDL-CPLD-course/tut14-VHDL-adder/