Digital ASIC Fabrication

SDMAY24-21

Haris Khan
Jonathan Hess
Samuel Heikens
Yu Wei Tan

Client & Advisor: Dr. Henry Duwe

Project Vision

Problem Statement:

- Digital ASIC fabrication is a valuable skill, but it is not in the undergraduate curriculum
- Traditional guitar pedals are incapable of having multiple effects

Objective:

- Learn EFabless tools & ASIC fabrication flow
- Develop an ASIC for guitar pedals that allow multiple effects
- Develop documentation to pass on knowledge to future teams

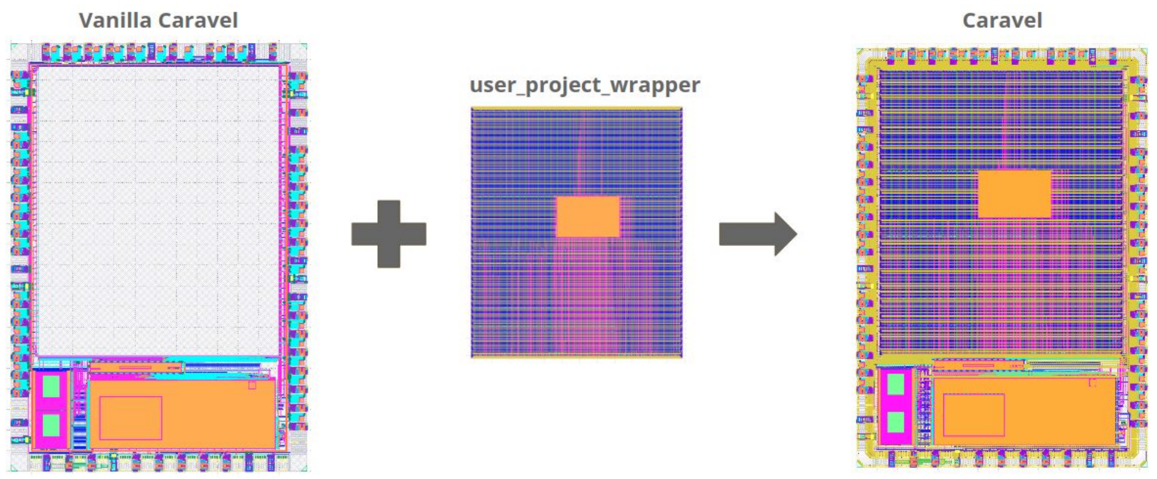
Project Purpose

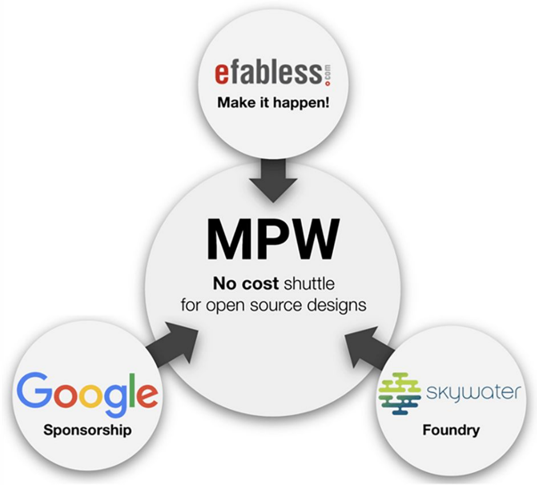
- Reduced need for multiple guitar pedals
- Standardized pedal for Studio Artists
- Provide more knowledge and resources to open source pool
- Create project that can be worked on by future students with CPRE 288 experience without knowledge of ASIC fabrication

Project Scope

- Design and send to EFabless chip design
  - Setup workspace in docker
  - Design modules in verilog
  - Test Individual Modules
  - Compile Verilog Modules
  - Test Compiled Verilog Modules
  - Select Off Chip components like data converters and memory
- Generate Layout
- Send to EFabless

ASIC Fabrication





Who:

- EFabless Open MPW Program

What:

- ASIC Fabrication at zero cost to developers

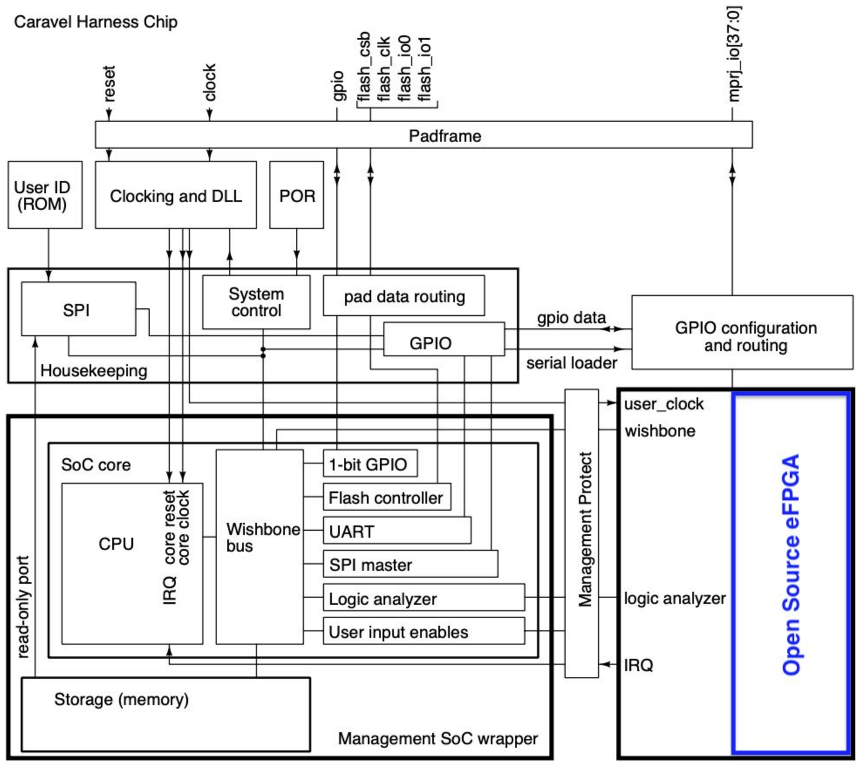
Why:

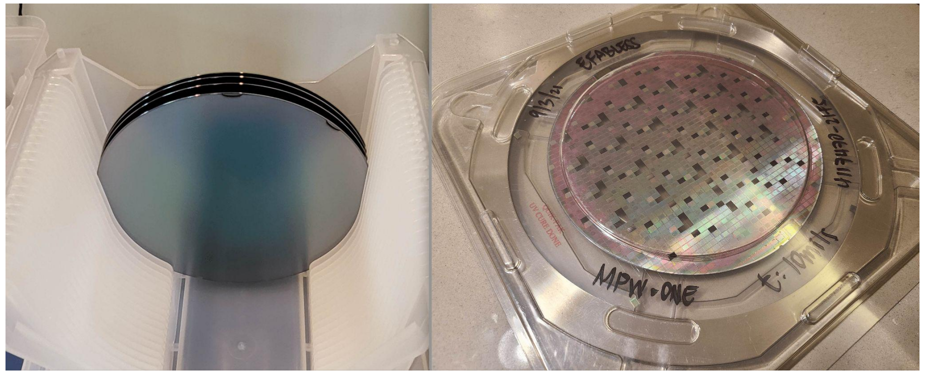
- Open-source platform

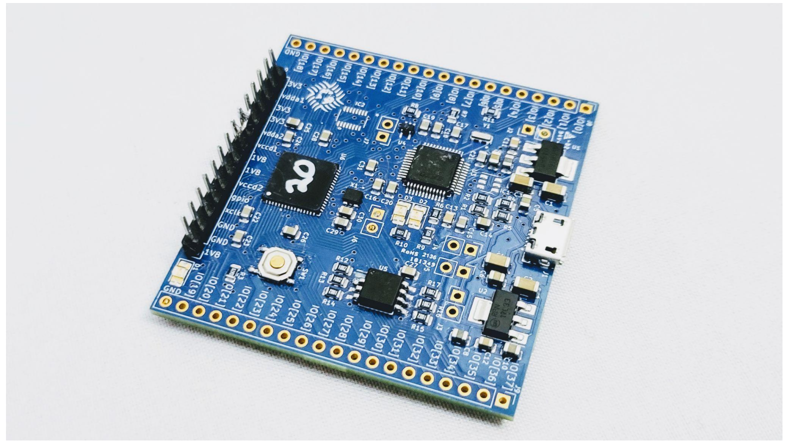
ASIC Fabrication

Caravel Harness:

- Microcontroller (PicoRV32)
- Logic Analyser
- I/O Pads



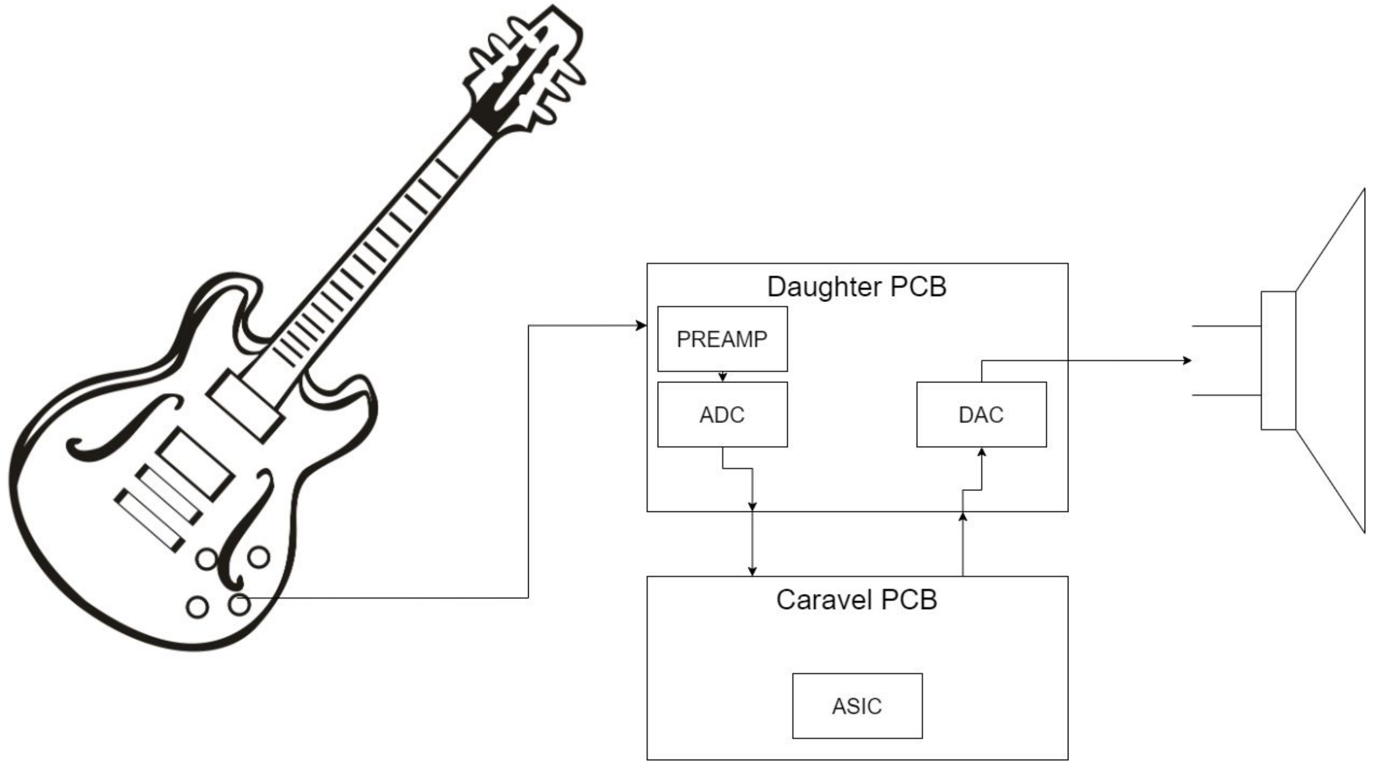




[CLEAR - The Open Source FPGA ASIC](#)

Top Level Basic Design

- Allow for creation of pedal with inputs using the caravel PCB.
- A finalized design could potentially be added to the guitar itself rather than an individual pedal.



Current Pedalboards

- Current pedalboards use multiple effects requires multiple pedals connected in series.
- Our design will reduce multiple of these pedals down to one.

Requirements

- 3 types of effects
  - Delay
  - Reverb
  - Compression
- Delay and reverb will each have two analog inputs
  - Delay
  - Magnitude
- Compression will have two analog inputs
  - Threshold
  - Ratio

- Audio Sample Size
  - 16 bits
- Sampling rate
  - 10 kHz

Engineering Standards

- Audio in/out: 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).
- SPI (Serial Peripheral Interface) for connecting ASIC to ADC, DAC, and SRAM
- IEEE 1364-2005 – IEEE Standard Verilog Hardware Description Language
- ISO/IEC 9899:2018 – C programming language (C17)

Setup workspace

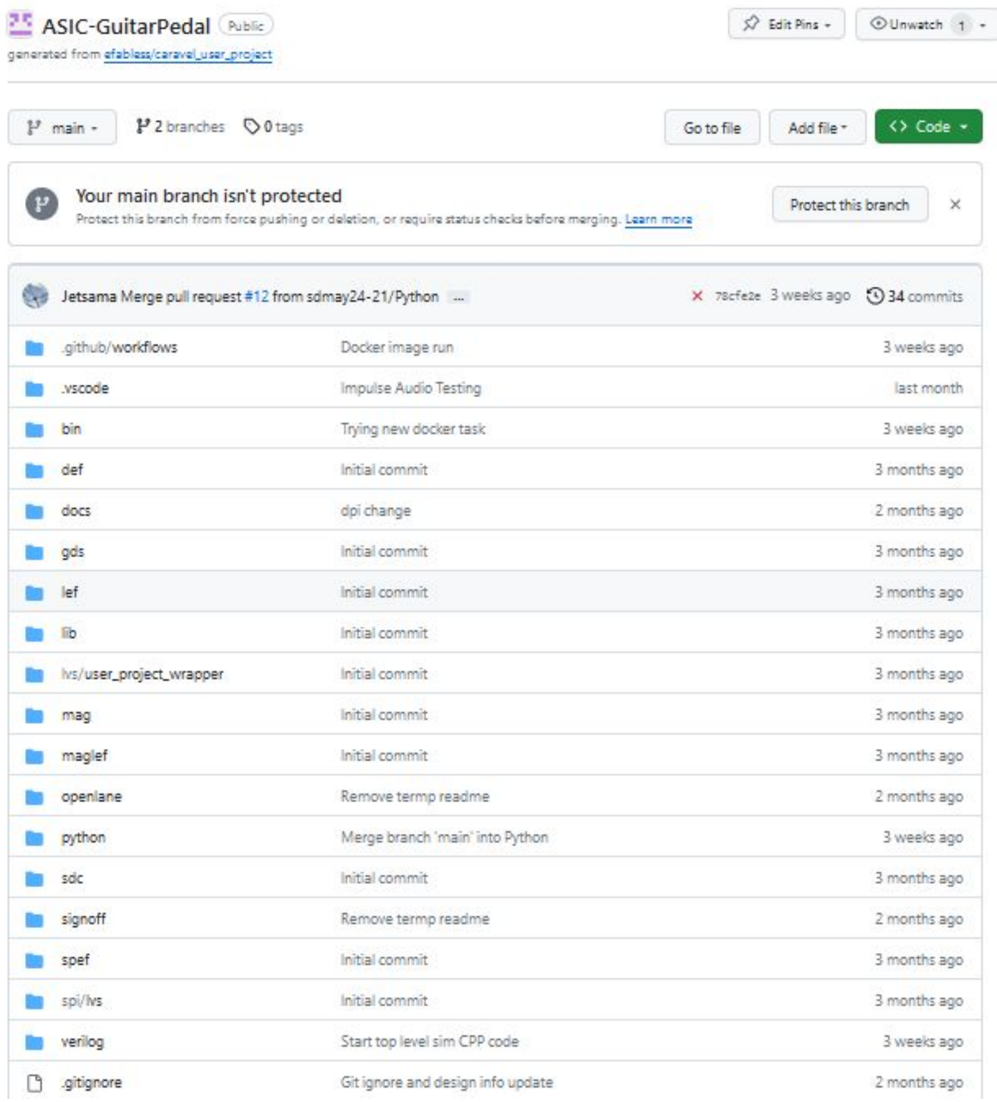
Created GitHub repository with wiki, issues, and milestones

Setup CI/CD for top level and module level testing

[Organization]

[Repository]

[WIKI]


ASIC-GuitarPedal Public

generated from efabless/caravel_user_project

Edit Pins · Unwatch 1

main · 2 branches · 0 tags · Go to file · Add file · Code

Your main branch isn't protected

Protect this branch from force pushing or deletion, or require status checks before merging. Learn more

Protect this branch

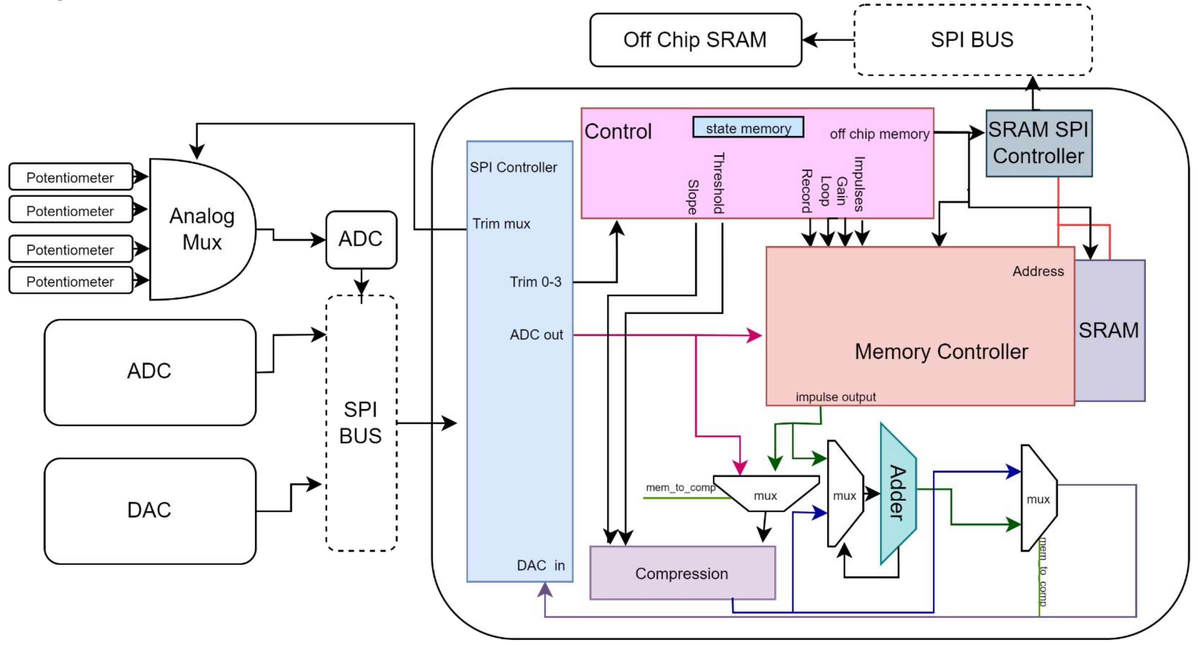
Jetsama Merge pull request #12 from sdmay24-21/Python ... × 78cfe2e 3 weeks ago 34 commits

.github/workflows	Docker image run	3 weeks ago
.vscode	Impulse Audio Testing	last month
bin	Trying new docker task	3 weeks ago
def	Initial commit	3 months ago
docs	dpi change	2 months ago
gds	Initial commit	3 months ago
lef	Initial commit	3 months ago
lib	Initial commit	3 months ago
lvs/user_project_wrapper	Initial commit	3 months ago
mag	Initial commit	3 months ago
maglef	Initial commit	3 months ago
openlane	Remove termp readme	2 months ago
python	Merge branch 'main' into Python	3 weeks ago
sdc	Initial commit	3 months ago
signoff	Remove termp readme	2 months ago
spef	Initial commit	3 months ago
spi/lvs	Initial commit	3 months ago
verilog	Start top level sim CPP code	3 weeks ago
.gitignore	Git ignore and design info update	2 months ago

Conceptual Final Design Diagram

After deciding on bit width, clock speed, memory allocation, and protocols the following design was created.

- Using a Finite Impulse Response implementation of many effects such as delay, reverb, or looping can be implemented
- For non linear effects the compression module has been added.



System Design-Control

We will be using the microcontroller, PicoRV32, as well as logic analyzer provided in the caravel harness

This allows to be programmed and revised via the SRAM

Will be used for Reverb and Delay functions and used to control compression module usage

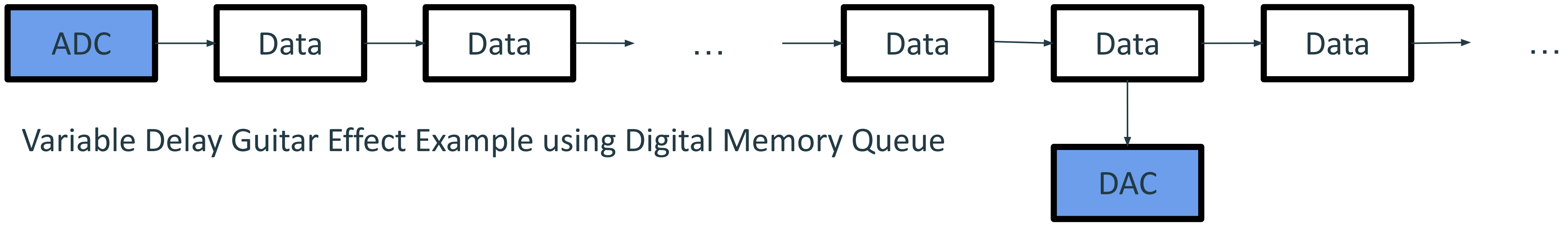
More information can be found in the wiki:
https://github.com/sdmay24-21/ASIC-GuitarPedal/wiki/Modules_Control_Module

System Design - Memory

The memory is organized in a queue format with 16 bit values.

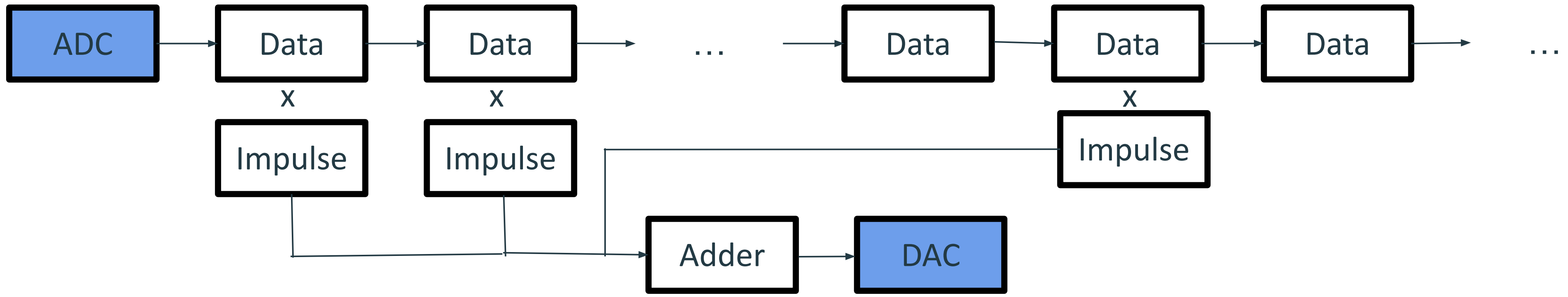
Every ADC Clock Cycle the data is added to the queue and the address is increased.

Depending on the mode, when memory runs out it will overwrite old data.

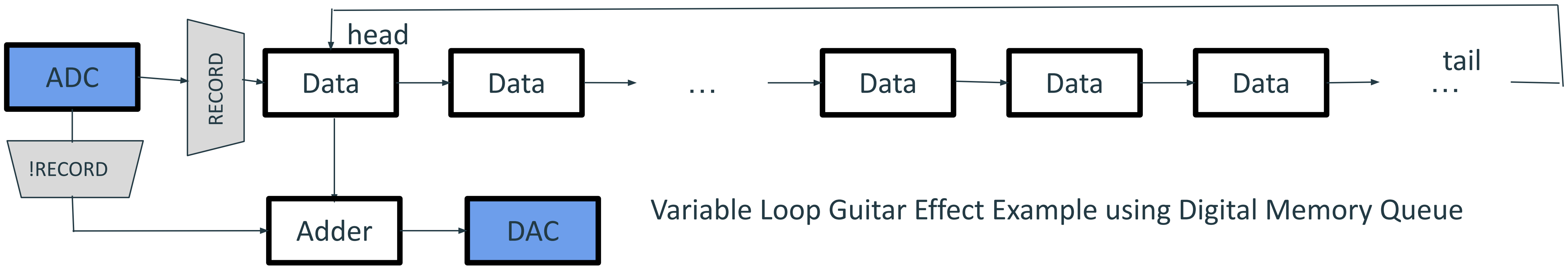


Variable Delay Guitar Effect Example using Digital Memory Queue

Other Pedal Effects using Queue Memory



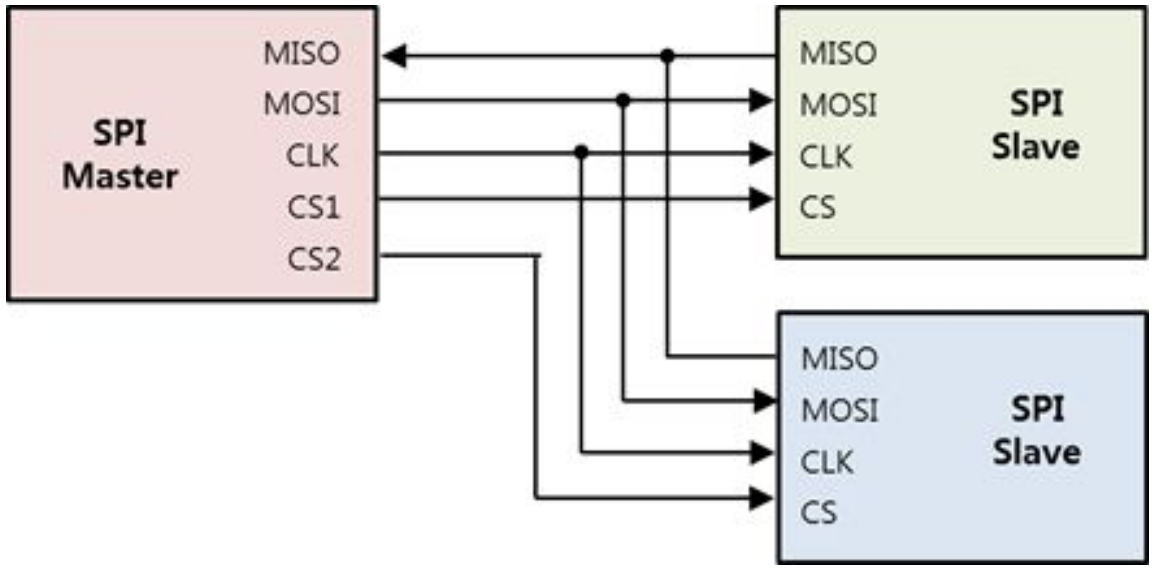
Variable Impulse Response (Reverb) Guitar Effect Example using Digital Memory Queue



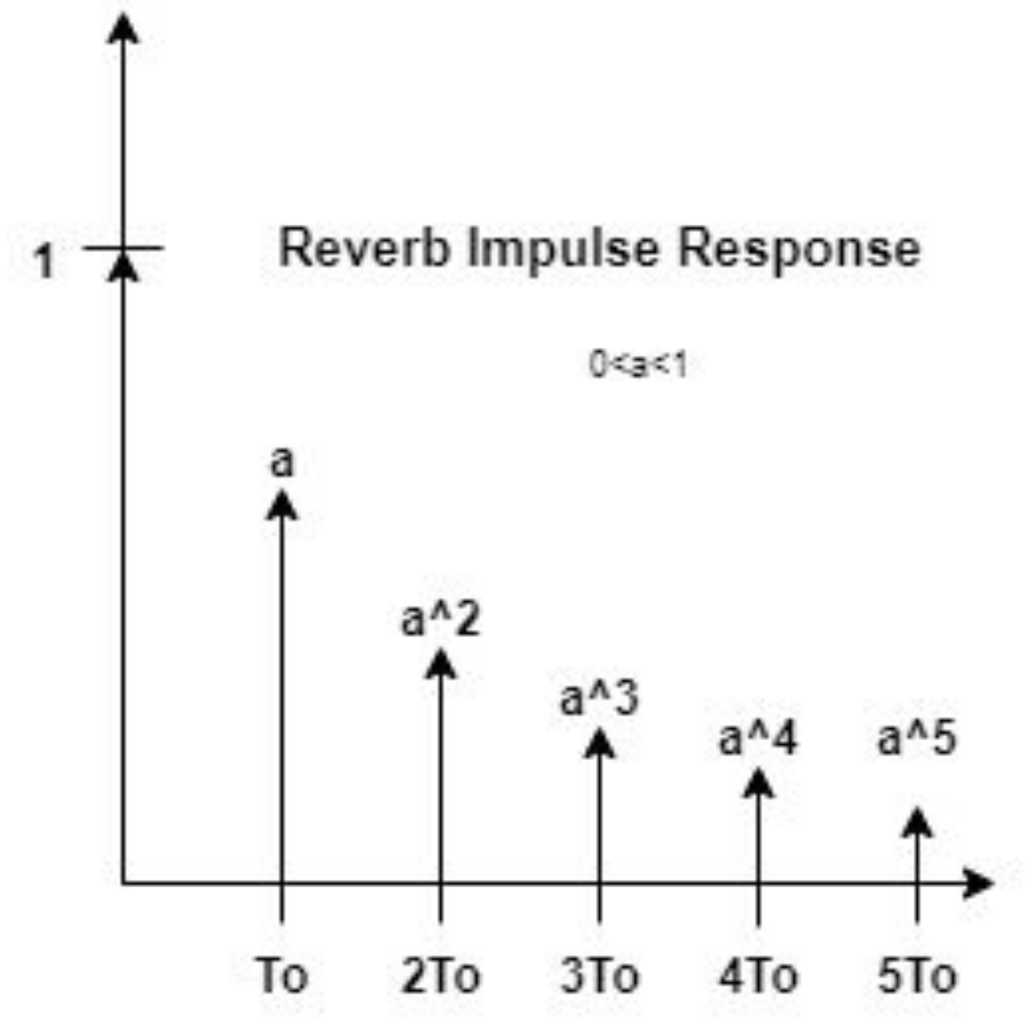
Variable Loop Guitar Effect Example using Digital Memory Queue

System Design-SPI

- Used to interface between memory on and off chip
- it allows for simultaneous data transmission and reception
- It is a synchronous device so the off chip components that it communicates to need to be on the same clock



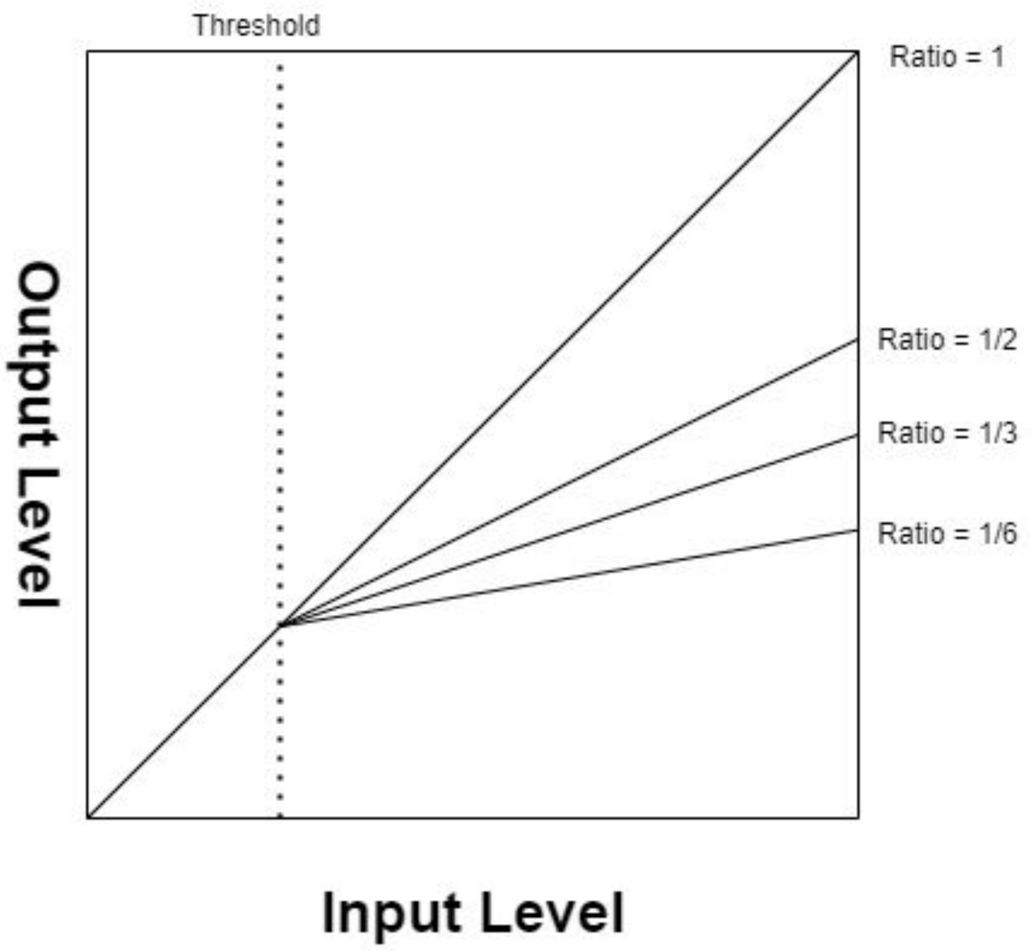
What is Reverb?



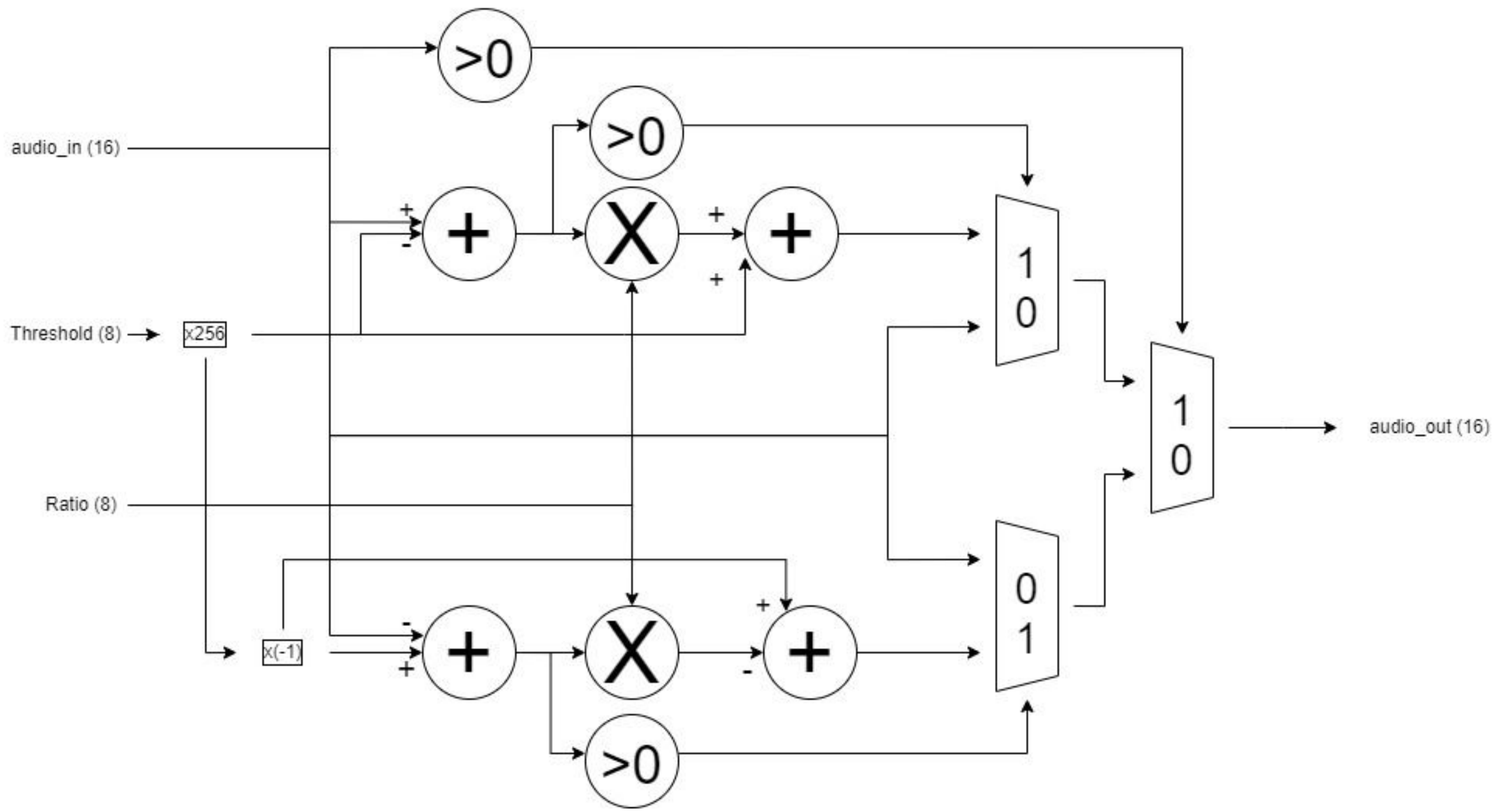
Equally Spaced Delays with a scaling factor for each delay

System Design-Compression

- What is Compression?
- Inputs
  - audio(16)
  - threshold(8)
  - ratio(8)
- When input is less than threshold, output = input
- Otherwise, a portion of the input is added beyond the threshold

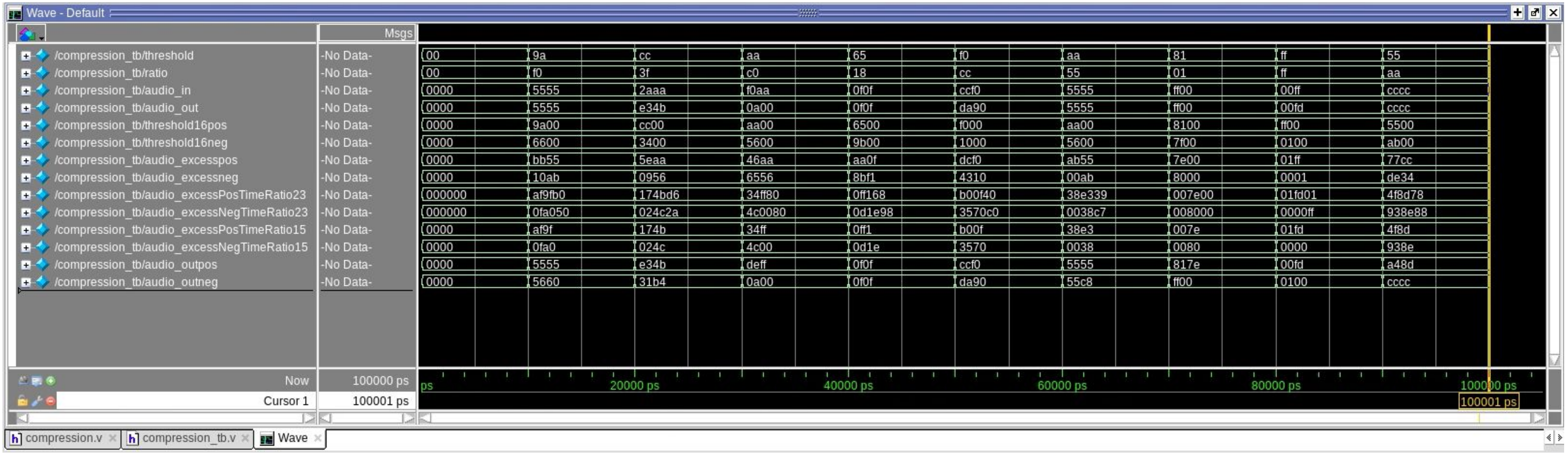


System Design-Compression



System Design-Compression

- Prototype Results
- The test results match the signal flow diagram

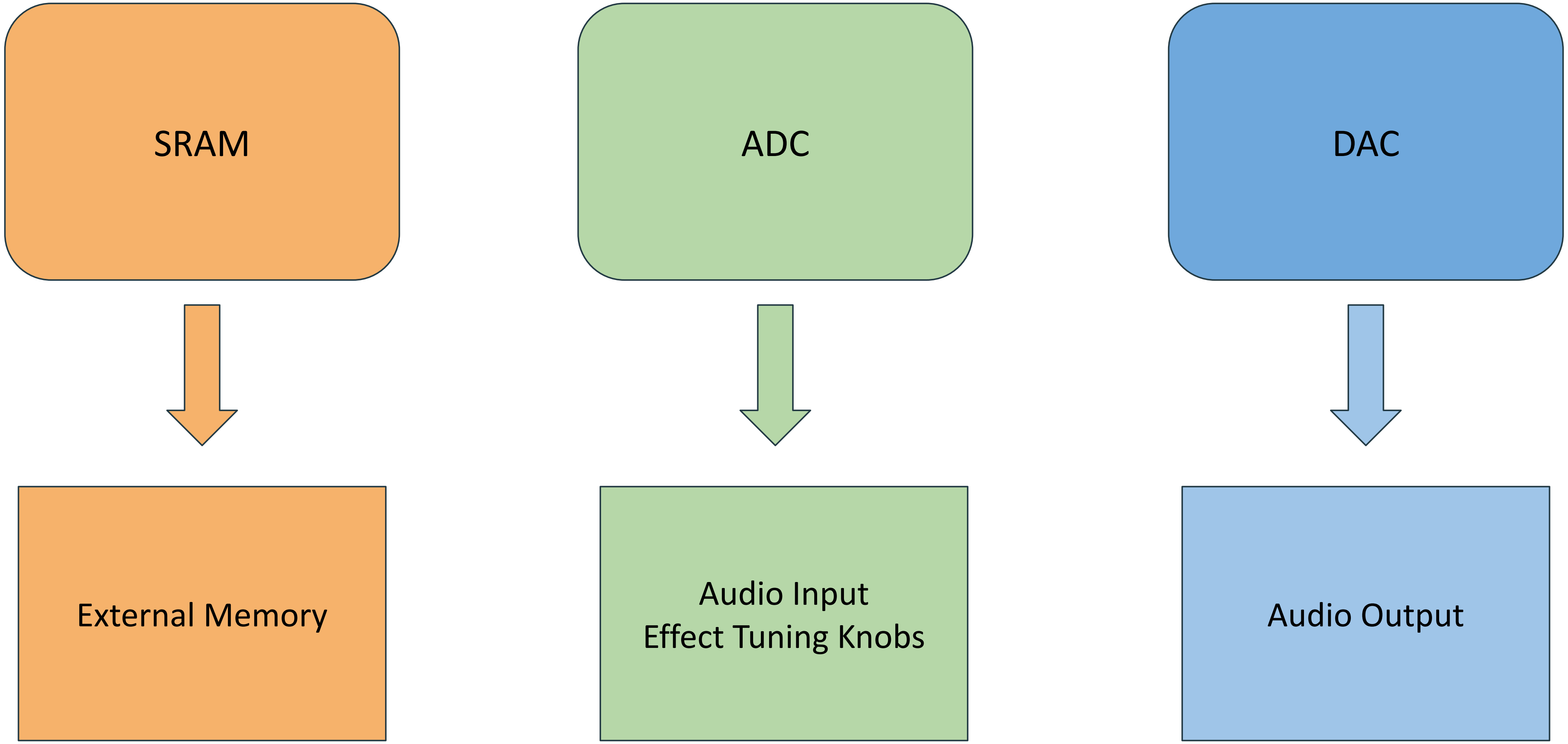


Wave - Default

	Msgs										
/compression_tb/threshold	-No Data-	00	9a	cc	aa	65	f0	aa	81	ff	55
/compression_tb/ratio	-No Data-	00	f0	3f	c0	18	cc	55	01	ff	aa
/compression_tb/audio_in	-No Data-	0000	5555	2aaa	f0aa	0f0f	ccf0	5555	ff00	00ff	cccc
/compression_tb/audio_out	-No Data-	0000	5555	e34b	0a00	0f0f	da90	5555	ff00	00fd	cccc
/compression_tb/threshold16pos	-No Data-	0000	9a00	cc00	aa00	6500	f000	aa00	8100	ff00	5500
/compression_tb/threshold16neg	-No Data-	0000	6600	3400	5600	9b00	1000	5600	7f00	0100	ab00
/compression_tb/audio_excesspos	-No Data-	0000	bb55	5eaa	46aa	aa0f	dcf0	ab55	7e00	01ff	77cc
/compression_tb/audio_excessneg	-No Data-	0000	10ab	0956	6556	8bf1	4310	00ab	8000	0001	de34
/compression_tb/audio_excessPosTimeRatio23	-No Data-	000000	af9fb0	174bd6	34ff80	0ff168	b00f40	38e339	007e00	01fd01	4f8d78
/compression_tb/audio_excessNegTimeRatio23	-No Data-	000000	0fa050	024c2a	4c0080	0d1e98	3570c0	0038c7	008000	0000ff	938e88
/compression_tb/audio_excessPosTimeRatio15	-No Data-	0000	af9f	174b	34ff	0ff1	b00f	38e3	007e	01fd	4f8d
/compression_tb/audio_excessNegTimeRatio15	-No Data-	0000	0fa0	024c	4c00	0d1e	3570	0038	0080	0000	938e
/compression_tb/audio_outpos	-No Data-	0000	5555	e34b	deff	0f0f	ccf0	5555	817e	00fd	a48d
/compression_tb/audio_outneg	-No Data-	0000	5660	31b4	0a00	0f0f	da90	55c8	ff00	0100	cccc

Now 100000 ps
Cursor 1 100001 ps

Off-Chip Components



Selection Criteria

SRAM →

- Memory Size (>400 kB)
- Access Time

ADC →

- Sampling Rate (>10 kHz)
- Caravel Absolute Maximum Ratings

DAC →

- Sampling Rate (>10 kHz)
- Caravel Absolute Maximum Ratings

Prototyping - Using Python

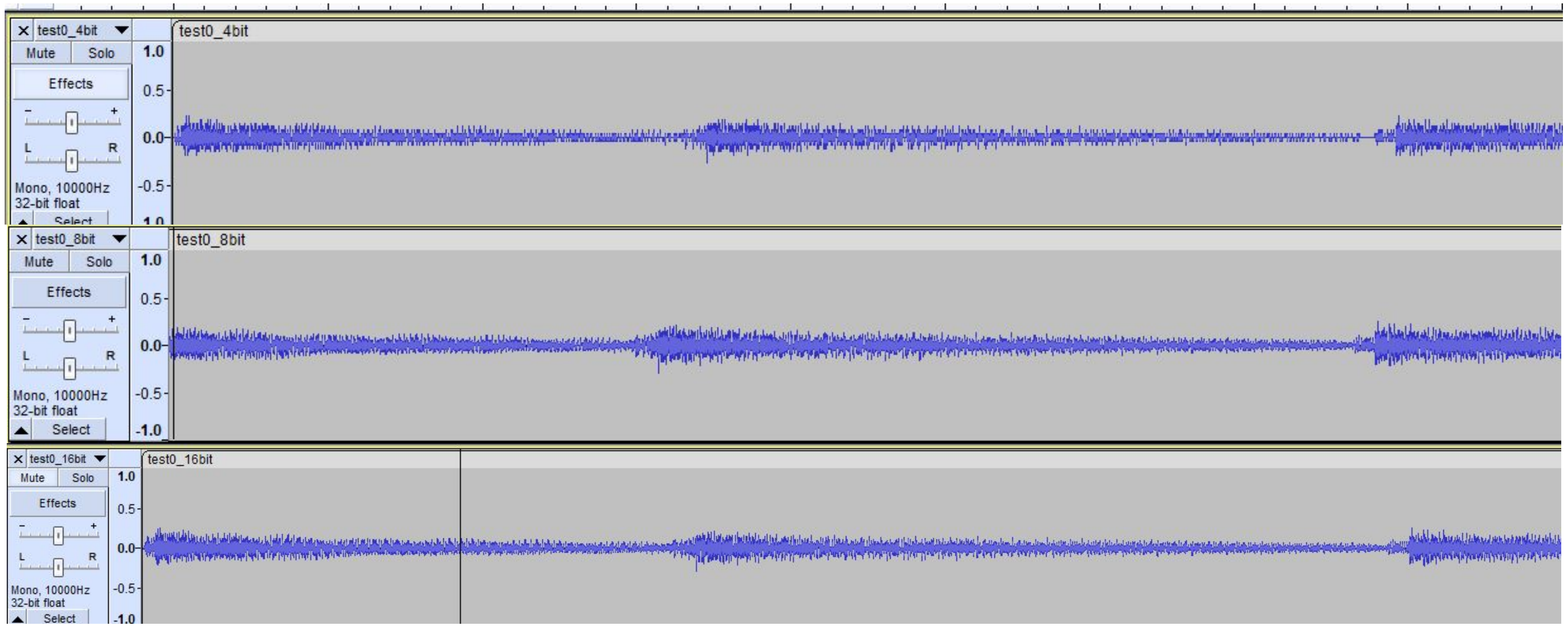
Python Module shows current implementation of verilog design.

Reflects the sampling rate change and bit width of design.

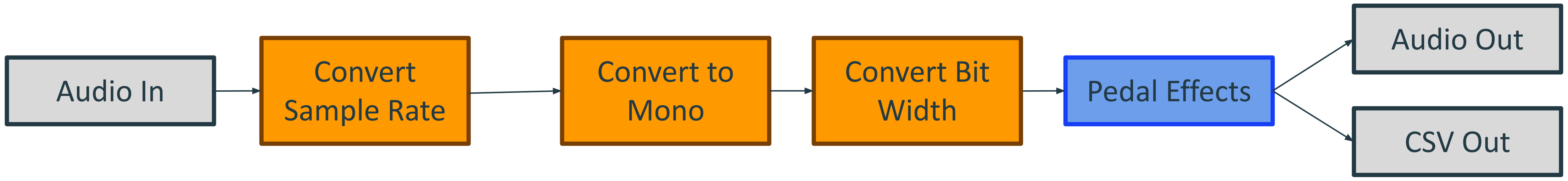
4 bit:

8 bit:

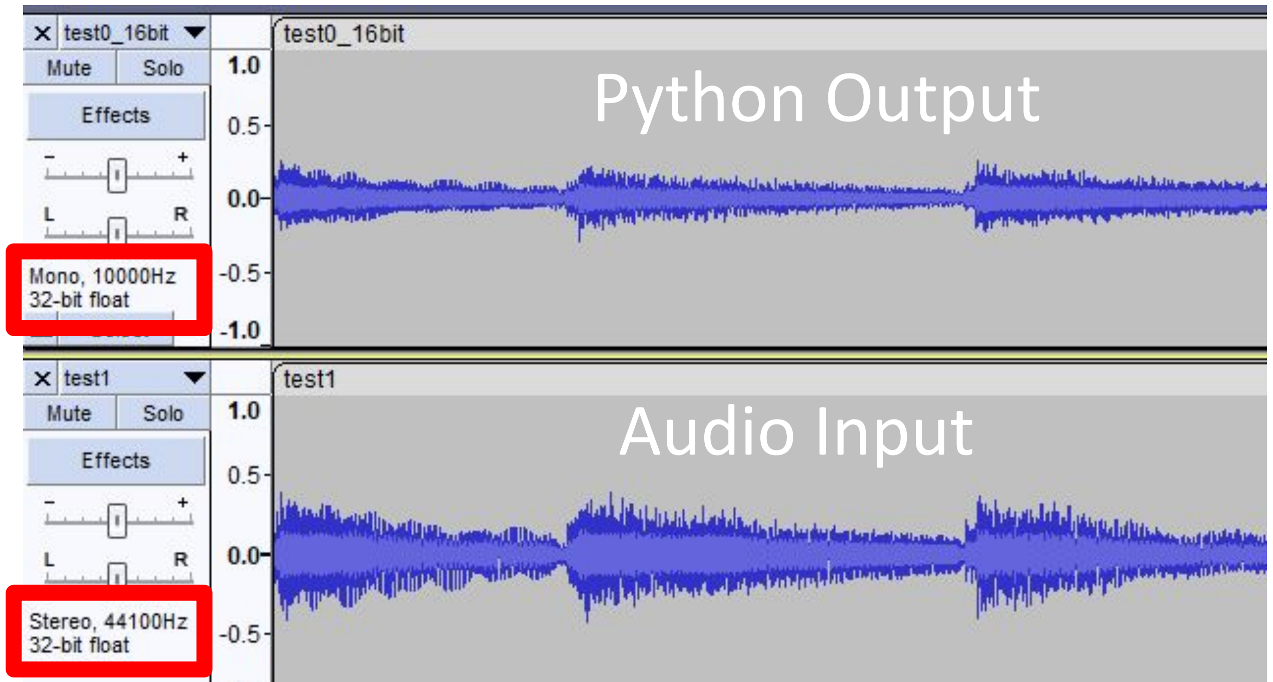
16bit:



Python Module Steps



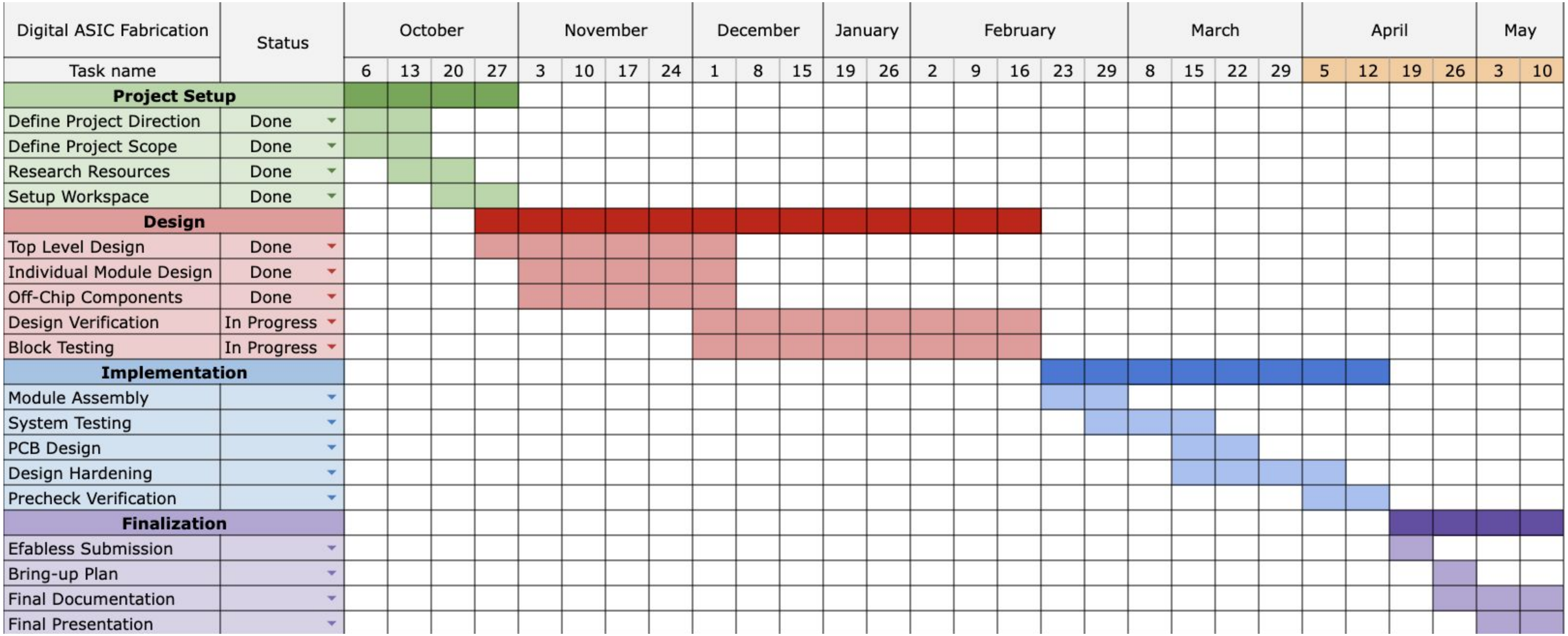
This process let us determine acceptable audio bit width and sample rates.



Design Complexity

- What made your design hard?
    - keeping track of multiple clocks and managing on chip and off chip memory
- What kind of design iterations were needed?
    - Initially our design was over simplified only having the function blocks in verilog
    - We then decided that off chip memory was needed to support more storage if we wanted to record a reasonable length of audio, no previous group has had off chip components
- Layout space
- Keeping track of clock speeds to determine cycles needed for certain operations such as accessing memory, which is determined based on the SPI

Project Schedule



Digital ASIC Fabrication	Status	October				November				December			January		February					March				April				May	
Task name		6	13	20	27	3	10	17	24	1	8	15	19	26	2	9	16	23	29	8	15	22	29	5	12	19	26	3	10
Project Setup		🟩	🟩	🟩	🟩																								
Define Project Direction	Done	🟩	🟩																										
Define Project Scope	Done	🟩	🟩																										
Research Resources	Done		🟩	🟩																									
Setup Workspace	Done			🟩	🟩																								
Design					🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥												
Top Level Design	Done				🟥	🟥	🟥	🟥	🟥	🟥																			
Individual Module Design	Done					🟥	🟥	🟥	🟥	🟥																			
Off-Chip Components	Done					🟥	🟥	🟥	🟥	🟥																			
Design Verification	In Progress									🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥												
Block Testing	In Progress									🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥												
Implementation																		🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦				
Module Assembly																		🟦	🟦										
System Testing																			🟦	🟦	🟦								
PCB Design																					🟦	🟦							
Design Hardening																					🟦	🟦	🟦	🟦					
Precheck Verification																								🟦	🟦				
Finalization																										🟪	🟪	🟪	🟪
Efabless Submission																										🟪			
Bring-up Plan																											🟪		
Final Documentation																											🟪	🟪	🟪
Final Presentation																												🟪	🟪

Project Plan

Jonathan Hess

- Python and Verilator Testing
- Top Level Design
- Establish usage of EFabless tools

Yu Wei Tan

- Off-Chip Design
- PCB Design

Samuel Heikens

- Compression Module
- PCB Design

Haris Khan

- SPI Design
- Memory

Python Module Testing

Python Module has been created which can:

- Simulate expected outputs
- Create test data for all 3 levels of top level verilog testing
- Create audio from module outputs
- Create audio from ASIC verilog output

Useful for hearing results of design before fabrication

Verilog Top Level Testing

Using Verilator script along with python test data, testing will be done on 3 top level testbenches:

1 - Without Control or SPI

2 - Without SPI

3 - Fully controlled by SPI

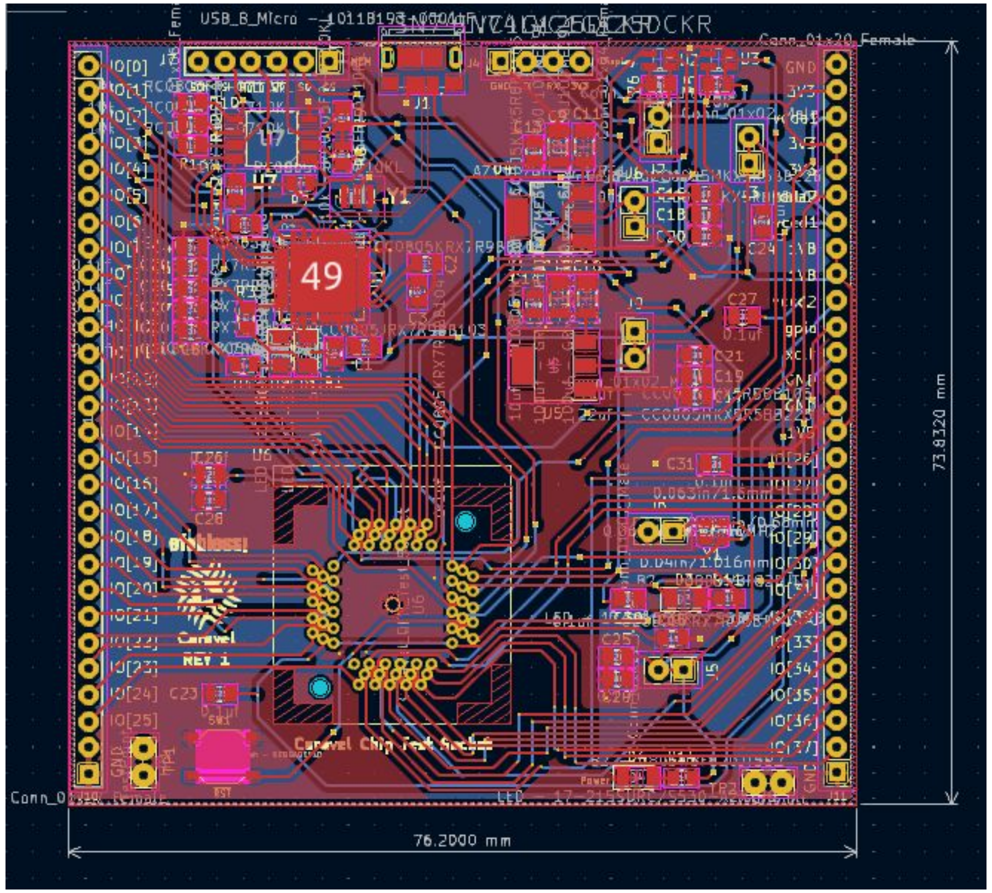
These will allow for verification that the verilog matches the python module. Also helps us with debugging to see which component is at fault

PCB Level Testing

Create PCB using Kicad

Then create test plan for future students to do the following:

- Use soldering instructions to place parts
- Check connections with multimeter
- Use python code with [Caravel PCB]()
- Use python script for loading and testing individual components



Caravel Test Socket PCB

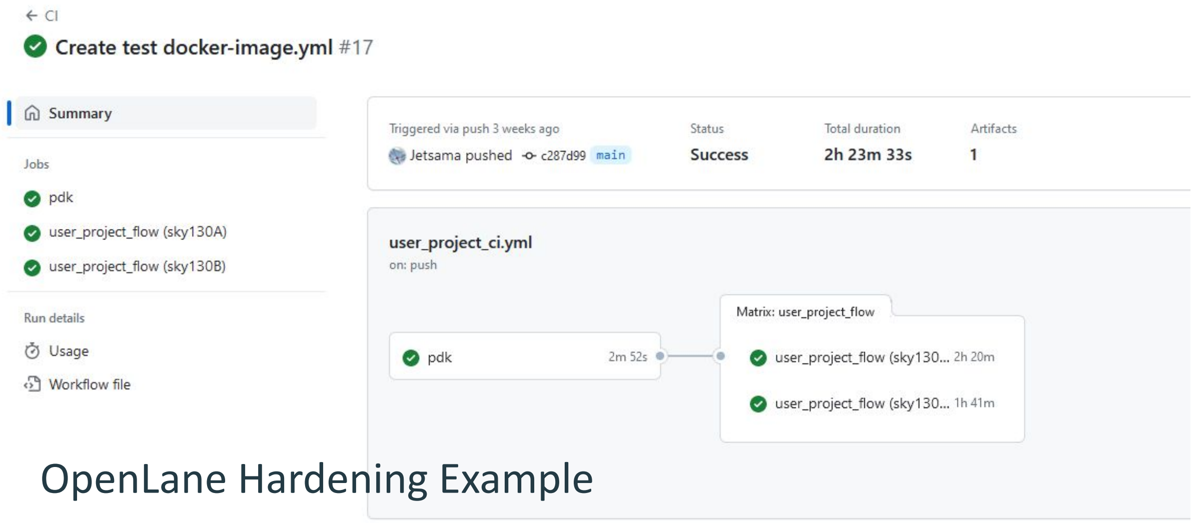
GitHub Runners

Verilator Runner

- Runs Verilator Testbench on changed files if exists

OpenLane Runner

- Runs OpenLane Hardening on design



← CI

Create test docker-image.yml #17

Summary

Jobs

pdk

user_project_flow (sky130A)

user_project_flow (sky130B)

Run details

Usage

Workflow file

Triggered via push 3 weeks ago	Status	Total duration	Artifacts
Jetsama pushed c287d99 main	Success	2h 23m 33s	1

user_project_ci.yml

on: push

pdk 2m 52s
Matrix: user_project_flow
user_project_flow (sky130... 2h 20m
user_project_flow (sky130... 1h 41m

OpenLane Hardening Example

Conclusion

Project Status

- Modules are design and need to be tested using python script

Next Project Steps

- Create PCB test plan to check connections with test points
- Python script for loading and testing individual components

References

https://efabless.com/open_shuttle_program

Q & A