Fachhochschule Dortmund
University of Applied Sciences and Arts

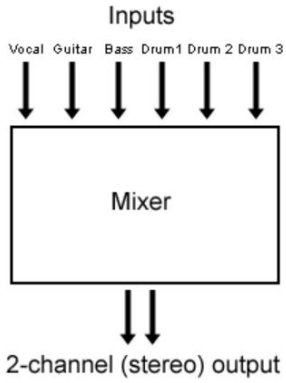
Audio Mixer

An Assignment Work for Microelectronics Hardware Software Co-design Course

Guided by Prof. Dr. Peter Schulz

Introduction

Audio Mixing is one of the major kinds of Signal processing and an interesting study of all time. We have moved and improved from a technology where Audio Mixers weren't compact, but enormous and ponderous, to the technology where even the AR/VR audio professionals could use it in their devices. The audio engineer, the maestro behind such developments, can be thought of being analogous to an orchestra conductor, ensuring that all of the individual audio sources mesh together.



Audio Mixer

An audio mixer is a device with the primary function to accept, combine, process and monitor audio. Apart from combining signals, mixers also allow to adjust levels, enhance sound with equalization and effects, create monitor feeds, record various mixes, etc. [2]

Applications:

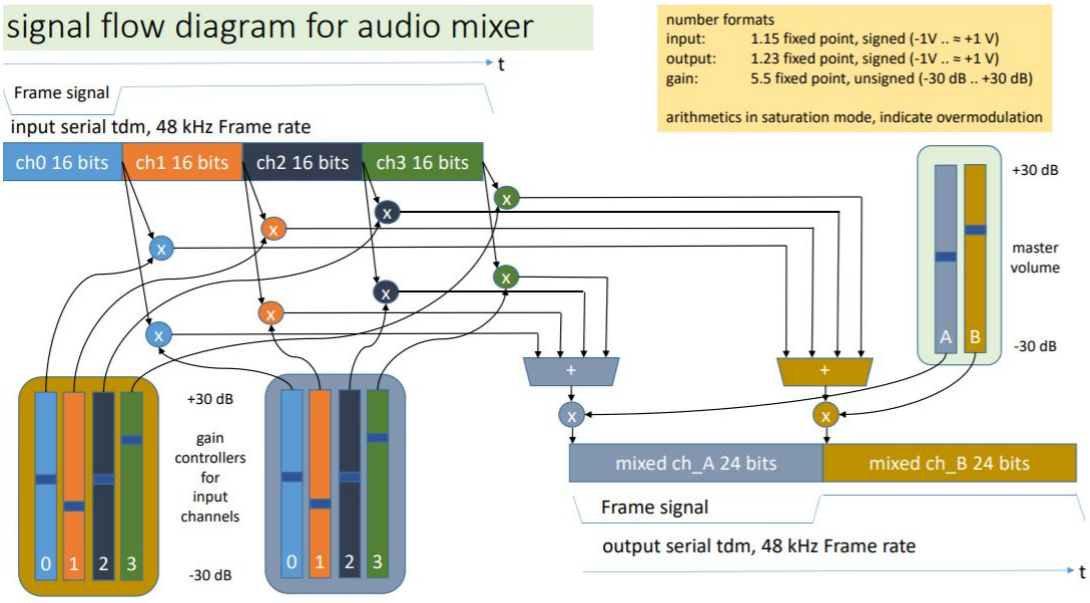
Some of the most common uses for sound mixers include:

Music studios and live performances: Combining different instruments into a stereo master mix and additional monitoring mixes.
Television studios: Combining sound from microphones, tape machines and other sources.
Field shoots: Combining multiple microphones into 2 or 4 channels for easier recording.

Authors:
Swathi Sudeendra Rao: 7207100; Mostafa Elmasry:; Mariam Jamal:7206873

Problem Statement

The given task is to design an audio mixer which has 4 input channels each of 16 bits. Each of these input channels are multiplied by different gain factors each of 10 bits for the target channel. Additionally, each channel is also multiplied by master volume control. These target channels are the sum of weighted input channels and are then sent as a TDM stream to generate a 24 bit output.



Requirements:

- Input:
  4 channels: 16 bit
  Serial TDM, 48kHZ Frame Rate
  Format: 1.15 fixed point, signed

- Output
  2 channels: 24 bit
  Serial TDM, 48kHZ Frame Rate
  Format: 1.23 fixed point, signed

Authors:
Swathi Sudeendra Rao: 7207100; Mostafa Elmasry: 7207073; Mariam Jamal:7206873.

- Gain Controller:
  -30dB to +30dB
  Format: 5.5 fixed point, unsigned

- Master Volume Controller:
  -30dB to +30dB

- Total number of multiplier instances should be 2

For simulation, sample audio wave files have to be generated using either MATLAB or Octave. The design can then be verified using VHDL testbench and Design under test in different modules.

Software and Tools

Lattice Diamond

The XP2 chip that we have created our design for, is made to go through HDL design flow of Simulation, Synthesis, logic analysis, Mapping, Place and Route. The tool has been used for verifying if the code can successfully undergo synthesis, could be mapped, placing and route design as per the given processor requirements. It gives the Synthesis Library support before the Synthesizer tool, Synplify Pro is used for it.

For the timing verification, Diamond tool gets the timing data from routing and follows a reverse engineering process to get back the intended result. Thus, this tool has served as the environment for design entry of the audio mixer.

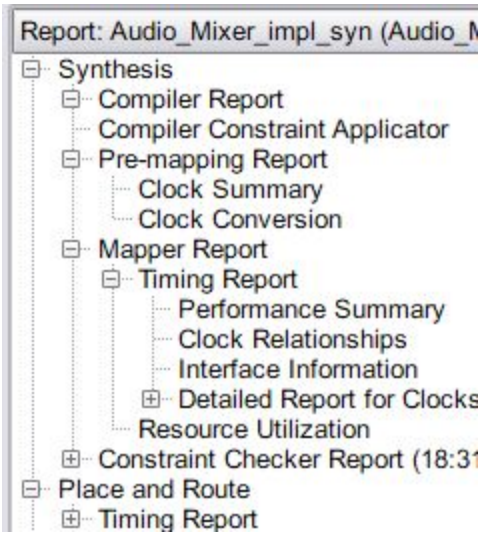
Matlab

Matlab is a very popular tool that is used for many mathematical calculation and simulation results. This takes and provides data in the form of a matrix that makes it more understandable to both the computing system as well as the users. Due to this very property of MATLAB, we could use it for converting our data into a desirable format.

The MATLAB libraries have a wide range of usage and we used to read a sound file from it and convert it into required 1.15 fixed point format input data with the given sampling rate. This means that the continuous sound waves were brought into the discrete data samples of 16 bit using this tool.

Synplify Pro

Synplify Pro was installed along with the Lattice tool and it has been used for Synthesizing the design. We made use of the tool for generating the RTL synthesis of the design. It has got Behaviour Extracting Synthesis Technology algorithms.



Since this is a large design, handling of timing constraints are very important. We don't need bad data or the loss of data. So, this serves as a tool for optimizing the design at a high level before RTL synthesis is performed.

Active HDL

This tool was used for compiling the design as it provides an integrated environment for the development of the VHDL code. This is the simulation tool that was used to generate the output waveform simulation with the help of advanced debugging features.

Solution

1. Input Sample Audio File

The audio files (mp3) cannot directly be used as an input and hence needed to be converted into binary bits before being fed into the input channels. Therefore, we used MATLAB for generating the input audio files.

Code for generating the binarized audio files using MATLAB:

```
filename = 'C:\Users\Swathi\Desktop\test.mp3'          %file with its location
[y1,Fs] = audioread(filename)                          %reading data into y1
```

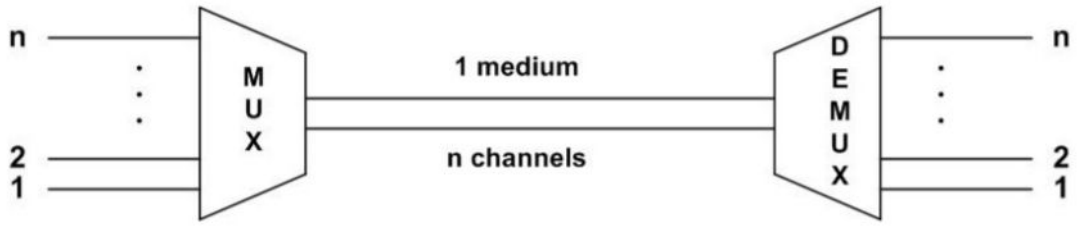
```
Fs2= 48000                                          %change the sampling rate
newfile = 'C:\Users\Swathi\Desktop\test.wav'         %create new .wav file
audiowrite(newfile,y1,Fs2)                          %write the same data into .wav file
[y2,Fs2] = audioread(newfile)                       %Read the data back with fs=48kHz
c1=y2(:,1)                                          %read ch1 data of newfile, test.wav
c2=y2(:,2)
y3=sfi(c1)                                          %convert into fixed point
y=bin(y3)                                           %convert to binary
t = cellstr(y)                                      %stringify all the 16 columns to one
                                                    %create/open a file myFile.txt to extract matrix t
fid = fopen('D:\MSc-ESM\2nd Semester\ME HW -SW Co-Design\Ref materials and homework\myFile.txt', 'w')
fprintf(fid, '%s\n', t{:})
fclose(fid)
```

2. Serial input with Time Division Multiplexing

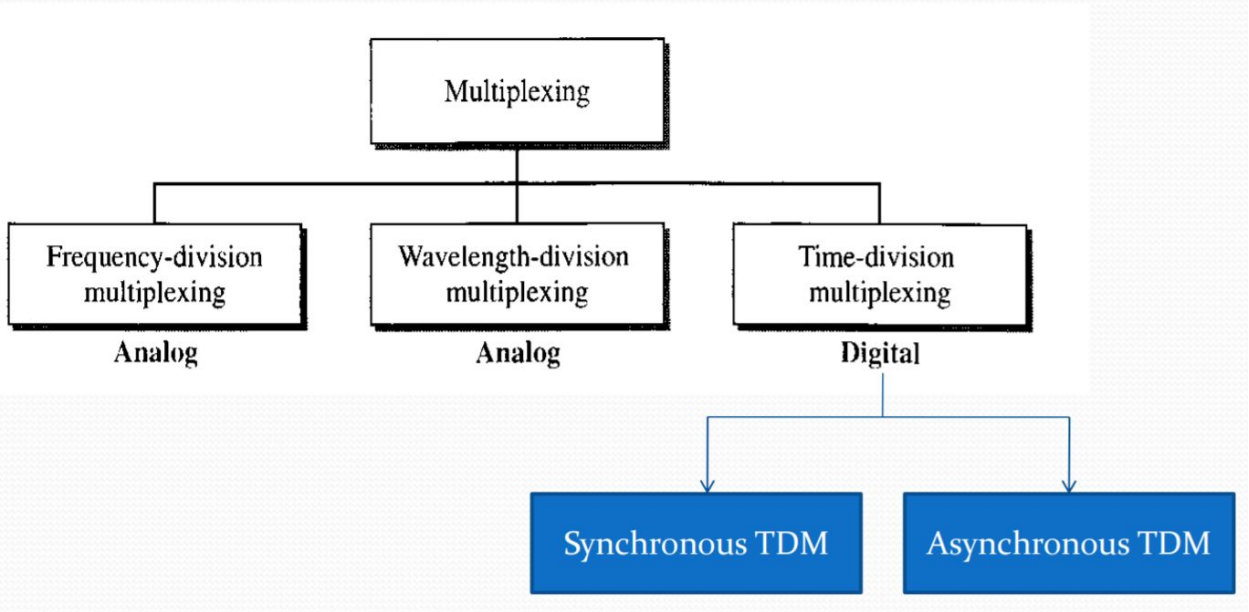
Once the audio file is converted into binary format, it needs to be serially fed using Time Division Multiplexing

Multiplexing:

It is the set of techniques that allows the simultaneous transmission of multiple signals across a single data link. Multiplexing is done using a device called Multiplexer (MUX) that combines n input lines to generate one output line i.e. (many to one). At the receiving end a device called Demultiplexer (DEMUX) is used to separate signal into its component signals i.e. one input and several outputs (one to many).

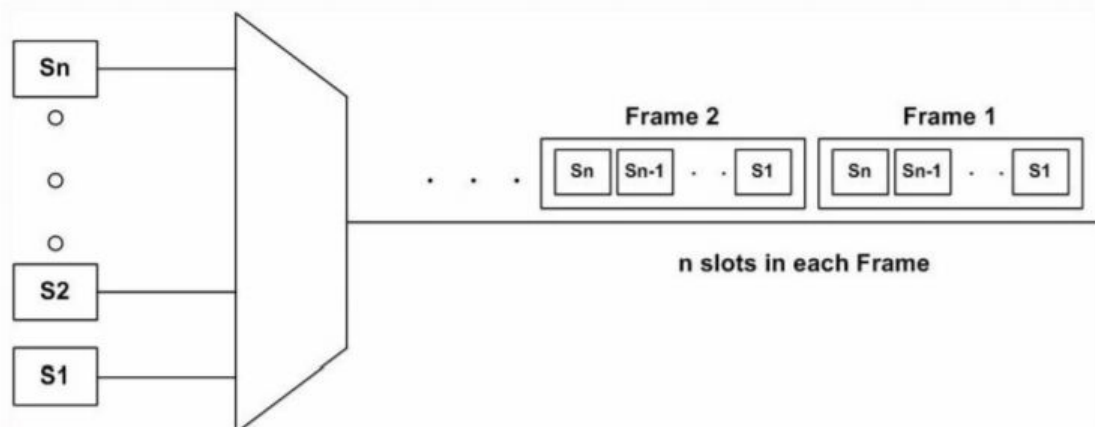


Types of Multiplexing:

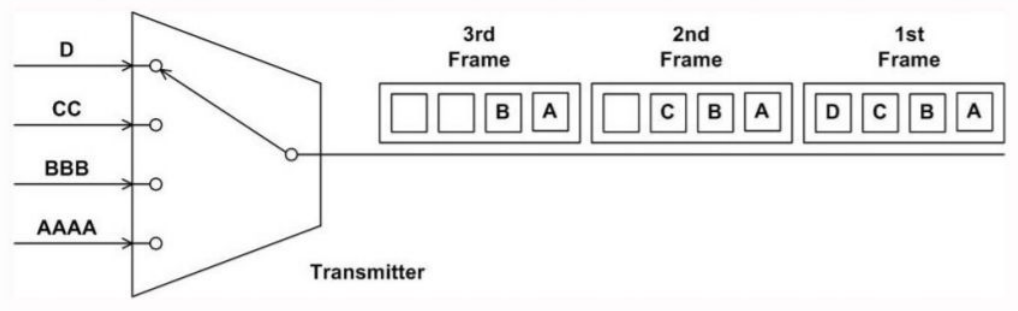


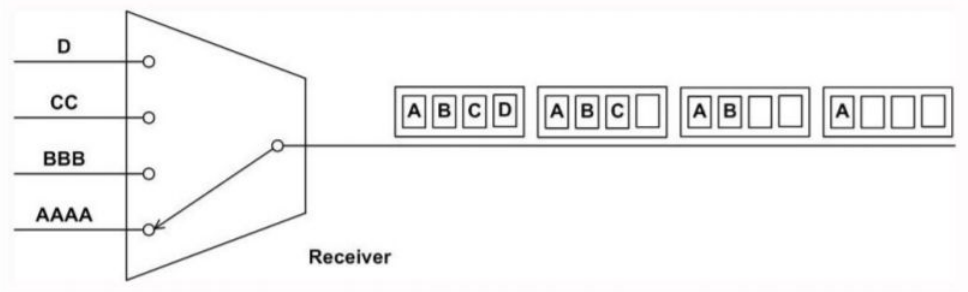
Time Division Multiplexing:

It is the digital multiplexing technique. Thd Channel/Link is not divided on the basis of frequency but on the basis of time. Here, the total time available in the channel is divided between several users. Each user is allotted a particular time interval called time slot or slice. In TDM, the data rate capacity of the transmission medium should be greater than the data rate required by sending or receiving devices.



Authors:
Swathi Sudeendra Rao: 7207100; Mostafa Elmasry: 7207073; Mariam Jamal:7206873.





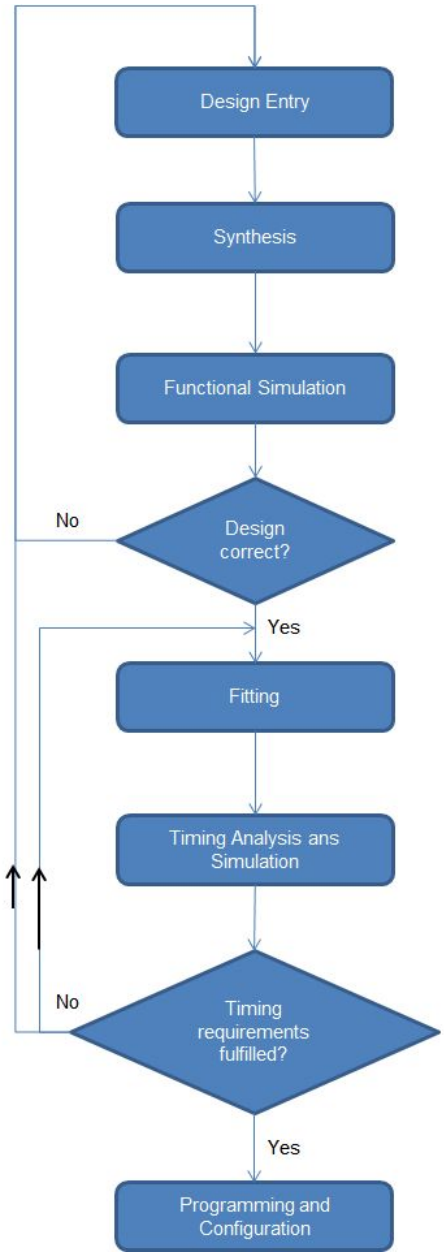
Types of TDM:

1. Synchronous TDM
    - Each device is given the same Time Slot to transmit the data over the link, whether the device has any data to transmit or not.
    - Each device places its data onto the link when it's Time Slot arrives, each device is given the possession of line turn by turn.
    - If any device does not have data to send then its time slot remains empty.
    - Time slots are organized into Frames and each frame consists of one or more time slots.
    - If there are n sending devices there will be n slots in the frame.

2. Asynchronous TDM
    - Also known as Statistical Time Division multiplexing.
    - In this time slots are not Fixed i.e. slots are Flexible.
    - Total speed of the input lines can be greater than the capacity of the path.
    - In ASTDM we have n input lines and m slots i.e. m less than n (m<n).
    - Slots are not predefined rather slots are allocated to any of the devices that has data to send.

Design

The basic HDL design could be explained as follows:



1. Design Entry: This is the VHDL code that has been written (referring to the 5 units that have been created). This could also have been created by the means of a schematic or with the help of any other hardware design creator.
2. Synthesis: Logic elements are created and thus the vhdl design is synthesized into a circuit here

3. Functional Simulation: Without considering the timing issue, the functional correctness could be verified during this process. Active HDL was used for this process.
4. Fitting: The Synplify Pro tool checks if the design fits into the given IC, ie., checks for all the parts, IO constraints or any other peripheral constraints. For eg., if there are 200k LUTs, the design must not consume 250k LUTs. That would mean that it has to be redesigned.
5. Timing Analysis: Analysis of various delays are made here
6. Timing Simulation: The simulation of the design along with the compatibility to the device and keeping the timing constraints into consideration, is executed.
7. Programming and configuration

Tool used: Lattice Diamond, Synplify PRO
Device used: Lattice XP2- 17 PQFP 208

The data type for different signals are constrained as follows:
Input: 1.15 fixed point, signed (-1V .. ≈ +1 V)
Output: 1.23 fixed point, signed (-1V .. ≈ +1 V)
Gain: 5.5 fixed point, unsigned (-30 dB .. +30 dB)

Concept:

Fixed point notation:
For the signed numbers, to find the negative numbers in the decimal form, 2's complement is first taken and the result will be converted to decimal by binary weights multiplication and addition. The last bit (MSB) is represented as Signed bit, the bits before the decimal point are integer numbers and the bits after decimal points are fractional numbers.

To define a Signed fixed point numbers the syntax used is:
variables/signals signum : sfixed(a downto b)
Example of Signed number:
signal num : sfixed (4 downto -3) :="11101101"
2's complement of 11101.101 is 00010.011
Finally the Decimal equivalent is
1 × 2 2 + 1 × 2 −2 + 1 × 2 −3 = −4.375

For the Give input number format 1.15 fixed point, signed (-1V to approximately +1V) and this can be defined as:
signal input : sfixed (1 downto -15) := "0111111111111111"

the range is from 1.0000000000000002 = -110 to 0.1111111111111112 = +0.99996948210 with increment of 1/32768 and the initial value of 01111111111111112 = +0.99996948210

For the Give input number format 1.23 fixed point, signed (-1V to approximately +1V) and this can be defined as:
signal output : sfixed (1 downto -23) := "01111111111111111111111"
the range is from 1.00000000000000000000002 = -110 to 0.11111111111111111111112 = +0.9999998810 with increment of 1.192092896 × 10−07 and the initial value of 0.11111111111111111111112 = +0.9999998810

Unsigned Fixed Point Numbers

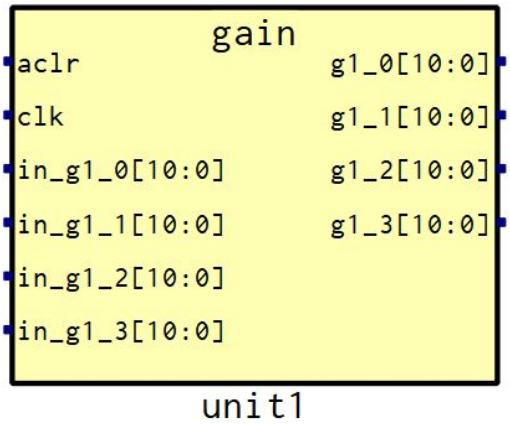
There is no negative number representation in the unsigned fixed point numbering, so the calculation is a bit simple when compared with signed fixed point numbers. For the conversation of binary to decimal numbers, binary weights multiplication and addition is performed.

To define a Unsigned fixed point numbers the syntax used is:
variables/signals unsigned num : ufixed(a downto b)
Example for unsigned number: signal num : ufixed (2 downto -6) :=“01001111”
01.001111 Decimal equivalent is 1 × 2 0 + 1 × 2 −3 + 1 × 2 −4 + 1 × 2 −5 + 1 × 2 −6 = 1.234375

For the Give input number format 5.5 fixed point, unsigned (-30 dB to approximately +30dB) and this can be defined as:
signal gain : ufixed (4 downto -5) := "1111100111"
the range is from 0 to 11111.111112 = 31.9687510 in an increment of 1 32 and with initial value as 11111.001112 = 31.2187510

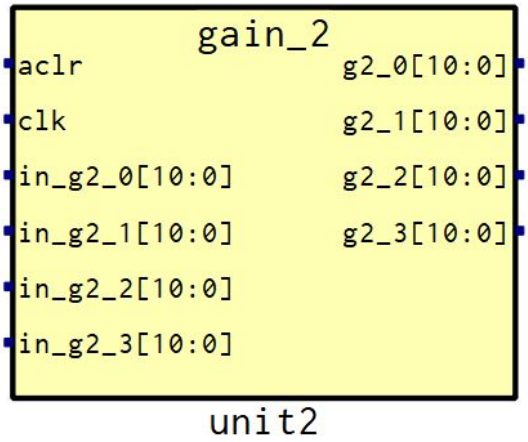
For the designing of the audio mixer, we approached it with a divide and conquer method. We defined different components as follows:

- Gain Controller 1




```
component gain is
PORT (
        clk :           IN STD_LOGIC
        aclr :          IN STD_LOGIC
        in_g1_0:        IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        in_g1_1:        IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        in_g1_2:        IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        in_g1_3:        iN STD_LOGIC_VECTOR (10 DOWNTO 0)
        g1_0:           OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
        g1_1:           OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
        g1_2:           OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
        g1_3:           OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
    )
End component
```

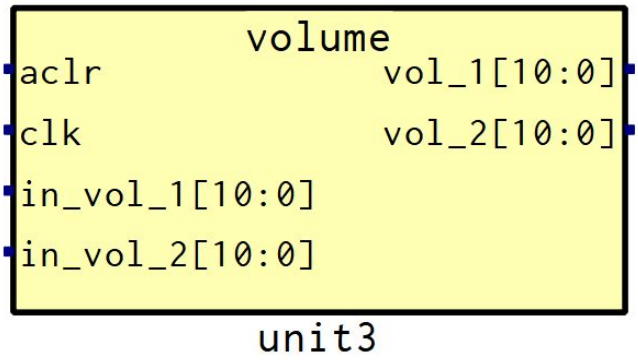
- Gain Controller 2




```
component gain_2 is
PORT (
        clk :           IN STD_LOGIC
        aclr :          IN STD_LOGIC
        in_g2_0:        IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        in_g2_1:        IN STD_LOGIC_VECTOR (10 DOWNTO 0)
```

```
        in_g2_2:        IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        in_g2_3:        IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        g2_0:           OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
        g2_1:           OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
        g2_2:           OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
        g2_3:           OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
    )
End component
```

- Volume Controller

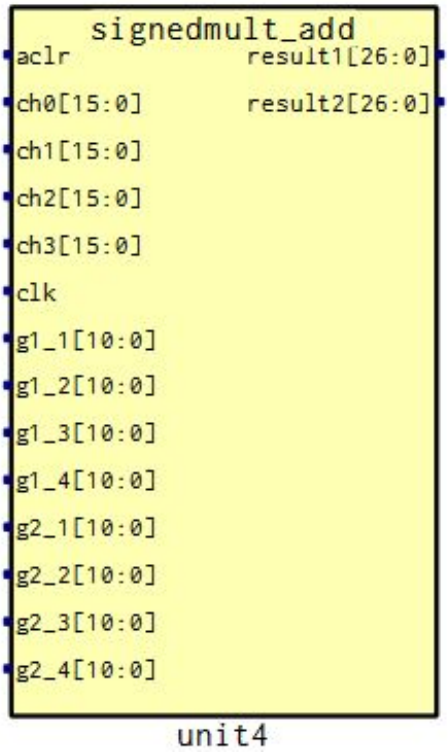



```
component volume is
PORT (
        clk :           IN STD_LOGIC
        aclr :          IN STD_LOGIC
        in_vol_1:       IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        in_vol_2:       IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        vol_1:          OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
        vol_2:          OUT STD_LOGIC_VECTOR (10 DOWNTO 0)
        )
End component
```

- Multiplier and Adder

  Since, the task had a constraint of using only two multiplier instances we constructed a multiplier + adder block.

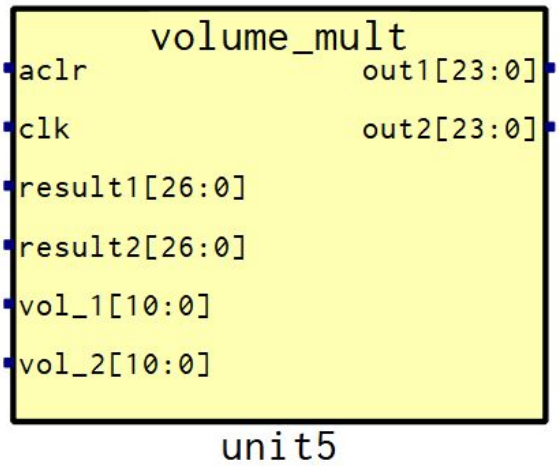
  This block first multiplies the input channels with different gain factors and then adds them together to give two outputs i.e result1 and result2.



component signedmult_add is

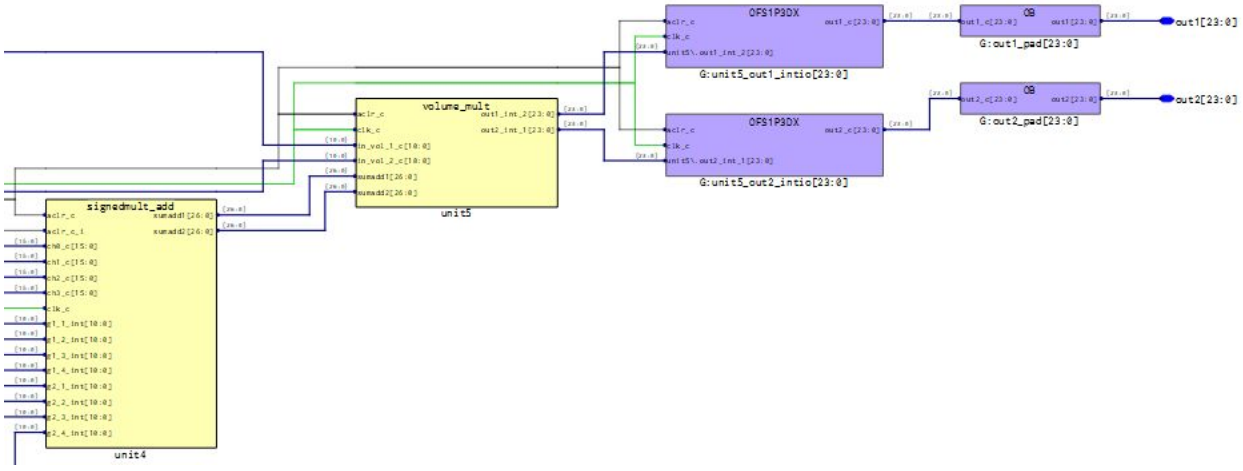
```
PORT(
        clk :           IN STD_LOGIC
        aclr :          IN STD_LOGIC
        ch0:            IN STD_LOGIC_VECTOR (15 DOWNTO 0)
        ch1:            IN STD_LOGIC_VECTOR (15 DOWNTO 0)
        ch2:            IN STD_LOGIC_VECTOR (15 DOWNTO 0)
        ch3:            IN STD_LOGIC_VECTOR (15 DOWNTO 0)
        g1_1:           IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        g1_2:           IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        g1_3:           IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        g1_4:           IN STD_LOGIC_VECTOR(10 DOWNTO 0)
        g2_1:           IN STD_LOGIC_VECTOR(10 DOWNTO 0)
        g2_2:           IN STD_LOGIC_VECTOR(10 DOWNTO 0)
        g2_3:           IN STD_LOGIC_VECTOR(10 DOWNTO 0)
        g2_4:           IN STD_LOGIC_VECTOR (10 DOWNTO 0)
        result1:        OUT STD_LOGIC_VECTOR (26 DOWNTO 0)
        result2:        OUT STD_LOGIC_VECTOR (26 DOWNTO 0)
        )
END  component
```

- Volume Multiplier
  This block generates the final output by multiplying the processed input channels from the above block with master volume.



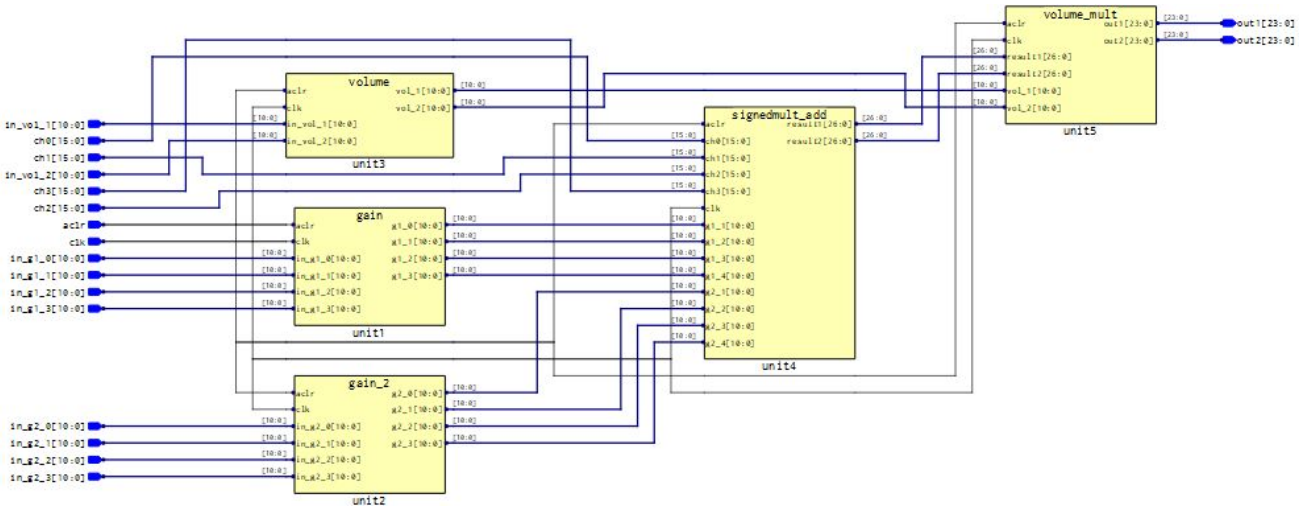
component volume_mult is
PORT (
clk : IN STD_LOGIC
aclr : IN STD_LOGIC
vol_1: IN STD_LOGIC_VECTOR (10 DOWNTO 0)
vol_2: IN STD_LOGIC_VECTOR (10 DOWNTO 0)
result1: IN STD_LOGIC_VECTOR (26 DOWNTO 0)
result2: IN STD_LOGIC_VECTOR (26 DOWNTO 0)
out1: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
out2: OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
)
End component

Technology view



Authors:
Swathi Sudeendra Rao: 7207100; Mostafa Elmasry: 7207073; Mariam Jamal:7206873.

RTL Schematics Diagram:



This is the RTL view for the design that has been made. The units that have been previously mentioned in the previous section.

Simulation result

Signal name	Value	229,44	229,48	229,52
aclr	0			
ch0	0012			0012
ch1	0270			0270
ch2	0319			0319
ch3	0402			0402
clk	1			
in_g1_0	00A			00A
in_g1_1	00F			00F
in_g1_2	014			014
in_g1_3	019			019
in_g2_0	010			010
in_g2_1	011			011
in_g2_2	012			012
in_g2_3	013			013
in_vol_1	002			002
in_vol_2	003			003
out1	018ED4			018ED4
out2	020B68			020B68

This is the simulation result for the sample of the inputs that were provided in the testbench.

Authors:
Swathi Sudeendra Rao: 7207100; Mostafa Elmasry: 7207073; Mariam Jamal:7206873.

ch0 = "0000000000010010b" x gain1_0 = "00000001010b" (10db) = 18d × 10d = 180d
ch1 ="0000001001110000b" x gain1_1 ="00000001111b" (15db)= 624d × 15d = 9360d
ch2 ="0000011100011001b" x gain1_2 ="00000010100b" (20db) 793d × 20d = 15860d
ch3 ="0000010000000010b" x gain1_3 ="00000011001b" (25db) 1026d × 25d = 25650d
SOP for the above is 51050d × volume_1 ="00000000010b" (2d) = 102,100d = **1 8ED4**

Similarly the SOP times the volume gain for the same input would be **20B68**.

Future Work

- After getting the bitstream with the correct timing constraints, one can verify the correct working of code that has been designed could be actually implemented on the Lattice XP2 IC.
- There could be a lot of other features added for the optimization of the output results by using noise cancellation techniques.
- An UI could be developed based on this design so that the commoners could use this in an everyday life.

Bibliography

[1] https://www.mediacollege.com/audio/mixer/intro.html
[2] https://www.techopedia.com/definition/9669/time-division-multiplexing-tdm

Authors:
Swathi Sudeendra Rao: 7207100; Mostafa Elmasry: 7207073; Mariam Jamal:7206873.