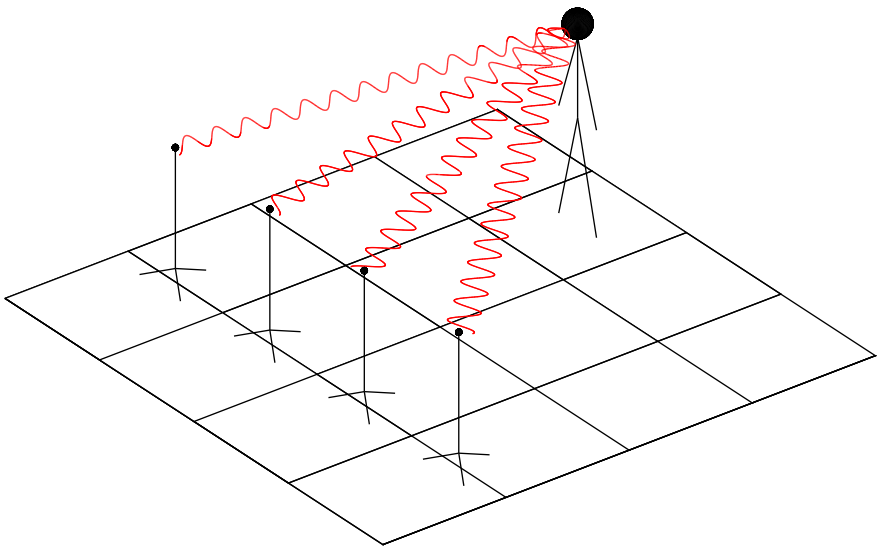


Speech Enhancement for Theatre Stages Using Linear Microphone Array

DAT096 Embedded system design project
Group 5

Daniel Eliasson
Weihan Gao
Aravind P S
Ammar Shihabi
Lexuan Tang

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

DAT096 Embedded system design project 2023

Speech Enhancement for Theatre Stages Using Linear Microphone Array

Daniel Eliasson
Weihan Gao
Aravind P S
Ammar Shihabi
Lexuan Tang

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Speech Enhancement for Theatre Stages Using Linear Microphone Array
Daniel Eliasson
Weihan Gao
Aravind P S
Ammar Shihabi
Lexuan Tang

Examiner, scrum & team support: Lena Peterson, Embedded Electronics Systems and Computer Graphics
Co-product owner & technical expert: Sven Knutsson, Department of Computer Science and Engineering

DAT096 Embedded system design project 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Speech Enhancement for Theatre Stages Using Linear Microphone Array
Daniel Eliasson, Weihan Gao, Aravind P S, Ammar Shihabi, Lexuan Tang
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This report presents the design and implementation of a speech enhancement system for a theatre stage using linear microphone array. The implementation is done using: Four microphones, an ADC, an FPGA and a DAC. Two different algorithms are introduced; the first implementation determines the output based on a power estimation calculation from the loudest microphone. The second implementation uses multichannel cross-correlation for position localization, to later compensate for attenuation due to distance. A reference model for both algorithms has been designed and simulated in MATLAB. The models are then implemented on FPGA, and later verified through some formal verification methods. The system is designed upon having an audio path and a control path. This approach makes the audio delay close to zero. Moreover, alternating between the two implementations becomes more practical.

In conclusion, we have a system with two working implementations. The quality and complexity of the two algorithms is exponentially correlated to the cost of hardware. The output results of the algorithms after being implemented on FPGA is very consistent with the results from the MATLAB simulation model. The performance of the algorithms is evaluated by analyzing the system behavior in MATLAB simulation. After implementation, the audio quality is evaluated by subjective perception. But in general, the audio quality of the system as well as the audio delay is pleasant and inside acceptable range for the listener.

Keywords: *Speech enhancement, microphone array, FPGA, cross correlation, power estimation, ADC, DAC, DSP, digital filters, SNR.*

Acknowledgements

We would like to thank Xu Wang for the big technical help as well as for the good discussions we had throughout the project. They have opened many locked doors during the development of the complex algorithm.
We would like to thank Lena Peterson for the help and discussions provided during weekly meetings.
We would like to thank Sven Knutsson for the technical lecture and borrowing of microphones.
We would like to thank Becky for the thoughtful lectures and discussion on team dynamics.
We would like to thank Anne Hsu Nilsson for the lectures and discussions on writing.

Daniel Eliasson, Weihan Gao, Aravind P S, Ammar Shihabi, Lexuan Tang
Gothenburg, May 2023

Contents

Abbreviations and acronyms

A/D : Analog to Digital
ADC : Analog to Digital Converter
D/A : Digital to Analog
DAC : Digital to Analog Converter
DSP : Digital signal processing
FIR : Finite Impulse Response
FPGA : Field-Programmable Gate Array
IIR : Infinite Impulse Response
IC : Integrated Circuit
I^2C : Inter-Integrated Circuit
I^2S : Inter-IC Sound
LC1 : Left channel 1 (microphone 1)
LC2 : Left channel 2 (microphone 2)
NRDF : Non-Recursive Discrete Filter
RC1 : Right channel 1 (microphone 3)
RC2 : Right channel 2 (microphone 4)
RDF : Recursive Discrete Filter
TDOA : Time Difference Of Arrival
VLSI : Very Large-Scale Integration

1

Introduction

Speech enhancement system is quite common in entertainment environments, to provide better experience to the audience such as theatre. Microphones, as the input of the enhancement system are necessary. There are several problems with using a single microphone for each actor: First, the system is not scalable, which means that the number of microphones increases linearly with the number of actors. The authenticity of the play might be undermined since actors appeal will be affected by wearing microphones. When using lavalier microphones, actors may forget to turn off their equipment, causing inconvenience and unnecessary troubles. So a microphone array can gradually replace the single microphone in speech processing not only because it can avoid the problems mentioned earlier, but also because they have the ability to collect much more information. This offers the opportunity to carry out complicated operations, such as position localization and speech enhancement. In this report, we propose a speech enhancement system using linear microphone array built using multiple microphones positioned on different positions of the stage as inputs.

1.1 Background

The voice signal from the microphones need to be converted to digital signals using an analog-to-digital converter (ADC). The converted signals will be piped to a field programmable gate array (FPGA), where the algorithm is running. The result of the enhanced voice will be fed into a digital-to-analog converter (DAC). The output of the DAC is connected to a speaker to output the processed sound coming from the actors on the scene.

1.2 Purpose and aim

The purpose of this project is to implement an FPGA hardware solution for enhancing theatre audio. Four microphone inputs are used, where each microphone would receive the voice with different delay and different amplitudes. The microphone signals are sent to a digital signal processing algorithm to later output an enhanced and cleaner version of the talking actor. The talking actor can stand at any given place in the stage. The output should be a combination of the modulated microphone signals. We aim to have a stable output volume, with audio delay that's acceptable

for the listener. This implies that the output audio must not be significantly delayed through the speaker, as it would create an unpleasant experience for the listener.

1.3 Scope

We develop this system for indoor stage performance. The ideal behavior of the system will be providing a stable output when the actor is moving. The robustness of the system under limited noise and reverberation is also considered. Since our system is designed for indoor application, the performance will not be guaranteed in other test environment. The performance is also limited by the hardware platforms. One example is that the number of inputs will definitely limit the system performance, especially for position location.

1.4 Organization of the report

In introduction chapter, we bring up the question we need to solve, designing and implementing a speech enhancement system for theatre stage. And we briefly give an overview of the whole project.

In theory chapter, a theory background for the design is first established. Communication protocols related to the system are first introduced. Then, we will summarize the existing research on speech enhancement algorithm, as well as introduce the necessary tools for the design, such as power estimation, cross-correlation, position solver and digital filters.

In design and implementation chapter, the system is described from top to bottom in detail. We will start from a system overview and introduce the two parts of the system: the audio path and the control path. Then we will go into detail of design and implementation of audio path and algorithms. The key parameters which have effects on system performance will be specified at the end.

In result chapter, we will evaluate the performance of the system by presenting and analyzing the outcome from MATLAB simulation, behavioral simulation of register-transfer level (RTL) and hardware implementation.

In the conclusion chapter, we will have a conclusion on system performance and discuss the possibility to modify and optimize the system.

2

Theory

In this part, we provide necessary hardware knowledge for the whole system and theory support for our algorithm design.

For hardware part, we focus on the establishment of audio path from analog to digital converter (ADC) to digital to analog converter (DAC) through digital communication interface, and algorithm mapping in hardware. Field programmable gate array (FPGA) supports algorithm mapping for audio processing by leveraging plenty of available programmable logic units and memory resources. The hardware description at the RTL level, along with multiple interfaces on FPGA, enables the implementation of the underlying logic for transmission protocols. The audio path between ADC and FPGA utilize standard protocols, including I^2S, TDM, and LJ protocols. I^2C protocol and SPI are two typical serial communication protocols for control signal path of ADC. To get valid the scene audio data for subsequent algorithm test, Ethernet port with user datagram protocol (UDP) is considered as high speed data transmission interface between FPGA and personal computer (PC).

For theory part in the algorithm, we start from the overview of speech enhancement and discuss existing methods. Based on these methods, we design the two algorithms: One is a microphone selection algorithm which outputs the signal from the loudest microphone based on power estimation, which is implemented by root mean square method (RMS) in our system; The other one uses multichannel cross-correlation to localize the position of the actor so that we can compensate for attenuation. The principle of function blocks including cross-correlation, position solver and digital filters is also discussed in this part.

2.1 Audio Protocols

The audio data transfer between the transmitter and receiver in datapath is configured using an audio serial interface (ASI). PCM6240-Q1 supports Inter-IC sound (I^2S), time division multiplexing (TDM), and left-justified (LJ) audio serial interfaces.

2.1.1 I^2S protocol

Inter-IC sound (I^2S) is a standardized serial sound protocol that is used for digital sound applications. This protocol is mainly used in A/D, D/A applications for Very Large-Scale Integration (VLSI) and Integrated Circuits (ICs) [1]. Additionally, the main goal of the bus is to only capture audio data. Therefore, only three lines are required. The control commands are transferred separately to the transmitter. The transmitter then sends the serial data (SD) back to the receiver. The remaining two lines are the word select (WS or FSYNC) which sets the sample rate, and the master clock (MCLK or BCLK) that is a multiple of the sample rate. The higher the multiple, the higher quality is delivered for each sample. FSYNC specifies also which channel is read at a given time. The left channel is a corresponding zero in FSYNC, meanwhile right channel corresponds to a logical one. The serial data is transmitted with the MSB first in two's complement. Check Fig. 2.1 to see how the serial data is transferred from four input channels.

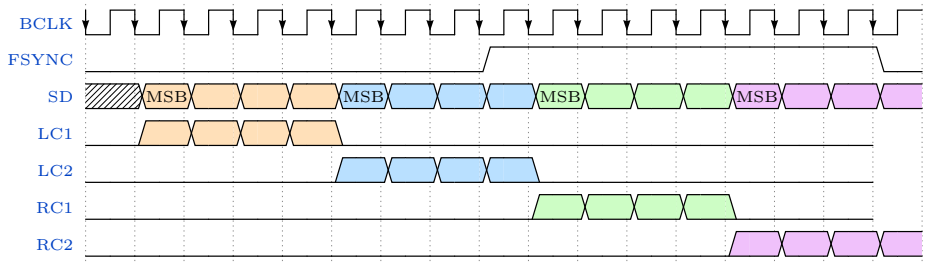


Figure 2.1: Timing of the I^2S protocol. The different channels for the number of microphones are shown as Left channel 1 to right channel 2. The data resolution is set to be four bits in this example

The bus can be configured to run in either master or slave modes [1]. In slave mode the transmitter gets the BCLK and FSYNC signals from a reciever, and based on that sends out the serial data. In master mode, all of the three lines are generated on the transmitter side. Fig. 2.2 shows how the bus can be controlled through different modes.

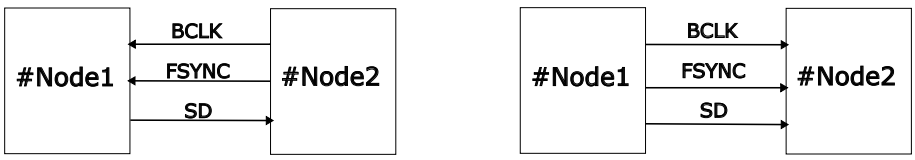


Figure 2.2: The slave respectively master configuration of the I^2S bus

2.1.2 TDM protocol

Time Division Multiplexing (TDM) protocol is used for transmitting and receiving multiple signals through a common channel. TDM protocol samples the signals from the transmitter at regular intervals and assigns specific time slots. Here, the time slot will be determined by the word length.

In TDM mode, the data transfer begins at the first rising edge of the BCLK after the rising edge of the FSYNC. Initially, the data from slot 0 will be transferred followed by the transmission of data from the rest of the slots. Each slot represents the microphones connected to the ADC. For using TDM mode for a higher BCLK frequency operation, a TX_OFFSET value higher than 0 is recommended. The timing diagram of the TDM protocol is shown in Fig. 2.3.

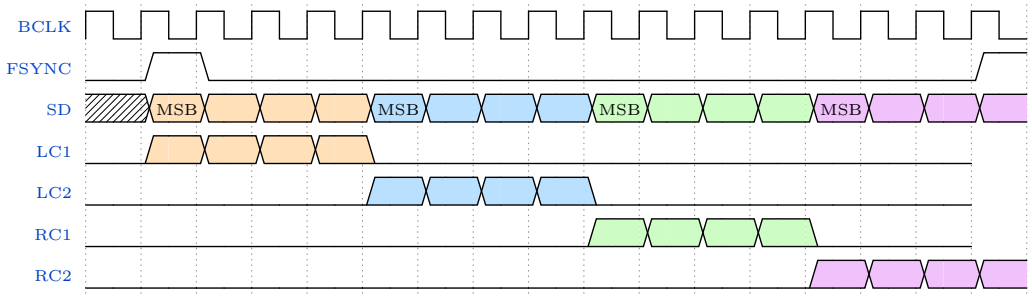


Figure 2.3: Timing diagram of TDM protocol.

2.1.3 LJ protocol

The left-justified (LJ) protocol is a commonly used protocol in ADC and DAC interfaces. This protocol uses two channels, left and right, for data transfer. LJ protocol would output each data in the falling edge of the clock.

The MSB of the left channel 1 will be transmitted in the following falling edge of BCLK after the rising edge of the FSYNC. The transmission of data will be shifted to the right channel on the falling edge of the FSYNC. The LJ protocol is implemented by configuring the two microphone channels in the left channel and the other two channels in the right channel. The timing diagram of the LJ protocol is shown in Fig. 2.4.

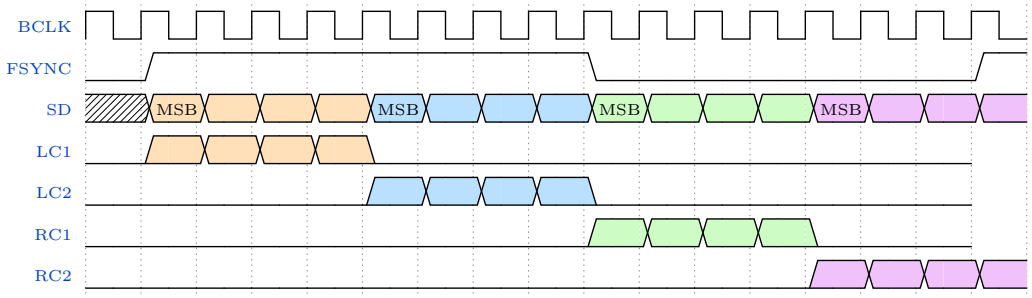


Figure 2.4: Timing diagram of left-justified protocol.

2.2 Control Protocols

The ADC provides several control protocols for customization of ADC parameters. The parameters will shape the datapath in timing, data format, number of channels and needs to be set before datapath work.

2.2.1 I^2C protocol

The selected ADC can be controlled through I^2C or SPI interface, making it easy to manage its registers. The I^2C protocol uses two wires - a serial data line (SDA) and a serial clock line (SCL) - to transmit data between devices. SDA is used to transfer data, while the SCL synchronizes data transmission and reception. Multiple masters and slaves can be connected on a single bus, with the master accessing the slave through its unique device address. As we are only using one ADC, we will only consider a single pair of master and slave.

We need to understand how data is transmitted and what kind of data is transmitted. Both SDA and SCL are high during idle state. When the transmission starts or ends, SDA transitions from high to low or low to high while SCL remains high. Then, SCL starts to change and SDA remains stable when SCL is high and can change only when SCL is low. Therefore, each SCL represents one bit of data. The effective data must be 8-bit long with MSB first. Each transmitted byte must be followed by an acknowledgment bit from the receiver, requiring 9 SCL cycles to transmit one byte of data.

To identify the location of the slave, the master sends an 8-bit addressing byte to the slave at the beginning, where bits [7:1] make up the slave's address, and the LSB bit is the data transfer direction bit. When it is '1' or '0', it indicates that the master is reading respectively writing data. Once the data transfer direction is determined, the master will transceive the data.

2.2.2 SPI protocol

The Serial peripheral interface (SPI) is a synchronous serial bus that is widely used in embedded electronic applications [2]. It is used to communicate between microcontrollers, sensors and other digital applications. The bus contains of four line: Serial clock (SCL), master out slave in (MOSI), master in slave out (MISO) and chip select (CS). This communication bus is bidirectional, meaning that both the controlling unit and the peripheral device can send and receive data at the same time.

The serial clock signal is derived from the master device and is used for synchronization. When data is about to be sent, the master device pulls the CS signal low to choose a certain device. It will later send out the SCL and the data through its MOSI output. The receiving device gets the data from MOSI as an SD input. At the same time, the receiver responds with data back to the transmitter through its MISO pin. Fig. 2.5 describes how the pins are connected to each other between the transmitter and the receiver.

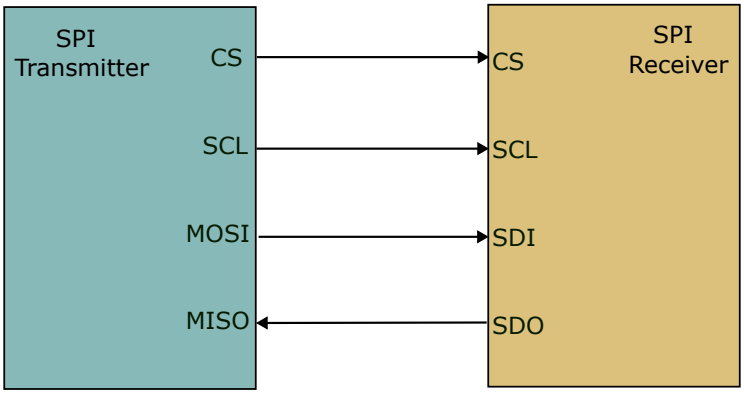


Figure 2.5: The connections of the SPI bus

There is no official rules of how to use the SPI [2]. Which means that timing specification of the bus can differ very much depending on the circuitry and the application. Therefore, sensors and other components usually include how fast the clock should be and how to use the SPI inside their datasheet. But in general SPI can operate at higher frequencies than I^2C.

2.3 Communication between FPGA and PC

The algorithm decides the quality of whole system, but the performance of the algorithm is highly dependent on input data. Therefore, the test data collected from the actual application environment, containing noise and reverberation, can help us improve the design.

Nexys 4 FPGA provides Ethernet and UART port to realize the data communication between FPGA and PC. However, it is impossible to completely transmit 64-bit sampling data within one sampling period (1/48 KHz) by UART. In contrast, high-speed Ethernet (100 MHz) with the corresponding protocol ensures no missing data and adequate time for communication with PC, where we can improve the algorithm according to the data we got.

In this part, we introduce the Open Systems Interconnection (OSI) model, how the information from layers of OSI is compressed into an Ethernet frame format and the tool to monitor and receive Ethernet data.

2.3.1 OSI Model

The Open Systems Interconnection (OSI) Model is the fundamental model for computer network communication. Its layered structure separates different protocols into different layers, so that it can simplify our design because we only need to consider the layers we need; in our case, the **concrete layers** in left part of Fig. 2.6. Based on this model, the data to be transmitted is encapsulated layer by layer from the top layer to the bottom layer, and finally reaches the physical layer (PHY), where it is sent to the destination physical layer through the network cable.

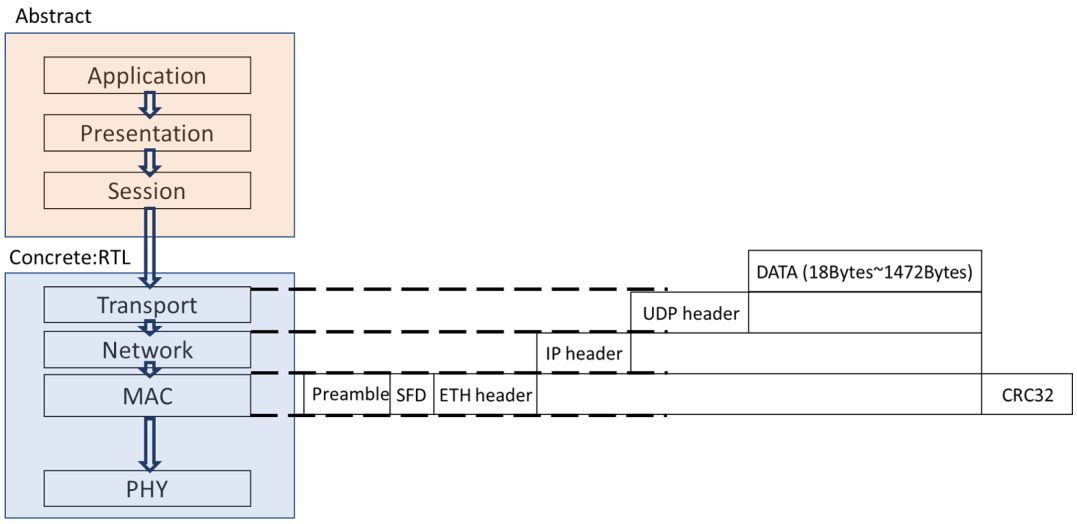


Figure 2.6: Layers in OSI model and relationship of Ethernet frame.

To implement Ethernet without abstract layers in FPGA, the concrete layers considered are the transport layer, network layer, and media access control (MAC) address layer. The Transport layer is responsible for providing data transmission

services between application processes (including different processes on the same terminal), and mainly defines which protocol will be used, the transmission control protocol (TCP) or the user datagram protocol (UDP). The UDP protocol has a faster transmission speed than the TCP protocol, so it is more suitable for real-time transmission of audio data. The Network layer adds an IP header to the data and provides connection services and data transmission services between different computer networks. The MAC layer is responsible for encapsulating the data from the Network layer into a frame format that can be transmitted by the physical layer (PHY) in reduced media independent interface (RMII) supported by FPGA.

2.3.2 Ethernet frame format

The right part of Fig. 2.6 shows the format of an Ethernet frame. As mentioned in section 2.3.1, only concrete layers are necessary for our system. The key information from three concrete layers, including data length, protocol information and address, is extracted and integrated into a header of an Ethernet frame, with valid data (18 Bytes at least) following. The Transport layer attaches UDP header to data directly since it is the highest layer in hardware when we select UDP protocol. It contains destination and source UDP address. IP header of 20 Bytes contains much detailed information on IP protocol and data type used before UDP header. And Ethernet header collects the MAC address and data length wrapped in front of Ethernet frame. To synchronize and sign the start, the preamble and Start of Frame Delimiter(SFD) are added at beginning. To ensure the accuracy of data transmission, Ethernet standard supports the use of 32-bit cyclic redundancy check (CRC-32) checksum for frame verification. Each Ethernet frame contains a 4-Byte CRC checksum that is used to verify whether there are any errors during the transmission of the 18 Bytes of data. The CRC-32 algorithm is capable of effectively detecting single-bit, double-bit, and multiple-bit errors.

2.4 Speech enhancement

Speech enhancement has been a well-covered topic for decades as a branch of speech processing, including other topics such as speech recognition. The goal of speech enhancement is using signal processing algorithm to improve the quality or intelligibility of the speech signal [3]. According to the Introduction part, our system aimed for an output with improved quality which matches the goal of speech enhancement. It is therefore reasonable to classify our system as a speech enhancement system.

Speech enhancement is a broad topic; noise reduction, reinforcement of specific signal, speech separation and other related subjects are all included in this subject. Noise reduction is the main concern at the early stage of research. Although characteristic of noise depends on specific application, additive noise and reverberation are two general types of noise in speech processing.

Two widely used approaches for reducing additive noise are spectral subtraction and signal subspace decomposition [3]. The key of both approaches are estimation algorithms. In spectral subtraction, we estimate the spectrum of noise and subtract it from the spectrum of combination signal of speech and noise to remove the normal broad band additive noise [4]. Signal subspace decomposition, a new method compared to spectral subtraction, directly uses a mathematical model to estimate the signal and noise based on principal component analysis to remove the noise, circumventing the frequency-domain [5]. During decades, researches have devoted to refine these two approaches by using different estimation methods, including Max likelihood estimation in 1980 [6], minimum mean square error estimation in 1984 [7], joint Maximum a posteriori estimation in 2001 [8] and weighted noise estimation in 2005 [9]. These methods are all based on the assumption that characteristics of the noise are unknown so we need to do some estimate of the signal and noise. In other words, if we have knowledge about the noise beforehand, we may just use normal digital filters to remove noise. Details about digital filters will be introduced in section 2.8.

Dereverberation is more complicated than additive noise since reverberation is highly related to source signal and environment. There existing two dereverberation methods applied on single-channel microphones: blind inverse filtering and dereverberation based on speech signal features [10]. Since ideally these methods should be adpative to test environment, so the available information is limited to the source signal itself, which leads to the limitation of the performance of these method. To improve past these limitations, researchers also exploit the relation between reverberation and speech harmonicity for dereverberation [11].

There is also a trend of using a microphone array instead of a single microphone in speech enhancement because the former one provides redundancy [12]. It is easy to imagine that the microphone array is just multiple microphones receive the same signal with different delay and attenuation. The redundancy brings more information for noise reduction and speech signal itself. Many theories on multichannel-signal processing have been proposed and verified in other area like radar and antenna systems, before being applied to speech processing [13]. One example is to use beamforming to combine signals in a way that signals from a specific direction is

enhanced. This redundancy is also beneficial for dereverberation. When we only have single channel it is impossible to estimate the impulse response, but if we have more microphones than the interference sources we could use mathematical models to estimate the impulse response and then remove the reverberation.

Another benefit brought up by the redundancy is sound source localization including both direction and distance, corresponding to two important parameters for sound source localization: time difference of arrival (TDOA) and direction of arrival [14]. In this project, we mainly focus on the estimation of TDOA. The first application of cross-correlation in time delay estimation is proposed in [15]. In the article, the author suggested that by converting the multiplication in time-domain to convolution in frequency domain and using other convolution processor with different weight may achieve better performance than normal cross-correlation. But the method to use cross-correlation in time delay estimation is verified and applied to many researches [16], [17], [18], [19]. We introduce the normal cross-correlation we use in our algorithm in section 2.6 and how we localize the sound source with cross-correlation in section 3.3.2.

2.5 Power estimation

The power of a signal can be calculated using the root mean square method (RMS) see (2.1).

$$power(x) = \sqrt{\overline{x^2}} \tag{2.1}$$

For the purpose of comparing multiple signals, seeking the signal with the largest power, the root is superfluous as a larger value also has a larger root.

When we look into the mean operation we can see that more simplifications is possible. The mean is done over 'n' samples, and we therefore divide by n but this is also superfluous as it's simply a linear scaling applied to every power estimate.

$$powerEstimate(x) = \overline{x^2} = \sum \frac{x^2}{n} \tag{2.2}$$

We can also look in to how the equation look at different points in time. Assume some general time $t = i$ then the following would be the equation for it and the sample just after.

$$PE(x(t = i)) = \sum_{t=i-n}^{i} x(t)^2 \tag{2.3}$$

$$PE(x(t = i + 1)) = \sum_{t=i+1-n}^{i+1} x(t)^2 \tag{2.4}$$

What we observe is that a lot of the sum is the same and we can rewrite the equation for the time $t = i + 1$ as a function of the previous sampled power.

$$PE(x(t = i + 1)) = PE(x(t = i)) + x(i + 1)^2 - x(i - n)^2 \tag{2.5}$$

A visual representation of the equation(2.5) is in Fig. 2.7.

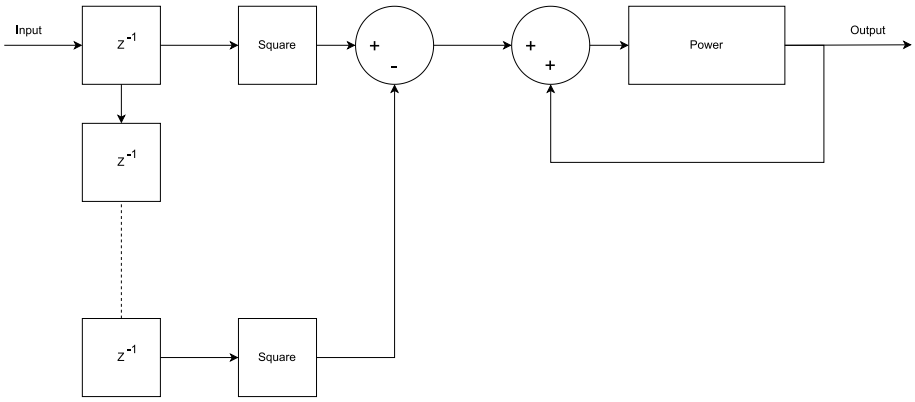


Figure 2.7: Block diagram of the power estimate function

2.6 Cross-correlation

Cross-correlation is the mathematical function in which two signals are compared and their similarity is measured. The general equation for the time continues and the time discrete variant is (2.6) respectively (2.7). Cross-correlation is a function of lag; t respectively n. The lag is the amount of time skew we apply to signal x to align it with signal y.

$$\phi_{xy}(t) = \int_{-\infty}^{\infty} x(t+\tau)y(\tau)d\tau \tag{2.6}$$

$$\phi_{xy}(n) = \sum_{m=-\infty}^{\infty} x[m+n]y[m] \tag{2.7}$$

One of the application of cross-correlation is to calculate the delay between two signals. By varying the lag across some range and observing the cross-correlation, we get how similar the signals are over lag. The maximum peak would correspond to the best value to lag one signal to match the other, in most applications the delay.

For a system calculating cross-correlation over a moving time window then you can reduce the computation by the part that is common between two time windows. For a discrete system calculating the cross correlation between x and y over a time window of t_w, then for the general time t the equation would be (2.8).

$$\phi_{1xy}(n) = \sum_{m=t-t_w}^{t} x[m+n]y[m] \tag{2.8}$$

Equally for the time $t+1$ the time would be (2.9).

$$\phi_{2xy}(n) = \sum_{m=t+1-t_w}^{t+1} x[m+n]y[m] \tag{2.9}$$

What we observe is that a large part of the sum would be the same between the two equations. Now writing the equation for the time $t+1$ as a function of (2.8) we get the much less computational heavy equation (2.10).

$$\phi_{2xy}(n) = \phi_{1xy}(n) + x[t+1+n]y[t+1] - x[t-t_w+n]y[t-t_w] \tag{2.10}$$

2.7 Intersection of circles

The intersection of two circles may be two points if there's overlap between the circles, a singular point if they barely touch and two points in imaginary space in the case of no touch no overlap [20].

We may solve these intersection points given we know the two circles position and radii. For simplification of solving the equation we can move the circles to the $y = 0$ line and the larger circle to $x = 0$ see Fig. 2.8. Using the notation of R and r for the radii of the larger respectively smaller circle and a distance of d it would give us the following equations:

$$x^2 + y^2 = R^2 \tag{2.11}$$

$$(x - d)^2 + y^2 = r^2 \tag{2.12}$$

The equations may be combined and rearranged for the following result of x and y: [20]

$$x = \frac{d^2 - r^2 + R^2}{2d} \tag{2.13}$$

$$y = \pm\sqrt{\frac{4d^2R^2 - (d^2 - r^2 + R^2)^2}{4d^2}} \tag{2.14}$$

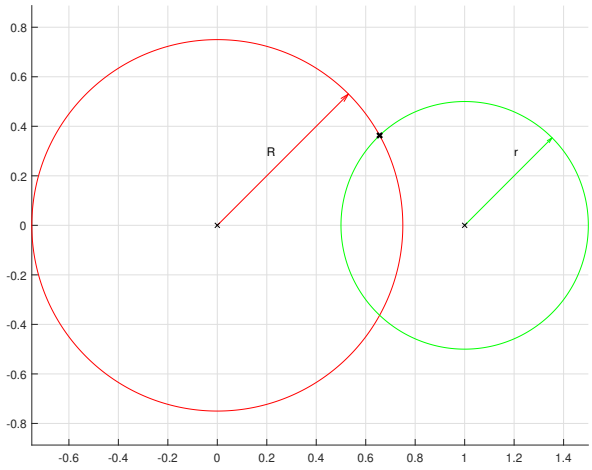


Figure 2.8: Intersection of two circles with a distance of 1 between

2.8 Digital Filters

Filters are used to attenuate noise components in an audio system. Filters can be designed to attenuate signals above or below a particular frequency. Filters that operate on discrete-time signals are referred as digital filters. Digital filters come in many different variants but the two most common filters are divided into Finite Impulse Response filters (FIR filters), and infinite Impulse Response filters (IIR filters). The FIR filter is a non-recursive Discrete Filter (NRDF), it has no feedback path and so the output is only dependent on the (current and previous) input. IIR filter is a Recursive Discrete Filter (RDF) and contrary to FIR, IIR utilizes one or more feedback paths. This means that an IIR filter is dependent on both the (current and previous) input and output [21].

An example of a FIR filter is the transversal filter, see Fig. 2.9. A special case of the transversal filter is the rolling average, this is when all of the filter constants b are the same. As such the output y[n] would simply be the sum over a time window of x[n]. Another special case of the transversal filter is the folding of FIR filters, this is when the coefficient values are symmetric around the middle value. Then, the multiplication operations in the filter can be reduced by a factor of 2. The two main benefits of an FIR filter are that it is inherently stable and that it can be designed to have a linear phase [21].

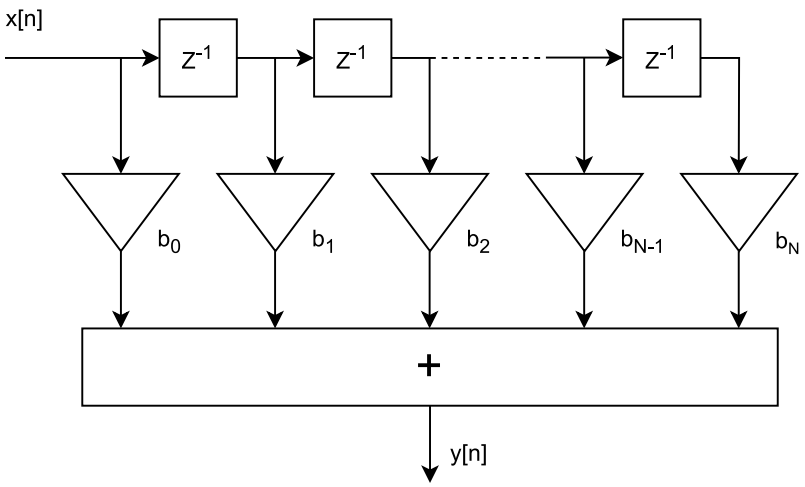


Figure 2.9: Transversal filter

IIR filters are useful for high-speed designs compared to FIR filters due to the lower number of filter coefficients [22]. Digital IIR filters are designed using the bilinear transformation of classical analog filters [23]. An example of the IIR filter is the Butterworth filter, shown in Fig. 2.10. The Butterworth filter has a smooth response at all frequencies and does not allow any ripples in the passband and stopband.

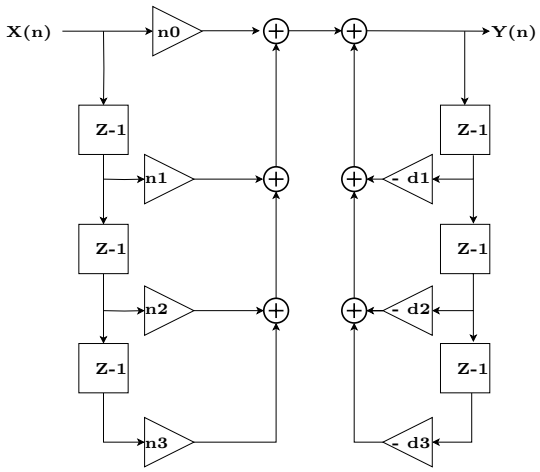


Figure 2.10: Butterworth Filter

The output of the IIR filter is calculated based on the filter coefficients and (current and previous) input and output. The output of a k^{th} order Butterworth filter is represented as (2.15), where (n,d) denote the filter coefficients, and X and Y denote the input and the output respectively.

$$Y(j) = \sum_{i=0}^{k} n_i X(j-i) - \sum_{i=1}^{k} d_i Y(j-i) \tag{2.15}$$

3

Design and Implementation

In this part, we introduce our system in detail. We will bring up a novel perspective to inspect the structure of our system, the control and audio path referred aptly so, see Fig. 3.1. After establishing a basic structure of the system, we go into the detail on design and implementation of the algorithms. Finally, we specify the key parameters in the system, which have effect on system performance.

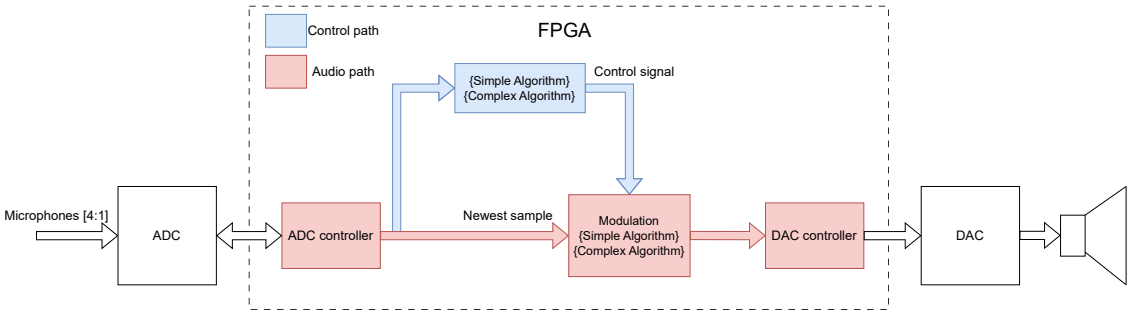


Figure 3.1: Simple overview of system

The benefit of dividing the system into audio path and control path is that the audio delay of the system will be close to zero, due to the newest samples that are directly sent to the modulation block. This can potentially give a very stable and acceptable result for the audience. Another benefit is the ability to easily transfer between the two algorithms in the design as the common blocks can be shared. The response time of the algorithms will however differ, meaning that a longer response time should be expected before the control signal is achieved its correct output.

3.1 System overview

In this section a more detailed overview is presented. Fig. 3.2 shows the major blocks between the ADC, FPGA and the DAC. Note that the same approach with audio-control paths are still present.

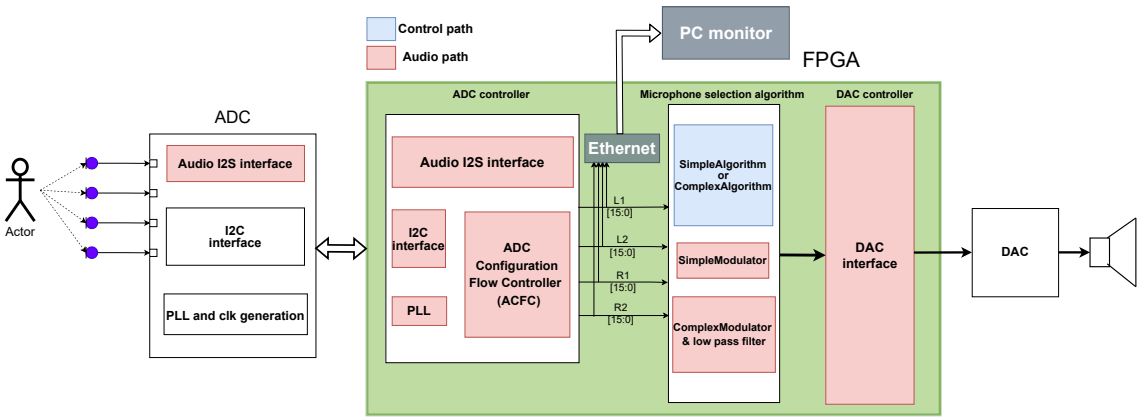


Figure 3.2: In depth overview of the blocks used in this system

To accomplish a successful audio path, some interface blocks are needed between the ADC and the DAC blocks. We have decided to use the I^{2}S protocol to capture the microphones input from the ADC, and the I^{2}C to configure the registers of the ADC module. The two interfaces are placed inside the ADC controller blocks seen in Fig. 3.2. The control path will include two solutions, called the simple and complex algorithm respectively.

When an actor is standing in front of the microphone array, the voice will be recorded by the array and the ADC should sample and convert the data. The ADC is transitioned into active mode after power-up, enabling the FPGA to send active control signals. During the process of controlling an ADC, it is necessary for the developer to configure the parameters within the registers of the device in order to establish the desired operating mode and enable the serial data output. All the registers for the different configurations are found inside the datasheet of the ADC [24].

We designed an I^{2}C master, and matched it with a finite state automaton controller to transmit control signals. The **ADC Configuration Flow Controller** module manages the data transmission protocol and relation between BCLK and FSYNC, also the IO port for MCLK from **PLL** module. The master_mode switch is also available since the ADC's master mode can guarantee precise timing signals generated by MCLK. When the finish_config_input switch is set high, it triggers additional necessary configurations, including the number of audio channels, after completing all switch-defined configurations.

The serial audio is captured in digital form using the I^{2}S protocol. The I^{2}S protocol supports standard left and right channel data formats, and provides a variety of clock signals to synchronize data transmission. The ADC will transmit data based on a sampling rate of 48 kHz and a word length of 16-bits. In addition, the BCLK frequency is 12.288 MHz, which is used as a clock signal in the I^{2}S bus protocol to

synchronize the transmission rate of audio data. The sample values of the captured microphones are held inside registers until a falling edge of FSYNC occurs, where all four samples are released together to the rest of the system.

3.2 Audio path

As shown in Fig. 3.1, the audio path contains ADC controller and DAC controller which are the beginning and end of audio path. The modulation part in audio path combines the four-channels input into one-channel output. The control path calculates the necessary parameters for combining the input signals. The simple algorithm and the complex algorithm share some of the same audio path but are completely different in the control path (the algorithm itself).

For the simple algorithm, the modulation blocks are called **Fader** and **Mixer**. They are two blocks in the simple algorithm, but belong to the audio path part of the system. The **Fader** gets the control signal from the algorithm, where it would attenuate the quietest microphones and only bypass the loudest microphone. In practice this is done by weighting the four channels signals with different gain. Fig. 3.3 shows the expected behavior of the **Fader** block.

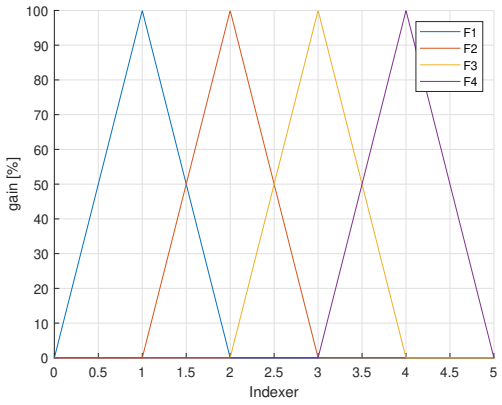


Figure 3.3: Operation of the fader block showing the corresponding gain it applies to each channel (F1 to F4) depending on the indexer value

The result of the **Fader** blocks for each input needs to be added together, because the DAC only takes one 16-bit input. As shown in Fig. 3.4, a **Mixer** block is created. The **Mixer** will add the newest samples coming from the I^2S and apply the new settings/parameters on them.

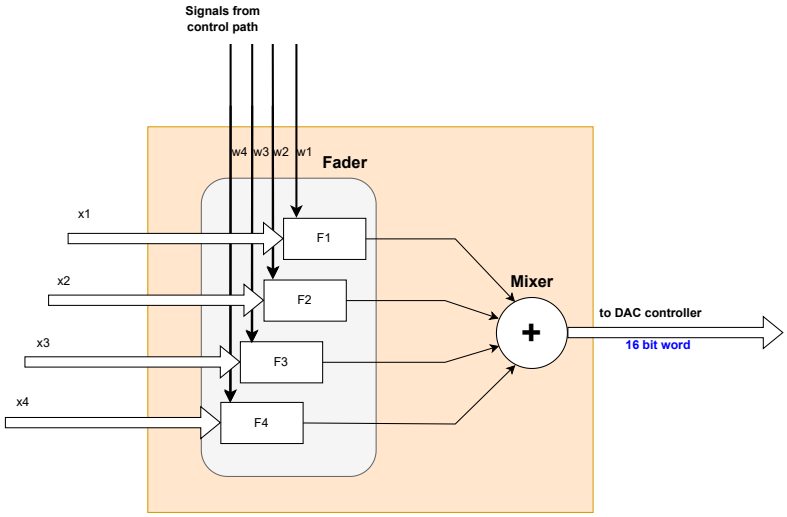


Figure 3.4: The modulation block consists of fader and mixer in simple algorithm design

In contrast, the complex algorithm makes use of delay between microphone signals to determine the location of the actor. With the location known it is possible to compensate for various factors such as attenuation and delay. The modulating block of the complex algorithm will use an **Attenuation compensation** block instead of the Fader from the simple algorithm. This **Attenuation compensation** block will multiply every microphone with the distance in Y direction. This is done to compensate for attentuation due to distance. An addition to the complex algorithm is filters, in particular the summed output from the same **Mixer** block is passed through a low pass filter. It is essential to note that in this implementation the actor should be heard equally loud wherever he is moving. Fig. 3.5 shows the behavior of the **Attenuation compensation** block.

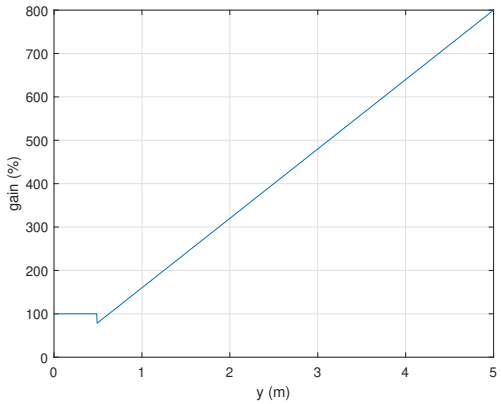


Figure 3.5: Operation of the attenuation compensation block. The algorithm compensates with gain the further away from the microphone array an actor is located.

3.3 Control path

The control path of the system includes the algorithm design. Here, the simple algorithm as well as the complex algorithm are introduced.

3.3.1 Simple Algorithm

The simple algorithm performs a power estimation of each microphone signal. The power estimation needs a time window from which the signal is summed over, this is done through a **Shift register** hardware module. From the **Shift register** the power is estimated according to Fig. 2.7. The output of this **Power estimation** block is the average amplitude throughout this sampling window. A **Max** function would then return the index of the loudest microphone, it will compare the audio sample from every microphone [1-4] to later determine the loudest microphone. In Fig. 3.6, we describe how the simple algorithm works based on power estimation/rolling average.

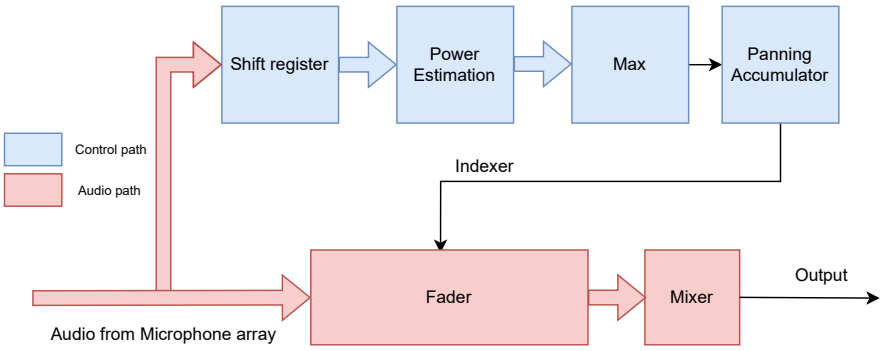


Figure 3.6: Block diagram of the simple algorithm

To be able to achieve a stable output you need a large shift register, this proves to be very costly hardware wise. We decided to add a block, the **Panning accumulator**. The **Panning accumulator** adds a fractional component to the output index. It stores a value **Indexer**, this is incremented each clock by a small fractional value **Delta**. The sign of **Delta** is switched so that **Indexer** approaches the index input value from the max function. The end result is that we can control the slew rate of the indexing and can to some degree smooth out jittery response caused by a small shift register. This also means if noise occurs for a short time, the accumulation of **Delta** will take longer time causing the system to not instantly switch the microphone towards the source of this noise. In this way a more reliable system is achieved with lower hardware utilization. The last two blocks of the algorithm are the **Fader** and the **Mixer** blocks, are earlier referred to in the audio path. The **Fader** will basically either attenuate or redirect a signal to the **Mixer** block, where every signal is added generating one output signal.

3.3.2 Complex Algorithm

For the complex algorithm we wish to calculate the location of the actor to be able to compensate for attenuation. We would need to reuse some of the blocks, but also create more VHDL blocks. A block diagram of the complex algorithm is found in Fig. 3.7 below.

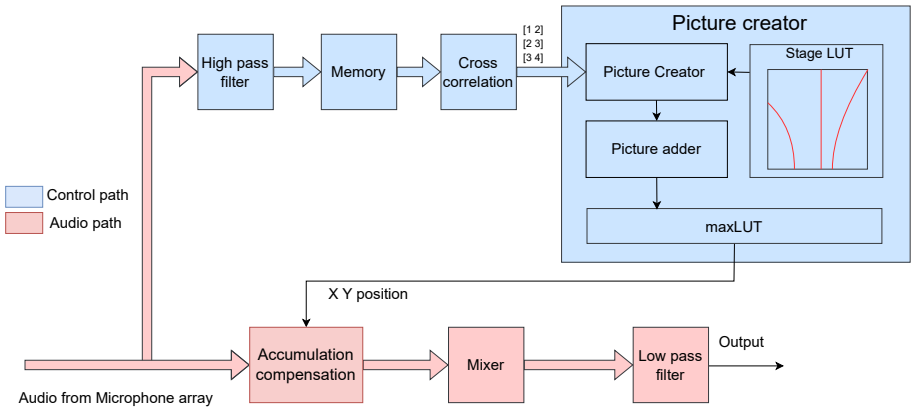


Figure 3.7: Simple block diagram of the complex algorithm.

In this implementation, our control path needs to store a larger sample window to perform the cross correlation. This sample window is stored in **Memory** similar to the simple algorithm. Three **Cross-correlations** are then made between microphone one and two, two and three, and three and four. In this case more redundant and accurate value of the localization problem can be obtained.

The fundamental operation to achieve this is cross-correlation, where the highest point in the signal shows where two signals are to appear most similar. A typical cross-correlation may look like Fig. 3.8.

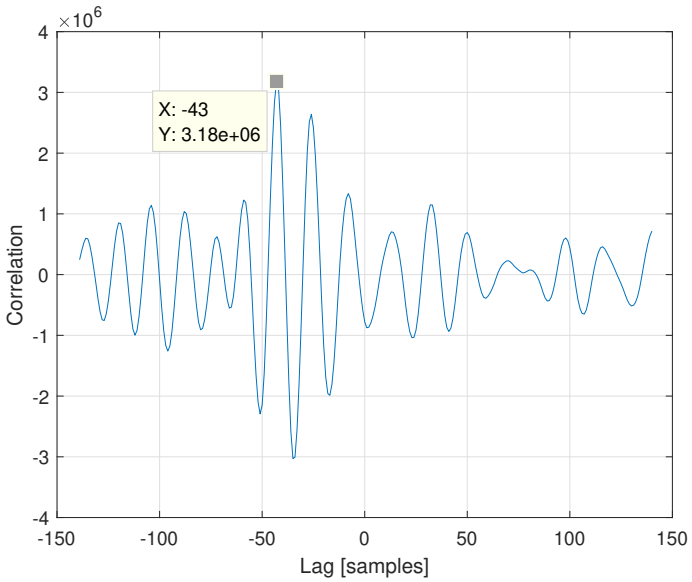
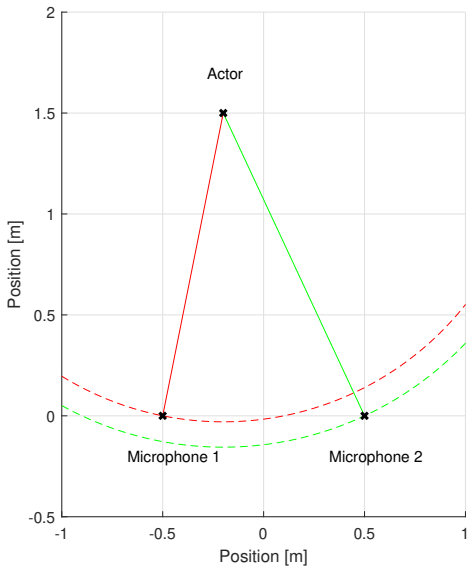


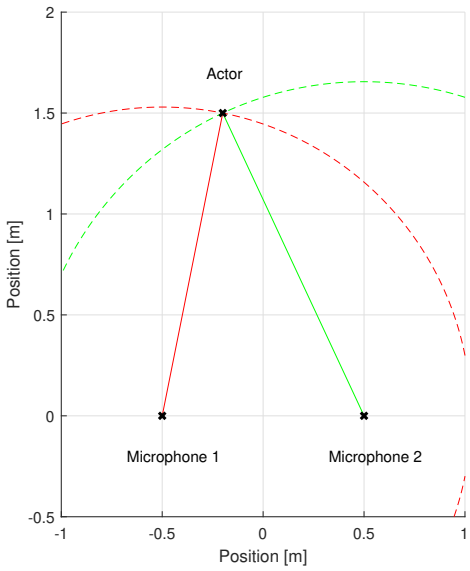
Figure 3.8: Typical cross correlation with maximum correlation marked

To add more redundancy we aim to include all cross-correlations, as such we will begin by focusing on what one cross-correlation contributes. Each lag value corresponds to a delay between microphones. This delay directly corresponds to the difference in distance between microphones and the actor. In reference to Fig. 3.9a this delay would be the difference in circle radii, note that the definitive radii is unknown.

Plotting the circle with their origin in the actors position may be intuitive, however you could equally plot their origin in each corresponding microphone as in Fig. 3.9b. What we observe is that the actor is at the intersection of these two circles.



(a) Delay value representing a difference in path length between green and red



(b) The reverse of the green and red arches will create a point of intersection

Figure 3.9: Visualization of delay and the corresponding path length difference

As previously stated the definitive radii is unknown, only the relation between the two circles is known. Plotting multiple circles which adheres to this rule we get Fig. 3.10. We can now observe that there is a line of valid positions.

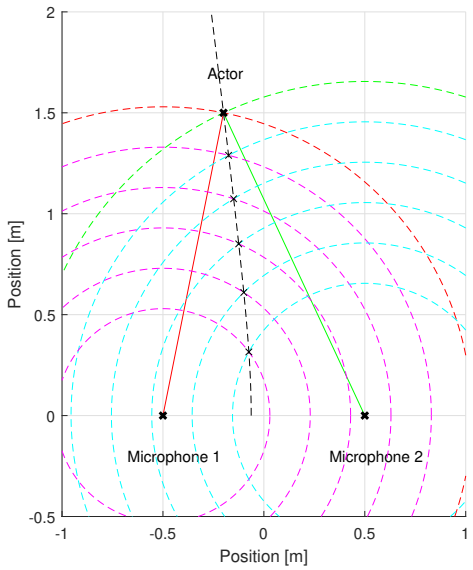


Figure 3.10: Correlation line from intersection of circles.

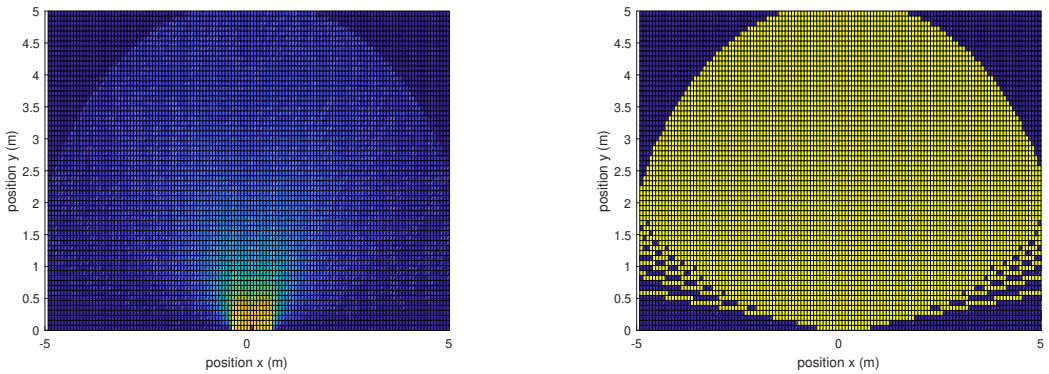
As a cross-correlations lag value gives us a line of valid positions, two lag values from different cross-correlations would thereby give you two lines which should intersect at the actors location. Adding a third line do however present a problem, mainly it may not intersect at exactly the same point, as such you would have three intersection points. Furthermore, we would ideally want to include all lag values not just pick the lag value corresponding to the maximum correlation as we have observed that sometimes the maximum correlation doesn't correspond to the delay of the signal but that it is inherent to the frequency content of the signals.

To fulfill both the above we chose to implement a image rastering system in which the lines are plotted in a limited resolution and with a brightness value equal to its correlation. This is done via the **Picture creator** block in the block diagram in complex algorithm in Fig. 3.7.

Each correlation generates its own image to later get summed together with other correlation images using the **Picture adder**. From the summed image the brightest pixel is equivalent to the intersection point from the highest correlating lines. This is calculated using the **maxLUT** block. The limited resolution is key for the intersection point of multiple lines to coincide at a singular point.

In practice we would use a look up table (**LUT**) representing the stage to store the lines in a vector format, from which a control loop may draw each line. The brightness of a pixel isn't simply added to the previous value. Instead it's compared and only if the correlation is higher then previous stored value of the pixel then it is replaced. This is key to prevent a natural bias towards close location values in

respect to the microphone position as the line density per pixel is the highest close up. By replacing the pixel brightness completely and only storing the highest value, the line density has no effect on the final image, see Fig. 3.11. It also guarantees that any intersection between two cross correlation images will add to a brighter pixels than any correlation image contain on its own.



(a) Addition of pixel brightness

(b) Comparison and replacement of pixel brightness

Figure 3.11: Single correlation image with same correlation value for every lag

For the audio path in this algorithm, the **Attenuation compensation** and the **Mixer** blocks are used as a modulation block. The **Attenuation compensation** block, earlier referred to will multiply each input with the distance in Y of where the actor is standing. In this way we can compensate for attenuation, and again have zero audio delay. The **Mixer** takes the modulated signals from the previous block, but now it will send this signal to a **Low-pass filter** to remove some of the low frequency noise in the system. The output of the filter goes then to the DAC and at the end to the speaker.

3.4 Parameter

System parameter

- System clock frequency: 100 MHz
- Sample frequency/FSYNC: 48 kHz
- BCLK: 12,288 MHz
- Signal width: 16 bits
- Microphones: Shure SM57 and AKG C568EB
- ADC Board: TI PCM6240Q1EVM-PDK
- DAC board: LT DC2459A
- FPGA board: Digilent Nexys A7-100T

Block parameter

Simple algorithm

- Time window for power estimation: 100 samples
- Accumulating unit for panning accumulator: 2^{-10}

Complex algorithm

- Time window for cross-correlation: 10 000 samples
- Cut-off frequency of high pass filter: 1 kHz
- Cut-off frequency of low pass filter: 7 kHz
- LUT resolution: 64 x 64 pixels or approximately 8 pixels/m
- LUT size: 5 x 5 m
- Correlation image: 10 x 5 m
- Final image: 8 x 5 m
- Microphone spacing: 1 m

4

Results

In this chapter, we first introduce the verification methods. Then based on these methods, we analyze our results from MATLAB simulation, behavioral simulation and FPGA implementation to verify and then evaluate the system.

The ideal version of system behavior is presented in MATLAB simulation part. Then we use behavioral simulation of RTL for verification. For audio path, we present the behavior of I^2C and I^2S interface. For the control path, we compare the output from behavioral simulation with output from MATLAB. In simple algorithm, the most important parameter is the indexer of the selected microphone. While in complex algorithm, we focus on the position of the actor.

After behavioral simulation, we synthesize and implement our design on FPGA. We compare the implementation of the two algorithms, in terms of timing, utilization as well as performance.

4.1 Verification methods

Verification methods of the designs are required to be able to carry out the development of the different blocks in the system. It effective to reduce integration problems when multiple smaller blocks are integrated together. The first test was a simple ADC-DAC pipeline. The ADC required a control interface as well as a sound capturing interface. The DAC required also an interface to order the bits on the physical pins. Therefore, a simple pipeline test was made to check if the output is similar to a random input. This test is essential to make sure that the audio path would be functional before starting to implement the algorithms. See Fig. 4.1.

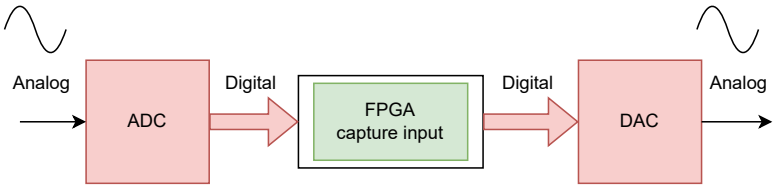


Figure 4.1: A simple pipeline between the ADC and DAC blocks

To design the algorithms, a MATLAB model was first made to verify that the high level thinking meets specification. The MATLAB model can consequently be used as a reference model when writing the VHDL code. The verification of the smaller blocks in VHDL was done using testbenches in ModelSim.

A more robust behavioral simulation was done to verify each algorithm design when the smaller blocks got combined. This testbench included four microphone input files. Each file contains 12 million stimuli input vectors. The input files contain real microphone data captured using the Ethernet interface. WireShark is used as to monitor and save the captured data from the ADC though Ethernet. The audio was captured in a similar environment as the system is aimed to be placed in. This will give authentic behavior of the algorithms as it captures the desired audio as well as inherent noise from live capture.

The RTL of each corresponding algorithm would then run through these test vectors and save important parameters in LOG files. Example of such parameters: The output, panning value and the indexer of the simple algorithm to verify that it picks the loudest microphone. But also the output of the complex algorithm as well as the calculated distance to the actor from the cross correlation and the picture creator. Fig. 4.2 below explains how the behavioral test is done for the algorithms.

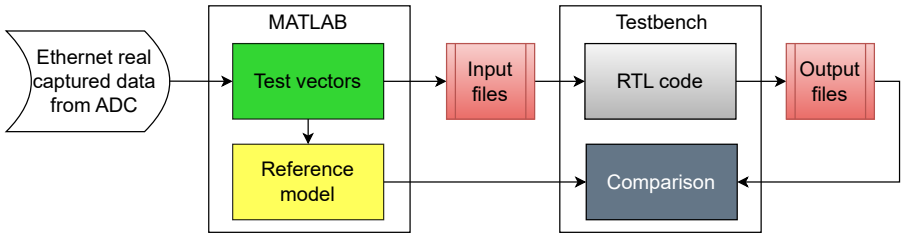
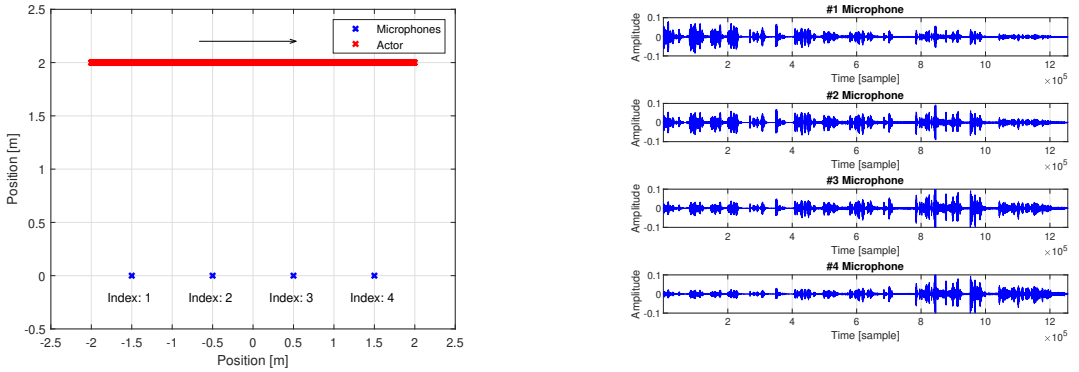


Figure 4.2: The VHDL design is tested and compared to the MATLAB model

4.2 MATLAB simulation

In this section, we first present the simulation result of both simple and complex algorithm from MATLAB software prototype. Simulation results of digital filters will also be discussed. Finally, a performance evaluation of the two algorithms is presented.

The control path of the algorithms was tested with test vectors created from a simulation of an actor speaking while moving on a stage. The microphones and the actor were positioned according to Fig. 4.3a. The test vectors are generated by delaying and attenuating an input sound file accordingly to the distance between the actor and each microphone. The test vectors can be seen in Fig. 4.3b.



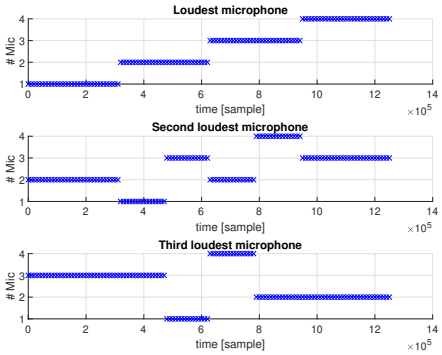
(a) Positions during simulation

(b) Audio test vectors from the simulation

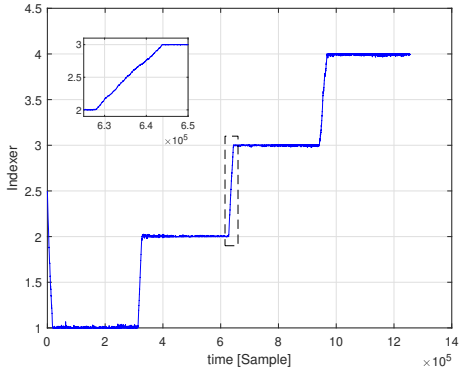
Figure 4.3: Setup of audio file into MATLAB. The sound is moving across all microphones

4.2.1 Simple Algorithm

When comparing the indexes generated directly out of the soundstage simulation with the indexer from the simple algorithm, we can see that they correlate very well. This is visible in Fig. 4.4a and Fig. 4.4b. There is some noise which induce error in the target index but the panning accumulator dampens the system, this is visible as a slew in the figure. Of note in regards to the slew is that it is very fast despite achieving a stable output, as such it should be adequate to switch microphones for fast moving actors and alternating sound sources.



(a) Ideal indexes of microphones during simulation

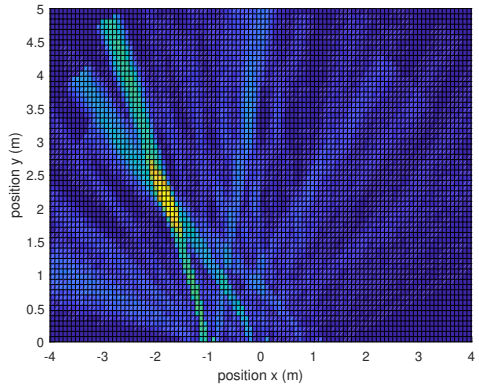


(b) Result of the algorithms indexer

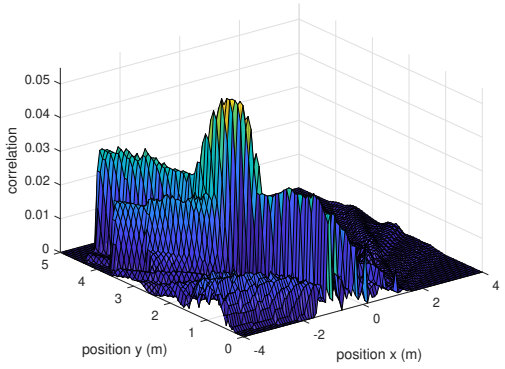
Figure 4.4: Ideal microphone indexes compared to algorithms indexer based on loudest microphone

4.2.2 Complex Algorithm

The more sophisticated algorithm uses cross-correlation for position calculation. The advantage being that the delay is not affected by non-ideal factors such as microphone pickup-pattern, attenuation as well as the directionality of the sound source. This leads to more accurate identification of the sound source position. This is a considerable advantage compared to the power estimation method used in the simple algorithm. In Fig. 4.5 we see the result of three correlation images added on top of each other. These images represent the LUT of the stage.



(a) Top view of the stage actor position

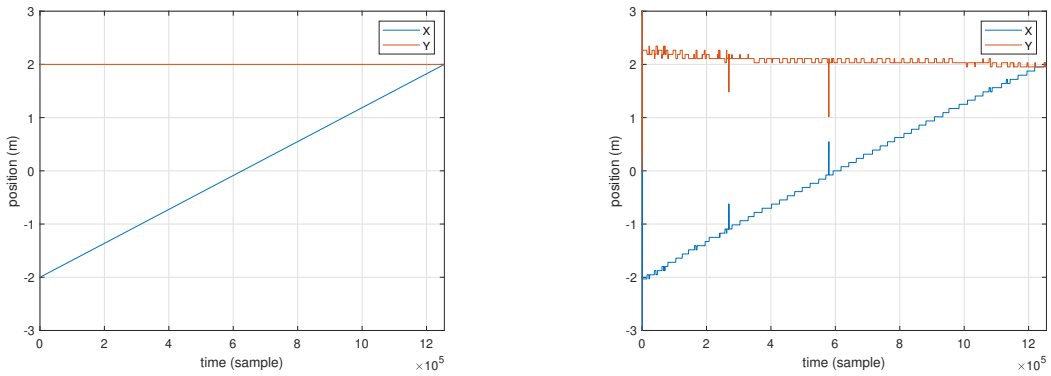


(b) Side view of the stage

Figure 4.5: Sum of three correlation images. Actor position appears at (-1,8; 2,2)

The most interesting thing to observe here is due to redundancy in the number of cross-correlations, a clearer localization result is obtained. A lighter points in the left figure is seen which indicated that the actor is standing around these pixels. From the right figure, the complex algorithm should pick the pixel with the highest amplitude. This maximum in amplitude pixel should be the exact position of the actor.

In Fig. 4.6 the complex algorithm is simulated using the sound input file that was previously presented in Fig. 4.3b. The value presented in Fig. 4.6 is the calculated distance to the microphone an actor is standing. The result turns out to be very accurate.



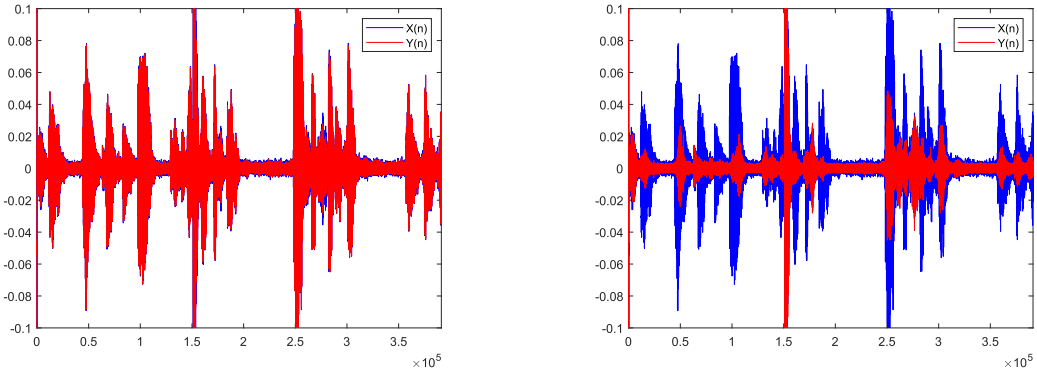
(a) Ideal algorithm result for actor position

(b) Result of the algorithms location calculation

Figure 4.6: Algorithm positional result compared to ideal result

4.2.3 Digital filter

IIR Butterworth filters are designed for reducing the noise and thereby improving the performance of the design. A high pass filter in the control path and a low pass filter in the audio path will improve the performance of the complex algorithm. The real audio data captured from the ADC using the Ethernet was used as input for designing the filters. Filter functions in MATLAB were used for designing the filters. Filter operations were performed for the same input with different orders and cut-off frequencies. When considering hardware limitations and the performance of the design, the calculated order and cut-off frequency of the high pass filter is 3 and 1kHz respectively. Similarly, the order and cut-off frequency of the low pass filter is 3 and 7kHz respectively. The behavior of the designed filter in MATLAB is shown in Fig. 4.7, where X(n) and Y(n) represent the input and the output respectively. The designed filter coefficients of the high pass filter and low filter in the normalized form are shown in Table 4.1.



(a) Low pass filter **(b)** High pass filter

Figure 4.7: MATLAB simulation of IIR Butterworth filters

Table 4.1: Filter coefficients of high pass and low pass filters

	High pass filter		Low pass filter	
i	ni	di	ni	di
0	0.8772	1.0000	0.0463	1.0000
1	-2.6317	-2.7384	0.1388	-1.2112
2	2.6317	2.5099	0.1388	0.7279
3	-0.8772	-0.7695	0.0463	-0.1466

4.2.4 Performance evaluation

To evaluate the performance of two algorithms, we use a sine wave signal to plot the response curves in two different test cases: The actor moving in X direction and Y direction. We did not add any external noise to the system, so the only noise source is the MATLAB and algorithm itself so that we can focus on the system behavior. Except the simple algorithm and the complex algorithm, we also use a mixer, which just sum up the input signal from four channel without any processing, as reference.

Fig. 4.8 and Fig. 4.9 show the relationship between power and position of the actor. Since this enhancement system is used to provide a stable output, the ideal power will be a constant while the actor is moving. And since the actual speech signal is not a single frequency signal, we used a low frequency signal of 10 Hz and a high frequency signal of 2 KHz.

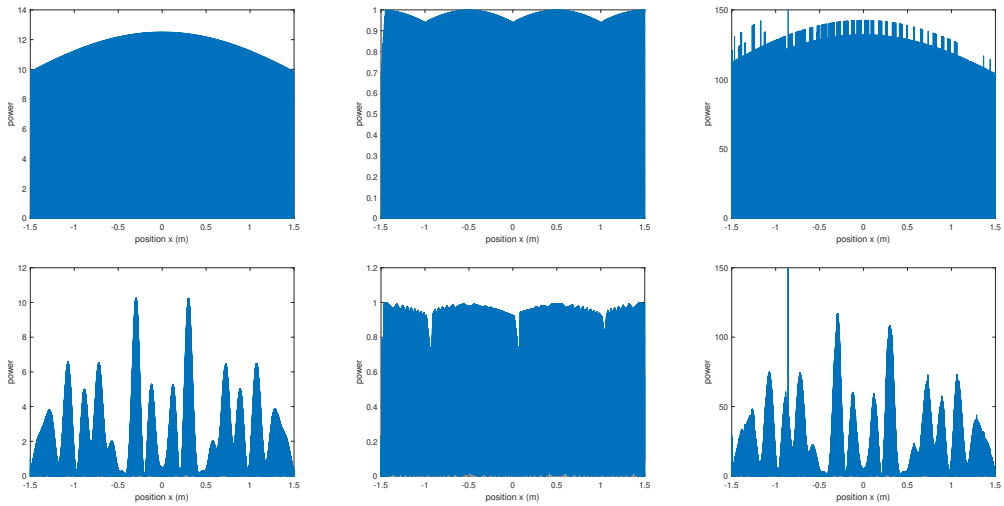


Figure 4.8: System response curve from moving actor in X direction. From left to right; normal mixer, simple algorithm, complex algorithm. Top 10 Hz signal, bottom 2 kHz signal

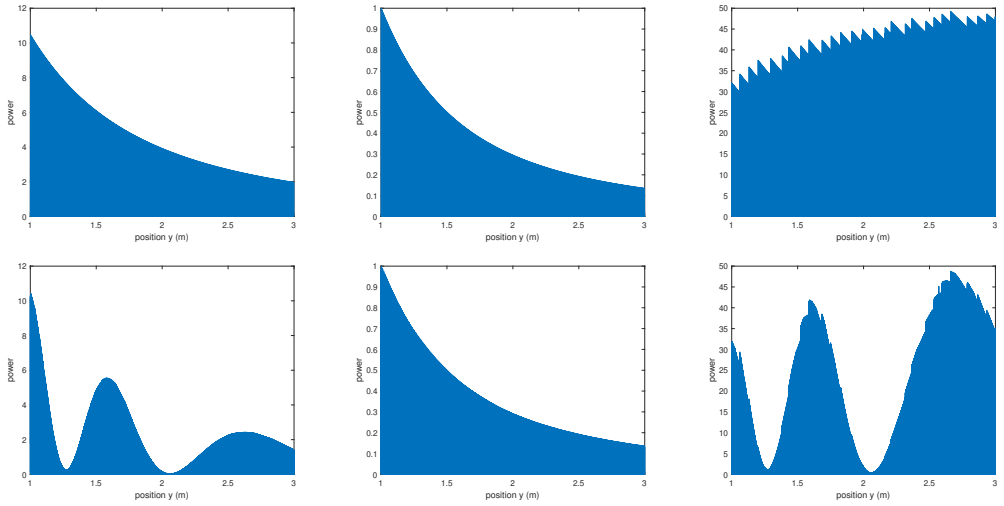


Figure 4.9: System response curve from moving actor in Y direction. From left to right; normal mixer, simple algorithm, complex algorithm. Top 10 Hz signal, bottom 2 kHz signal

For the X direction test case, we can see that the simple algorithm works better than the complex algorithm in terms of consistent power. When we use the 10 Hz signal, it is reasonable that power of the mixer is symmetrical by the middle point when the total power of four channels is maximum. As shown in Fig. 4.8, the complex algorithm has similar behavior with the mixer because the change in X direction does not contribute to update of compensation based on Y position, although the signal is amplified by a fixed gain. We also see some spikes which is caused by a inaccuracy in position estimation. The performance of the simple algorithm is quite good, because the loudest microphone is switching as the actor is moving and

the power of the loudest microphone almost constant. When we use the 2 KHz signal, the envelope of the curves are similar to of the 10 Hz scenario. When we use a high frequency signal, the phase of the signals arriving at each microphone changes more as the actor moves compared to a low frequency. When two signals with opposite phase add together, they will cancel out each other, which cause the power to decrease in bottom line of Fig. 4.8.

For the Y direction test case, the complex algorithm has better performance than the simple algorithm as shown in Fig. 4.9. When the actor moves further away from the microphones, it is reasonable that the power of mixer decrease because the power of each channel decreases. Similar behavior is observed from the simple algorithm. Since the complex algorithm compensate the attenuation based on Y position, the system will add more gain as the actor moving further. The power of the complex algorithm is in this case almost constant. We can see some discontinuity in the curve which is caused by the limited pixel resolution of the picture(discussed in section 3.3.2). In the bottom line, we can still see the power fluctuate due to the high frequency phasing.

In summary, the simple algorithm performs better in X direction test and the complex performs better in Y direction test. And the response curve of complex algorithm has a frequency-dependent feature while the response curve of simple algorithm does not.

4.3 Behavioral simulation

In this section we will present the results from behavioral simulation stage during the hardware design of our system, including both simulation results from audio path and control path.

4.3.1 Audio path

We implemented I^{2}C communication in accordance with the timing requirements specified in the datasheet as shown in Fig. 4.10. We also verified the timing and the logic of the design using the ModelSim tool in Fig. 4.11 and 4.12. Our implementation followed the sequence of slave addressing, register addressing, and register writing, coupled with a finite-state machine in the controller and input from switches, to modify the values of destination registers for configuring the ADC.

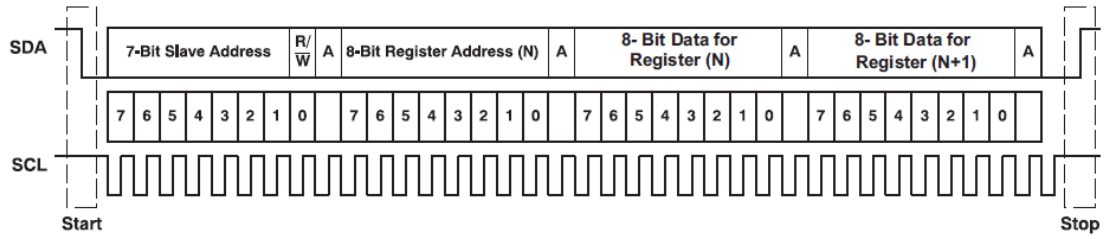


Figure 4.10: I^{2}C write rule. From [24]. Reproduced with permission.

As the registers we controlled had explicit behaviors, such as the output frequency and ratio relationship of BCLK synchronous clock and FSYNC sampling clock, we adopted a direct observation method to detect the control path (which also includes signal path) and avoid excessive state settings.

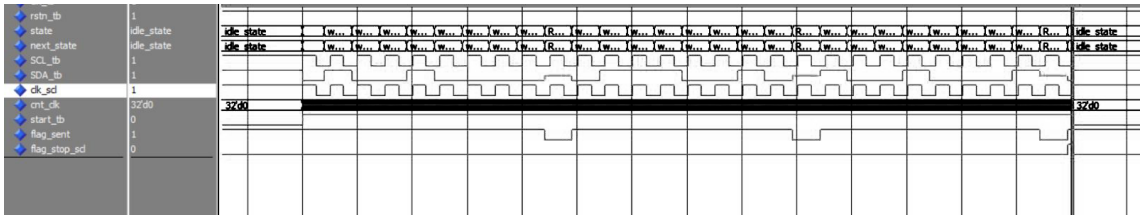


Figure 4.11: I^{2}C Modelsim waveform

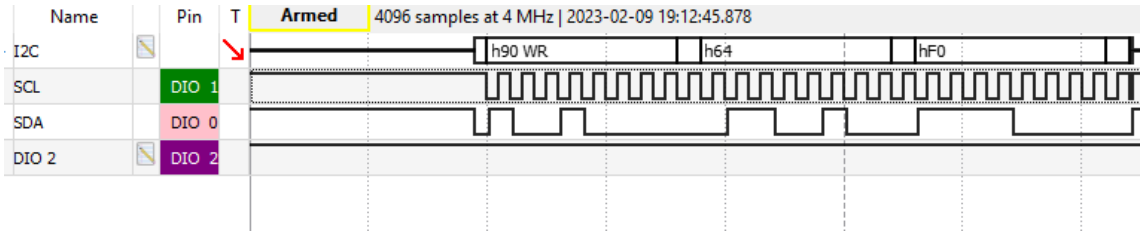


Figure 4.12: I^{2}C waveform in logic analyzer.

In addition to the control protocol, the I^{2}S sound capturing interface was developed on the FPGA. Four input channels are read on each falling edge of FSYNC signal. After achieving both control and sound capturing protocols, we could test a simple pipeline between the ADC and the DAC. This simple test would be to send a sine-wave through each of the ADC inputs and using an oscilloscope observe if the corresponding output on the DAC would be the same as the input. This test is very efficient to verify multiple things including: Correct configuration of the ADC, correct timing and behaviour of the I^{2}S protocol and correct interface for the DAC module. See Fig 4.13.

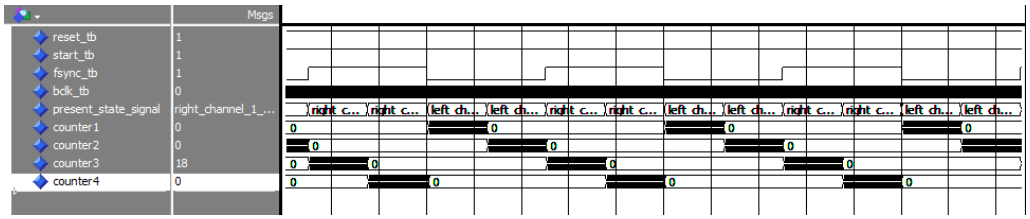


Figure 4.13: I^{2}S Modelsim wave simulation.

4.3.2 Simple algorithm

After we implemented the system in VHDL, we used the same audio file that was tested in the MATLAB simulation of the simple algorithm. We converted the microphones inputs to 16 bits binary numbers and saved them into four files. We then simulated the algorithm and saved the output from the behavioral simulation into an output log file. This output file was later exported to MATLAB and plotted together with the ideal output from the MATLAB version. In Fig 4.14 we can clearly see that the output from the behavioral simulation is very similar to the expected output except that the setup time of the system at the beginning. In total a file of 1.2 million input vectors was tested.

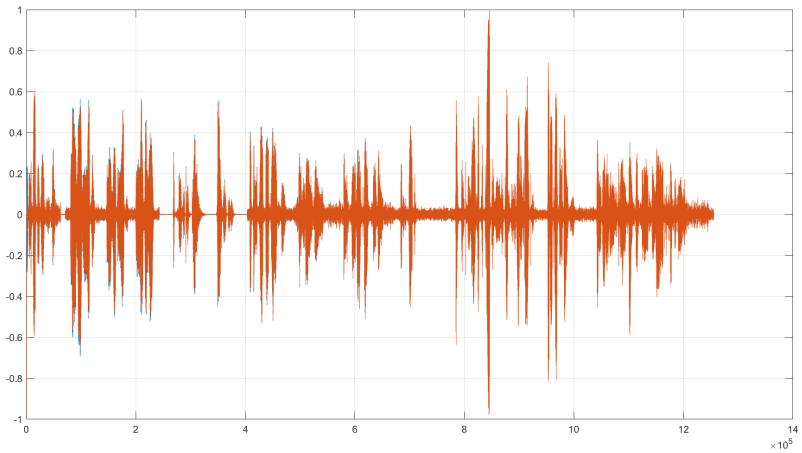
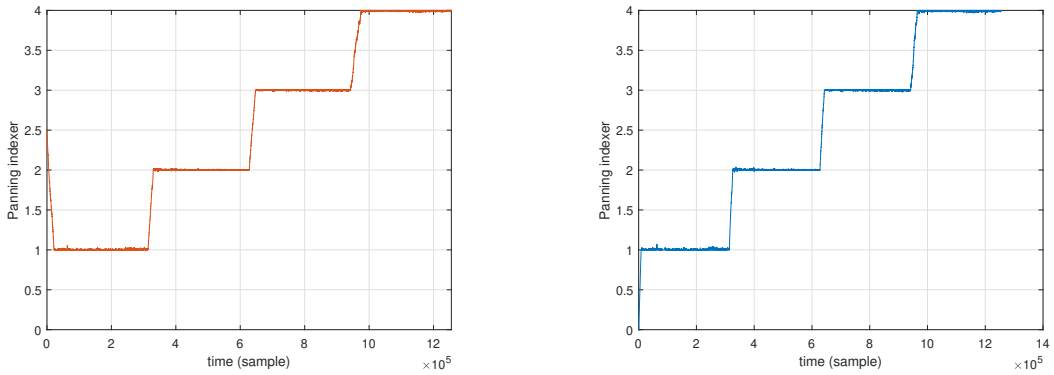


Figure 4.14: Comparison between expected output from MATLAB (Blue) and output from behavioral simulation (Red).

As introduced in section 3.3.1, we add panning accumulator to the system to avoid

a sudden change in microphone selection. To see if we get the correct behavior, we also check the indexer from the control path in Fig. 4.15. We observe that the MATLAB prototype and the behavioral simulation have the same behavior, except in the beginning. It is reasonable because we set different initial value in the two implementation.

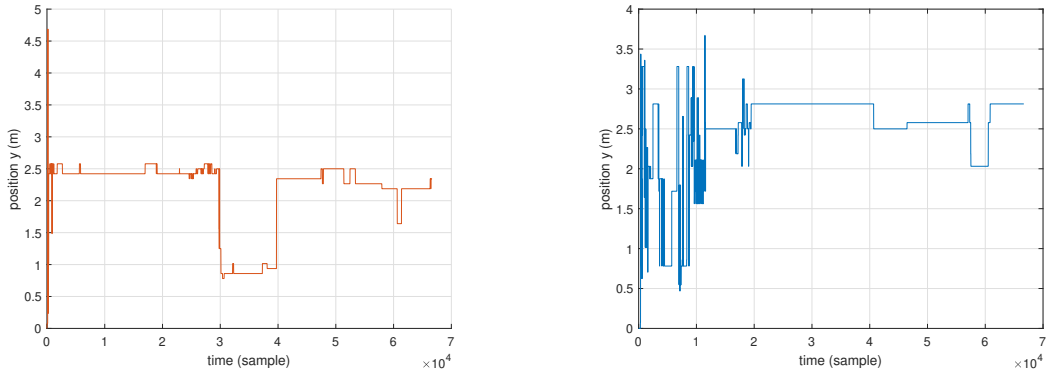


(a) MATLAB **(b)** Behavioral simulation

Figure 4.15: Indexer of selected microphone from MATLAB

4.3.3 Complex algorithm

We used the same method to test the complex algorithm as we used for the simple algorithm but with real data caught from Ethernet, which is explained in section 4.1. Because we believe that the performance of cross-correlation and position solver is more sensitive to real data than power estimation. As shown in Fig. 4.16, the behavioral simulation takes longer than MATLAB simulation before it gets a stable position. But the outputs from both algorithm are similar: In MATLAB simulation the position is stable around 2.5 m; In behavioral simulation the position is stable around 2.8 m.



(a) MATLAB **(b)** Behavioral simulation

Figure 4.16: Index of selected microphone from MATLAB

4.3.4 Digital filters

The coefficients from the MATLAB are represented in VHDL using fixed point representation. The output of the filter was calculated by using (2.15). The real audio data was used as the input stimuli for functional verification of the digital filter designs. The filtering of the lower frequency noise can be observed from the behavioural simulation of the high pass filter shown in Fig. 4.17.

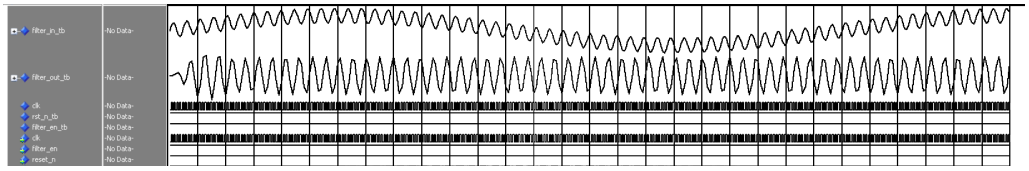


Figure 4.17: Behavioural simulation of high pass IIR Butterworth filter

Similarly, using the same input stimuli, the behaviour of the low pass filter design is shown in Fig. 4.18. Here we can observe the filtering of high-frequency noise.

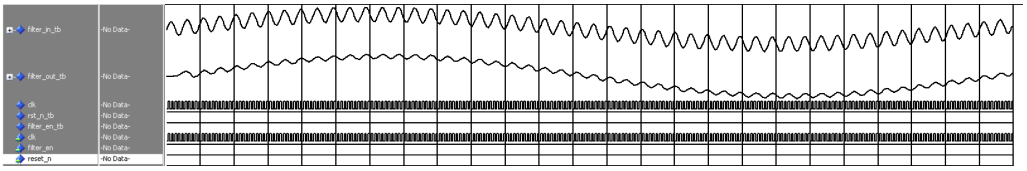


Figure 4.18: Behavioural simulation of low pass IIR Butterworth filter

4.4 FPGA implementation

In this section we will present the result from synthesis and implementation and discuss the performance of the system after we have implemented it on FPGA.

4.4.1 Timing

For the simple algorithm, the whole system runs at sample clock, which is the 48 kHz FSYNC in section 3.4. From the timing report provided by Vivado, our system has no timing violation. The worst negative slack for setup is 10 412 ns and the worst negative slack for hold is 0.058 ns.

For the complex algorithm, the position solver block runs at 100 MHz system clock while other blocks run at 48 kHz FSYNC. From the timing report provided by Vivado, the complex algorithm has no timing violation. The worst negative slack for setup is 0.498 ns and the worst negative slack for hold is 0.079 ns.

4.4.2 Utilization

The utilization reports of the two systems are shown in table 4.2. We can clearly see that the complex algorithm has higher utilization than simple algorithm.

Table 4.2: Utilization comparison of two systems

	LUT (%)	LUTRAM (%)	BRAM (%)	FF (%)	DSP (%)
Simple Algorithm	2,05	1,18	-	0,64	1,67
Complex Algorithm	32,32	-	45,56	44,16	63,75

Table 4.3: Utilization comparison of shift register instance in two systems

	Slice LUTs	Slice Registers	LUT as Logic	LUT as Memory	Block RAM
Simple Algorithm	115	48	59	56	-
Complex Algorithm	273	105	273	-	8

The result is quite understandable because the function of complex algorithm is more computational demanding than the simple algorithm, and if we want to run them at the same speed or run the complex algorithm at higher speed, the cost of hardware will increase significantly.

But we also did some optimization on certain blocks, such as the shift register to reduce utilization. We can see in table 4.3, that the shift register is implemented by slice Registers in the simple algorithm. This implementation is not compatible with complex algorithm, so we use block RAM instead of slice Register therefore there is no significant increase in utilization of slice registers in the the complex algorithm compared to the simple algorithm.

4.4.3 Performance

The performance of the two algorithms is evaluated by subjective perception.

We first test the two systems separately. For the simple algorithm, when the microphone selection is correct and there is only one speaker, the output signal sounds similar to the input. In theory, it should sound less noisy because we choose the channel with highest SNR. When the speaker moves from one microphone to another, due to the feature of fader introduced in section 3.2, the level of the output signal will first decrease and then increase. For complex algorithm, during the test, the output signal appears to be at the same level when the speaker moves within the range of five meters (in Y direction), giving the feeling that the actors have a virtually placed microphone one them. This is the result we expect from our model. In both system, we did not hear an obvious delay between the actor and the output. This is because the algorithm (control path) will determine the settings of the audio enhancement, meanwhile this settings is only applied to the newest sample that we get from the (audio path). To verify this assumption, we observe the input and output by oscilloscope. As shown in Fig. 4.19, the delay is 600 µs, which is very small and will not affect the performance of the system.

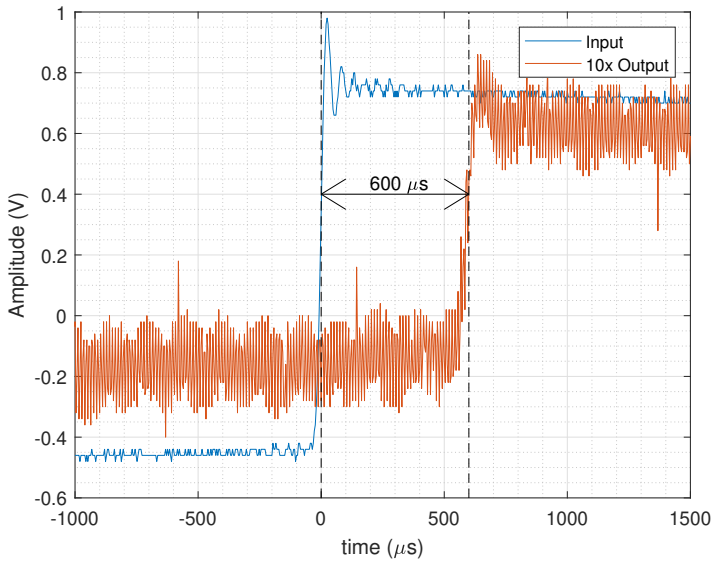


Figure 4.19: Input and output captured through oscilloscope

To compare the performance of the two systems, we test both algorithm with the test case where the actor moves in X direction and in Y direction. For X direction test, the outputs of two systems sound similar. But as discussed in section 4.2.4, the output of the simple algorithm is more stable. For Y direction test, we can clearly hear that the output of the simple algorithm is attenuated when the actor moves away from microphones while the output of the complex algorithm keeps volume level, which is consistent with the ideal behavior.

We also test both algorithms with test case where two speakers talk to each other, both algorithm perform as expected. Due to the panning accumulator, we avoid the clipping sound when switching microphone in simple algorithm. The complex algorithm can also update the position to the current speaker, causing both actors at different distances to the microphone array to sound equally the same. As for the test case where two speakers talk at the same time, both algorithm seems to work. Considering the application of the system we discussed in section 1.3, this test case is not our main concern.

5

Conclusion

The demand of an outstanding electronic sound system exists in many places that we aimed to design a speech enhancement system. We first established the audio path working at 48 kHz sample rate from the ADC, passing the FPGA and the DAC, to the speaker. Two speech enhancement algorithms are implemented; The simple algorithm based on power estimation and, the complex algorithm compensating attenuation based on position localization using cross-correlation.

To get proper audio data for system test, we prepared a series of test scenes; fixed and moving actor and recorded the audio. The high-speed Ethernet with UDP protocol sends the data frames as test vectors to PC. Using this interface we were able to improve our algorithm design based on real data captured by the ADC.

The enhancement algorithm have achieved satisfactory results on both audio quality and delay when we tested the system in real application environment. The simple algorithm is better than complex in terms of hardware resources. In terms of performance, the simple algorithm is better than complex when using X direction test case while the complex algorithm performs better when using Y direction test, which means when the actor moves in Y direction the output keeps at the same volume level and the delay is within the acceptable range of audience.

We also found a new view to inspect the algorithm design better. That is, we divided it to control path and audio path. This perspective not only interprets algorithm more easily, but also helps us to analyze the hardware implementation more effectively: The control path is the worst case path since it is longer than audio path. Therefore, we replaced some computing modules with LUT such that we could save intermediate result in advance and can read them to save the computing time. It is also important to enlarge the width of control path. For example we increased the length of sample memory in the complex algorithm, to generate more stable data for accurate position localization. However, wider control path will slow the speed due to the addressing of intermediate data, which are relation to the timing constraint of system. This is a trade-off between width and length in this worst case path, which is crucial to get a performance-speed balance.

In future work, the system can be improved in three ways: First, changing to high-performing hardware, such as ADC, DAC, FPGA and microphones. Second, optimizing the VHDL code of the current design. The optimization can be done in utilization and power. One example is that we can decrease the utilization of DSP in complex algorithm. Third, improving the algorithm. In current system, we use

cross-correlation values for all 280 lags. We could also compensate for delay in this implementation instead of only compensating for attenuation. Using some selection or optimization algorithm, we may decrease this number to less than 10. Except for basing on current system, we can also choose other algorithm, such as using a different way to calculate cross-correlation, or add more blocks for better performance such as inverse filtering blocks to remove reverberation.

Bibliography

[1] *"I2S bus specification"*, Philips Semiconductors, February 1986, accessed: 2023-05-24. [Online]. Available: https://www.nxp.com/docs/en/user-manual/UM11732.pdf

[2] P. Dhaker, "Introduction to SPI interface," *Analog Devices*, 2018, accessed: 2023-05-24. [Online]. Available: https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html

[3] J. Benesty, S. Makino, and J. Chen, "Chapter 1 - Introduction," in *Speech Enhancement*, J. Benesty, S. Makino, and J. Chen, Eds. Berlin Heidelberg: Springer, 2005, pp. 3–7.

[4] M. Berouti, R. Schwartz, and J. Makhoul, "Enhancement of speech corrupted by acoustic noise," in *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 1979, pp. 208–211.

[5] M. Dendrinos, S. Bakamidis, and G. Carayannis, "Speech enhancement from noise: A regenerative approach," *Speech Communication*, vol. 10, no. 1, pp. 45–57, Feb. 1991.

[6] R. McAulay and M. Malpass, "Speech enhancement using a soft-decision noise suppression filter," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 28, no. 2, pp. 137–145, Apr. 1980.

[7] Y. Ephraim and D. Malah, "Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 32, no. 6, pp. 1109–1121, Dec. 1984.

[8] P. J. Wolfe and S. J. Godsill, "Simple alternatives to the Ephraim and Malah suppression rule for speech enhancement," in *Proc. of the 11th IEEE Signal Processing Workshop on Statistical Signal Processing*, 2001, pp. 496–499.

[9] A. Sugiyama, M. Kato, and M. Serizawa, "Chapter 6 - Single-Microphone Noise Suppression for 3G Handsets Based on Weighted Noise Estimation," in *Speech Enhancement*, J. Benesty, S. Makino, and J. Chen, Eds. Berlin Heidelberg: Springer, 2005, pp. 116–123.

[10] T. Nakatani, M. Miyoshi, and K. Kinoshita, "Chapter 11 - Single-Microphone Blind Dereverberation," in *Speech Enhancement*, J. Benesty, S. Makino, and J. Chen, Eds. Berlin Heidelberg: Springer, 2005, pp. 247–259.

[11] T. Nakatani, K. Kinoshita, and M. Miyoshi, "Harmonicity-based blind dereverberation for single-channel speech signals," in *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 2007, pp. 80–95.

[12] J. Benesty, S. Makino, and J. Chen, “Chapter 1 - Introduction,” in *Microphone Array Signal Processing*, J. Benesty, S. Makino, and J. Chen, Eds. Berlin Heidelberg: Springer, 2008, pp. 1–5.

[13] ——, “Chapter 3 - Conventional Beamforming Techniques,” in *Microphone Array Signal Processing*, J. Benesty, S. Makino, and J. Chen, Eds. Berlin Heidelberg: Springer, 2008, pp. 39–40.

[14] ——, “Chapter 9 - Direction-of-Arrival and Time-Difference-of-Arrival Estimation ,” in *Microphone Array Signal Processing*, J. Benesty, S. Makino, and J. Chen, Eds. Berlin Heidelberg: Springer, 2008, pp. 181–184.

[15] C. Knapp and G. Carter, “The generalized correlation method for estimation of time delay,” *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 24, no. 4, pp. 320–327, Aug. 1976.

[16] B. Champagne, S. Bedard, and A. Stephenne, “Performance of time-delay estimation in presence of room reverberation,” *IEEE Trans. on Speech and Audio Processing*, vol. 4, no. 2, pp. 148–152, Mar. 1996.

[17] M. Omologo and P. Svaizer, “Acoustic event localization using a crosspower-spectrum phase based technique,” in *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 1994, pp. 273–276.

[18] A. N. Dwiputra, R. Hasiando, and G. Nainggolan, “3 dimension sound source localization with cross-correlation and cordic algorithm on FPGA,” in *2016 International Symposium on Electronics and Smart Devices (ISESD)*, 2016, pp. 365–370.

[19] P. Jeyasingh and M. M. Ismail, “Real-time multi source speech enhancement based on sound source separation using microphone array,” in *2018 Conference on Emerging Devices and Smart Systems (ICEDSS)*, 2018, pp. 183–187.

[20] E. W. Weisstein. "circle-circle intersection". Accessed: 2023-05-02. [Online]. Available: https://mathworld.wolfram.com/Circle-CircleIntersection.html

[21] A. W. M. van den Enden and N. A. M. Verhoeckx, *Discrete-time signal processing : an introduction*. Prentice Hall, 1989. [Online]. Available: https://archive.org/details/discretetimesign0000ende

[22] L. Litwin, “Fir and iir digital filters,” *IEEE Potentials*, vol. 19, no. 4, pp. 28–31, 2000.

[23] T. W. Parks and C. S. Burrus, *Digital filter design*. Wiley-Interscience, 1987.

[24] *"PCM6xx0-Q1 Automotive, 4-Channel and 6-Channel, 768-kHz, Audio ADC With Integrated Microphone Bias and Input Fault Diagnostics"*, Texas Instruments, 2020, accessed: 2023-05-24. [Online]. Available: https://www.ti.com/product/PCM6240-Q1