Digital ASIC Fabrication

DESIGN DOCUMENT

sdmay24-21

Client / Advisor: Dr. Henry Duwe

Team Members:
Yu Wei Tan
Jonathan Hess
Samuel Heikens
Haris Khan

sdmay24-21@iastate.edu
https://sdmay24-21.sd.ece.iastate.edu/

Revised: April 26th 2023, Version 2

Executive Summary

Development Standards & Practices Used

- IEEE 1364-2005: IEEE Standard for Verilog Hardware Description Language
- IEEE 1801-2018: IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems
- ISO/IEC 9899:2018: C programming language (C17)
- IEEE 1076.4-2000: IEEE Standard VITAL ASIC Modelling Specification
- Efabless Open Source Software
- Serial Peripheral Interface (SPI) Protocol

Summary of Requirements

- The design is approved for fabrication from eFabless
- The design can be mounted and used with a PCB board
- The design has to pass the MPW precheck tool

Applicable Courses from Iowa State University Curriculum

- CPRE 281 — Digital Logic
- CPRE 288 — Embedded Systems
- CPRE 381 — Computer Organization and Assembly Level Programming
- CPRE 488 — Embedded Systems Design
- EE 330 — Integrating Electronics
- EE 435 — Analog VLSI Design
- EE 465 — Digital VLSI Design
- EE 501 — Analog and Mixed-Signal VLSI Circuit Design Techniques

New Skills/Knowledge acquired that was not taught in courses

Skills:

- ASIC design
- Verilog implementation
- Working with open-sourced tools
- Learning to work with unfamiliar tools
- Agile workflow

Tools:

- KiCAD
- Verilator
- OpenLane

Table of Contents

1. The Team **5**
1.1 Team Members 5
1.2 Required Skill Sets for Your Project 6
1.3 Skill Sets Covered by the Team 6
1.4 Project Management Style Adopted by the Team 7
1.5 Initial Project Management Roles 7
2. Introduction **8**
2.1 Problem Statement 8
2.2 Requirements & Constraints 8
2.3 Engineering Standards 11
2.4 Intended Users and Uses 12
3. Project Plan **13**
3.1 Task Decomposition 13
3.2 Project Management/Tracking Procedures 15
3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria 16
3.4 Project Timeline/Schedule 18
3.5 Risks and Risk Management/Mitigation 18
3.6 Personnel Effort Requirements 19
3.7 OTHER RESOURCE REQUIREMENTS 20
4. Design **21**
4.1 Design Content 21
4.2 Design Complexity 21
4.3 Modern Engineering Tools 22
4.4 Design Context 22
4.5 Prior Work/Solutions 23
4.6 Design Decisions 24
4.7 Proposed Design 25
4.7.1 Design 0 (Initial Design) 25
Functionality: 26
4.7.2 Design 1 26
4.7.3 Design 2 27
4.8 Module Design 33
i. Control Module: 33
ii. SPI-Module 33
iii. Compression Module 34
iv. Memory Controller Module 35
v. On-Chip SRAM Module 36
4.9 Hardening 37
5 Testing **37**

5.1 Unit Testing 37
5.2 Interface Testing 39
5.3 Integration Testing 39
5.4 System Testing 39
5.5 Regression Testing 40
5.6 Acceptance Testing 40
6. Professionalism 40
6.1 Areas of Responsibility 40
Differences Between IEEE and NSPE: 43
6.2 Project Specific Professional Responsibility Areas 44
6.3 MOST APPLICABLE PROFESSIONAL RESPONSIBILITY AREA 44
7. Closing Material 45
7.1 Discussion 45
7.2 Conclusion 45
7.3 References 45
7.4 Appendices 45
7.4.1 Team Contract 46
Team Members 46
Team Procedures 46
Participation Expectations 46
Leadership 47
Collaboration and Inclusion 48
Goal-Setting, Planning, and Execution 49
Consequences for Not Adhering to Team Contract 49

List symbols/definitions

ADC - Analog-to-Digital Converter

DAC - Digital-to-Analog Converter

SRAM - Static Random Access Memory

SPI - Serial Peripheral Interface

PCB - Printed Circuit Board

ASIC - Application-Specific Integrated Circuit

Verilog HDL - Verilog Hardware Description Language

SoC - System on chip

eFabless - Open source fabrication company that will fabricate our chip, and provide us with design resources/tools

SkyWater 130nm - Fabrication process used by eFabless

Caravel Harness - Provided wrapper around our design which includes an SoC

User Area - Our design space within the Caravel Harness

Management Area - Part of the Caravel Harness that includes management utilities, SoC, and logic analyzer probes

OpenLane - The collection of open-sourced tools provided by eFabless

KiCAD - PCB design tool

Verilator - Verilog HDL design tool

1. The Team

1.1 Team Members

- Yu Wei Tan
- Jonathan Hess
- Samuel Heikens
- Haris Khan

1.2 Required Skill Sets for Your Project

- Digital Logic
- Embedded Systems
- Verilog
- Python
- PCB Design
- ASIC layout/hardening
- Debugging
- Open communication

1.3 Skill Sets Covered by the Team

- Digital Logic - All members
- Embedded Systems - All members
- Verilog - All members
- Python -Jonathan, Haris
- PCB Design - Yu Wei
- ASIC layout/hardening - Samuel
- Debugging - All members
- Open communication - All members

1.4 Project Management Style Adopted by the Team

Goal-Setting, Planning, and Execution

1. Team goals:
   We want to create a justifiable top-level design with the expected inputs and outputs of each module. We also want to have prototypes of each module, so that we know that our project is feasible.

2. Strategies for planning and assigning individual and teamwork:
   We will assign work based on specialization, skills prior experiences, and workload balance.

3. Strategies for keeping on task:
   We will communicate with each other in order to receive updates on tasks for our project. We will also be using Git in order to track issues that need work. We will also communicate with faculty to judge the status of assignments.

1.5 Initial Project Management Roles

- Haris Khan - Digital Design
- Jonathan Hess - Digital Design
- Yu Wei Tan - Analog Design
- Samuel Heikens - Analog Design

2. Introduction

2.1 Problem Statement

Traditionally, a guitar pedal can only perform one specific sound effect, such as looping the signal. If a guitar player were to try to play with a different sound effect, they would need to use a different pedal. This requires more physical space, more inputs to update, and creates more e-waste at the end of life. Our project eliminates the need to have multiple guitar pedals by creating a chip that can allow a pedal to perform multiple different sound effects.

2.2 Requirements & Constraints

i. For the current Google eFabless program, the following requirements must be fulfilled:

- "The project must be targeted on the currently-supported SkyWater Open PDK for the 130nm process."

  This requirement is fulfilled as we are using the Open PDK for the memory components as well as in the Caravel harness.

- "The project must be posted on a git-compatible repo and be publicly accessible."

  This requirement is fulfilled with our GitHub repository.

- "The top level of the project must include a LICENSE file for an approved open-source license agreement. Third-party source code must be identified, and source code must contain proper headers."

  This requirement is fulfilled in the git repository's LICENSE file.

- The repo must include project documentation and adhere to Google's inclusive language guidelines.
- "The project must be fully open. The project must contain a GDSII layout, which must be reproducible from the source contained in the project."

- "Projects must use a common test harness and padframe based on the Caravel repo. New projects should start by duplicating or forking the Caravel User Project repo and implementing their project using the

user_project_wrapper. The Caravel repo is configured as a submodule in the project under the 'Caravel' directory. Note -- you do not need to initialize nor clone the Caravel sub-directory to complete or submit your project. See the project README for further instructions. The projects must be implemented within the user space of the layout and meet all requirements for the Caravel."

- "Projects must successfully pass the Open MPW precheck tool, including LVS and DRC clean using the referenced versions of OpenLane flow. Projects should implement and pass a simulation test bench for their design integrated into Caravel. The Caravel User Project provides an example of how to implement this."

    We have a git runner script that tests our hardens and tests our design with the OpenLane flow.

ii. Voltage Input from Guitar: 740 mV peak to peak

iii. Our chip will have three different filters. Each of these filters can be used one at a time. The filters will be **delay**, **reverb**, and **compression**.

Delay:

- analog input for delay settings
- analog input for magnitude settings
- max delay of 5 seconds (stretch goal of 20 seconds with off-chip memory)

Reverb:

- analog input for delay settings
- analog input for magnitude settings
- max reverb of 5 seconds (stretch goal of 20 seconds with off-chip memory)

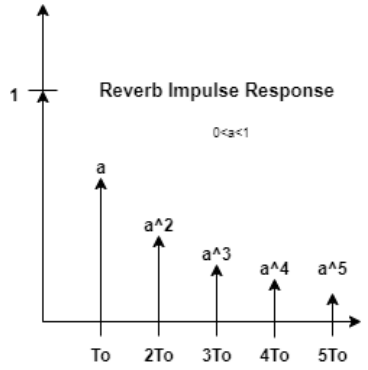

Figure: Plot representation of the reverb impulse response

Compression:

- analog input for threshold-linear (8-bit)
- analog input for ratio-linear (8-bit)
- The output will be scaled by a factor greater than 1 in order to keep the volume similar to before the effect is added

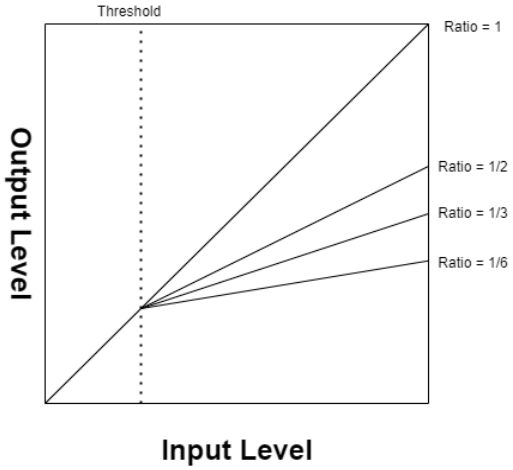

Figure: Plot representation of how compression works

iv. We will be using the EFabless process for our design, including the breakout board (PCB), and any design tools (e.g. Verilator - for Verilog design)

v. Design constraints & considerations:

- 10 kHz sampling rate - > 2x of 2637Hz (Highest note)
- Clock will need to be at least 160kHz, 16 bits * 10kHz
- Memory: 10kHz*2 bytes*5 seconds = 100k Bytes for audio data
- OnChipSRAM: 10kHz*2 bytes*1 second = 20kbytes
- The audio ADC and DAC will use 16-bit signals for audio data
- Other analog inputs will be converted to 8-bit digital signals
- Stretch Goal: Making a custom impulse response

vi. Design limitations:

- 38 I/O pins are available for use on the breakout board
- 4 power pads

2.3 Engineering Standards

Below are the engineering standards that we have used/abided by for our design:

- IEEE 1364-2005: IEEE Standard for Verilog Hardware Description Language
- IEEE 1801-2018: IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems
- ISO/IEC 9899:2018: C programming language (C17)
- IEEE 1076.4-2000: IEEE Standard VITAL ASIC Modelling Specification
- Wishbone Bus
- SPI Protocol
- Audio I/O: The cable is a 2.1 mm DC input at 9 volts, with the center contact carrying the negative and the outer contact carrying the positive part of the current. Boss Standard. Requires negative polarity. Uses the ¼ TS cables for input and output.

2.4 Intended Users and Uses

Guitar players

By creating this pedal, guitarists will not have to purchase as many pedals and will have more flexibility. While trying things out they will not have to edit the physical setup of their guitar and can spend more time creating and playing music.

Studio artists

By creating this pedal, new studio artists can standardize the pedals that they carry, reducing the price and variability.

Analog and digital engineers

By creating a new open source project these engineers will gain more knowledge and a larger resources pool.

3. Project Plan

3.1 Task Decomposition

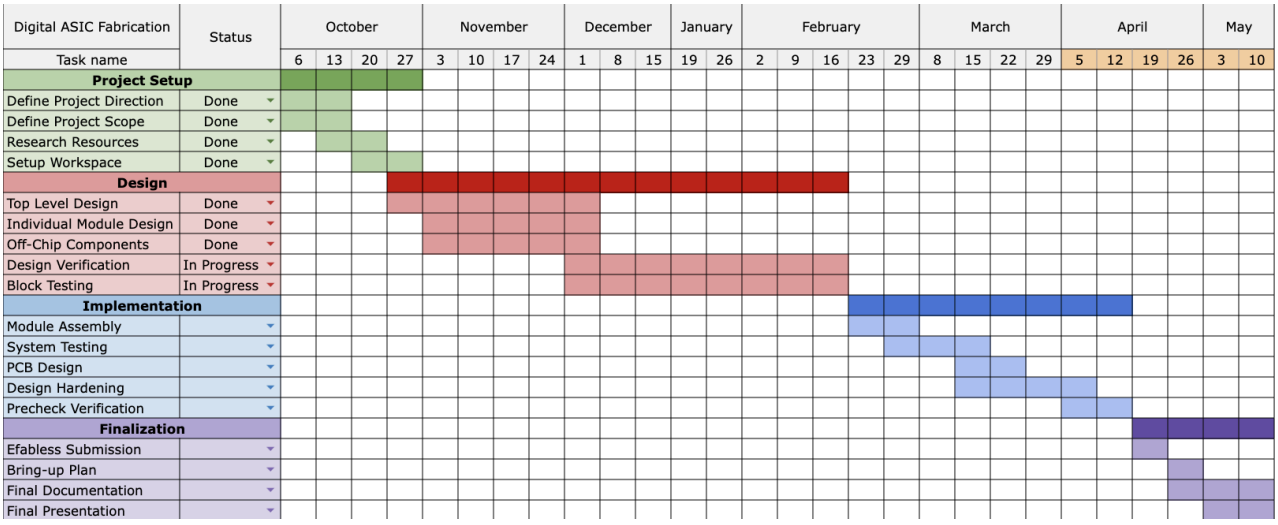

Figure: Gantt chart of our initial projected project timeline

Below are the steps that the team has taken to complete the project:

i. Define the scope & specifications of our project:

a. Brainstorm a specific application for our project
b. Compile a list of functions that may be relevant to our project
c. Need a project that can be finished within the time frame necessary and also challenging enough

ii. Install and familiarize ourselves with design tools/workspace:

a. Install digital tools that we will use to design our project
b. Familiarize ourselves with how to do basic design in the tool and how it can be integrated with EFabless
c. This task is necessary so we can design our chip with these tools

iii. Design a top-level diagram for the user area:

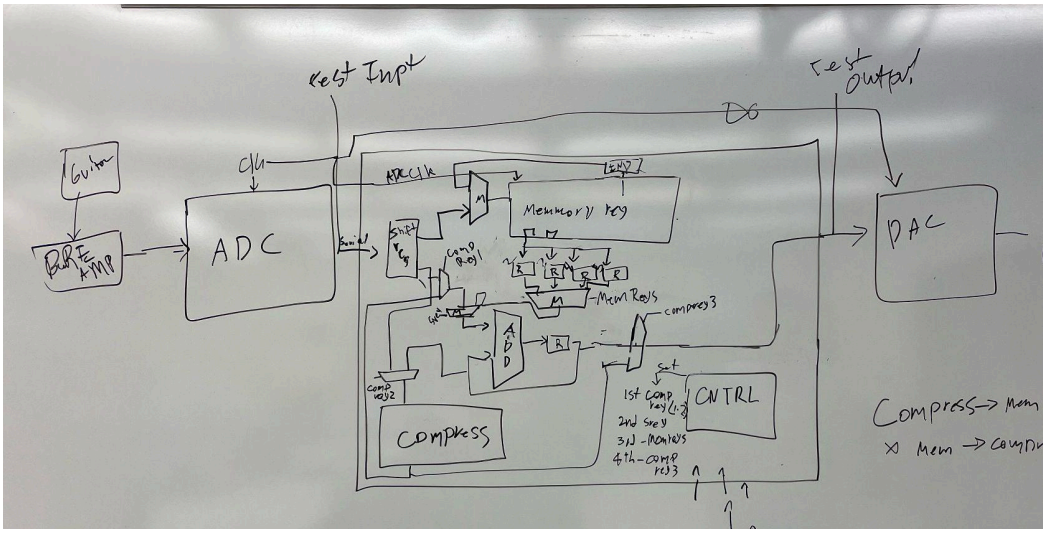

Figure: Our (poorly drawn) initial top-level design

This task is necessary for us to decompose our project into blocks. Further revisions were made to this top-level design as we progressed further in the project.

iv. Meticulously design each module in Verilog:

a. Each module will be given a set of inputs, outputs and requirements to be met and implemented
b. Each module will be uniquely designed
c. Each module will be verified for proper functionality
d. This task is necessary so that we can assemble all of our modules
e. Off-chip converters will be selected based on design considerations

v. Assemble all modules together:

a. During this process, we will verify the modules as we put them together
b. We need to have a working project that is testable

vi. Verify the final design:

a. Our final design will be verified for the design requirements
b. We need to have a verified design before it can be sent to EFabless

vii. Generate Layout/DRC and LVS Checks:

a. This is necessary to make sure our design follows the checks of the Skywater 130nm process

viii. Submit to eFabless shuttle:

a. We need to have our design verified by Efabless so that they can fabricate our chip

3.2 Project Management/Tracking Procedures

We will be utilizing the waterfall-style project management. Due to the nature of our project, as we complete modules of our project it would be challenging to go back and change design choices in the previous steps, as most steps of the project are dependent on previous steps. In addition, we will be using Git to keep track of our progress with issues and milestones.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Task	Milestones
i. Define the scope & specifications of our project: a. Brainstorm a specific application for our project b. Compile a list of functions that may be relevant to our project c. Need a project that can be finished within the time frame necessary and also challenging enough	Having the project approved by our client/faculty advisor
ii. Install and familiarize ourselves with design tools/workspaces: a. Install digital tools that we will use to design our project b. Familiarize ourselves with how to do basic design in the tool and how it can be integrated with EFabless c. This task is necessary so we can design our chip with these tools	Each member successfully familiarized themselves with their respective design tools/workspaces
iii. Design a top-level diagram for the user area: a. This task is necessary for us to decompose our project into blocks	A top-level diagram with specified inputs and outputs for the full design.
iv. Meticulously design each module in Verilog: f. Each module will be given a set of inputs, outputs, and requirements to be implemented	Each module will be verified for proper functionality

g. Each module will be designed h. Each module will be verified for proper functionality i. This task is necessary so that we can assemble all of our modules j. Off-chip converters will be selected based on design considerations	
v. Assemble all modules together: a. During this process, we will verify the modules as we put them together b. We need to have a working project that is testable	Compile all project files and make sure there are no errors
vi. Verify the final design: a. Our final design will be verified for the design requirements b. We need to have a verified design before it can be sent to EFabless	Create test benches to check outputs and complete Verilog synthesis
vii. Generate Layout/DRC and LVS Checks: a. This is necessary to make sure our design follows the checks of the Skywater 130nm process	DRC and LVS Pass
viii. Submit to eFabless shuttle a. We need to have our design verified by eFabless so that they can fabricate our chip	Create layout using synthesis tools.

3.4 Project Timeline/Schedule

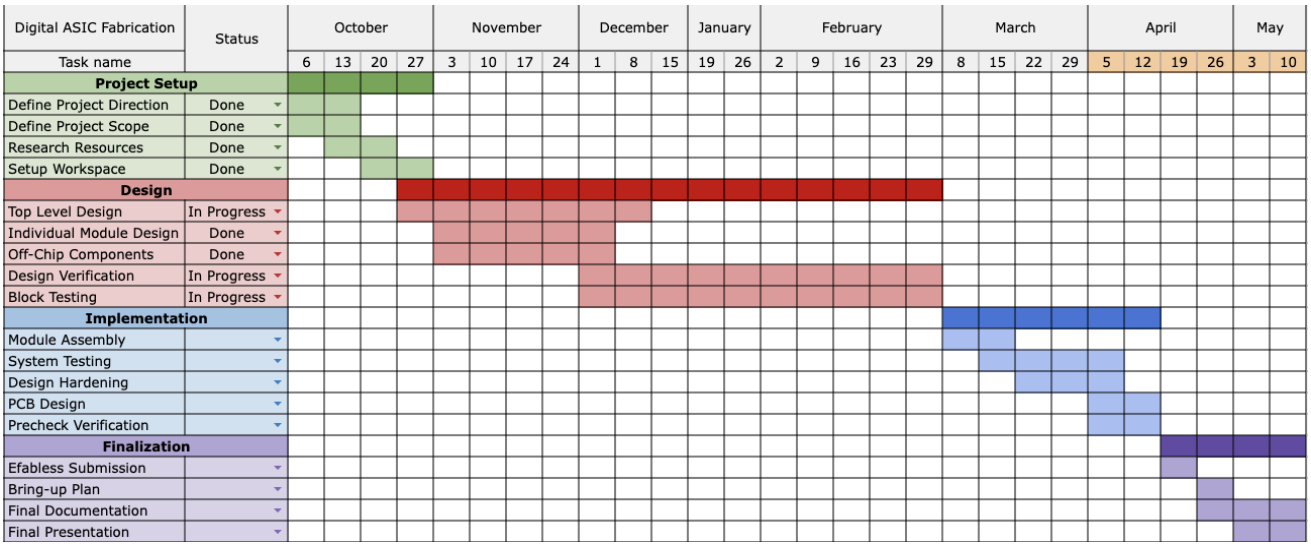

Figure: Gantt chart of our final projected project timeline

All updates & progress that we have made can be viewed on our GitHub page:

https://github.com/sdmay24-21/ASIC-GuitarPedal

3.5 Risks and Risk Management/Mitigation

Task	Risk
i. Define the scope & specifications of our project	0.1
ii. Install and familiarize ourselves with design tools/workspaces	0.1
iii. Design a top-level diagram for the user area	0.1

iv. Meticulously design each module in Verilog	0.5 -needs to get done put in as much time as possible -Split tasks between team members
v. Assemble all modules together	0.4
vi. Verify the final design	0.4
vii. Generate Layout/DRC and LVS Checks	0.4
viii. Submit to eFabless/generate layouts	0.4

3.6 Personnel Effort Requirements

Task	Hours and Explanation
i. Define the scope & specifications of our project	20 hours - creating ideas that will be approved and will work in practice.
ii. Install and familiarize ourselves with design tools/workspaces	20 hours-It takes about 10 hours to get the software installed and another 10 hours to become familiar.
iii. Design a top-level diagram for the user area	20 hours - taking our original idea from step one and creating modules of the project and how they interconnect with each other.

iv. Meticulously design each module in Verilog	112 hours - We have 7 Verilog modules. This will give us about 16 hours to design each module.
v. Assemble all modules together	32 hours - Assembling the modules will take a significant amount of time to verify that all modules work and to go back and fix any bugs if necessary.
vi. Verify the final design	24 hours - Verifying the final design will involve putting in a wide range of inputs and making sure the outputs are correct for a given set of inputs
vii. Generate Layout/DRC and LVS Checks	32 hours - This may take a significant amount of time due to errors that are generated from LVS and checks. We will also need to familiarize ourselves with the synthesis process.
viii. Submit to eFabless shuttle	24 hours - We will need to become very familiar with the EFabless requirements in order to put our design into EFabless and know the chip will be fabricated properly.

3.7 OTHER RESOURCE REQUIREMENTS

i. Our design requires the following off-chip components:

- ADC - audio input
- DAC - audio output
- SRAM - off-chip memory
- Miscellaneous parts necessary for a working PCB

ii. We will be sending in our design to eFabless to fabricate our chip.

iii. All other design resources are either provided by eFabless or open-sourced.

4. Design

4.1 Design Content

For our project, we are designing a Digital ASIC (Application Specific Integrated Circuit). Our chip will modify a guitar's analog input signal using one of many different digital effects, and produce an altered analog output signal. The ASIC design will be done entirely in the digital domain using Verilog HDL, but we will also implement analog off-chip components like ADCs, DACs, and SRAM.

4.2 Design Complexity

i. The design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, or engineering principles:

Reverb - the design uses a digital finite impulse response to create effects like reverb and looping

Function block - the design allows the creation of nonlinear functions such as compression, which uses piecewise functions on individual points in a nonlinear way to create compression. There are potentially a limitless number of functions that can be created via this function block.

ii. The problem scope contains multiple challenging requirements that match or exceed current solutions or industry standards.

The solution uses an application-specific integrated chip instead of an FPGA for audio effects yet stays configurable and user-editable.

Our chip also has a delay and reverb setting that can repeat the signal up to 5 seconds later. This is much longer than the typical ¼ - ½ second setting that is present in most reverb and delay systems.

4.3 Modern Engineering Tools

Verilator - to write Verilog HDL code, which is used to design the bulk of the ASIC.

KiCad - To design a PCB that can accommodate both the ASIC and the off-chip analog components.

Git - To ensure all members have access to the same, most updated version of the design.

4.4 Design Context

We are designing a guitar pedal that is able to produce multiple different audio effects. As such, our primary audience would be guitarists, artists, and audio engineers. Ergo, The overall music community may be affected.

The objective of our project is to address the electronic waste, and, to a lesser degree, time waste that is generated by single-effect guitar pedals. Our scalable, multi-effect guitar pedal will eliminate the electronic waste generated by unused/obsolete guitar pedals, and conserve time. Ultimately, our design principle is based on technology that helps both people and the environment.

Below is a list of relevant considerations related to our project in each of the following areas:

Area	**Description**
Public health, safety, and welfare	Although indirectly, our project is based on the idea of generating less waste, and improving public health, safety, and welfare via reducing electronic waste in our society.
Global, cultural, and social	It is aimed at musicians/artists, specifically guitar players, and audio engineers who want to be able to improve the music they make without compromising the environment.

Environmental	Our pedals will use materials that are already very common like silicon, plastic, and metal. There are no additional detrimental effects on the environment. In addition, the purpose of our multi-purpose guitar pedal is to reduce electronic waste by reducing the number of guitar pedals needed.
Economic	We may see a decrease in the sales of FPGA and regular guitar pedals if our ASIC pedals become popular. We may also see fewer pedals being purchased in general because our pedal can do more than one function.

4.5 Prior Work/Solutions

Traditionally, electric keyboard pedals have been non-programmable, and they provide only a single effect. In contrast to an electric keyboard, where a user could have multiple audio effects at once, electric guitars could only have one audio effect at a given time. Our design will eliminate the need for multiple pedals, and the possibilities of the numbers and types of effects that can be implemented are plenty. FPGA guitar pedals, which are similar to our ASIC guitar pedals, have been done in the past. An example would be a project from CalPoly which had multiple basic effects that consisted of an input signal and an impulse response (Robles, 23).

The advantages of creating a project using an FPGA instead of an ASIC would be that the audio effects provided could always be tweaked and improved, and more audio effects could also be added in the future. However, the cost of an FPGA is significantly greater than the cost of an ASIC. In addition, the FPGA guitar pedal only has linear time-invariant (LTI) effects, whereas our ASIC guitar pedal has a function block that allows for non-linear effects, such as compression.

4.6 Design Decisions

i. One of the key decisions that we will have to make is choosing the appropriate off-chip components that are electronically compatible and ideal. The off-chip components have to adhere to the Caravel Maximum ratings:

Type	minimum	typical	maximum	units
Supply voltage (VDDIO)	1.8	3.3	5.0	V
Core digital supply voltage (VCCD)	1.62	1.8	1.98	V
Junction temperature	-40	27	100	$^{\circ}C$
V_{OH}	$0.8 \cdot VDDIO$		V	
V_{OL}			0.4	V
Management area power		TBD		mW
Storage area power		TBD		mW
GPIO voltage	0		VDDIO	V
GPIO frequency	0		50	MHz
Management clock	0		40	MHz

Figure: Table of Caravel Maximum Ratings

ii. Another key decision is the structure and addressing of the memory. In order to save on write speeds and read speeds an efficient data structure is needed. We have decided to use off-chip memory to allow for more, and longer effects. In addition, off-chip memory also frees up design space on the chip (user area), allowing us more design freedom.

iii. Another key decision was choosing the specific guitar pedal effects that we wanted to implement. We have chosen three guitar pedal effects for the following reasons:

- Multiple effects to show off the capabilities of our ASIC
- Our design is scalable, so it is unnecessary to create more effects
- Our primary focus is to create a scalable design, not showing off signal processing skills
- The effects we have chosen (reverb, compression, delay) are three of the most common guitar pedal effects

4.7 Proposed Design

At the start of Senior Design 1 (first semester), we created a top-level design that breaks down the design into multiple modules that can be worked on by different members. We also planned to implement off-chip components such as an ADC (to convert analog input to digital signals) as well as a DAC (to convert digital signals to analog output). In addition, we might include an off-chip memory as well to allow for more design room on-chip. The design choices can be seen in section 4.7.1 & 4.7.2.

At the start of Senior Design 2 (second semester), we finalized our design as seen in section 4.7.3, and decided on the off-chip components that we needed (ADC, DAC, SRAM).

4.7.1 Design 0 (Initial Design)

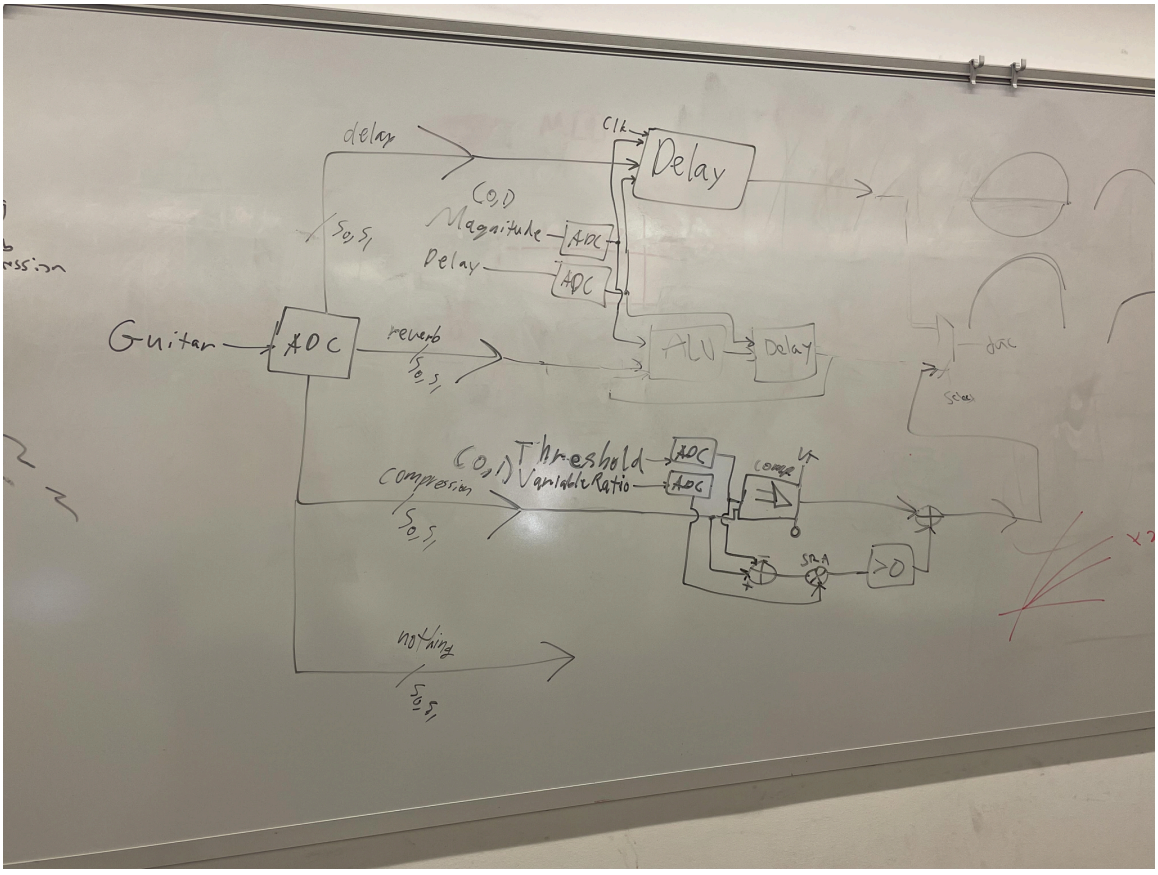

Figure: Initial idea of the project

Initial design notes:

The ASIC works by taking in input from the ADC and then going to the functions depending on the control bits. Each section simply does what we require to make that affect work, and then it gets output to the DAC.

Functionality:

It is intended for the user to be able to play their guitar and select one function at a time that they want to have enabled. The user will start their recording by pressing the pedal down and then finish when they release it. The function parameters can be changed through the use of knobs, each changing the strength of each function such as the time and the tone, compression, and loop with the level. The manipulated audio is then output.

4.7.2 Design 1

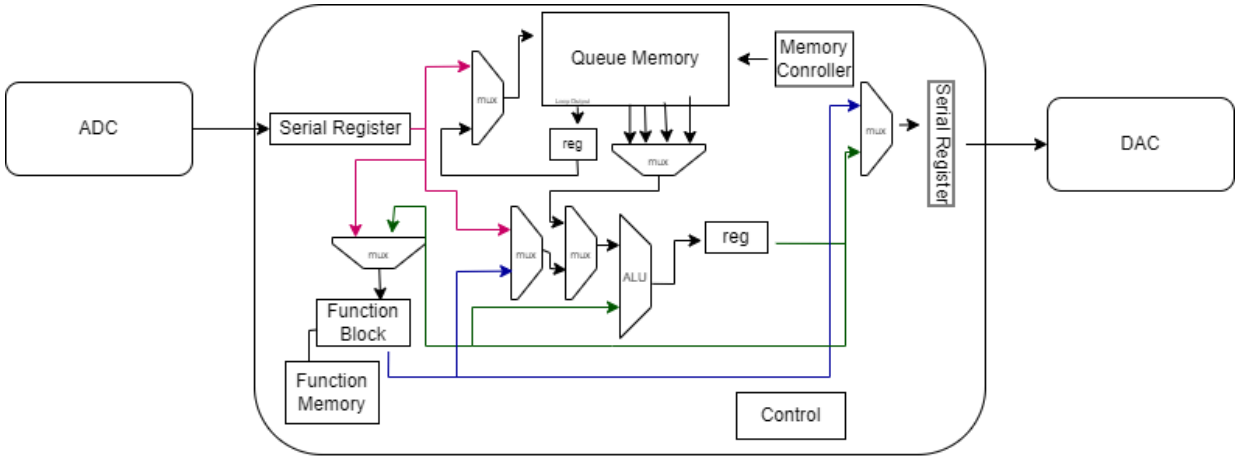

Figure: Initial top-level design

Design notes:

The input from the ADC is the same however from there it either goes to memory or compression depending on the state of the control bits.

We added the queue memory to store the last output of the serial register for either looping that value, such as for a chord progression, or for the reverb effect.

The memory controller is overall the same as it determines which way the memory is used, whether it is looping or reverb as well as holding the tail and head of the current memory.

The function block applies any nonlinear effect that is needed such as a compression effect by using comparators and multipliers.

4.7.3 Design 2

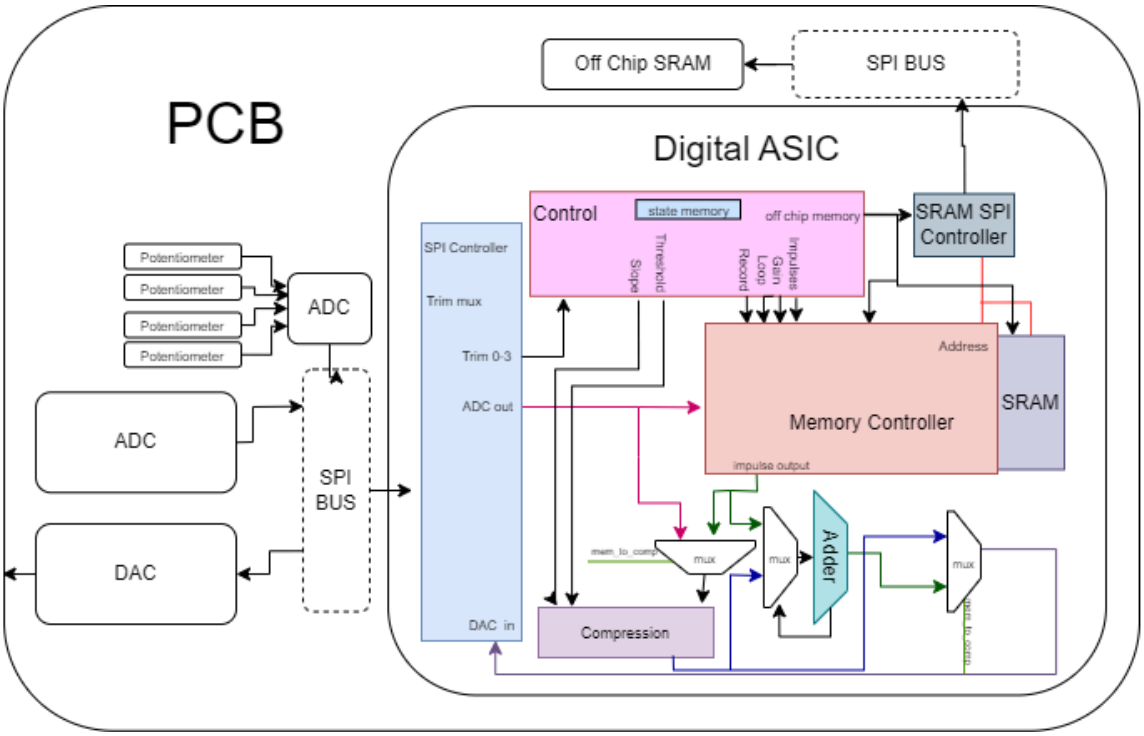

Figure: Top-level design of the final design

Design notes:

Compared to design 1, there are more details on the main protocols and top-level signals. The addition of impulses is now the responsibility of the memory controller rather than the top-level design control. The top-level design is also more detailed with the inclusion of off-chip components and SPI buses. In addition, we have decided on the off-chip components that would be placed on our PCB design. Each of these components adheres to our design constraints and the Caravel Absolute Ratings.

List of off-chip components:

i. ADC: **MCP3464**

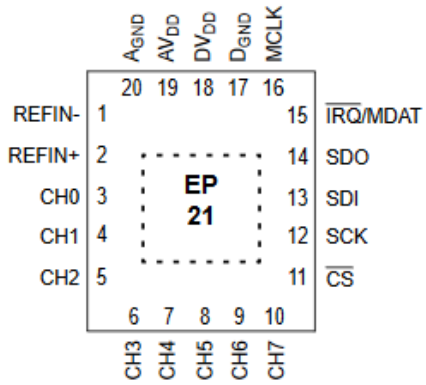

Figure: Schematic of MCP3464

Important Specifications:

- 16-bit Resolution
- 153.6 kHz sampling rate
- Low noise
- 2.7 to 3.6 analog Vdd
- 1.8 to 3.6 digital Vdd
- 8 channels
- SPI compatible

ii. DAC: **AD5676BRUZ**

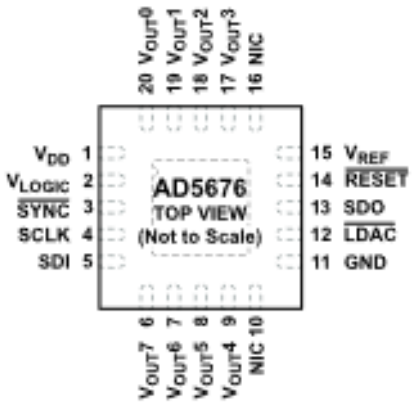

Figure: Schematic of AD5676BRUZ

Important Specifications:

- 16-bit Resolution
- +/- LSB maximum at 16-bits
- Low power
- 2.7 to 5.5 Vdd
- 8 channels
- SPI compatible (50 MHz)

iii. SRAM: **23K256T**

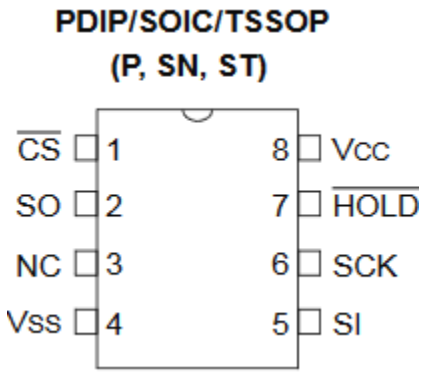

Figure: Schematic of 23K256T

Important Specifications:

- 20 MHz clock rate
- Low power
- 32,768 x 8-bit organization
- 2.7 to 3.6 Vdd
- 256 kBits (4 needed)
- SPI compatible

iv. Audio jack: **SJ_63062A**

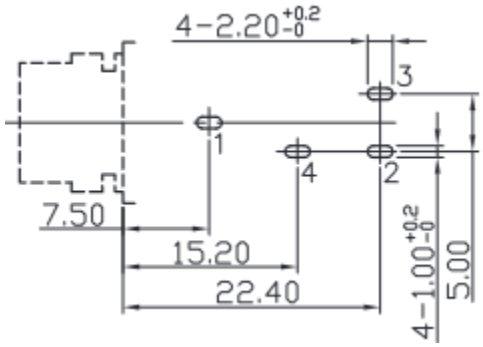

Figure: PCB layout view

Important Specifications:

- 6.355 mm

v. Knobs: **3360Y1103LF**

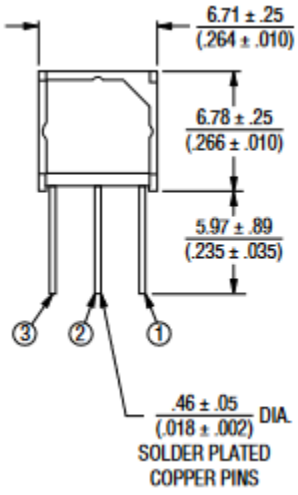

Figure: PCB layout view

Important Specifications:

- Resistance: 1kΩ - 1MΩ
- Resistance tolerance: +/- 2%

vi. Pre-amp: **LMV1032UP06NOPB**

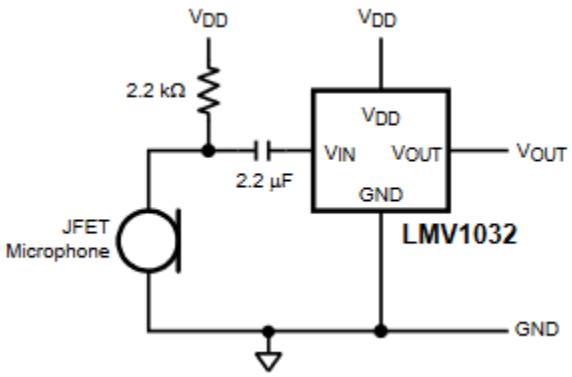

Figure: Schematic view

Important Specifications:

- 1.7 to 5.0 Vdd
- 0.11% THD + noise
- Single channel

The preamp is attached as an intermediary between the input audio jack and the ADC to reduce noise and increase signal gain.

vii. PCB (hat-board):

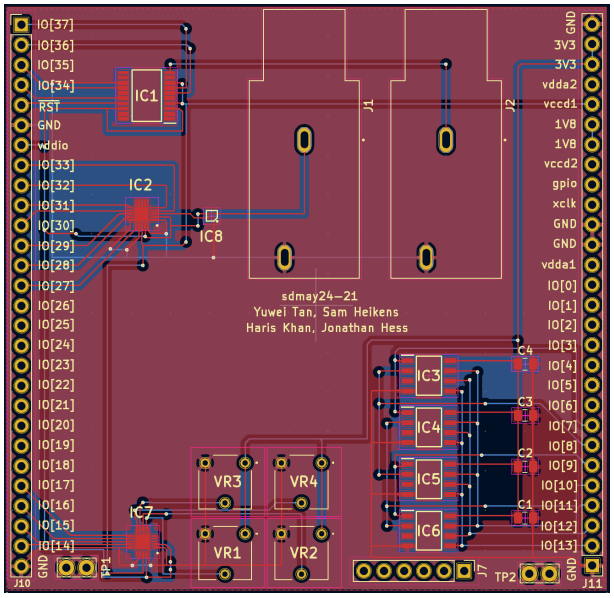

Figure: Layout view of the PCB

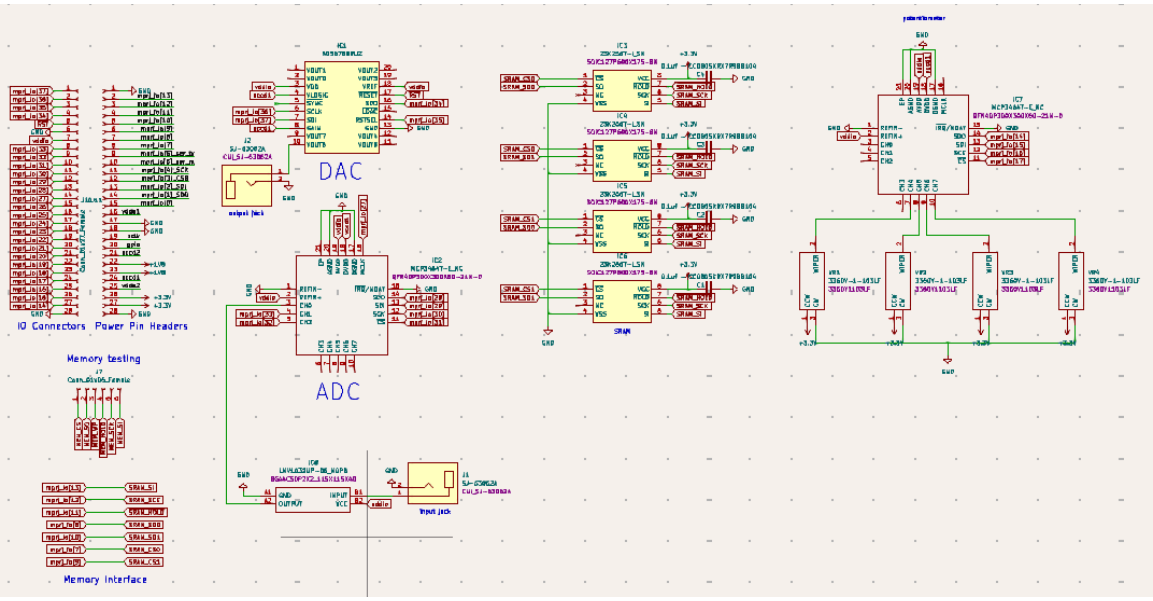

Figure: Schematic view of the PCB

Legend:

- IC1: DAC
- 1C2, IC7: ADC
- VR1-4: Knobs
- IC3-6: SRAM
- J1,7: Audio Jack
- IC8: Pre-amp

Additional notes:

The PCB design has been verified via DRC and LVS checks.

4.8 Module Design

i. Control Module:

https://github.com/sdmay24-21/ASIC-GuitarPedal/wiki/Modules_Control_Module

The control module will be using the MCU provided in the Caravel Wrapper. This uses the PicoRV32 (https://github.com/YosysHQ/picorv32). This is programmed using the gcc compiler, more information on this programming procedure can be found in the caravel documentation, https://caravel-harness.readthedocs.io/en/latest/programming.html. This allows to debug and edit the system values directly. Here are cases where this customizability would be helpful:

- Converting to input values.

  If an end user wants to use a linear or logarithmic potentiometer but get the opposite effect, code can be used to convert the logarithmic values to linear or vice versa.

- Adding extra effects.

  If an end user wanted to add extra effects, they could do so with code rather than in hardware. Because we are using an impulse response design, many other effects would be possible with changes to the code. (such as looping and filters)

ii. SPI-Module

Connects the off-chip Sram to the on-chip memory. This makes it possible to have more space to store our converted audio signal. It is a synchronous device which means that the devices must be on the same clock as the module. Integration testing is most important to see how this will work in our system. A visual representation is below with the slave devices being our memory

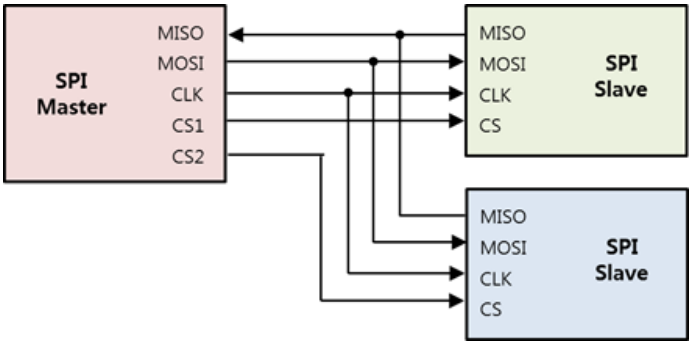

Figure: Diagram of SPI protocol

iii. Compression Module

https://github.com/sdmay24-21/ASIC-GuitarPedal/wiki/Modules_Compression

The purpose of this module is to provide the compression sound effect to the rest of the systems on the chip. Compression is a nonlinear function that will take audio inputs of 16 bits and compress them to a threshold range. Anything input that is beyond this range will be multiplied by a fraction to keep the signal close to the threshold value. A visual representing the usage of the Threshold and Ratio Values is below.

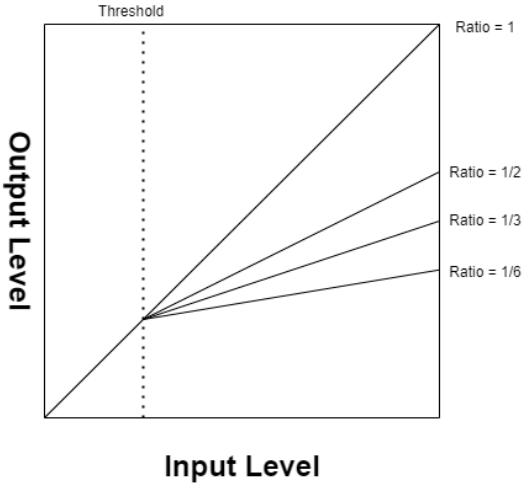

Figure: Visual aid for the compression module

When the input is below the threshold, the output equals the threshold, and beyond the threshold, The output is compressed depending on the value of the ratio.

For our design, a piecewise model will be used to implement this nonlinear function. Depending on the region, a set of MUXes will be used to select which equation will be used. A block diagram of our compression module is shown below.

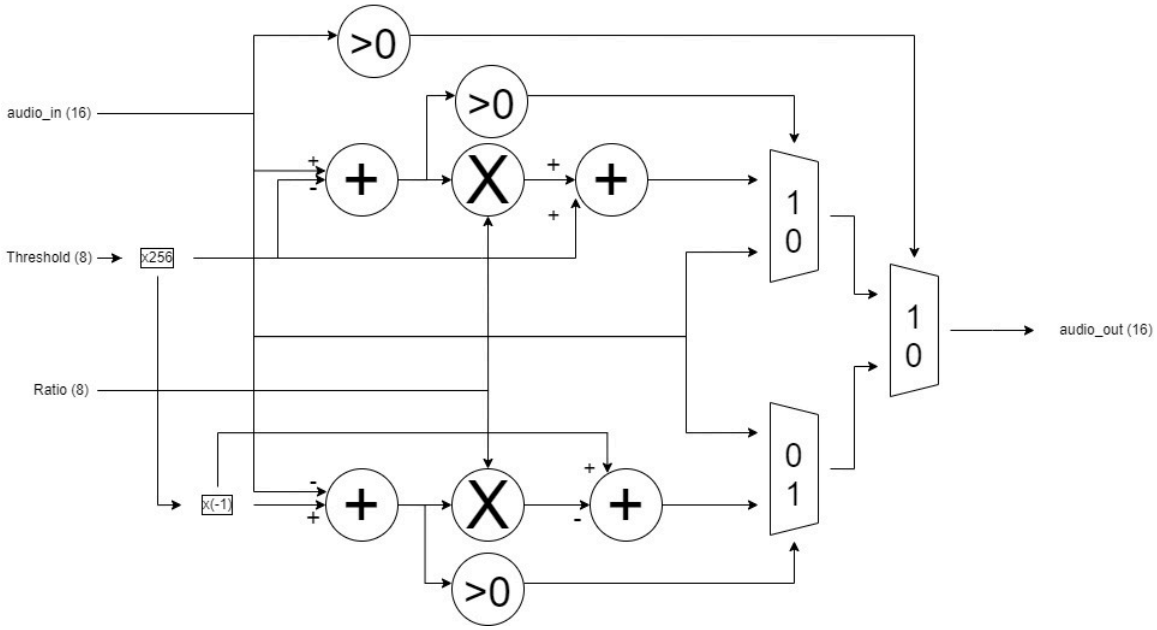

Figure: Block diagram of the compression module

When the magnitude of our audio input is less than the magnitude of the threshold, the audio is sent directly to the output. Otherwise, the compression function is done as shown above.

iv. Memory Controller Module

https://github.com/sdmay24-21/ASIC-GuitarPedal/wiki/Modules_Memory_Controller

The memory controller controls both memory on-chip and off-chip (with the use of an SPI controller). It sets and gets the values from the memory related to impulse response.

The value is grabbed from memory and the impulse and multiplied by Gain (an 8-bit value) added to the buffer and held on the data_out data line. Then it raises a flag for the control to ready the next impulse point and gain. After all the impulses the values will still be held at data_out. Then at the start of the ADC_CLK the buffer (and data_out) is erased for the next ADC value.

The impulse values are also 16-bit but are broken up in the following way:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LJ	JUMP VALUE					MULTIPLIER									

Where LJ is LARGE JUMP

WHEN LARGE JUMP =0

The Jump Value indicates the value to be subtracted from the current read address. The default Jump Value is 1, which means a value of 0 translates to **addr = addr+1**.

curr_r_adr <= curr_r_adr - ({10'b0, jump_value}+1);

WHEN LARGE JUMP =1

If the Large jump is set then the Jump value is multiplied by (2^6) (or is shifted to the left 6 bits). It doesn't have a base offset so a Jump Value of 0 yields a 0 jump.

curr_r_adr <= curr_r_adr - jump_value*(2^6);

This was designed to allow large jumps for very long delays or long reverb.

The impulses are stored in the beginning section of the memory. This can be configured by changing the “impulses” input on the memory controller.

v. On-Chip SRAM Module

Our SRAM Module is a predesign SRAM from EFabless. We adjusted the address width to make this module usable for our application.

4.9 Hardening

Hardening was attempted using the provided tools in a virtual Linux environment. Many issues were encountered during hardening which resulted in this process being incomplete. While our design was not hardened, we learned more about the ASIC hardening process and helped improve ASIC resources.

If future teams take our project, we will provide all available resources to help them continue fixing hardening errors.

Here is the most recent error:

```
10. Executing Verilog-2005 frontend: /home/sheikens/ASIC-GuitarPedal2/op
enlane/pedal_ASIC/../../verilog/rtl/uprj_netlists.v
/home/sheikens/ASIC-GuitarPedal2/openlane/pedal_ASIC/../../verilog/rtl/p
edal_wrapper.v:32: ERROR: Re-definition of module '\pedal_wrapper'!
child process exited abnormally
```

5 Testing

5.1 Unit Testing

Each individual top level Verilog module will have a corresponding Python script that simulates the output values that the tests should yield. Each module will then be tested via a Verilog testbench, and the testbench results will be compared against the values yielded by the Python script. When the modules are assembled, they will be tested using the logic analyzer on the microcontroller provided by Efabless. The aforementioned tests will be conducted using OpenLane tools.

Memory Controller Testing:

The test data was generated with the Python module. It takes in a test audio signal and an impulse audio signal. Then it outputs a CSV file holding the ADC values and control to the memory controller.

verilog > dv > modules > memorycontroller > debug0.txt

```
1   input file is testdata/test0.csv
2   input impulse file is testdata/test0impulse.csv
3   output file is outputs/test0output.txt
4   error file is outputs/test0error.txt
5   Off chip max memory is dff1 or 57329
6   - Verilated::debug attempted, but compiled without VL_DEBUG, so messages suppressed.
7   - Suggest remake using 'make ... CPPFLAGS=-DVL_DEBUG'
8   Loading impulses
9   Starting test!
10  Write to mem addr 3ff0 : ffffea4c
11  Write to mem addr 3ff0 : ffffea4c
12  [1] expected output =  0000 and data_out = 0000 ADC_IN=ea4c Current Write: 3ff1 Error = 0
13  Write to mem addr 3ff1 : 534
14  Write to mem addr 3ff1 : 534
15  [2] expected output =  ea4c and data_out = ea4c ADC_IN=0534 Current Write: 3ff2 Error = 0
16  Write to mem addr 3ff2 : fffffd0e
17  Write to mem addr 3ff2 : fffffd0e
18  [3] expected output =  fcb9 and data_out = fcb9 ADC_IN=fd0e Current Write: 3ff3 Error = 0
19  Write to mem addr 3ff3 : 219
20  Write to mem addr 3ff3 : 219
21  [4] expected output =  11bd and data_out = 11bd ADC_IN=0219 Current Write: 3ff4 Error = 0
22  Write to mem addr 3ff4 : fffffe6e
23  Write to mem addr 3ff4 : fffffe6e
24  [5] expected output =  f83c and data_out = f83c ADC_IN=fe6e Current Write: 3ff5 Error = 0
```

Figure: Generated Debug file output

It then checks the generated verilog output to the expected output generated by the python module. All of the outputs are saved in an output file and all errors are then saved in an error file. The output file can be used to generate the audio. This was used to both see and hear while debugging.

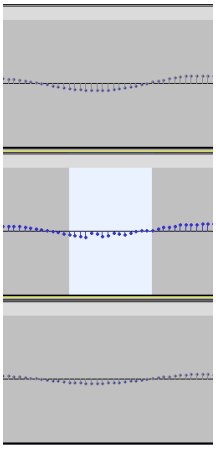

Figure: Example of a caught error in audio form (input, output, expected)

5.2 Interface Testing

The interface being tested is the SPI (Serial Peripheral Interface), where the interface test will verify the speed and consistency of our SPI controllers. The first testing will be done in a Verilog testbench with a simulation of an SPI bus for the SRAM and ADC/DAC SPI controllers. This will verify that the maximum speed for memory operations conforms to our specifications and confirm that the ADC/DAC is compatible. Then, testing will be done on the physical hardware once the part has been fabricated. The tests will be conducted using OpenLane tools.

5.3 Integration Testing

The critical data path will be determined once we synthesize our design. We expect this to be our impulse which has to go through memory, the longest module.

5.4 System Testing

Create test benches for the three levels of Verilog and then one for the physical chip. The first level of the Verilog testing is without control or SPI. In this testbench, the input and output will be set and checked directly. The next level of testing will use a testbench with control but without SPI. Then finally we. Last testing on the PCB and physical part to verify and get the specifications of the fabricated part.

A full-scale test will be conducted on the final integrated design using a large Verilog Testbench, and the outputs will be compared against Python simulation results. Three testbenches, each for one level of Verilog code, and one testbench for the physical chip will be made. The first level of Verilog testing will have the inputs/outputs checked directly. The next level of testing will include the control module, and the third level of Verilog testing will involve testing the entire design using SPI communication via the GPIO pins.

5.5 Regression Testing

We will ensure new modules or changes won't break the system with merges in GitHub as they actively run tests whenever a new merge is done to the main branch. We will have to write a sufficient amount of tests to make sure that the full functionality of the larger modules remains the same.

5.6 Acceptance Testing

We will demonstrate that the design requirements are met by comparing the Python simulation results to match the values of our Verilog testbenches. We will run the synthesized user area against the Efabless precheck that is provided for us to confirm that the layout matches the Verilog design.

6. Professionalism

*This discussion is with respect to the paper titled “Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment”, International Journal of Engineering Education Vol. 28, No. 2, pp. 416–424, 2012

6.1 Areas of Responsibility

We will be using the IEEE code of ethics.

Area of responsibility	Definition	NSPE Canon	IEEE
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	5. to improve the understanding of technology; its appropriate application, and potential consequences; 6. to maintain and improve our technical competence and

			to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations; 7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others; 10. to assist colleagues and co-workers in their professional development and to support them in following this code of ethics.
Financial Responsibility	Deliver products and services of realizable value at reasonable costs	Act for each employer or client as faithful agents or trustees.	
Communication Honesty	Report work truthfully, without deception, and are understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	3. to be honest and realistic in stating claims or estimates based on available data;
Health, Safety, Wellbeing	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public.	1. to accept responsibility in making decisions consistent with the safety, health,

			and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;
Property Ownership	Respect the property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	8. to treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual 35 orientation, gender identity, or gender expression; 9. to avoid injuring others, their property, reputation, or employment by false or malicious action;
Sustainability	Protect the environment and natural resources locally and globally.		1. to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly,	2. to avoid real or perceived conflicts of interest whenever

		ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	possible, and to disclose them to affected parties when they do exist; 4. to reject bribery in all its forms;

Work Competence: Works towards an understanding of necessary material, and does good work based on knowledge. There is also care taken to critique and improve existing work from others and him/herself.

Financial Responsibility: The components we choose will be adequately priced,

Communication Honesty: Communicates with others with integrity with respect to data and ideas.

Health, Safety and Wellbeing: Makes responsible decisions with respect to the general wellbeing of all people.

Property Ownership: Uses what they own responsibly.

Sustainability: Understand how the products we are designing may affect the environment.

Social Responsibility: Avoids conflict of Interest and makes others aware when there is a possible conflict of interest.

Differences Between IEEE and NSPE:

Work Competence: IEEE puts more focus on this area and gives a stronger description of what it means to have work competence.

Financial Responsibility: IEEE does not provide guidance on financial responsibility.

Communication Honesty: IEEE focuses more on general communication as opposed to public statements.

Health, Safety, and Wellbeing: IEEE goes more into disclosing content, instead of just saying to hold them paramount.

Property Ownership: IEEE gives a list of things to not discriminate on as opposed to just acting fairly.

Sustainability: NSPE does not provide guidance on Sustainability.

Social Responsibility: NSPE just says to act lawfully, but IEEE gets into the details of avoiding conflict of interests and rejecting bribery.

6.2 Project Specific Professional Responsibility Areas

Work Competence: This is relevant to our project as it is important for us to learn and ensure we are competent when working on the design of our chip. I think we are doing just ok in this area. We usually understand what we need to do and do it, but we are also behind where we would like to be.

Financial Responsibility: This applies to our project as we will be purchasing data converters and off-chip memory. We are doing well in this area as we have been working to find more affordable components.

Communication Honesty: N/A

Health, Safety and Wellbeing: N/A

Property Ownership: N/A

Sustainability: This is applicable to our project as we should account for waste generated from our product. We have been doing well in this area as our project will reduce E-waste.

Social Responsibility: N/A

6.3 Most Applicable Professional Responsibility Area

Work Competence is the most applicable as it is important for us to learn and understand our areas of strength and experience on a daily basis.

7. Closing Material

7.1 Discussion

Discuss the main results of your project – for a product discuss if the requirements are met, for experiments-oriented project – what are the results of the experiment, if you were validating a hypothesis – did it work?

7.2 Conclusion

Summarize the work you have done so far. Briefly reiterate your goals. Then, re-iterate the best plan of action (or solution) to achieving your goals. What constrained you from achieving these goals (if something did)? What could be done differently in a future design/implementation iteration to achieve these goals?

7.3 References

Robles, Carson. "An FPGA Implementation of Digital Guitar Effects." Digital Commons, California Polytechnic State University, https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1326&context=cpesp. Accessed September 20, 2023.

7.4 Appendices

Any additional information that would be helpful to the evaluation of your design document. If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc,. PCB testing issues etc., Software bugs etc.

7.4.1 Team Contract

Team Members

1) Yu Wei Tan

2) Haris Khan

3) Samuel Heikens

4) Jonathan Hess

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:

Monday: 1pm, Library or Coover, virtual if needed

Thursday: 2pm, Library or Coover, virtual if needed

Friday: 4:30pm, Library or Coover, virtual if needed

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g.,

e-mail, phone, app, face-to-face):

Text Message for immediate communication, git lab and email for content messages.

3. Decision-making policy (e.g., consensus, majority vote):

Discuss and try to win consensus, if this is not possible ¾ vote is required.

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be

shared/archived):

Record Minutes and Share them in GitLab. We may switch who records each meeting.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

Attendance is expected at all team meetings. If a member is not able to attend they should give

advance notice.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

At meetings, we will assign/split assignments to members. At subsequent meetings, we will follow up and hold members accountable.

3. Expected level of communication with other team members:

Provide detailed communication in meetings and outside of meetings. Any time there is an

update that is relevant to the project, this should be shared in detail. If team member(s) is/are

struggling with part of the project they should notify the team if delays are expected or if they

need more help.

4. Expected level of commitment to team decisions and tasks:

Team members should be able to commit about 6 hours per week outside of class, TA meetings,

and faculty meetings.

Leadership

1. Leadership roles for each team member (e.g. team organization, client interaction, individual component design, testing, etc.):

Haris Khan - Digital Design

Jonathan Hess - Scrum Master

Yu Wei Tan - Systems

Samuel Heikens - Scribe

2. Strategies for supporting and guiding the work of all team members:

We will talk regularly at team meetings. After the team meetings we will project work for all team members. Each member will report back their results. Other members can step in to help or give tips to support other members. Team members should also use Slack liberally to find solutions to issues. Slack has been very helpful to past teams, and we would like to use this resource as well.

3. Strategies for recognizing the contributions of all team members:

At each team meeting, members from each part of the project will debrief their contributions.

This will be a time to recognize what they have accomplished.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the

team.

Haris Khan - VHDL, Verilog, Assembly, C

Jonathan Hess - VHDL, Verilog, Python, C/C++, Embedded System, and PCB Design

Yu Wei Tan - ADS, integration simulations/testing, embedded programming

Samuel Heikens - Cadence, Packaging Knowledge, internship at Texas Instruments

2. Strategies for encouraging and supporting contributions and ideas from all team members:

When team members share ideas, we will consider all ideas. We may have brainstorming

sessions where we just give ideas about how to solve problems. We will thank and encourage our

team members for their contributions.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will

a team member inform the team that the team environment is obstructing their

opportunity or ability to contribute?)

If a team member is struggling to work on the project or feels included in the project, they should

notify the team. The team can respond by making proper accommodations.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

Create a justifiable top-level design with the expected inputs and outputs of each module.

Create state diagrams

2. Strategies for planning and assigning individual and teamwork:

We will assign work based on specialization, skills and prior experiences, and workload

balance

3. Strategies for keeping on task:

Weekly update to GitLab Gantt chart using issues to lay out goals and deliverables.

Communication with faculty to judge the status of assignments

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

Our team will discuss it at our next team meeting. The team member will have to wear a dunce

hat during team meetings if the other team members find their behavior violated the team

contract.

2. What will your team do if the infractions continue?

Our team will notify the TA or professor.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the

consequences as stated in this contract.

1) Yu Wei Tan DATE 9/7/23

2) Jonathan Hess DATE 9/7/23

3) Samuel Heikens DATE 9/7/23

4) Haris Khan DATE 9/7/23