Audio Processing on Field-Programmable Gate Arrays

A thesis presented in partial fulfilment of the requirements for the degree of Bachelor of Science

Mitchell Sharum

S.B. Degree Candidate in Electrical Engineering

Faculty Advisor: Flavio Calmon

Harvard University School of Engineering and Applied Science

Cambridge MA

Date

Contents

List of Figures **iii**

List of Tables **iv**

1 Introduction 1

2 Background 2

2.1 Motivation and Purpose 2

2.2 System in which the Solution Resides 3

2.3 End-User 4

2.4 Stakeholders 4

2.5 Problem Statement 5

3 Existing Solutions and Previous Work 5

3.1 Existing Solutions 5

3.2 Previous Work 7

4 Technical Specifications 7

4.1 Justification of Technical Specifications 7

4.2 Measurement of Technical Specifications 10

5 Systems Diagram 12

6 Consideration of Multiple Design Approaches 13

6.1 Hardware Platform 13

6.2 Design Methodology 13

6.3 Audio Codec Selection 14
7 Quantitative Evaluation and Justification of Design Approaches 14
7.1 Audio Codec Interface Analysis 15
7.2 DSP Implementation Methodology 15
7.3 Distortion Algorithm Selection 17
7.4 Low-pass Filter Selection 19
8 Required Laboratory Training 21
Bibliography **25**

List of Figures

1 Signal Flow through the Music Performance System 3
2 Stakeholder Map 4
3 High Level Systems Diagram 12

List of Tables

1 Technical Specifications 8

2 Audio Codec Selection Analysis 16

3 DSP Implementation Approach Analysis 17

4 DSP Distortion Algorithm Comparison 20

5 DSP Filter Implementation Analysis - FIR vs IIR 22

6 Project Milestones 23

7 Project Budget 24

This paper outlines the background, technical specifications, design approaches, and implementation of a synthesizable audio processing system that runs on board a field-programmable gate array (FPGA). Running high-compute digital signal processing (DSP) algorithms on a secondary board significantly reduces latency that is often associated with CPU-based audio processing.

Acknowledgements

A special thanks to the advisor for this project Flavio Calmon, whose support and enthusiasm made this project possible, as well as to the ES100 teaching staff, and especially Salma Abu Ayyash, whose consistent and helpful presence ensured that all parts of the project moves smoothly and at-pace.

1 Introduction

Live music performance environments require that many audio effects be processed with low enough latency so as to be imperceptible to the human ear. Today this processing is done with either a digital audio workstation (DAW) run on a CPU, often introducing audible latency into the performance [1], or on an array of analog and digital effects pedals, the size and cost of which places a low ceiling on the range of effects at the musician's disposal. Selecting one of these two technologies for live audio processing constrains artists along either the axis of performance timing or emotive expression. An ideal audio processing solution must avoid forcing this tradeoff between latency and expressive range upon the end-user.

Highly reconfigurable electrical devices can be used to implement these same audio effects, leveraging digital signal processing hardware constructs to execute common operations (like multiply-accumulate) on the sub-millisecond scale. Finally, the range of effects represented using this reconfigurable hardware must be very large in order to guarantee that replacing a DAW or group of effects pedals with the audio processing system will not decrease the number of effects available to the end-user.

2 Background

$SNR = 20 \cdot \log_{10}\left(\frac{V_{S+N}}{V_N}\right)$

2.1 Motivation and Purpose

Often in live performance settings, musicians use computer programs that run algorithms on their CPU (central processing unit) to perform audio effects processing [2]. These algorithms implement effects like reverb, delay, distortion, etc, and play a crucial role in live performance. In such an environment, timing is critical and can be negatively impacted by CPU latency [3], which grows more severe with the number of effects the CPU is tasked with processing. Modern CPUs are not optimized for audio effects processing, and any programs/applications/tasks that the performer leaves running during their performance require hardware resources, further slowing down the CPU.

Hardware optimized for digital signal processing (DSP) can implement the same signal processing algorithms to offload work from the CPU and reduce latency in the live performance environment. Field-programmable gate arrays (FPGAs) are commonly used in sensor applications, control systems, and other fields that place a heavy emphasis on digital signal processing. This is because embedded in the logical fabric of an FPGA, there are many blocks of hardware specifically optimized for multiply-accumulate operations, which are integral to DSP applications. These aptly-named DSP blocks can be combined with other parts of the FPGA fabric by a hardware design engineer to instantiate audio effects in hardware rather than in software run on a CPU.

Application-specific integrated circuits (ASICs) were considered as an alternative to FPGAs for this design problem, due to their highly specialized nature and even lower latency. FPGAs were selected for this solution for two primary reasons. First, FPGAs allow for the rapid configuration of hardware solutions, as their logical interconnects can be reprogrammed in a matter of minutes. This means that a correct and well tested design can be

Figure 1: Signal Flow through the Music Performance System

actualized in circuitry extremely quickly and cheaply on FPGAs in comparison with ASICs, which can take months to produce and only become financially viable at mass-production. Second, the reconfigurability of an FPGA lends itself to the incremental development of many different effects on the same piece of silicon, increasing the feasibility of prototyping, which why in the development of large commercial projects, FPGAs are often used to verify the logical design of ASICs before tapeout) [4].

FPGAs offer a unique solution between the two possible extrema for audio effects processing, boasting much of the programmability of a CPU along with minimal latency. The selection of FPGAs as a solution to the problem of audio effects processing latency in live performance environments was motivated by their positioning as a near perfect middle ground, optimizing for both configurability and speed simultaneously.

2.2 System in which the Solution Resides

The solution to this problem (the effects processing hardware) resides in an intermediary position in the larger music performance system. The source of the sound (guitar strings, human voice) is translated into an analog signal by some electrical device (magnetic pickups, microphone), this analog signal then goes through an analog to digital converter (ADC) so that the digitally sampled representation of the signal can be routed to the FPGA effects processor. This processor outputs the processed digital signal to an output DAC (digital to analog converter), which is then sent to a speaker or audio interface, driving the processed sound into a performance/recording venue.

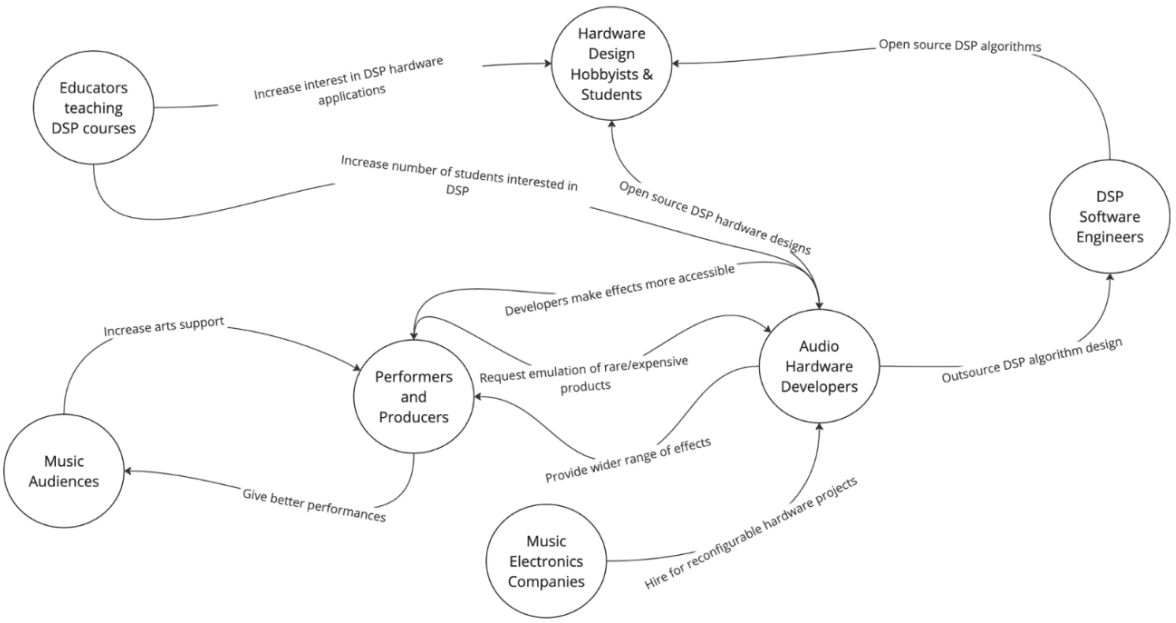

Figure 2: Stakeholder Map

2.3 End-User

The end-user of this solution is the professional or amateur musician, who will be able to toggle audio effects on and off based on the demands of the music that they are performing. Keeping this in mind, user-friendliness and ease of operability will remain a focal point of the design process. In the final implementation of the circuit, simple push buttons (and possibly turn dials) will be used to adjust and apply effects.

2.4 Stakeholders

The stakeholders affected by this solution include the performers, recording engineers and producers, music audiences, music electronics companies, individual audio hardware and software engineers, educators teaching DSP curricula, and hobbyists/students studying digital signal processing hardware and software. The potential interactions between these groups are marked by labeled arrows.

2.5 Problem Statement

Live music performance environments require that many audio effects be processed with low enough latency so as to be imperceptible to the human ear. Today this processing is done with either a digital audio workstation (DAW) run on a CPU, often introducing audible latency into the performance [1], or on an array of analog and digital effects pedals, the size and cost of which places a low ceiling on the range of effects at the musician's disposal. Selecting one of these two technologies for live audio processing constrains artists along either the axis of performance timing or emotive expression, which would be avoidable with a sufficiently configurable low-latency processor.

3 Existing Solutions and Previous Work

3.1 Existing Solutions

Antelope Audio is a company based out of Santa Monica, CA that specializes in the design of audio performance and recording equipment, audio interfaces, and other analog and digital devices relevant to music production. In their product lineup is the Synergy Core [5], a tool developed on FPGA for real time effects processing. They use extensive testing and measurement processes to replicate rare and expensive analog equipment extremely closely on using FPGAs with the goal of making "elite" audio effects more accessible to performers and producers. The Antelope Orion Studio is a $3,000 audio processing solution that near-perfectly emulates famous historical analog audio hardware setups. Like the Orion Studio, many of Antelope Audio's effects processing solutions aim to provide an extremely accurate approximation of famous (and often very expensive) analog effects pedals. The use of FPGAs to simulate the non-linear behavior of analog components allows for highly complex digital designs that emulate analog circuits. This level of specificity allows for the digital reconstruction of specific analog designs. Such a fine-grained level of approximation

is highly resource-intensive, both in the contexts of engineering and hardware costs, and contributes to the high price of antelope audio's effects processors. By focusing on the implementation of DSP algorithms to implement audio effects rather than analog emulation, the design and hardware resource costs of an audio processing device can be greatly reduced.

Other academic projects similar in nature to this have been completed in the past. At California Polytechnic State University in 2019, Carson Robles [6] implemented a similar FPGA guitar effects pedal, which included five parameterized effects. In his project, Carson Robles designed a custom PCB to interface with the FPGA effects processor. This PCB featured an array of potentiometers and an analog multiplexor that allowed for a different effective Voltage to be delivered to each audio FX module on the FPGA. This enables finer-grained control over the effects implemented, providing the user with the ability to mix the levels of each processed signal, "dialing in" a tone of their own. Dean Cannon, Tianyang Fang, and Jafar Saniie (researchers at the University of Chicago) published an article on the implementation of a delay effect using FPGAs [7]. The documentation of their project features the development of a simple graphical user interface (GUI) for interfacing with the board to adjust the delay parameters, as well as a detailed table detaling the resource utilization of the module. Their research project revealed that delay processing demands more of the FPGA's on board block RAM than any other resource. Memory optimization tactics, like data compression before storage (trading speed for memory efficiency), state minimization, or memory sharing between modules. No considerations were made for commercial production within this project, thus design and hardware costs were not discussed in the documentation. Ciprian Dragoi, Cristian Anghel, Cristian Stanciu, and Constantin Paleologu from the University of Bucharest have also collaborated on a multi-effect FPGA [8] audio processing application. In this processor implementation. The design of DSP algorithms was accelerated with signal processing tools in MATLAB. These algorithms were then translated into RTL for synthesis on the FPGA, which allowed for a more efficient design process. This RTL was analyzed using tools built into the Xilinx design suite to determine

on-board resource usage Each of these projects will provide guidance and inspiration to draw on for the initial implementation.

Each of these projects implements at least one effect that will be included in the final design of this project's FPGA based effects processor, and their approaches to the design of each effect will be considered. However, the actual implementation of these effects will be done independently using electric design automation (EDA) tools in the Xilinx Vivado suite, possibly integrating MATLAB for RTL code generation.

3.2 Previous Work

No previous design work has been done on this project. In order to familiarize myself with DSP on FPGAs, I intended to complete a self-guided project for extra-credit in ES156, but was unable to begin work on the project past the phase of reading a little bit of general DSP content. The project was going to be the implementation of a simple FIR low-pass filter in SystemVerilog, and to verify the filter in simulation. A low-pass filter (as well as a high-pass) will likely end up being implemented as one of multiple effects on the FPGA based audio processor for ES100.

This project is different from that considered in ES156 as it includes many more effects and synthesis on real-world hardware as opposed to simulation. The ES156 projected would have accounted for less than 5% of the hardware design that I anticipate creating in ES100, if it had ever gotten written.

4 Technical Specifications

4.1 Justification of Technical Specifications

- LAT - Signal delay through the audio processor's critical path should be comfortably within the minimum delay perceptible to the human ear (12-15 milliseconds) [9]. If the delay between a signal entering and exiting the processor were to approach 15 mil-

Table 1: Technical Specifications

Label	Requirement	Specification	Value	Measurement
LAT	Latency	Maximum Tolerable Latency	5 ms	Xsim
NFX	Audio FX	Minimum number of unique effects	4	Counting
IGN	Input Gain	Minimum necessary input boost	10 dB	Oscilloscope
LUT	Lookup Tables	Maximum percentage of on-board combinatorial logic elements	20%	Vivado
BRM	Block RAM	Maximum percentage of on-board memory used	50%	Vivado
DSP	DSP Blocks	Maximum number of signal signal processing-optimized resources used	15%	Vivado
SNR	Signal-Noise Ratio	Ratio between the Signal to be processed, and unwanted noise	90dB	Oscilloscope

liseconds, then the processor would not be suitable for a live performance environment.

- NFX - The number of on-board effects included in the final implementation should reflect the FPGA's ability to support a wide range of custom hardware, and to perform parallelized computations efficiently. Thus multiple effects must be able to be processed simultaneously.

- IGN - The audio effects processor takes as its input an instrument signal. Generally speaking, these signals are high-impedance, and not compatible with most FPGA audio processing peripherals. For sampling and processing, the instrument signal must be boosted to "line level," i.e. the Voltage at which most common 3.5mm headphone jacks operate. The necessary gain to boost a high-impedance guitar signal to line level

is roughly 10dB.

- LUT - On each FPGA there exist a certain number of dedicated lookup tables (LUTs) that implement basic digital logic primitives. In order to ensure design efficiency, the number of these base combinatorial elements that are used in the design should be minimized. This also allows space for additional effects instances to be added to the same board at a later date. A 20% constraint on total on-board LUT usage is based on similarly scaled implementations in *Existing Solutions and Previous Work*, and helps to ensure an efficient use of power-consuming resources [6, 10].

- BRM - Dedicated block RAM on the FPGA will be used rather extensively for audio effects that require memory of previous sound bytes, or that implement some form of feedback (delay, reverb). These effects are memory intensive, and thus justify a sizeable amount of block RAM. Sound buffering to meet effects timing requirements will require dedicated memory as well. Previous implementations [10] have been shown to demand 50% of the available block RAM on Xilinx Zynq 7 series development boards.

- DSP - DSP blocks are on-board resources specifically designed to efficiently compute multiply-accumulate and other complex operations very efficiently. These computations are especially critical in DSP hardware applications, and will be used appropriately in the implementation of audio effects processing. Multi-effect processors like the subject of this project tend to run with up to 15% DSP utilization. Thus an efficient design should use an upper bound of 15% of the on-board DSP blocks, again facilitating the future addition of new audio effects on the same board.

- SNR - The signal-to noise ratio for most consumer audio electronics is roughly 90dB

(headphones like the FiiO A3 are rated at an SNR of 93dB, whereas the PreSonus AduioBox (audio interface) is rated at 96dB. A drastic difference between the amplitude of signal and noise Voltages will help to ensure that the processed signal is produced cleanly, and that input error due to noise does not propagate excessively.

4.2 Measurement of Technical Specifications

- LAT - Two methods of measurement will be used to measure latency. First, the number of cycles that it takes in simulation (Vivado Xsim) for data to propagate from the input to the output will be divided by the clock frequency of the design, yielding the number of seconds it takes for one sound sample to run from input to output (latency). Alternatively, once the design has been actualized in hardware, an oscilloscope will be used to probe the input and output to the system, and the time difference between the two signals will be measured in-lab.

- NFX - The number of functional effects will simply be counted once the design has been completed. Using the percentage of on-board resources used in the initial design, an estimate will be made of the maximum number of audio effects that can fit on one Zynq7000 FPGA.

- IGN - A voltmeter probing the output of the instrument and the output of the preamp will suffice, as the ratio between these two Voltages is the input gain for the system.

- LUT - Vivado's built in synthesis tools provide logs after the place and route process on an FPGA. These tools provide information about the number of LUTs used in the design, which can then be divided by the total number of LUTs on the chip in the datasheet for the FPGA that is being used (within the scope of this project, the

Zynq7000)

- BRM - Vivado's built in synthesis tools provide logs after the place and route process on an FPGA. These tools provide information about the number of BRAMs used in the design, which can then be divided by the total number of BRAMs on the chip in the datasheet for the FPGA that is being used (within the scope of this project, the Zynq7000)

- DSP - Vivado's built in synthesis tools provide logs after the place and route process on an FPGA. These tools provide information about the number of DSPs used in the design, which can then be divided by the total number of DSPs on the chip in the datasheet for the FPGA that is being used (within the scope of this project, the Zynq7000)

- SNR - The signal to noise ratio for the FPGA audio effects processor will be measure using an oscilloscope probing the board's input and output. A synthesized sine wave will be passed into the module, and noise will be measured at the ouput in comparison with the input (the sine wave will most likely be synthesized in Logic Pro X and passed into the board via the same audio interface as the guitar signal).

- *Though not a quantifiable technical specification, the sound produced by the audio processor must be evaluated subjectively in order to verify that the effects applied match their expectations for that musical effect (ideally roughly 25 musicians will evaluate the board). The finished product will therefore be presented to musicians with knowledge of basic principles of audio production and effects utilization. These musicians will be asked to identify the effects that are applied to an incoming guitar signal by listening*

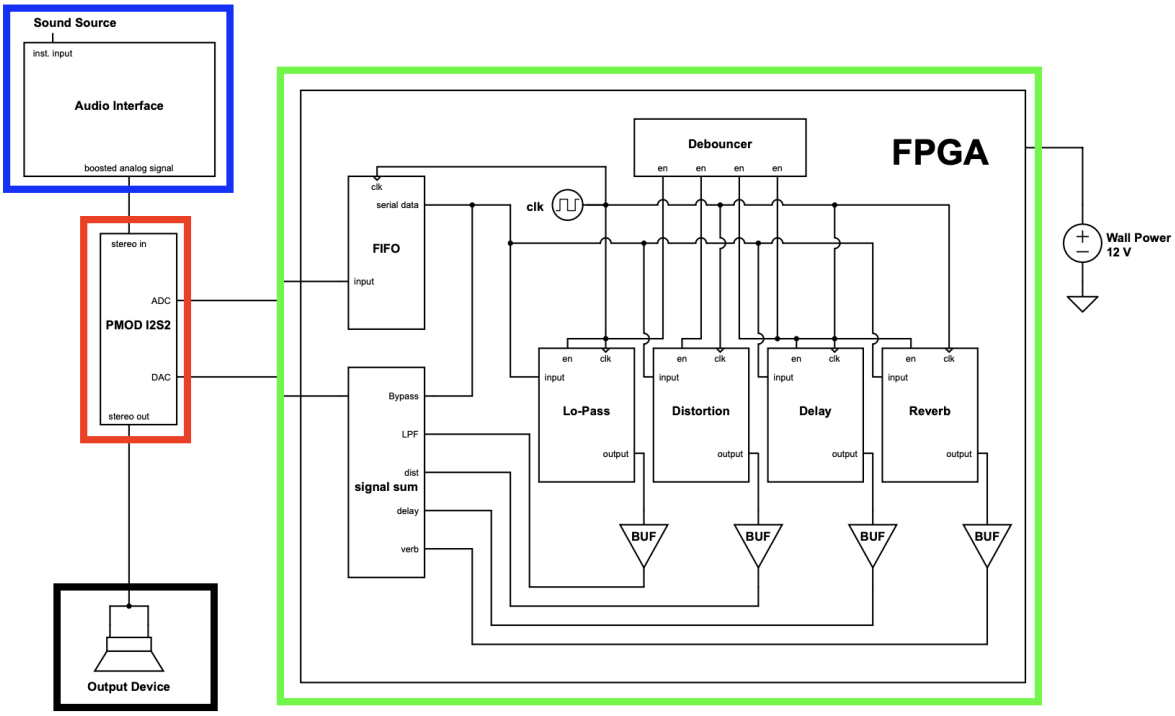

Figure 3: High Level Systems Diagram

to the system's output. A success rate in identification of 90% or higher will indicate that the effect has been correctly implemented from a musical standpoint.

5 Systems Diagram

The high-level system diagram can be broken into four primary signal treatment blocks: signal generation (blue), conversion (red), processing (green), and driving (black). The Signal generation is characterized by a sound source, likely an instrument of some sort, that generates a high-impedance signal, which is then passed to the audio interface for amplification to line level. This signal is ready to be sampled, and handed off to the conversion block (the PMOD I2S2 audio codec). This sampled audio is fed through the audio processing module synthesized on the FPGA, where each effect will be applied according to which enable signals are asserted (via mechanical switches on the board that will be appropriately debounced to reduce error). After processing, buffering for synchronization, the sound signals (with var-

ious FX) will be summed and routed out of the board. There they be reconverted into an analog signal, which will be used to drive an output device (speakers, headphones, etc.).

6 Consideration of Multiple Design Approaches

6.1 Hardware Platform

Two design approaches were considered at the top level: ASIC and FPGA as platforms on which the audio processor design would live. Each solution has its merits, with the primary benefits of ASICs being lower cost at large-scale production, and lower latency due to highly optimized signal routing. With latency being the most critical technical specification to the success of the design, this advantage is the most significant held by an ASIC-focused design, rather than a design for FPGA. FPGAs also boast very low latency, enough to still pass the technical specifications for the implementation. The added benefit of rapid reconfigurability and fast design changes resulted in the selection of FPGAs over ASICs for the audio processor design. A design actualized on an FPGA makes more sense from an engineering perspective due to the massive difference in prototyping cost and time.

6.2 Design Methodology

The Xilinx ISE Design Suite is the most commonly used for FPGA-based projects. It hosts many tools for simulation, synthesis, build, and evaluation of FPGA projects. Many of these tools allow for one design to be achieved through various approaches. The design of the audio processor can be completed either by leveraging the intellectual property (IP) blocks available in the Xilinx ISE, by using high-level synthesis (HLS) to compile hardware from high level descriptions, or at the register-transfer level (RTL). While creating circuit designs in the Vivado GUI with the IP block integrator offers the greatest ease of implementation, it offers the lowest degree of control over the implementation. HLS offers slightly more control over individual implementation decisions than the IP block interface, but not as much room

for control and optimization as handwritten RTL. Both as an engineering exercise (to ensure a high enough degree of challenge for ES100), and for the purpose of writing maximally optimized hardware descriptions, the majority of the design will be implemented in RTL. The IP block integrator will be used sparingly for implementing predetermined protocols to communicate with off-board peripherals (I2S, I2C, etc.).

6.3 Audio Codec Selection

For conversion between analog and digital signals, two primary options were available for interfacing with the programmable logic fabric of the Zynq7000 FPGA. The first is the on-board ADAU1761, a 24-bit audio codec that interfaces directly with the arm processor on the Zedboard Zynq7000. The spatial locality of this audio codec contributes to a speedup over the alternative, but interfacing the programmable logic of the Zynq7000 FPGA with the onboard arm core poses an additional integration challenge. The second option, the PMOD I2S2, is a peripheral audio codec (also 24-bit), that operates with the I2S (inter-integrated circuit sound) communications protocol. This module can be plugged into any of the 8-pin PMOD peripheral slots on the Zedboard, and integrated into the digital design without the need for an interface with a CPU core. For this reason, and due to the fact that the added latency is not significant enough to endanger technical specifications for the project (adding microseconds will not approach a 5 millisecond limit), the PMOD I2S2 was selected as the codec that will be used in the initial design.

7 Quantitative Evaluation and Justification of Design Approaches

The implementation of an FPGA-based audio effects processor requires careful quantitative analysis of design tradeoffs such that the design meets all technical specifications. Two critical design decisions significantly impact the system's performance: audio codec selection

and the implementation approach (RTL vs HLS) for the digital signal processing modules.

7.1 Audio Codec Interface Analysis

The selection between the PMOD I2S2 peripheral codec and the onboard ADAU1761 codec represents a fundamental architectural decision in the audio signal chain. This choice directly impacts several technical specifications, most critically LAT (maximum tolerable latency) and SNR (signal-to-noise ratio). The total signal path latency through the codec can be modeled as:

$$T_{total} = T_{conversion} + T_{interface} + T_{processing} + T_{output} \tag{1}$$

where $T_{conversion}$ represents the ADC/DAC conversion time at the selected 48kHz sampling rate, $T_{interface}$ encompasses protocol and communication overhead, $T_{processing}$ accounts for on-chip processing delays, and T_{output} represents output buffering and synchronization delays.

A comprehensive evaluation of both codecs was conducted using a Pugh matrix, making considerations for key performance metrics and effects on the processof implementation:

7.2 DSP Implementation Methodology

The implementation of digital signal processing algorithms presents another critical design decision. The choice between Register Transfer Level (RTL) design and High-Level Synthesis (HLS) impacts ease of implementation, resource utilization, and system performance. One such resource is block RAM, which is used in effects like delaya dn reverb. For a simple delay implementation, the fundamental memory requirements can be expressed as:

$$\text{Effect Buffer Size} = f_s \times T_{delay} \times W_{sample} \tag{2}$$

where f_s is the sampling frequency (48kHz), T_{delay} is the maximum delay time, and

Table 2: Audio Codec Selection Analysis

Criterion	**Weight**	**PMOD I2S2**	**ADAU1761**	**Justification**
Integration Complexity	0.30	+1	-1	PMOD interfaces directly with programmable logic, ADAU1761 requires complex ARM core integration
Signal Path Latency	0.30	+1	-1	Direct PL connection minimizes interface latency compared to ARM core routing
Resource Utilization	0.20	+1	0	PMOD requires minimal FPGA resources while ADAU1761 needs additional interface logic
Signal Quality	0.20	-1	+1	ADAU1761's integrated design provides marginally better noise performance
Weighted Total:		**+0.60**	**-0.40**	

W_{sample} is the sample width in bits.

Other resource usage, such as that of DSP blocks and lookup tables (LUTs), are also critical considerations, and are used in almost every digital effect instantiation.

To balance the pros and cons of RTL and HLS implementations, a quantitative comparison was conducted via Pugh matrix:

The quantitative analysis presented in these sections demonstrates that while certain design choices present clear advantages (such as PMOD I2S2 for codec interface), others (such as RTL vs HLS implementation) require careful balancing of competing factors. These analyses directly informed the design decisions, ensuring that the implementation would meet the technical specifications outlined in section 4 while maintaining practical feasibility for the target FPGA platform.

Table 3: DSP Implementation Approach Analysis

Criterion	**Weight**	**RTL**	**HLS**	**Justification**
Resource Efficiency	0.30	+1	-1	RTL enables precise control over hardware resource allocation
Development Time	0.25	0	+1	HLS significantly reduces initial development and verification time
Performance Optimization	0.20	+1	-1	RTL allows cycle-accurate optimization and timing control
Code Maintainability	0.15	-1	+1	HLS code is more readable and easier to modify
Design Reusability	0.10	0	+1	HLS facilitates easier porting to different hardware
Weighted Total:		**+0.30**	**+0.00**	

7.3 Distortion Algorithm Selection

In order to implement the distortion effect, the first necessary step was the selection of a DSP algorithm that would be implemented in hardware. Similar auditory effects can result from various processing algorithms, and finding an optimal DSP algorithm for deployment on the FPGA greatly impacts both the feasibility and the efficiency of the resulting module.

Two primary options were considered for the distortion algorithm that was implemented. First, hard clipping, which amplifies the signal via multiplication by a gain constant. This amplified digital signal is next compared to a predetermined threshold, and if it exceeds the threshold in magnitude, then the amplified wave is cut down to the threshold value. This algorithm can be expressed via a simple piecewise function:

$$y[n] = \begin{cases} \text{threshold}, & \text{if } x[n] > \text{threshold} \\ -\text{threshold}, & \text{if } x[n] < -\text{threshold} \\ x[n], & \text{otherwise} \end{cases}$$

The second form of distortion algorithm is implementation via wave-shaping algorithm. This method enables finer grained control of the distortion's sound (beyond gain and clipping threshold, greater frequency control). There are many common methods for wave shaping distortion, each of which involve the use of a different polynomial or trigonometric function for the generation of an output signal, which becomes more distorted as the amplitude of the input signal increases. A few examples of wave shaping distortion algorithms are as follows:

1. Soft clipping via hyperbolic tangent - By making the sound signal the input to a tan function, multiplied by a gain parameter a, the output signal becomes similar to the input for small magnitudes, and is rounded down to around $1.0 * a$ for large signals. This results into a more dynamically sensitive clipping than regular, hard clipping.

$$y[n] = \tanh(ax[n])$$

2. Cubic Distortion - Distortion is achieved by cubing the input signal, dividing it by three, and subtracting this newly found value from the original signal. This results in small signals being largely unaffected, and large amplitudes being reduced greatly, sometimes even flipping signs, or "folding over." This results in a much harsher distortion sound than tangent wave shaping, introducing odd harmonics into the output signal.

$$y[n] = x[n] - \frac{x[n]^3}{3}$$

3. Distortion via multiplication with an approximated polynomial - Using a custom built polynomial with gain parameters as the coefficient to each term, a highly specific

distortion effect, with a parametrically specified mix of input, even, and odd harmonics resulting at the output. This wave shaping algorithm can also be implemented as a linear filter, and offers a wide range of distorted sounds at the expense of much more intensive computation.

$$y[n] = \sum_{k=0}^{N} a_k (x[n])^k$$

The pros and cons of each form of distortion were weighed against each other quantitatively using a Pugh decision matrix, where the most important decision criteria were determined to be resource usage, complexity of implementation, and latency. These factors served to determine both which distortion would be optimal for the scope of this project and which effect would be the most efficient in synthesis on FPGA. Ultimately clip distortion was selected as the algorithm to be implemented for this project, but even much more computationally intensive implementations can be designed to run on an FPGA, with virtually any form of wave shaping being feasible given enough time and digital design expertise. Though both methods for implementing a distortion algorithm have their benefits, the scope of this project lends itself more readily towards clipping, primarily for reasons of resource utilization and ease of implementation.

7.4 Low-pass Filter Selection

When it came to the selection of a method for implementing a DSP low-pass filtering algorithm, two options were the most desirable: finite impulse response (FIR) filtering and an infinite impulse response (IIR) filtering approach. Just as was the case with distortion, each of the two have benefits and drawbacks related to resource efficiency, complexity of implementation, etc.

Both FIR and IIR filters are implemented similarly, with the key distinction that IIT filters rely not only on previous inputs, but previous outputs as well when generating the current term.

Table 4: DSP Distortion Algorithm Comparison

Criterion	**Weight**	**Clip**	**Wave Shaping**	**Justification**
Resource Usage	0.30	+1	-1	Clip distortion requires minimal logic resources (comparators only), while wave shaping needs multipliers and memory
Implementation Complexity	0.25	+1	-1	Clip implementation is straightforward with simple threshold comparison logic, wave shaping requires complex polynomial calculations
Latency	0.20	+1	-1	Clip achieves single-cycle latency, wave shaping requires multiple cycles for polynomial evaluation
Sound Quality Control	0.15	-1	+1	Wave shaping offers more flexible control over the distortion characteristics
Numerical Stability	0.10	+1	-1	Clip has inherent output bounds, wave shaping prone to overflow and precision issues
Weighted Total:		**+0.65**	**-0.65**	

FIR filters can be viewed as a moving window average, where the last N terms are summed to generate the current output. Each previous term is multiplied by a selected coefficient. The input-coefficient pairs are called the "taps" of the filter. Coefficient values affect the aggression of the filtering algorithm, and the number of taps determine the filter roll-off after the cutoff frequency. The cutoff frequency of an FIR can be determined by calculating filter coefficients with the following formula:

$$b[n] = \frac{\sin(2\pi f_c n)}{\pi n}$$

Where $f_c = \frac{f_{cutoff}}{f_{sampling}}$ is the normalized cutoff frequency, n is the coefficient index, and the coefficient $b[0]$ is handled as a special case: $b[0] = 2f_c$.

FIR filters are implemented very similarly to FIRs, but they incorporate a feedback loop that consists of previous outputs multiplied with coefficients. These recursive values are also summed during the generation of the current term. The addition of this feedback loop often results in peaks at certain frequencies, which can add an "analog" feel to the filter's sound, due to added resonance. One key issue of IIR filters is that they also pose potential feedback issues, and failr to maintain linear phase when applied to a signal.

The benefits and drawbacks posed by each of the two filtering strategies were compared using a Pugh matrix, and as a result, the finite impulse response filter was selected for implementation and synthesis on the FPGA.

8 Required Laboratory Training

Due to the very limited physicality of this project, the only required laboratory training is for the use of the EE lab in SEC 1.111 (for oscilloscopes to make measurements on tech specs). All other lab work will be done in the computing lab in the SEC, which requires no training to use. (General Laboratory Safety (EHS) training was completed on April 10th, 2024)

Table 5: DSP Filter Implementation Analysis - FIR vs IIR

Criterion	**Weight**	**FIR**	**IIR**	**Justification**
Resource Efficiency	0.30	-1	+1	IIR requires fewer coefficients and multipliers for similar frequency response
Phase Response	0.25	+1	-1	FIR provides linear phase, IIR has nonlinear phase response
Stability	0.20	+1	-1	FIR is inherently stable, IIR can become unstable due to feedback
Implementation Complexity	0.15	+1	-1	FIR is straightforward to implement, IIR requires careful consideration of feedback paths
Filter Sharpness	0.10	-1	+1	IIR achieves sharper cutoffs with fewer coefficients
Weighted Total:		**+0.15**	**-0.15**	

Table 6: Project Milestones

Week	**Phase**	**Preliminary Milestones**
1	Define	Decide on effects list and user interaction with the board
2	Define	Define interconnect methods for instrument-to-FPGA signal
3	Design	Design bypass mode on FPGA, integrating PMOD, ADC/DAC
4	Verify	Verify bypass mode, simulate and pass guitar/vocal signals
5	Design	Design effect #1 (delay?)
6	Verify	Verify effect #1 using physical and simulation tests
7	Design	Design effect #2 (overdrive?)
8	Verify	Verify effect #2 (using SystemVerilog or UVM)
9	Design	Design effect #3 and #4 (filtering or reverb)
10	Verify	Verify effects #3 and #4
11	Measure	Measure latency of signal through each effect/bypass mode
12	Measure	Adjust signal gain and latency based on measurements
13	Compile	Compile results, complete write-up, prepare for demonstration

Table 7: Project Budget

Item	**Source**	**Price**	**Obtained**
Zedboard Zynq7000 FPGA	Digilent	$589.00	Yes
Digilent PMOD I2S2	Digilent B	$24.99	Yes
Xilinx Vivado Design Suite	Xilinx	$0.00	Yes
MATLAB Student License	MathWorks	$0.00	Yes
Scarlett 2i2 Audio Interface	FocusRite	$0.00	Yes
Generic Audio Speaker	–	$0.00	Yes
Total	–	$613.99	Yes

Bibliography

[1] "DAW Latency and CPU," EDMProd, Accessed: Sept. 1, 2024. [Online]. Available: https://www.edmprod.com/daw-latency-and-cpu/

[2] "Inspiring Ways to Use Logic Pro for Live Performance and DJing," LogicXX, Accessed: Sept. 1, 2024. [Online]. Available: https://logicxx.com/blogs/news/inspiring-ways-to-use-logic-pro-for-live-performance-and-djing

[3] "Advice on the Best Setup for Live Performance Using Logic," Logic Pro Help, Accessed: Sept. 1, 2024. [Online]. Available: https://www.logicprohelp.com/forums/topic/152400-advice-on-the-best-setup-for-live-performance-using-logic/

[4] "ASIC vs FPGA: What's the Difference?" ASIC North, Accessed: Sept. 1, 2024. [Online]. Available: https://www.asicnorth.com/blog/asic-vs-fpga-difference/

[5] "Synergy Core," Antelope Audio, Accessed: Sept. 1, 2024. [Online]. Available: https://en.antelopeaudio.com/synergy-core/

[6] J. Lipshy, "Development of a Digital Sound Processor Using FPGAs," California Polytechnic State University, San Luis Obispo, 2013. [Online]. Available:

https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1326& context=cpesp

[7] G. T. Metcalf et al., "Low Power Neural Architecture Search for TinyML Applications," IEEE J. Emerg. Sel. Topics Circuits Syst., vol. 12, no. 2, pp. 255-266, Jun. 2022, doi: 10.1109/JETCAS.2022.9813875.

[8] S. Wang et al., "High-Fidelity FPGA-Based Emulation of Power Systems with Modular Multilevel Converters," IEEE Access, vol. 9, pp. 100771-100784, 2021, doi: 10.1109/ACCESS.2021.9515041.

[9] R. Y. Litovsky, H. S. Colburn, W. A. Yost, and S. J. Guzman, "The Precedence Effect," J. Acoust. Soc. Am., vol. 106, no. 4, pp. 1633-1654, 1999. [Online]. Available: https://www.cogsci.msu.edu/DSS/2019-2020/Hartmann/Litovsky_et_al_1999.pdf

[10] C. Dragoi, C. Anghel, C. Stanciu, and C. Paleologu, "Efficient FPGA Implementation of Classic Audio Effects," in *Proceedings of the 2021 IEEE International Symposium on Signals, Circuits and Systems (ISSCS)*, 2021, doi: 10.1109/ISSCS9515041.