**Logo

Description automatically generated**

Department of Electrical Engineering

University of Cape Town

**EEE4120F**

**High-Performance Embedded Systems**

**YODA Project**

Audio Min-Maxing with FPGA

|  |  |  |
| --- | --- | --- |
| Team Leader | Christal Sima | SMXCHR002 |
|  | //Description here | |
| Co-ordinator | Joyce Ejumu | EJMJOY001 |
|  | Puts together and proofreads the final documents before submission. | |
| Secretary | Phalo Mathungana | MTHPHA010 |
|  | Takes minutes and reads them before following meeting. | |
| Archivist | Tyran Naidoo | NDXTYR006 |
|  | //Description here | |

# Project Description

## Introduction

The purpose of this project is to design, simulate and test and FPGA that can take in an audio input and output its minimum and maximum values. Furthermore, the FPGA should be able to take a user input of sampling intervals and sample an audio input at this rate. It should then be able to filter out values that fall one standard deviation outside the mean values. Finally, the filtering process will be implements for both the minimum and maximum values, and result in and two sets of output.

## Section 1 – Basic Min-Max Audio Processing

The FPGA should be able to find determine the minimum and maximum values of an audio input. To do this, integer values for minimum and maximum values are initialised and set to zero. These are then compared against the input values and replaced by values that are greater than the current maximum, or less than the current minimum.

A simple Algorithm achieving this is shown below.

Data = Read Data From location

Max = Data(0)

Min = Data(0)

For i = 1:(length(Data) – 1)

If Max < Data(i) then

Max = Data(i)

Else If Min > Data(i) then

Min = Data(i)

End

End

## Section 2 – Interval Min-Max Audio Processing

While most of the functionality for section 2 is like that outlined in section one, there are a few differences. The first being that a user can input an interval range at which they want the data sampled. Once this is specified by the user, the FPGA records values at the specified interval sample rate.

Following this, the FPGA then calculates the standard deviation and mean of the audio input. The input values are then compared against the standard deviation and mean to filter out the values that fall one standard deviation outside the mean value.

Finally, two sets of values are output. One set being the maximum values, and the other being the minimum values that fall within one standard deviation of the mean.

Building from section 1, the algorithm is adapted to determine an array of the minimum and maximum values for separate intervals.

Data = Read data from location

X = N //Some interval length

\*/Error Checking to confirm that length(Data)/X is integer /\*

For k = 0:(length(Data)/X – 1)

Max(k) = Data(k\*X) //First element of data block

Min(k) = Data(k\*X)

For i = 1:X

If Max(k) < Data(k\*X + i) then

Max(k) = Data(k\*X + i)

Else If Min(k) > Data(k\*X + i) then

Min(k) = Data(k\*X + i)

End

End

From the arrays of minimum and maximum values obtained for the audio signal. The mean and standard deviation are calculated according to the equations:

SumMax = 0

SumMin = 0

SumMaxTerms = 0

SumMinTerms = 0

For i = 0:(X-1)

SumMax += Max(i)

SumMin += Min(i)

End

AvgMax = SumMax/X

AvgMin = SumMin/X

For i = 0:(X-1)

SumMaxTerms += (Max(i) - AvgMax)^2

SumMinTerms += (Max(i) – AvgMin)^2

End

StdDevMax = sqrt(SumMaxTerms/X)

StdDevMin = sqrt(SumMinTerms/X)

For i = 0:(X-1)

If Max(i)>=(AvgMax-StdDevMax) AND Max(i)<=(AvgMax+StdDevMax) then

FltMax.append(Max(i))

End

If Min(i)>=(AvgMin-StdDevMin) AND Min(i)<=(AvgMin+StdDevMin) then

FltMin.append(Min(i))

End

End

FPGA’s can very effectively process audio inputs via a set of codecs as well as our internal algorithms. Companies such as Nexys and Xilinx have great on-board support for a receiver and codex system. Once these sounds are on board and identified by the FPGA system the sounds can be processed via a codex. The Nexys boards can read signals via an on-board audio amplifier, which is part of the simulation on Vivado, and hence can be coded using verilog to activate and read a simulated audio input. It is then encoded by the onboard codec to be readable by the board and the results can be seen on Vivado and further processing, mentioned in, the above algorithms, can be executed within Vivado, i.e., the board.

# Prototype Specification

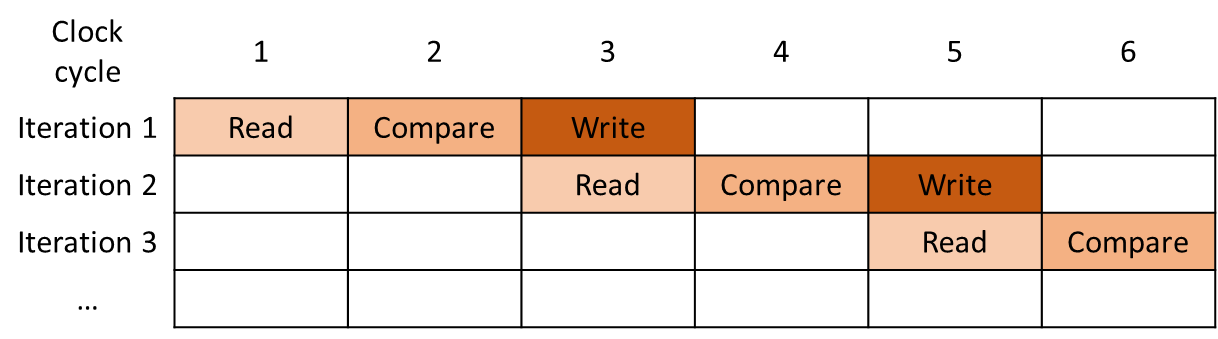
## Section 1 – Basic Min-Max Audio Processing

As this project is primarily simulated, the choice of simulation software is of high regard, seeing that the integration and functionality is a large factor on how effectively the system can be designed and tested. For this reason, we have chosen Xilinx Vivado as our simulation software.

This choice came down to multiple factors, including how close to the real device we could get the simulation, the accuracy and response time as well as the compatibility and modularity of the different simulacra. The Vivado suite reigned on top in these numerous factors. First, it was able to utilize a maximum potential with regards to logic board capabilities, allowing the simulation to closely mimic the real board. This allows the architecture and hardware to seamlessly function together with minimum clashing.

Vivado is also an extremely robust program, utilising proficient algorithms to bypass the bottleneck problems that many similar programmes face. It also keeps the results consistent between high value logic processors and low device utilization to evenly distribute the logic.

The algorithm presented above in section 1 is somewhat coarsely grained. However, while this entails those operations may take place independently, the reading and writing to certain memory blocks cannot take place simultaneously. The algorithm presented may be optimised by the use of pipe-lining to ensure that while multiple operations may take place, reading and writing of certain memory spaces do not overlap. An illustration of this is presented below.



## Section 2 – Interval Min-Max Audio Processing

The algorithm presented above for Section 2 can be approached in a similar manner as with section 1. However, this algorithm also features iterations that can be freely performed simultaneously. Therefore, the outer most loops can be separated into parallel processes.

## Implementation

For the purposes of this project, the group has decided to simulate the FPGA rather than building it on a physical board. This will be done using software programs such as Vivado. This decision does mean that there will have to be thorough testing to ensure that the board will behave as expected when loaded onto a physical board. This will be done by creating a test bench that is able to test the accelerator and virtual peripherals, to yield simulation results that are like what would be present on a real board. The following development boards are made available for the purpose of this project.

**Nexys2**

The Nexys2 board is a development platform that is based on a Xilinx Spartan 3E FPGA. It features a high speed USB2 port, 16Mb of ROM and RAM. It also has an embedded processor such as the MicroBlaze used in the Xilinx FPGA.

**Nexys3**

The Nexys3 is based on the Spartan-6 LX16 FPGA. This development boards is made for high performance logic. In comparison to the Nexys2, the Nexys3 has a higher capacity, performance, and more resources available.

**Nexys4**

This is a development board based on the Atrix-7 FPGA. It has various USB, Ethernet, and other ports. The Nexys board can also host many unique designs of combinational circuits, powerful embedded processors, and various other applications. In addition to all of this, this board has built in peripherals such as an accelerometer, temperature sensor, and digital microphone a speaker amplifier amongst others.

**Nexys A7**

This development platform is based on the latest Artix-7 FPGA. It is a large high-capacity FPGA with external memory, and various ports. It also features an accelerometer, temperature sensor, digital microphone, speaker amplifier and other input out devices. Like the Nexys4, it can be used for a wide range of applications without the need for other components.

While each board presents their own merits, be usability, resources available or price of the FPGA, it should be noted that both the Nexys2 and Nexys3 are no longer manufactured by Xilinx. In addition to this, the boards are not readily available for simulation on Vivado. While they may be the more cost-effective option, they are the more difficult boards to find or simulate.

Therefore, the boards we shall mostly likely use for the duration of this project are the Nexys4 or Nexys A7.

# Project Goals

The first project goal is to be able to design, build and test an FPGA that can accurately output both the minimum and maximum value of an audio clip input. To fulfil this goal, the following smaller tasks have been outlined. The aim of these tasks is to form a team, understand the project, and work towards designing, building, and testing a working prototype by the end of the semester.

1. Form a team, assign roles, and agree on a consistent method of communication and sharing documents
2. Form a team that shall work together for the duration of the project
3. Define team roles, and assign a role to each person
4. Ensure all group members have access to the Microsoft team and can attend meetings using the agreed-upon method
5. Ensure all members have access to the shared documents for the project
6. Select a topic and decide on which board to use
7. Decide on a topic for the project based on the interests and skillsets of the group
8. Find out which boards are available for use
9. Look into the pros and cons of various boards
10. Look into ways in which various boards are used
11. Brainstorm ideas and discuss a way to approach the project
12. Look into methods of simulation of the board
13. Decide on an algorithm
14. Decide on what tasks for which each team member will be primarily responsible
15. Think about ways to test the prototype
16. Do individual research to gain a deeper understanding of the project and how best to approach it
17. Research on the theory behind FPGAs and how they function
18. Research on ways in which they are used
19. Look into similar projects and how they were approached
20. Meet with the assigned team manager, and discuss any progress made on the design
21. Ensure that the team correctly understands what the project entails and what is expected for the prototype and final report
22. Discuss progress made with the team manager
23. Ask any questions that may arise
24. Discuss the best ways in which to contact the team manager when necessary
25. Write up a draft report
26. Build and test design
27. Ensure all goals set for the functionality of the FPGA are achieved
28. Find and note any issues that arise
29. Make any necessary improvements to the design based on the results of the testing
30. Create a final submission and work on a presentation and demo

## Section 1

1. The FPGA should be able to accurately output both the minimum and maximum value of an audio clip input
2. The FPGA should be able to read the input of an audio clip
3. From the audio clip, the FPGA should be able to find both the minimum and maximum value of the audio clip input
4. The FPGA should then be able to output the determined input and output values
5. These results must be accurate for various audio clips
6. The FPGA should be faster than the EDA playground equivalent
7. The time taken for the EDA to yield results will be recorded for both a cold start and after being run a few time
8. The time will then be compared to the EDA playground equivalent and golden measure, to ensure that the FPGA is not only accurate but faster than a simulation
9. The FPGA must be portable to board
10. Must be able to transfer from a simulation to the physical board
11. The FPGA sound sampling prototype should process information at least 1.2x faster than it is equivalent DSP software processing method.

## Section 2

The goals for section 2 of the project will be like those set out for section 1. There are however a few additional goals. These are listed below.

1. The FPGA should be able to accurately output both the minimum and maximum for a provided interval length value of an audio clip input
2. The FPGA should be able to sample at the audio an interval specified by the user
3. There must be a way in which a user can put an input
4. The FPGA must be able to sample at this rate
5. The FPGA should be able to calculate the mean and standard deviation of the audio signals
6. The FPGA must be able to store data for an interval
7. It must be able to calculate the mean of the signal based on this data
8. Lastly, the FPGA must be able to calculate the standard deviation of the audio input
9. The signals that are one standard deviation outside the mean of the audio clip should be filtered out
10. Using the calculated mean and standard deviation, the FPGA must be able to calculate what values fall outside of the range of one standard deviation from the mean
11. The values that fall outside of this range must then be filtered from the output
12. Two sets of data for the maximum and minimum points must be the output of the FPGA
13. Based on the values that remain after filtering, there should be two sets of values that fall within one standard deviation of the mean value
14. These values should be output
15. The results must be accurate for various input audio signals
16. These results must be accurate for various audio clips
17. The FPGA should be faster than the EDA playground equivalent
18. The FPGA must be portable to the physical board

# References

*ADAU1761 Datasheet and Product Info | Analog Devices*. (n.d.). Retrieved April 29, 2022, from <https://www.analog.com/en/products/adau1761.html>#

*Audio processing - FPGA - Digilent Forum*. (n.d.). Retrieved April 29, 2022, from <https://forum.digilent.com/topic/4972-audio-processing/>

*Audio Processing with the Snickerdoodle - Hackster.io*. (n.d.). Retrieved April 29, 2022, from <https://www.hackster.io/adam-taylor/audio-processing-with-the-snickerdoodle-727c40>

*Nexys 2 - Digilent Reference*. (n.d.). Retrieved April 29, 2022, from <https://digilent.com/reference/programmable-logic/nexys-2/start#:~:text=Nexys2%20circuit%20board%20is%20a,a%20Xilinx%20Spartan%203E%20FPGA>

*Nexys 3 - Digilent Reference*. (n.d.). Retrieved April 29, 2022, from <https://digilent.com/reference/programmable-logic/nexys-3/start>

*Nexys 4 - Digilent Reference*. (n.d.). Retrieved April 29, 2022, from <https://digilent.com/reference/programmable-logic/nexys-4/start>

*Nexys A7 - Digilent Reference*. (n.d.). Retrieved April 29, 2022, from <https://digilent.com/reference/programmable-logic/nexys-a7/start>

Reiche, O., Özkan, M. A., Hannig, F., Teich, J., & Schmid, M. (2017). Loop Parallelization Techniques for FPGA Accelerator Synthesis. *Journal of Signal Processing Systems 2017 90:1*, *90*(1), 3–27. <https://doi.org/10.1007/S11265-017-1229-7>

//Below is work outline – above is structure blog doc

# Project Description

//Section Description – Joyce

// Algorithm Specification - Christal

//Benefits of processing with FPGAs – Phalo

//Ways in which FPGAs can process sounds efficiently -Tyran

# Prototype Specification

//Simulation Software/choice and reasons/closeness to reality – Tyran

//Should move faster than benchmark (Our original Algorithm) – Phalo

//Algorithm Break down – how to parallelize it – Christal

//Specify the board we will – Specify whether will simulate or implement on real board, info about the board being used -Joyce

# References - Tyron

//Text here