Digital ASIC Fabrication

Design Document

Sddec23-06
Client: Dr. Duwe
Advisor: Dr. Duwe

Jake Hafele – Scribe, SPI Lead
Gregory Ling – Client PoC, Custom Cell Lead
Cade Breeding – Researcher, Clock Lead
Will Galles – Researcher, DSP Lead

sddec23-06@iastate.edu
http://sddec23-06.sd.ece.iastate.edu/

Revised: 5/3/2023 Revision: 1.1

Executive Summary

Problem Statement

The ability to create a custom digital ASIC (Application Specific Integrated Circuit) is often locked behind high barriers to entry and restricted to industry professionals. Many different groups would benefit from a method to create custom ASIC designs including research groups, future senior design teams, and other students who would benefit from the experience of designing custom parts. We will submit a design to the eFabless platform to test the limits of what they can manufacture and determine the limits of their processes including clock gating, power management, custom logic cells, and various other components of the framework. This learning will be passed down to future users who will use our design as a reference in creating their own for research or other learning tasks.

Development Standards & Practices Used

- **IEEE 1364-2005 - IEEE Standard Verilog Hardware Description Language**
    - Our ASIC will be written in Verilog and make use of Value Change Dump files for simulation results, specified in this standard
- **ISO/IEC 9899:2018 – C programming language (C17)**
    - Our test programs for the harness MCU will be written in the C17 language.
- **TIA/EIA 232-F – RS232 (UART) Protocol**
    - Our bring-up plan will use UART to communicate to the harness MCU to communicate test data.
- **SPI Protocol**
    - This is a de facto standard originally defined by Motorola and is a simple communication interface we will use as a backdoor into our design.
- **Wishbone Bus**
    - Open-source hardware bus for interconnecting peripherals like an AXI bus created by Silicore Corporation. We will use this bus for interfacing between the harness MCU and our peripheral modules we implement.
- **I2C Protocol**
    - The I2C protocol was a different form of bus protocol originally designed by NXP Semiconductor. We considered using I2C for managing the backdoor interface, but opted for SPI as it would be simpler to implement.

Summary of Requirements

- Test the limits of the eFabless process to the best of our ability including, but not limited to:
    - Test clock gating
    - Instantiating several standard cells provided by eFabless
    - Explore custom logic cells
    - Ensure our design is modular
    - Using the wishbone bus provided in the wrapper project
- Include a bring up plan for future team to test manufactured design once returned
- Test and verify functionality of previous team's manufactured design if it is shipped within a reasonable period
- Use the mpw_precheck tool at https://github.com/efabless/mpw_precheck to check our design requirements before submission

Applicable Courses from Iowa State University Curriculum

- CPR E 281 – Digital Logic
    - Basic hardware design in Verilog with simple digital circuits
- CPR E 288 – Embedded Systems I
    - C programming on embedded devices
- CPR E 381 – Computer Organization and Assembly Programming
    - Complex tasks in hardware design, creation of a MIPS processor
- CPR E 488 – Embedded Systems Design
    - Design of various embedded systems, introduction to AXI busses
- ENGL 250 – Written, Oral, Visual, and Electronic Composition
    - Introduction to English communication in various media formats
- ENGL 314 – Technical Communication. Writing technical documentation
    - Technical documentation and presentations on complex topics

New Skills/Knowledge acquired that was not taught in courses

Tools:

- KLayout
- GTKWave
- OpenROAD
- OpenLANE
- Magic
- Icarus Verilog

Skills:

- ASIC Design
- Using open-source tools
- Version Control
- Agile Workflow

- Parsing sparse documentation

Knowledge Gained:

- Clock Domain Crossing
- Clock Gating
- Digital Signals Processing
- Cell Layout
- Routing

Table of Contents

List of Acronyms

- ADC – Analog to Digital Converter
- ASIC – Application Specific Integrated Circuit
- AXI – Advanced eXtensible Interface
- DAC – Digital to Analog Converter
- DMA – Direct Memory Access
- DRC – Design Rule Check
- DSP – Digital Signal Processing
- FPGA – Field Programable Gate Array
- FRAM – Ferroelectric Random Access Memory
- GDS – Graphic Data System
- GPIO – General Purpose Input / Output
- I2C – Inter Integrated Circuit protocol
- IEEE – Institute of Electrical and Electronics Engineers
- IO – Input / Output
- LA – Logic Analyzer
- LVS – Layout Versus Schematic
- MPW – Multi-Project Wafer
- MS – Microsoft
- NSPE – National Society of Professional Engineers
- PCB – Printed Circuit Board
- PDK – Product Development Kit
- PDN – Power Delivery Network
- RTL – Register Transfer Level
- ReRAM – Resistive Random Access Memory
- SHA-1 – Secure Hash Algorithm 1
- SOC – System On Chip
- SPI – Serial Peripheral Interface
- SVB – Silicon Valley Bank
- UART – Universal Asynchronous Receiver and Transmitter
- WSL – Windows Subsystem for Linux

List of Definitions

- eFabless – Open-source fabrication company who will manufacture our design
- Caravel Harness – Provided wrapper around our design which includes a pre-built SoC
- User Area – The region inside the Caravel Harness we are allowed to edit
- Management Area / SoC – The part of the Caravel Harness which contains the management utilities including the SoC and logic analyzer probes
- Wishbone bus – The peripheral bus used by the Management SoC to communicate with peripherals in the User Area
- Verilog – The hardware design language specified by IEEE Std 1364-2005 which we will use for implementing our design in the user area
- SkyWater 130nm – The fabrication process used by eFabless supported by the SkyWater Foundry
- GTKwave – A cross-platform and open-source waveform viewer for viewing simulation results from VCD files
- KLayout – An open-source tool for viewing and editing mask layouts
- OpenROAD – The collection of open-source tools based on OpenLANE configured and provided by eFabless to generate production files from Verilog descriptions
- RISC-V – An open-source instruction set architecture that defines the group of commands that is used by software to communicate with the hardware of the processor.

List of Figures

List of Tables

1 Our Team

1.1 Team Members

- Cade Breeding
- Gregory Ling
- Jake Hafele
- Will Galles

1.2 Required Skill Sets for Your Project

- Digital logic (state machines & combinational logic)
- Embedded systems (microcontroller programming)
- Verilog
- Waveform analysis
- Digital signal processing
- Revision control (Git)
- Open communication
- Commitment to accountability
- ASIC layout/hardening
- Linux debugging

1.3 Skill Sets covered by the Team

- Digital logic (state machines & combinational logic): CB, GL, JH, WG
- Embedded systems (microcontroller programming): CB, GL, JH, WG
- Verilog: CB, GL, JH, WG
- Waveform analysis: CB, GL, JH, WG
- Digital signal processing: Minimal, need to learn
- Revision control (Git): CB, GL, JH, WG
- Open communication: CB, GL, JH, WG
- Commitment to accountability: CB, GL, JH, WG
- ASIC layout/hardening: None, need to learn
- Linux debugging: CB, GL, WG

1.4 Project Management Style Adopted by the team

- Kanban task tracking
    - Use the task manager in MS Teams to control workflow
    - Keeps client and advisor in loop
    - Each member tracks their own tasks
    - Each member can pull in new work once theirs is completed
    - The group can keep track of everyone's accomplishments and possible sticking points
- Agile workflow
    - Hold weekly intra-team update meetings on Sundays
    - Communicate with each other over teams outside of meeting times

    - Weekly meetings with Client/Advisor on Mondays to demonstrate progress and get feedback
    - Able to adapt quickly to changes caused by the OpenROAD tooling or other unforeseen challenges

1.5 Initial Project Management Roles

- Researcher, Clock Lead – Cade Breeding
- Client Point of contact, Custom Cell Lead – Gregory Ling
- Scribe, SPI Lead – Jake Hafele
- Researcher, DSP Lead – Will Galles

2 Introduction

2.1 Problem Statement

The ability to create a custom digital ASIC (Application Specific Integrated Circuit) is often locked behind high barriers to entry and restricted to industry professionals. Many different groups would benefit from a method to create custom ASIC designs including research groups, future senior design teams, and other students who would benefit from the experience of designing custom parts. We will submit a design to the eFabless platform to test the limits of what they can manufacture and determine the limits of their processes including clock gating, power management, custom logic cells, and various other components of the framework. This learning will be passed down to future users who will use our design as a reference in creating their own for research or other learning tasks.

2.2 Requirements & Constraints

Advisor Requirements

For this project, our advisor has several requirements for us to fulfill as a result of this project:

- Test the limits of the eFabless process to the best of our ability
- Include a bring up plan for a future team to test manufactured design once returned
- Test and verify functionality of previous team's manufactured design if it is shipped within a reasonable period
- Use the mpw_precheck tool at https://github.com/efabless/mpw_precheck to check our design requirements before submission

To satisfy the requirement of testing the eFabless process, we have defined the following sub-requirements which we will complete:

- Test clock gating
- Instantiating several standard cells provided by eFabless
- Explore custom logic cells
- Instantiate a small, yet relatively complex module
- Have multiple redundancy paths if one fails
- Ensure our design is modular
- Use the wishbone bus provided in the wrapper project

Design Constraints

The major constraint with this project is that it must use the eFabless manufacturing process. As a result, there are many sub-constraints which we are required to comply with for our design to be accepted by eFabless detailed below.

eFabless Open MPW (Multi-Project Wafer) submission Constraints:
https://platform.efabless.com/shuttles/MPW-8

The following project requirements must be met to qualify for inclusion on the open MPW shuttle program:

- The project must be targeted on the currently supported Open PDK.
- The project must be posted on a git-compatible repo and be publicly accessible.
- 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. See details here.
- The repo must include project documentation and adhere to Google's inclusive language guidelines. See details here.
- 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 testbench for their design integrated into Caravel. The Caravel User Project provides an example of how to implement this.

Caravel Harness Directory Structure Constraints
https://caravel-harness.readthedocs.io/en/latest/getting-started.html#required-directory-structure

Required Directory Structure

- gds/ : includes all the gds files used or produced from the project.
- def : includes all the def files used or produced from the project.
- lef/ : includes all the lef files used or produced from the project.
- mag/ : includes all the mag files used or produced from the project.
- maglef : includes all the maglef files used or produced from the project.
- spi/lvs/ : includes all the spice files used or produced from the project.
- verilog/dv : includes all the simulation test benches and how to run them.
- verilog/gl/ : includes all the synthesized/elaborated netlists.
- verilog/rtl : includes all the Verilog RTLs and source files.
- openlane/<macro>/ : includes all configuration files used to run openlane on your project.

- info.yaml: includes all the info required in this example. Please make sure that you are pointing to an elaborated caravel netlist as well as a synthesized gate-level-netlist for the user_project_wrapper

User Project Constraints
https://github.com/efabless/caravel_user_project/blob/main/docs/source/index.rst#user-project-wrapper-requirements
Your hardened user_project_wrapper must match the golden user_project_wrapper in the following:

- Area (2.920mm x 3.520mm)
- Top module name "user_project_wrapper"
- Pin Placement
- Pin Sizes
- Core Rings Width and Offset
- PDN Vertical ancd Horizontal Straps Width
- You are allowed to change the following if you need to:
- PDN Vertical and Horizontal Pitch & Offset
- To make sure that you adhere to these requirements, we run an exclusive-or (XOR) check between your hardened user_project_wrapperGDS and the golden wrapper GDS after processing both layouts to include only the boundary (pins and core rings). This check is done as part of the mpw-precheck tool.

2.3 Engineering Standards

The following list describes a set of engineering standards that we used or considered for our project:

- **IEEE 1364-2005 – IEEE Standard Verilog Hardware Description Language**
    - Our ASIC will be written in Verilog and make use of Value Change Dump files for simulation results, specified in this standard
- **ISO/IEC 9899:2018 – C programming language (C17)**
    - Our test programs for the harness MCU will be written in the C17 language.
- **TIA/EIA 232-F – RS232 (UART) Protocol**
    - Our bring-up plan will use UART to communicate to the harness MCU to communicate test data.
- **Serial Peripheral Interface (SPI) Protocol**
    - This is a de-facto standard originally defined by Motorola and is a simple communication interface we will use as a backdoor into our design.
- **Wishbone Bus**
    - Open-source hardware bus for interconnecting peripherals like an AXI bus created by Silicore Corporation. We will use this bus for interfacing between the harness MCU and our peripheral modules we implement.
- **Inter-Integrated Circuit (I2C) Protocol**

- The I2C protocol was a different form of bus protocol originally designed by NXP Semiconductor. We considered using I2C for managing the backdoor interface, but opted for SPI as it would be simpler to implement.

2.4 Intended Users and Uses

Our project has several different intended users, each with different reasons for using our design and different requirements for our project. For the most part, our design is going to be a silicon-proven reference to other groups interested in implementing their own circuit in hardware.

Our Team

The main user of interest for our project is us. Our project will teach us about the fabrication process, give us a thorough understanding of how an ASIC is designed from the ground up, and what limitations are in ASIC designs. We have used FPGAs in several of our classes to create specialized digital circuits before, but never at the level of a custom silicon hardware design. For us, learning and exploring will be one of the main requirements of this project.

Future Senior Design Groups

Future Senior Design groups will use the results of our project to guide their future designs. Our design will test the limits of the design process so future teams know what is possible and how to implement it. They will be creating a project for a specific task and use our design as a reference for how to design their projects and what the limitations are.

Research Teams

The research team use case is very similar to the future senior design groups. There are several groups who would benefit from being able to create low-cost custom ASICs for their research goals. Having a proven design that they can reference will allow them to perform more cutting-edge research using more efficient technology than FPGAs which tend to be significantly higher power.

eFabless Open-Source Community

The eFabless project is centered around the open-source community. Every submission to the eFabless project is required to be open source and visible to other groups for them to reference for their designs. Just as we will look at other designs to determine what to experiment with in this project, other potential designers will look at our designs to see what we have done to fabricate their own projects.

2.5 Market Survey

The market price for a minimal production run for a custom ASIC is about $1M. This is not a feasible option for small groups who only need to create a small quantity of a custom design and do not have enough funding to pay for a $1M production run (“How much does it cost”). As a result, several companies have begun offering Multi-Project Wafer options where multiple groups are bundled together onto a single wafer and the production run cost is split between them. Each group receives a smaller order quantity, but also a significantly reduced cost. We will be submitting to the OpenMPW project by eFabless which is a free option funded by Google that requires that all submissions be open source. eFabless has another option named ChipIgnite which has no open-source requirement, has a stricter schedule, and costs $9,750 per 10 mm^2 project (eFabless). Other companies also provide MPW submissions include MUSE Semiconductor which uses TSMC’s manufacturing at a comparable price of around $1,250 per mm^2.

One additional note is eFabless provides configured open-source tooling to use with their processes, MUSE’s educational package is $1,000 for the PDK, $22,000 per mm^2, and requires an NDA.

3 Project Plan

3.1 Project Management/Tracking Procedures

Our team has decided to follow an agile project management style. Due to the unforeseen complexity of many parts of our project, being able to adapt to different changes as they arise in our workflow will be a great benefit. We have weekly update meetings within our team, and weekly update meetings with our advisor to keep us synchronized and ensure progress is being made.

We track our progress primarily through the Tasks module in MS Teams, supporting our agile model by assigning specific tasks to each member and keeping track of past and future tasks which need to be completed. We also utilize git and the ECpE GitLab for version control purposes to ensure our code base is synchronized across our team.

3.2 Task Decomposition

Below is a numbered list of each decomposed task that is required to complete our project:

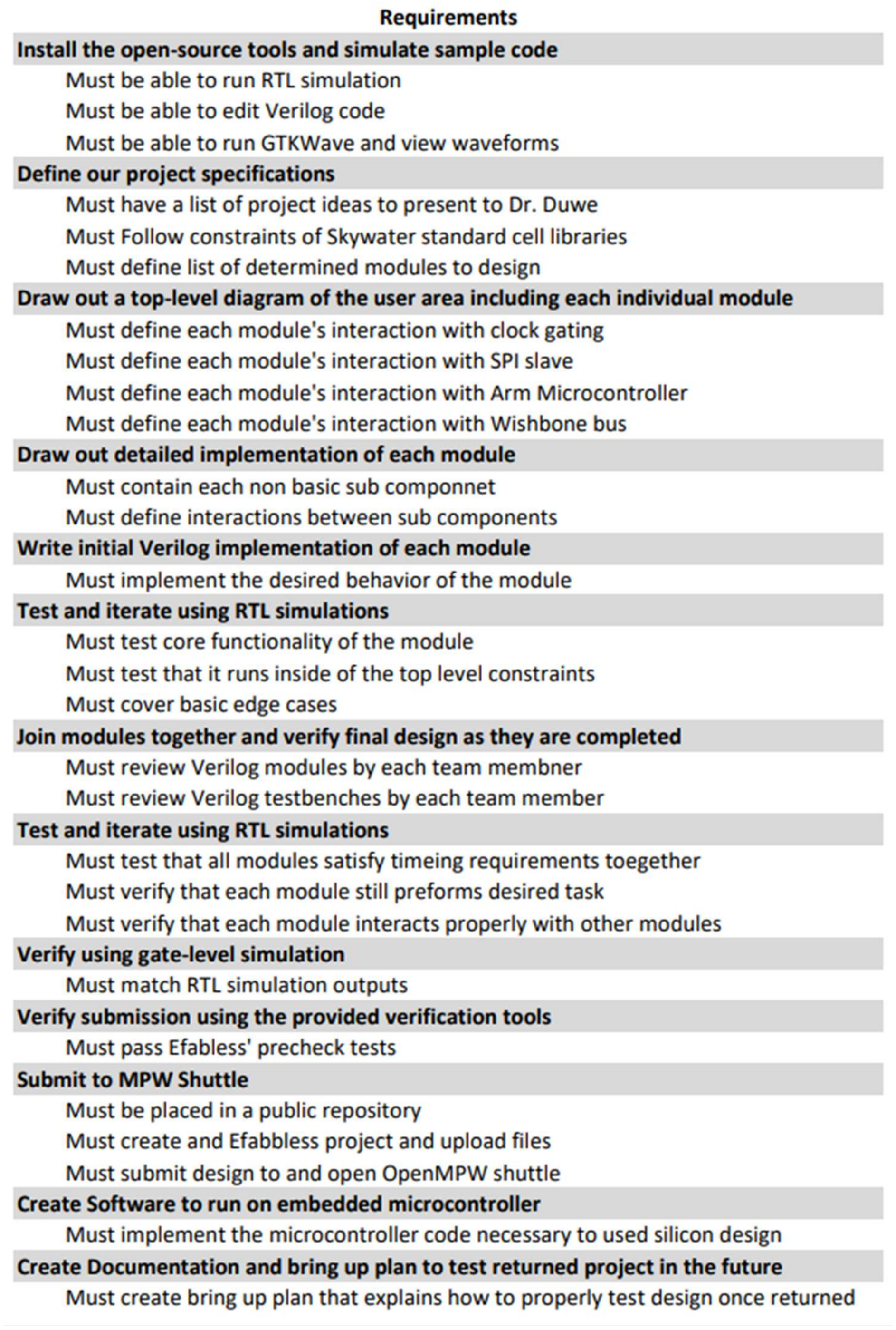
1. **Install the open-source tools and simulate sample code**
    a. Build a sample user project with the Caravel tool flow
    b. Complete a Verilog simulation of a sample user project
    c. View the output waveforms of a sample user project in GTKWave
2. **Define our project specifications**
    a. Create list of many possible modules that could be useful to implement
    b. Investigate Skywater standard cell libraries to determine what design modules are possible
    c. From that list narrow down and select the most important few that we would want to develop
3. **Draw out a top-level diagram of the user area including each individual module**
    a. Determine how each module will interact with clock gating
    b. Determine how each module will interact with SPI slave
    c. Determine how each module will interact with included ARM microcontroller
    d. Determine how each module will interact with the wishbone bus
4. **Draw out detailed implementation of each module**
    a. Draw out module to include each of the needed subcomponents that will need to be created
    b. Define interactions between subcomponents for data and control paths needed for full functionality
5. **Write initial Verilog implementation of each module**
    a. Create a Verilog implementation of each module assigned to individual members
6. **Test and iterate using RTL simulations**
    a. Create thorough tests that will cover main functionality of module
    b. Create tests that verify module satisfies top level constraints of the module
    c. Create basic tests to cover edge cases outside of typical operating state
7. **Join modules together and verify final design as they are completed**
    a. Review Verilog modules designed by each team member

    b. Review Verilog testbenches designed by each team member

8. **Test and iterate using RTL simulations**
    a. Create timing tests to ensure that all individual modules satisfy the timing requirements of the system
    b. Create main functionality tests that verify each module gives correct results for main desired task
    c. Create tests to verify functional interactions between modules
9. **Verify using gate-level simulation**
    a. Create tests to ensure modules operate with the same functionality as our RTL simulations
10. **Verify submission using the provided verification tools**
    a. Ensure that final project passes Efabless' precheck tests
11. **Submit to MPW Shuttle**
    a. Create public repository to satisfy Efabless' open-source requirement
    b. Create a project on Efabless' website and point at our public repository
    c. Submit the design to the time applicable OpenMPW shuttle
12. **Create Software to run on embedded microcontroller**
    a. Create a repository of tested sample code which will verify the integrity of our system
13. **Create Documentation and bring up plan to test returned project in the future**
    a. Document the bring-up plan in a detailed form for a future user (at the level of a 288 student) could use to test our design when returned from eFabless

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

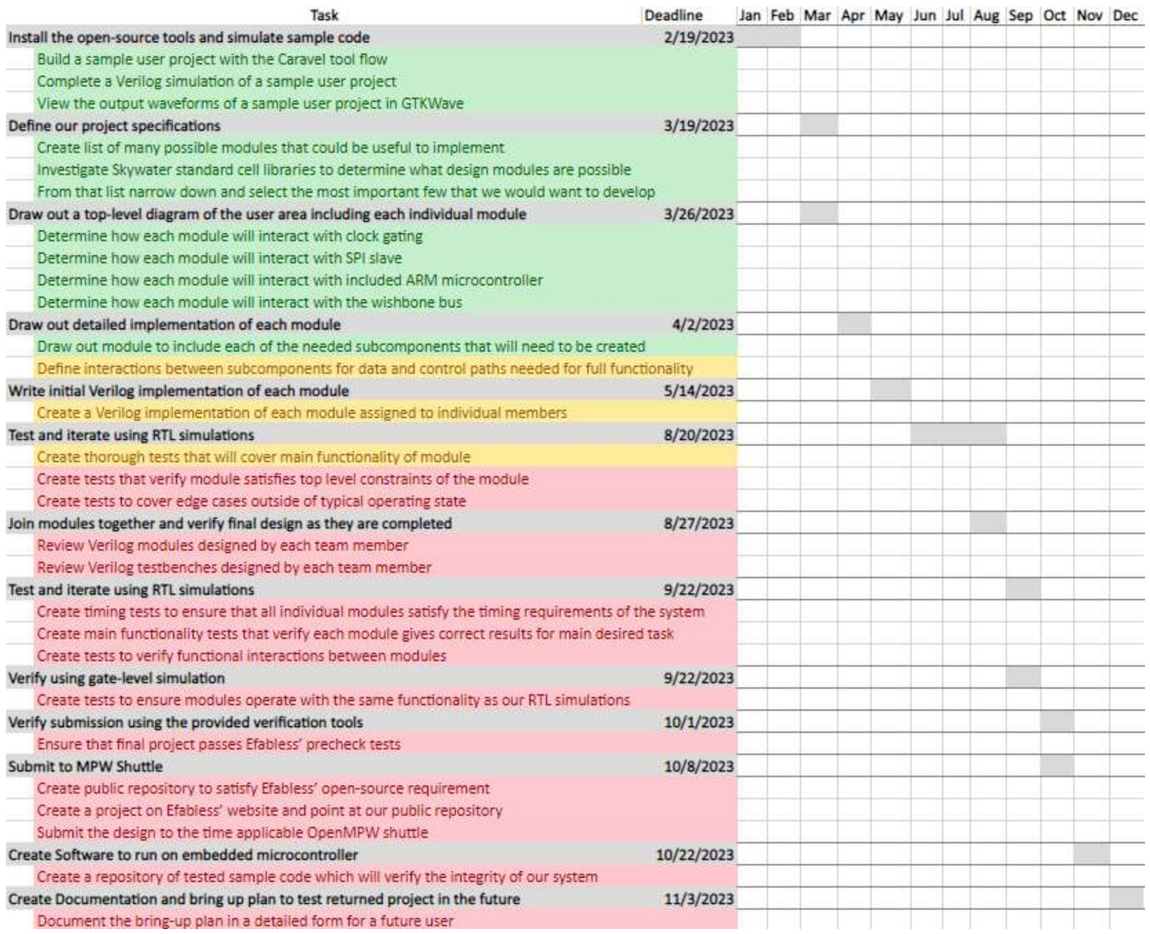
Due to the nature of our project, we rely on functional requirements, and our milestones are based on qualitative data. Our progress milestones are defined as a functional module or subunit of a module that has been tested and verified independently from the rest of the design which will provide small enough granularity for our group to keep up with task deadlines well.



Requirements
Install the open-source tools and simulate sample code
Must be able to run RTL simulation
Must be able to edit Verilog code
Must be able to run GTKWave and view waveforms
Define our project specifications
Must have a list of project ideas to present to Dr. Duwe
Must Follow constraints of Skywater standard cell libraries
Must define list of determined modules to design
Draw out a top-level diagram of the user area including each individual module
Must define each module's interaction with clock gating
Must define each module's interaction with SPI slave
Must define each module's interaction with Arm Microcontroller
Must define each module's interaction with Wishbone bus
Draw out detailed implementation of each module
Must contain each non basic sub componnet
Must define interactions between sub components
Write initial Verilog implementation of each module
Must implement the desired behavior of the module
Test and iterate using RTL simulations
Must test core functionality of the module
Must test that it runs inside of the top level constraints
Must cover basic edge cases
Join modules together and verify final design as they are completed
Must review Verilog modules by each team membner
Must review Verilog testbenches by each team member
Test and iterate using RTL simulations
Must test that all modules satisfy timeing requirements toegether
Must verify that each module still preforms desired task
Must verify that each module interacts properly with other modules
Verify using gate-level simulation
Must match RTL simulation outputs
Verify submission using the provided verification tools
Must pass Efabless' precheck tests
Submit to MPW Shuttle
Must be placed in a public repository
Must create and Efabbless project and upload files
Must submit design to and open OpenMPW shuttle
Create Software to run on embedded microcontroller
Must implement the microcontroller code necessary to used silicon design
Create Documentation and bring up plan to test returned project in the future
Must create bring up plan that explains how to properly test design once returned

Figure 1 Proposed Milestones

3.4 Project Timeline/Schedule



Task	Deadline	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Install the open-source tools and simulate sample code	2/19/2023	▨	▨										
Build a sample user project with the Caravel tool flow													
Complete a Verilog simulation of a sample user project													
View the output waveforms of a sample user project in GTKWave													
Define our project specifications	3/19/2023			▨									
Create list of many possible modules that could be useful to implement													
Investigate Skywater standard cell libraries to determine what design modules are possible													
From that list narrow down and select the most important few that we would want to develop													
Draw out a top-level diagram of the user area including each individual module	3/26/2023			▨									
Determine how each module will interact with clock gating													
Determine how each module will interact with SPI slave													
Determine how each module will interact with included ARM microcontroller													
Determine how each module will interact with the wishbone bus													
Draw out detailed implementation of each module	4/2/2023				▨								
Draw out module to include each of the needed subcomponents that will need to be created													
Define interactions between subcomponents for data and control paths needed for full functionality													
Write initial Verilog implementation of each module	5/14/2023					▨							
Create a Verilog implementation of each module assigned to individual members													
Test and iterate using RTL simulations	8/20/2023						▨	▨	▨				
Create thorough tests that will cover main functionality of module													
Create tests that verify module satisfies top level constraints of the module													
Create tests to cover edge cases outside of typical operating state													
Join modules together and verify final design as they are completed	8/27/2023								▨				
Review Verilog modules designed by each team member													
Review Verilog testbenches designed by each team member													
Test and iterate using RTL simulations	9/22/2023									▨			
Create timing tests to ensure that all individual modules satisfy the timing requirements of the system													
Create main functionality tests that verify each module gives correct results for main desired task													
Create tests to verify functional interactions between modules													
Verify using gate-level simulation	9/22/2023									▨			
Create tests to ensure modules operate with the same functionality as our RTL simulations													
Verify submission using the provided verification tools	10/1/2023										▨		
Ensure that final project passes Efabless' precheck tests													
Submit to MPW Shuttle	10/8/2023										▨		
Create public repository to satisfy Efabless' open-source requirement													
Create a project on Efabless' website and point at our public repository													
Submit the design to the time applicable OpenMPW shuttle													
Create Software to run on embedded microcontroller	10/22/2023											▨	
Create a repository of tested sample code which will verify the integrity of our system													
Create Documentation and bring up plan to test returned project in the future	11/3/2023												▨
Document the bring-up plan in a detailed form for a future user													

Figure 2 Project Timeline

3.5 Risks And Risk Management/Mitigation

All of our tasks carry significant risks. The overall project has its own set of risks, including the large risk that eFabless will not have an MPW submission open near the time we need to submit our design. This could be caused by several factors, a recent example being the SVB collapse which did affect eFabless in a minor capacity. The second major risk with our design is that anything can go wrong during fabrication, and we receive effectively a black box back. If we make a major design flaw in the clock system, the entire chip will be effectively unusable. Therefore, we will be ensuring throughout our design process that our project is as modular and separate as possible to minimize the risk that one small module's mistake will take down a large portion of the rest of our design with it. Any critical components will have backups to ensure redundancy, for example in the case the wishbone bus fails to operate, we will have a secondary SPI bus that can bypass the built-in wishbone bus and access all our modules independently. This also allows us to test the entire project even in the event that the processing system is unusable.

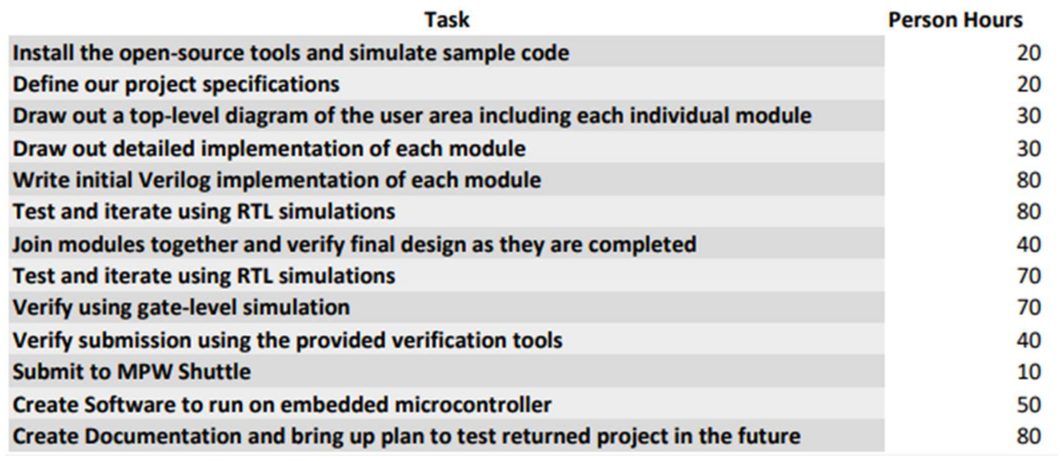
These risks will affect the outcome of our project, but not the viability of our product as a test of the system. If a submission does not open, our design will be left in a submittable form to be submitted whenever submissions do open, and the usefulness of our project will not be diminished. If our project fails due to a manufacturing flaw, then our project succeeded in determining a limitation of the eFabless system. However, if it is a major design defect, that will significantly limit the effectiveness of our tests, so we will use test cases to verify the logic of our design before submission.

Table 1 Project Risks

Risk	Estimated Probability
No MPW submission is available	40%
DSP module cannot both fit in user area and run at real time	40%
Wishbone bus is unable to interact with user modules after fabrication	5%
Fabrication error causes an individual module to fail	15%

3.6 Personnel Effort Requirements

These are the forecasted time requirements for this project. They are based on our initial couple of weeks working on the project and our interactions with previous senior design teams that have worked on similar projects.



Task	Person Hours
Install the open-source tools and simulate sample code	20
Define our project specifications	20
Draw out a top-level diagram of the user area including each individual module	30
Draw out detailed implementation of each module	30
Write initial Verilog implementation of each module	80
Test and iterate using RTL simulations	80
Join modules together and verify final design as they are completed	40
Test and iterate using RTL simulations	70
Verify using gate-level simulation	70
Verify submission using the provided verification tools	40
Submit to MPW Shuttle	10
Create Software to run on embedded microcontroller	50
Create Documentation and bring up plan to test returned project in the future	80

Figure 3 Time Requirements

3.7 Other Resource Requirements

We will have no direct financial dependencies to submit as eFabless is free. If we receive a past design, we may gain financial requirements for physical testing depending upon what we get back from eFabless.

The additional resources we require are all open-source tools which are provided by eFabless for use with their MPW submissions:

- eFabless OpenMPW shuttle program
- Skywater 130nm OpenPDK
- GTKwave
- KLayout
- OpenROAD

The GitLab instance we use is provided to us by the ECpE department.

4 Design

4.1 Design Context

4.1.1 Broader Context

Our project is situated inside the larger ecosystem of digital design. Digital design is broken down into multiple sectors to develop different kinds of electronics. Specifically, our project falls under the domain of developing Application Specific Integrated Circuits. This area of design is focused on implementing digital designs directly onto silicon chips. These chips are custom designed to perform a very specific task.

The main communities that our design will benefit will be our own team, future senior design groups, research teams, and the eFabless open-source community.

Our project tries to address the need for open-source ASIC design for students and research groups. These types of designs are typically locked behind high level academic research and major corporations with large capital backings. Our project strives to bring this technology to individuals that do not have the resources necessary to create one of these designs on their own in the current age. By creating our open-source project we are lowering the bar to entry for creating an ASIC chip for others referencing our work.

Table 2 Project Context

Area	**Description**	**Examples**
Public health, safety, and welfare	Our project strives to open the ability to design and manufacture ASIC chips to groups without the typical capital requirements. Our project will allow new designers to leverage our designs in creating their own designs later.	Our project will allow smaller businesses and individuals to enter the market in designing ASIC chips. This will increase competition and decrease costs for individual consumers down the road.
Global, cultural, and social	Our project will look to help the open-source digital design community. Our project will introduce more designs for new and existing members of the community to leverage in their own designs. This will create a better functioning and more active community.	The community will grow from more activity, and it will allow for newer members to be introduced with less effort. This will create an exponential growth in the community that will lead to more projects like ours that will in turn bolster the community.
Environmental	Our project strives to support smaller creators that can leverage the tools more effectively than larger corporations. Our project will create a better environment through the above process as a smaller scale design will allow for more efficient use of resources. It will allow for more	One example of how this will create a better environment is by allowing for more specific integrated circuits to be created. This will allow for smaller and more efficient devices to be created. This in turn will lower the energy footprint of these devices

	specialized devices that can reduce the overall material footprint.	which in turn will lower the overall carbon footprint of the devices utilizing these ASIC chips.
Economic	Our project, as part of the open-source partnership that this belongs to, will help make the design and fabrication of ASICs more financially available, so more available to many more interested groups such as future students, researchers, and hobbyist groups. With more individuals able to create these products there will be more competition in the market. This will allow smaller businesses and individuals to compete and make a profit in this market.	The nature of open-source work will allow individuals to create and market their own products. This will create a more competitive market that will increase innovation. This will overall create a better market for consumers.

4.1.2 User Needs

The user needs for each of our listed user groups in Section 2.4 are as follows:

Our Team

Our senior design team needs an accessible way to design and harden a digital ASIC because we are unfamiliar with the process and want to learn through using open-source tools and silicon proving another digital design.

Future Senior Design Groups

Future senior design groups need to execute a comprehensive test procedure because their main assignment for the project would be to verify the design of our digital ASIC deliverable.

They also need to have our project as a reference to create more complex designs with the knowledge that they will function as expected.

Research Teams

Research teams need more accessible tools to bring up digital hardware designs because there will are frequent changes or variations in designs, which can be near impossible to order with only one chip on a wafer due to costs and time.

eFabless Open-Source community

The open-source community needs more accessible resources to bring up digital chips because most ASICs currently require a large order of chips on one wafer, which is unrealistic for someone not a part of a large company.

4.1.3 Prior Work/Solutions

MPW-1 Shuttle:

- https://platform.efabless.com/projects/shuttle/1

Caravel Documentation:

- https://caravel-user-project.readthedocs.io/en/latest/

Prior Senior Design Teams:

- http://sddec22-17.sd.ece.iastate.edu
- http://sdmay23-28.sd.ece.iastate.edu
- http://sddec23-08.sd.ece.iastate.edu

We have access to all previous MPW shuttles which are designs that other groups or individuals have submitted to the eFabless project. These are made available as open-source reference designs for others to reference. We have briefly explored the MPW shuttles site; the projects range from simple adders to a 10-bit DAC or an AXI DMA. Ours is more of a spread of simple tests to see how the submission and fabrication process works with different parts of the ASIC.

We are following the previous and concurrent senior design teams which have worked on various ASIC projects (bitcoin mining, spiking neural networks, ReRAM). None of these projects have been returned from fabrication, so we have no results to reference yet. However, we do have their designs and documentation on how they created their designs. The major difference between our designs is theirs is a single coherent design which may give an advantage as it shows a specific application where this ASIC design process could be useful, but it comes with the shortcoming that it does not test as many individual aspects of the platform. Our design is more modular, which allows us to test more of the system and have more resiliency if one part fails, but it will not have a particular use-case aside from a test and playground module if it succeeds.

One other major difference is prior projects focused more on documentation while our group is more focused on exploring the platform's limits and not so much on documentation.

4.1.4 Technical Complexity

Our final ASIC design will include the following subcomponents to test different potential designs and reduce the risk factor of our overall design failing:

1. **Voice Road Noise Isolation Module**
   - This module will require the development of a large convolution network in hardware. Due to size constraints of the useable area this will also lead to the requirement to separate the operation out over multiple cycles.
2. **Backdoor SPI**
   - This module will require the development of our own protocol inside of the larger SPI interface. This will allow us to create our own method of communication to the interior modules without needing to interface through the integrated microcontroller. This protocol will need to interface between multiple clock domains and ensure data integrity through the process.
3. **Clock Gating**
   - This module will require development of a module capable of regulating the clocks inside our design. It will need to be able to shut down individual modules clock sources to shut them off. It will also need to be able to switch the chip from running on internal and external clock signals.
4. **Wishbone Test**
   - This test will require a functional test of the integrated data bus inside the ASIC. It will need to test the ability of the integrated microcontroller to send data to and receive from the user development area of the chip.
5. **Skywater Standard Cell Logic**
   - This module will require the implementation of a standard cell and development of a testing procedure to determine the integrity of the manufacturing process.
6. **Custom Cell Logic**
   - This module will require the development of our own cell in the sky water 103nm process. We will have to develop our design in the different layers of the actual manufacturing process to create a functional unit.

4.2 Design Exploration

4.2.1 Design Decisions

The following set of prompts include important design questions that we have considered during our ideation and planning phases for our project. We will focus on answering these

- Do we want a modular design where each team member implements a specific function, or a combined design that completes a larger task?
- What submodule designs will we choose to implement, and why will they be worthwhile?
- How can we reduce the risk of our overall design from failing with each submodule?
- How will we implement the Voice Road Noise Isolation Module that was previously designed for a microcontroller by another student for a different project?

4.2.2 Ideation

One of the more open design decisions was determining how we wanted to go about implementing the Voice Road Noise Isolation Module. The development of this module was inspired and guided by Issaac Rex's EE 529 Speech Enhancement project. The module had many considerations between different algorithms and processes to separate voice audio from road noise audio. Below are some of the different design ideas were considered:

- The first design idea was to implement a Weiner Filter using a direct convolution to apply the filter to the incoming audio all in the time domain.
- The second design idea was to implement a Weiner Filter again but this time to perform a Fourier transform on the incoming audio first. It would then apply the filter in the frequency domain before performing the inverse Fourier transform to generate the new output audio.
- The next idea was to use a Subspace algorithm that would isolate the clean audio using the Karhunen-Loeve Transform to reflect only the clean audio into a new subspace.
- The third method that we looked to utilize was the use of a Deep Neural Network that would be used to apply a correction to magnitudes generated from the Fourier transform. This method would again convert the incoming audio into the frequency domain before adjusting it and converting it back to the time domain.
- The last item that we looked at utilizing would be a Long Short Term Recurrent Neural Network. This design would differentiate from the previous as the neural network would both take in current input values along with previously generated output values to then apply an adjustment on the incoming audio.

In the end we decided to pursue the Weiner Filter via a direct convolution. This was decided as we were worried about complexity using the subspace method and had sizing concerns about both neural network designs. This eliminated all the options except the two Weiner Filter approaches. After more deliberation it was determined that performing the Fourier Transform would require a sizable portion of memory to be built into our hardware to support it, and we have limited space in the user area to put RAM. Therefore, we concluded the Weiner Filter via direct convolution would be the most feasible to implement.

4.2.3 Decision-Making and Trade-Off

There were many tradeoffs we had to consider and decide on that led us to our final design. One tradeoff decision we made was whether to attempt a single design or several smaller, more modular, designs. Prior teams working on this project have created single designs, a SHA-1 bitcoin miner, spiking neural network, ReRAM implementation, etc. However, when shown the project requirements, we decided that the requirements would be better fulfilled by implementing several mostly independent designs (see Appendix 8.4.2). The goal of our project is to experiment with different designs, so having several different independent designs will provide better resiliency for our test. If one module fails, we have usable results from that module, but we also have the results of all the other modules as well. This will give our project more value as a test than a single-design implementation because if the single design fails, you gain far less information than from the failure of one of several smaller modules.

4.3 Proposed Design

At the beginning of Senior Design 1, in January 2023, we begun discussing potential project ideas with our advisor and client Dr. Duwe. During this time, we decided to create a design that housed multiple modules of different digital functions, to help provide a lower risk of our entire module failing, while enabling a future design team to test more of our chip if one part had failed. After this, we explored different submodule designs, some of which we did not end up carrying through for our design such as an FRAM memory module, a power gating module, and a neural network implementation for the Voice Road Noise Isolation (DSP) module.

Each of these modules were deemed to be either too taxing on space or impossible to manufacture with the open-source fabrication process through eFabless. FRAM required a specialized fabrication process with two extra layers ("FRAM FAQs"), power gating is not supported in the eFabless tooling, and a neural network required too much complexity. In February 2023, we decided on implementing the submodules including the Voice Road Nosie Isolation (DSP) module, a SPI slave interface, clock gating, SkyWater standard cell logic, custom cell logic, and a wishbone bus test. These modules scale on complexity which will allow us to create both an earlier deliverable with some of the modules and reduce overall risk for our final fabricated chip.

During Senior Design 1, in January 2023, we began using the GitHub caravel repository provided by eFabless, which is how we simulate and harden our digital designs. We met with a previous senior design group who is implementing a spiking neural network system, to overview how to simulate a sample adder testbench through the caravel harness. This required us to install WSL-2 on Windows computers and install open-source tools including GTKWave, to view the output simulation waveforms from the sample adder that was given to us by the previous design team.

During March 2023, we began to draw submodule diagrams showcasing how each design could be implemented, such as the Voice Road Nosie Isolation (DSP) module or the Backdoor SPI Interface. This helped us greatly, since it allowed us to find more design questions that we had not yet thought of, and we could come to a conclusion as a group.

4.3.1 Design Visual and Description

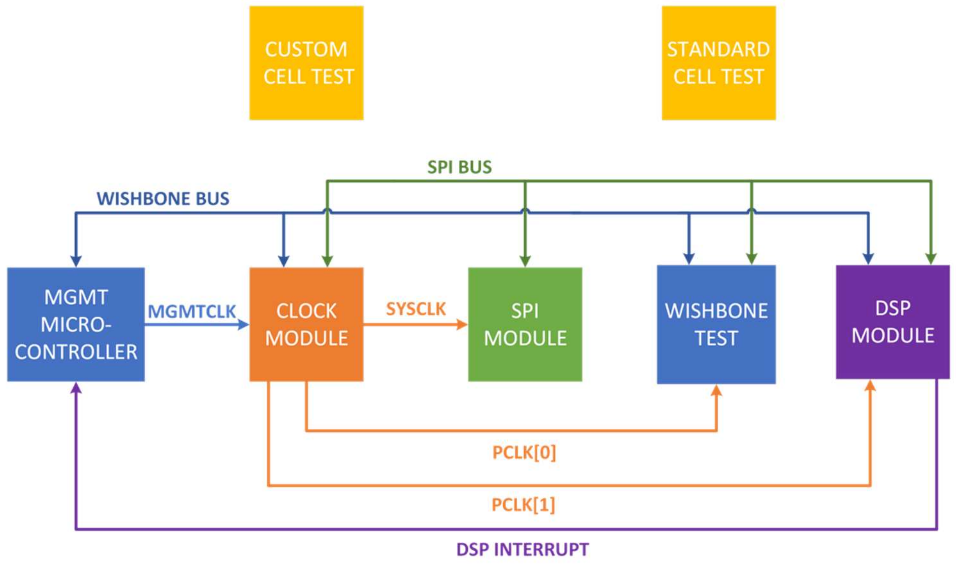


Figure 4 Caravel User Area Module Diagram

The user area diagram in Figure 4 depicts the user space area of our project which is composed of several distinct modular components. The management soc and GPIO blocks on the left side of the above diagram are constraints given to us by efabless where there is a RISC-V processor a wishbone bus and other GPIO as defined by the caravel documentation linked above. The rest of the blocks in the above diagram are the modules that we will be implementing in the project. All the modules are further defined in Appendix A, but to summarize the modules there is the standard cell test which takes an identical logic gate from each of the four standard cell libraries provided by the project and mux them together so that the propagation delay can be measured through the various implementations. The custom cell test, which is currently still being developed but will take two inputs and provide one output. The SPI interface will be a simple bidirectional 4-wire SPI bus on physical pins to the modules we create elsewhere on the device. Finally, the DSP module implements a Weiner Filter to isolate the human voice from background road noise while traveling in a vehicle.

4.3.2 Functionality

The intended operation of our design is to have an ASIC that can be tested by another group to allow them to benchmark some of the capabilities of the manufacturing process. It is designed with the possibilities of certain modules or signals failing and having alternative ways to still be able to test. An example of this is the clock gating module which will have an override so that a clock signal can still pass through if it doesn’t work or having an SPI to interface with the individual modules in case the wishbone bus fails. The results of these tests should allow for better utilization of the open-source process for future users as defined earlier in Section 2.4.

Currently our design should follow all the eFabless rules for submission to allow it to be part of a manufacturing shuttle, but some of the tests will not be able to be run until the full implementation of our design is complete. Our current design also fulfills all the requirements from our advisor and client in the form of the different modules we plan to implement.

4.3.3 Areas of Concern and Development

The primary concern at the moment is how ambitious the Voice Road Nosie Isolation module is to implement in an ASIC. Our user space available in this framework is not very large, so we are concerned about the feasibility of inserting a relatively large convolution into the area available to us. This concern will be fully addressed when we attempt to synthesize our final design before submission and see if the design fits in the user area, but we will be estimating the size of the required multiply and add units prior to full implementation to make design tradeoffs on how the convolution should be implemented. For example, if we find that we can fit 1,000 multiply units in the ASIC user area, then we can do a large convolution with near-single-cycle latency, but if we find that only 10 multiply units will fit, we can design a multi-cycle convolution which takes longer and may not run at real-time, but will still perform an intense test of the ASIC hardware.

4.4 Technology Considerations

Another technological consideration to be made is that our open-source project can only be provided through eFabless, meaning that if we have an issue with our final product we cannot go to any other company or foundry to create our digital ASIC. One upside to this is that both our project and others are required to be open source, meaning that we are contributing to a large library of RTL designs that can be referenced if another company or foundry opened a similar experimental service.

Another item that must be considered is the reliance on open-source software to complete this product. As an open-source product, the software is not guaranteed that it will always be functional. It is purely reliant on the unpaid maintainers to keep it updated and in a functional shape. This will have an impact on our ability to complete and maintain our project.

4.5 Design Analysis

Our project is focused on testing the manufacturing limits of the eFabless platform. We initially debated between creating a single coherent design or several smaller independent modules. We concluded that by using a modular design, we will be better able to test different aspects of the process. We can select which aspects we want to test, and if a manufacturing or design failure occurs in some part of a module, we will still be able to test the other modules independently. This was the main driving factor of why our group went modular instead of creating a singular more complex project that could have more points of failure.

As we researched the Caravel Harness and previous projects, our group was able to brainstorm ideas of possible modules and areas we would like to be able to test. From this we were able to exclude some of the ideas due to hardware limitations (see section 4.3 above) and we were then able to select the modules to include. We decided on six modules, further described in Section 4.1.4, to test aspects of the given Caravel Harness including the Wishbone test, Voice Road Noise Isolation module, a test of the manufacturing capabilities of the standard cell library, and a custom cell test. Lastly, we have both a Backdoor SPI to communicate with modules in case of failures with the Harness, and a clock gating module to be able to turn off the clock signal to any of the modules as clock gating was a significant area Dr. Duwe wished us to experiment with.

4.6 Design Plan

As described above, our design plan will fulfill the requirements for our users defined in Section 4.1.2:

Our Team

We have picked suitably complex modules to challenge us, yet also give us a reasonable chance of success. We will fulfill our requirement of learning through the completion of these modules and attempting to simulate and submit the final design.

Future Senior Design Groups

Our project's design files will be available to future senior design groups to reference, and we will leave a detailed bring-up plan with Dr. Duwe for a future group to use to finish testing our design when it is returned in the future.

Research Teams

Our design files will also be accessible to research teams. Specifically, our clock gating module will be of significant interest to low-power research groups, and our convolution will be of interest to DSP-related research groups, fulfilling our requirements for research teams.

eFabless Open-Source community

As we submit our design, our files will be published to eFabless for the community to reference, fulfilling our requirement to the community as a whole.

5 Testing

In this section, we sought out to create a comprehensive testing plan that would be able to thoroughly test not only our basic designs on their own but also our final integrated product as well. The focus was to create a testing system that would thoroughly verify individual modules as they are written. This will ensure that each module's desired functionality is achieved individually.

From there, once each module has been verified to be functional, it will then be integrated into the larger system. With the modules combined, we will then test that they both retain their own functionality yet also do not harm other integrated modules.

Once we integrate every module, we will focus on the total system together and test that the final test software can run and interact with all the modules. Lastly, we will then submit our design to the eFabless precheck system that will test to make sure that our final design passes all their pre-manufacturing tests. We believe that this testing plan will provide a comprehensive test of our product and give it the best chance of returning from manufacturing fully functional.

5.1 Unit Testing

Each module will have a single test which covers the individual module on its own. For complex modules (DSP Voice Road Noise Isolation), we will have more in-depth testing in each subcomponent including the adders, multipliers, and RAM used in the final design.

These tests will be performed in RTL and gate-level simulation using the OpenROAD tools and written in Verilog test benches.

5.2 Interface Testing

There are two main interfaces that the different modules in our design interact with. Those two would be the integrated wishbone bus along with the backdoor SPI protocol. Our interface testing will verify that each of the applicable modules is able to send and receive data over both bus protocols. We will also verify that each of the modules adheres to the bus protocol strictly.

These tests will again be performed by RTL and gate-level simulations using the OpenROAD tools written in Verilog test benches.

5.3 Integration Testing

The critical integration paths in our design are the Wishbone bus, SPI bus, and GPIO usage. We will ensure that each module has an independent address space which does not collide and run each of the per-module tests on this integrated design to ensure nothing broke during integration and each module can be accessed independently.

5.4 System Testing

For full integration testing, we will test all modules using a large test bench for the overall design after final integration to ensure each part is working as we expect. We also have the ability to simulate full C code (although it is super slow), so we have the ability to run our final test code to ensure everything should function correctly when fabricated and returned to a future tester. This will also test the clock gating to ensure that it is theoretically going to function with the overall design as we expect.

5.5 Regression Testing

Due to the nature of testbench driven development in Verilog, we will be able to continue running past test benches on previously tested components to ensure functionality does not break as we seek to implement more complex functionality.

5.6 Acceptance Testing

We will demonstrate that our design requirements are met by verifying the expected results from our register transfer level and gate level simulations match the output of each testbench we have designed. Each testbench will be designed to verify every functional requirement of the submodule will be satisfied, while ensuring the submodule will not interfere with other designs in our user area. We will also run our final user area wrapper against the eFabless precheck that is built into the provided GitHub repository that we are working out of.

5.7 Security Testing

None of our submodule designs will be primarily security focused. As a development test of the capabilities of eFabless, we are purposefully adding a backdoor SPI into the user area and are attempting to ensure that we can inspect as much of the design as possible. From that perspective, our design should be as minimally security focused as possible. However, there are security risks from other aspects of our project. The tools we use are open source, so there are security considerations that must be made there, but these are common and well-used programs and are not a major concern. here will be some security risk from the fact that we will not have clear documentation on the fabrication process that is being completed at the SkyWater foundry. Due to this, more rigorous testing should be done post fabrication with the interfacing, including GPIO and Wishbone tests. If this does prove to be an issue, we can utilize our own Backdoor SPI module to bypass the provided serial bus interface.

5.8 Results

As of now, we have successfully implemented and verified multiple RTL simulations across different submodule designs. We were able to verify the waveforms in GTKWave for the sample adder that was given to us by one of the previous senior design groups, to ensure our Caravel repository was running properly for RTL simulations. We have also succeeded in generating testbenches and waveforms for our own submodule designs, specifically the standard cell test and shift registers which will be integrated into the Backdoor SPI module. To verify the results, we compared the expected results to the actual resulting outputs using the waveform viewer GTKWave. These detailed results can be seen below in Appendix 8.4.3.

6 Implementation

6.1 Standard Cell Test

Current Status

We have looked at the cells available to us and the different categories they are part of, such as high density or low latency. As part of the work on the SPI a 2 to 1 mux from the standard cell library has been used and is part of Appendix 8.4.3,

Plan for Next Semester

Write the Verilog for the four and gates that are going to be tested and then verify through simulation that is working as intended.

6.2 Custom Cell Test

Current Status

We have found an initial guide from another contributor who submitted a custom logic cell in a previous MPW submission. We have read through the guide, and this appears feasible to implement, but no implementation progress has been made.

Plan for Next Semester

We will use the example guide as a reference to learn the tooling, then explore what is possible within these tools. Two possible ideas are a low-voltage retaining flip flop and a complex logic gate such as an AOI. The number of pins used is flexible on this module, but should be kept small. This will likely be tested using a Spice simulation.

6.3 Wishbone Test

Current Status

The Wishbone Bus test has been started, but the implementation has not been completed. We have read through documentation on how the Wishbone Bus works, and have run the provided testbenches which utilize the Wishbone Bus.

Plan for Next Semester

We will use the existing testbenches and assorted documentation to finish the implementation of a Wishbone Bus test and test it to ensure the module works as expected by writing C code for the harness MCU to be simulated in the tooling.

6.4 Clock Gating

Current Status

A design of the clock gating system with an override has been completed and has started to be implemented.

Plan for Next Semester

Finish the implementation and thoroughly test it including the override to make sure that everything works as to prevent this module from disallowing the clock signal to propagate through it in the case of failure.

6.5 Backdoor SPI

Current Status

Near the end of the first semester, the module design and verification for the shift in and shift out registers of the Backdoor SPI module have been completed. The test results for the shift in and shift out registers can be seen in Appendix 8.4.3. The block diagram and schematic for the Backdoor SPI module have also been developed. A robust description of the Backdoor SPI module is referenced below in Appendix 8.4.4, describing design decisions surrounding the shift registers, clock synchronization, and the external master SPI interface.

Plan for Next Semester

Next semester, the plan is to complete the Verilog for the Backdoor SPI module and verify it with RTL simulations, similar to the shift in and shift out registers. Since the shift register modules are already written and tested, there will be no other required submodules inside of the Backdoor SPI module. After this, gate level simulations will be conducted for each of the three separate modules, starting with the shift registers. When all tests have passed, the Backdoor SPI module will be ready for integration testing with the other modules.

6.6 Voice Road Noise Isolation (DSP)

Current Status

The Voice Road Noise Isolation module has recently been fully defined. The module has had the final algorithm selected to be implemented. The module's internal components have also been defined. With both of these defined the module's functionality has also been determined.

Plan for Next Semester

For the upcoming semester there is much that is needed to be completed for this module. The first item will be to write the Verilog implementing each of the subcomponents of the module. From there the subcomponents will need to be tested and verified to be functionally complete. From there the submodules can be compiled into a system together to implement the desired functionality of the system. With the overall module completed the next step will be to write the functional tests for the module. When the tests have all passed and the module is determined to be functional it can then be hardened. From there the last step is to test the module's gate level behavior through more tests.

6.7 Submission

Plan for Next Semester

Once all the modules have completed their individual tests the integration process can begin. Each of the modules will be compiled into one large system and routed together. With the fully integrated design complete the top-level system tests can be written and applied to the design. Once the design has been tested and verified to fulfil the requirements of the project the design can be run through the eFabless precheck process. Once the design passes the precheck process it can be submitted to the most recent Open MPW Shuttle.

6.8 Bring-up Plan

Plan for Next Semester

Once the design has been submitted for manufacturing, work can begin on compiling the firmware to run on the device along with the bring up plan to fully test the manufactured chip. The firmware will need to be written such that major changes will not be needed to be able to run on the returned chip. The bring up plan will need to be written such that individuals with little knowledge of digital design or the implemented design will be able to get the chip to function.

7 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

7.1 Areas of Responsibility

Below is a list of the responsibility areas for both the NSPE and IEEE Canon, outlining how each Canon defines their code of ethics. Below, we outline how the IEEE Canon code of ethics differs from the NSPE Canon.

Table 3 Areas of Responsibility

Area of responsibility	Definition	NSPE Canon	IEEE Canon
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	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;
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees	To reject bribery in all its forms;
Communication Honesty	Report work, truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	To be honest and realistic in stating claims or estimates based on available data;
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public	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 property, ideas, and information	Act for each employer or client	To avoid injuring others, their property, reputation, or

	of clients and others.	as faithful agents or trustees	employment by false or malicious action;
Sustainability	Protect environment and natural resources locally and globally.		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, ethically, and lawfully to enhance the honor, reputation, and usefulness of the profession	To treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression;

Work Competence

- IEEE states that we must only take on work that we are trained to do and qualified for and must disclose when we may not be able to be up to the necessary standards
- The IEEE code covers the Work Competence section by ensuring that we know how to properly preform the job we set out to do
- This differs from the NSPE cannon as the IEEE does not cover deceptive acts in this portion and does not cover malicious intentions here

Financial Responsibility

- IEEE states that you should not accept any form of bribe on the job
- This code covers Financial Responsibility by guaranteeing that engineers will not accept any financial gain at the expense of their employer or wellbeing of the public
- The NSPE cannon does not specifically state to reject bribes, and alludes more towards acting in good faith when handling finances for a project

Communication Honesty

- IEEE states that we must be honest and accurately describe situations and problems based on the information that we have available
- The IEEE code covers the Communication Honesty standard as it strives to provide honest information to those that need it based on in information that they have available themselves
- The NSPE cannon differs from the IEEE standards here as it does not require you to use all the information available to you when communicating with others

Health, Safety, Well-Being

- IEEE addresses that engineers must take responsibility in decisions regarding the safety, health, and welfare of the public.
- This IEEE code relates to the Health, Safety, and Well-being responsibility by ensuring engineers will make decisions of good faith with the public's best interests in mind
- The NSPE Code has a very similar statement, but lacks the statement to disclose any factors that may endanger the public, like the IEEE code states

Property Ownership

- IEEE addresses avoiding malicious action that could harm property or reputation
- In NSPE, they specifically mention acting as a faithful agent as opposed to IEEE which is more against malicious behavior. The difference in this could be interpreted as doing the best that is possible as compared to just not doing something harmful.

Sustainability

- The IEEE addresses preventing harm to the environment and a responsibility to disclose any factors that could pose a threat to the environment.
- In IEEE as compared to NSPE, they combine all health and safety factors of the public into one point as a responsibility to protect safety health and welfare. NSPE does not have any mention in the table for sustainability

Social Responsibility

- The IEEE code calls to treat everyone equally without regard to their background or other personal factors
- The IEE code covers the Social Responsibility portion as it calls for everyone to be treated equally and for the benefit for all
- The NSPE differs from the IEEE code as it calls for the person to enhance the honor of engineering professions

7.2 Project Specific Professional Responsibility Areas

Below, we described how each of the referenced professional responsibility areas apply to our project in a professional context. Each responsibility area is scored on a rating from LOW, MEDIUM, to HIGH, with a HIGH rating describing that our team is performing well in the respective responsibility area.

Work Competence

- Our team is performing well in the work competence context. We are working to learn the process that our chip fabrication will use, and we are not claiming any knowledge that we do not possess.

- We are also doing a lot of research into the resources the eFabless process gives us access to so that we can become more qualified or ask other groups that have done this process that have more experience than us.
- Overall, our team is performing well in this area and thus rates our proficiency as HIGH.

Financial Responsibility

- Financial Responsibility is very applicable to our project and team, due to our project being open-source and free to submit.
- This means that we should not accept any financial gain or bribes, like both the NSPE and IEEE canon warn against.
- Since we have not accepted any money and are using the open-source tools provided for our project, we are performing with a HIGH proficiency.

Communication Honesty

- Overall, our team is performing high with communication honesty responsibility. This responsibility is described as reasonable and realistic in the estimates of our project given the total data.
- We have access to a Slack channel with different teams and collaborators that communicate when the Open MPW Shuttle releases occur, which in turn could dictate when we would receive our final product in the future.
- This is important so that a future senior design team could bring up and test our design once the final product is shipped back.
- In the meantime, we are working with Dr. Duwe to provide timely updates and a realistic scope of what we are interested in and capable of.
- Overall, our team is preforming well in this area and thus rate our proficiency as HIGH

Health, Safety, Well-Being

- Since all designs will be kept open source, there is less risk to the general safety of the public.
- We are improving the area of digital electronics by silicon proving open-source designs, that can itself go into improving lives.
- Overall, our team is performing adequate in this area, and thus earns a proficiency rating of MEDIUM

Property Ownership

- Project ownership is an integral part of this project as a whole.
- As it is an open-source project anyone can use and modify our designs for the betterment of digital design.
- Our project only requires that those who wish to use and adapt our designs keep their iterations open source as well.
- This will create an ecosystem that encourages the sharing of ideas without them getting locked behind some company's proprietary IP.
- Overall, our team is performing well in this area and thus rates our proficiency as HIGH.

Sustainability

- The company that completes the fabrication, Skywater, is in the United States so there are more environmental regulations that ensure they are following the Sustainability ethics than compared to some other countries.
- Like the Health and Well Being responsibility area, our digital ASIC design could be seen as more sustainable due to the open-source nature of the project, enabling other design teams to pick up on where we started.
- Due to the project using open-source tools and being public online, we would rate our Sustainability score as MEDIUM

Social Responsibility

- This project has a great impact in the realm of social responsibility.
- Our project utilizes a manufacturer in the United States that emphasizes working conditions for their workers.
- This project is in line with the idea of creating better working standards for all and this is an important factor as to why we chose this manufacturing process.
- Overall, our team is performing ok in this area and thus rates our proficiency as HIGH.

7.3 Most Applicable Professional Responsibility Area

One area that is both important and that we have been very proficient in is the responsibility of property ownership. One of the most appealing factors of our project is that we will submit our digital ASIC design to the Open MPW Shuttle submission, which will require us to submit our project as open source. Being open source, our full design will be open for the public to view, edit, and use for their own purposes. We also intend for our project to be used with future senior design group(s), with a focus on bringing up the chip by implementing a tester circuit board and embedded code. We have been working on our design with the fact in mind that it will be used by others at a later date.

8 Closing Material

8.1 Discussion

As of the end of our first semester in senior design, we have successfully verified RTL simulations by viewing waveform outputs for a sample adder, SkyWater standard 2x1 multiplexer cell, and a pair of shift in and shift out registers. All waveform results can be seen below in Appendix 8.4.3. While the sample adder will not be included in our final design, it was useful in verifying that the RTL simulations and GTKWave functioned as expected which helped us ensure we could use the provided open-source tools that are required by eFabless. The SkyWater standard cell test will be included in our final design, alongside the shift registers which will be implemented into the Backdoor SPI module next semester. For more information on the Backdoor SPI module, refer to Appendix 8.4.4.

8.2 Conclusion

For our project, we are creating a modular digital ASIC design using open-source tooling and an OpenMPW Shuttle submission through eFabless. To confirm we meet our functional requirements for each module, we have planned to include risk mitigation considerations in each modular design. Through the work of an external SPI interface, clock gating, and a wishbone test, we will ensure that as much of the chip will come back functional after fabrication at the SkyWater foundry. During the second semester of senior design, we will focus on designing, verifying, and integrating each submodule in the user area together into one functional ASIC. During this time, we will also work to develop a bring up plan for a future senior design group to use our chip, after it is delivered to us from eFabless and the SkyWater foundry.

8.3 References

Technical References:

"Caravel user project," *Caravel User Project - CIIC Harness documentation*. [Online]. Available: https://caravel-user-project.readthedocs.io/en/latest/. [Accessed: 20-Apr-2023].

Efabless, "Efabless/caravel_user_project," *GitHub*. [Online]. Available: https://github.com/efabless/caravel_user_project/blob/main/docs/source/index.rst#user-project-wrapper-requirements. [Accessed: 20-Apr-2023].

Efabless, "Efabless/mpw_precheck," *GitHub*. [Online]. Available: https://github.com/efabless/mpw_precheck. [Accessed: 20-Apr-2023].

"Open MPW Shuttle Program," *Efabless*. [Online]. Available: https://platform.efabless.com/shuttles/MPW-8. [Accessed: 20-Apr-2023].

"FRAM FAQs," *Texas Instruments.* [Online]. Available: https://www.ti.com/lit/wp/slat151/slat151.pdf. [Accessed: 23-Apr-2023].

"How much does it cost to have a custom ASIC made?" *Electrical Engineering – Stack Exchange.* [Online]. Available: https://electronics.stackexchange.com/questions/7042/how-much-does-it-cost-to-have-a-custom-asic-made. [Accessed: 2-May-2023].

eFabless. [Online]. Available: https://efabless.com. [Accessed: 2-May-2023].

"TSMC MPW Shared Tapeouts." *MUSE Semiconductor.* [Online]. Available: https://www.musesemi.com/shared-block-tapeout-pricing. [Accessed: 2-May-2023].

Related Work:

A. Petersen, J. Thater, M. Ottersen, and R. Dukele, "Senior Design Team sddec23-08 • RERAM compute asic fabrication," *Iowa State University ECpE Senior Design*. [Online]. Available: http://sddec23-08.sd.ece.iastate.edu/. [Accessed: 20-Apr-2023].

C. Mantas, S. Szabo, C. Violett, and D. Ghauri, "Senior Design Team sdmay22-17 • Digital Chip Fabrication," *Iowa State University ECpE Senior Design*. [Online]. Available: http://sddec22-17.sd.ece.iastate.edu/. [Accessed: 20-Apr-2023].

"MPW-1 shuttle projects," *Efabless*. [Online]. Available: https://platform.efabless.com/projects/shuttle/1. [Accessed: 20-Apr-2023].

T. Green, A. Sledge, K. Gisi, F. Zhu, and W. Zogg, "Senior Design Team sdmay23-28 • Digital Chip Fabrication," *Iowa State University ECpE Senior Design*. [Online]. Available: http://sdmay23-28.sd.ece.iastate.edu/. [Accessed: 20-Apr-2023].

8.4 Appendices

8.4.1 Team Contract

Team Name sddec-06

Team Members:

1) Gregory Ling	2) Cade Breeding
3) Will Galles	4) Jake Hafele

Team Procedures

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

Face to Face in Coover TLA on weekends. Once a week for an hour with our client (Dr. Duwe).

Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

Everything should be communicated through our shared Teams page with our client and advisor Dr. Duwe

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

We will reach a consensus before major decisions because this affects everyone in the group.

Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

The scribe role will keep minutes and it will be saved in a document in a tab in teams weekly

Participation Expectations

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

We will all come to all meetings on time unless notified previously.

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

Each week, we will delegate different tasks for each member, and each member will be responsible for completing their tasks.

Expected level of communication with other team members:

We expect everyone will be responsive on MS Teams, as that will be our main method of communication between our team and Dr. Duwe.

Expected level of commitment to team decisions and tasks:

Each team member will take on a portion of the work best suited to their abilities and everyone would be engaged in directing the project and making decisions

Leadership

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

Researcher, Clock Lead – Cade

Client Point of Contact, Custom Cell Lead – Gregory

Scribe, SPI Lead – Jake

Researcher, DSP Lead – Will

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

We will have 2 weekly meetings, one to meet with our client and update him on our progress, and another to work together and ensure our jobs are completed.

Strategies for recognizing the contributions of all team members:

All contributions will be recorded on the team's Kanban board on MS Teams.

Collaboration and Inclusion

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

Will Galles - Embedded Systems Design, Digital Logic Design, C, VHDL, FPGA design and implementation, Digital logic test and analysis.

Jake Hafele – PCB Design, PCB Testing, Digital Logic Design, VHDL, Verilog, FPGA Design, Git, C

Gregory Ling - VHDL, C/C++, Verilog, experience using FPGAs in research scenarios and working with Dr. Duwe for other projects

Cade Breeding - VHDL, C/C++, Git/Source Control Management, Verilog

Strategies for encouraging and support contributions and ideas from all team members:

Our group will meet once a week separately from the client to discuss our progress and work together on the project. We will welcome contributions and ideas from all team members and consider everyone's ideas.

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?)

Ideally the group member feels comfortable enough to bring it up to the group either through teams or in person and we can start a discussion about it. If the issue is with another individual and they are uncomfortable addressing, it with the group then they can bring it up with the scrum master who can then address it with the team anonymously.

Goal-Setting, Planning, and Execution

Team goals for this semester:

- Prepare our digital design to be ready for the Open MPW Shuttle submission (estimated for Summer 2023 as of now)
- Learn more about the digital ASIC design process through the Efabless tooling
- Test the design that was submitted and manufactured from the Dec 2022 Senior Design group

Strategies for planning and assigning individual and teamwork:

Create a Kanban board in Teams that is open to all team members and our advisor/client

Strategies for keeping on task:

Set a concrete list of goals to complete when we hold our weekly meeting with our advisor/client, Dr. Duwe

Consequences for Not Adhering to Team Contract

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

It will initially be brought to the team members' attention to allow them a chance to make up any work that they are behind on and build a plan with the te8.4.3.am for how they will get to that point.

What will your team do if the infractions continue?

Bring up the issue with the respective team members to our advisor, to get their opinion on how it should be handled.

****

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) Jake Hafele	DATE 2/19/2023
2) Gregory Ling	DATE 2/19/2023
3) Cade Breeding	DATE 2/19/2023
4) Will Galles	DATE 2/19/2023

8.4.2 Module Design Specification

Backdoor SPI Interface

Primary Author: Jake Hafele

Inputs

- i_SYSCLK – System Clock
- i_BCLK – Bus Clock from Master
- i_SS – Bus Slave Select (Active-high reset)
- i_MOSI – Bus Master Out Slave In
- i_DATA_OUT[31:0] – Data to be sent to master SPI, loaded from other user modules

Outputs

- o_MISO – Bus Master In Slave Out
- o_ADDR[6:0] – Address to determine what the data is for
- REGISTER[6:3] - What data do you want from each module?
- MODULE[2:0] - Which module are you talking to?
- o_DATA_IN[31:0]: output data read from master SPI module, read by user modules
- o_DOUT_VALID: Flag that is asserted when DATA_OUT is ready to be read by submodules

Internal

- s_READ: Are we reading (1) or writing (0) to slave SPI? Read from ADDR[7] after shifted from MOSI

Basic Components

- One parallel-in serial-out shift register for MISO
- Two parallel-out serial-in shift register for MOSI
- Six D flip flops used to hold flags to address clock synchronization
- Basic combinational gates including 2 input AND, 3 input AND, and NOT gates

Implementation Description

The backdoor SPI interface will be a simple bidirectional 4-wire SPI bus on physical pins to the modules we create elsewhere on the device. The SS pin will reset the SPI module when high. The first 7 bits of transfer data from i_MOSI will be interpreted as the address from which to read and write, with the following 8th bit for the read/write identification, s_READ. At that time, s_ADDR will be set to the correct value to drive logic within the modules. It is expected that s_ADDR will drive a large multiplexer between all peripheral values into the i_DATA_OUT field which must be stable within 1 system clock cycle of ADDR or READ changing. The output of that mux will be sent in the following 32 bits to the master SPI module regardless of the s_READ bit. If s_READ is 0, the mux will be ignored, and the next 32 bits will be stored in DATA_IN. The o_DOUT_VALID flag will be high for one system clock cycle at the end of transfer during which time the data is guaranteed to be stable and the addressed module should act on the provided data.

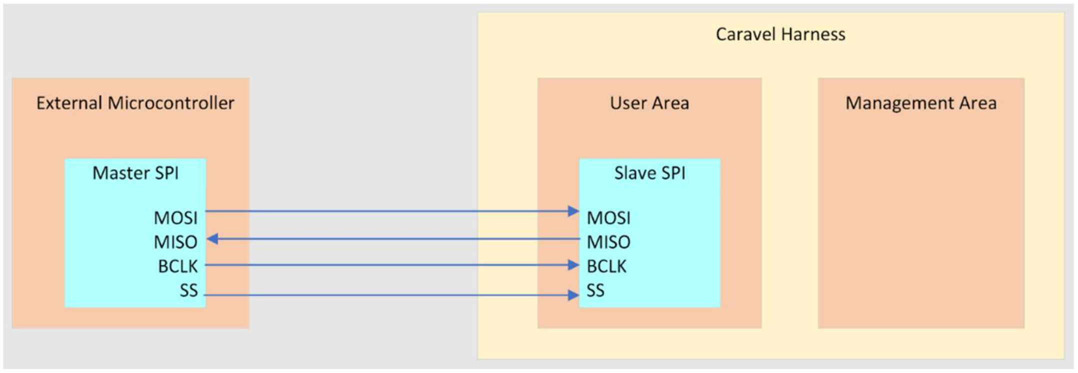


Figure 5 External SPI Connection

Voice Road Noise Isolation (DSP)

Primary Author: Will Galles

Primary Reference: Issaac Rex's EE 529 Speech Enhancement Project

Inputs

- PCLK0 – Clock input from clock gate
- Wishbone Bus
- Backdoor SPI

Outputs

- Wishbone Bus
- Backdoor SPI

Basic Components

- Convolution control unit
- Address up counter
- Address down counter
- N 16 bit multipliers
- 32 bit accumulator

Implementation Description

The Voice Road Noise Isolation (DSP) module will implement a Weiner Filter to clean a noisy audio signal to improve the underlying voice audio. This filter works by first creating a tuned filter that rejects signal frequencies that are not typically found in the human voice while letting all others pass through. This filter would then be applied to the incoming voice audio to perform the voice isolation. There are two main ways to go about applying the filter to the incoming data. The first method would be to first convert the filter to the time domain and then directly convolve the filter with the incoming data. The second method would be to transform some window of your incoming data into the frequency domain and then multiply it with the filter in the frequency domain before performing the inverse transform to convert the data back to the time domain.

In the end we selected to pursue the direct convolution route as a way to cut down on the hardware that we needed to implement. This allowed us to forgo the hardware to perform the Fourier Transform as we could convert the filter to the time domain off chip. With a final approach now decided we could focus on the implantation.

The module first begins with initialization where the filter is first loaded into the filter sram in the user area. Next, we load the first data values in to completely fill our data sram with junk data. Once both memories have been filled the initialization is complete and the module can begin to operate. In the normal operation mode one cycle begins with one new data point coming in on the wishbone bus. This new data point is loaded into the data memory at the location of the oldest previous value. Once it has been loaded the convolution can begin. The module begins by pulling the oldest data value and the last filter value and multiplies them together. The product is then fed into the accumulator unit and then added to the previous sum. From there the second lowest data value and the second to last filter values are then run through the same process. This continues until we reach the newest data value and the first filter value. When their product is fed into the accumulator the output sum is then sent back on the wishbone bus as our new output value. The accumulator is now reset and then a flag is raised to let the processor know that it can send over a new input value to the module. This process then continues indefinitely providing cleaned voice audio data.

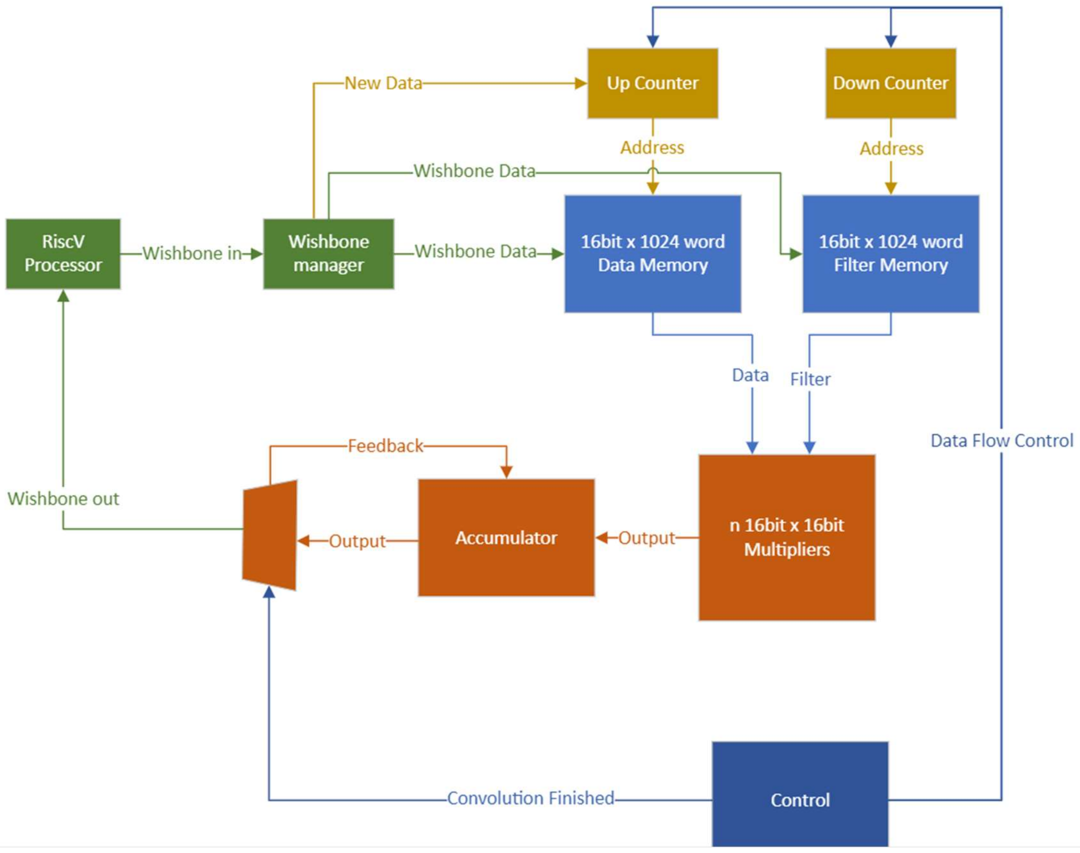


Figure 6 Voice Road Noise Component Diagram

Clock Gating Module

Primary Author: Cade Breeding

Inputs

- HARNESS_CLK – Harness Clock
- EXTCLK – External Clock
- Hardware gate override pin
- Hardware clock override pin
- Wishbone Bus
- Backdoor SPI

Outputs

- SYSCLK – System clock for all devices except wishbone bus (SPI bus)
- Wishbone Bus
- Backdoor SPI

- PCLK[2:0] – Individual Module Clock

Basic Components

- 3-bit register for downstream module control
- Clock Gate for each downstream module
- Single Clock Mux between two input clocks

Implementation Description

Clock gating – Every PCLK has a bit in a register (0 = enabled, 1 = disabled), defaults on, hardware pin override to turn all clocks on, accessible over SPI and wishbone. Every module clock comes from the clock module. One (or two) input clocks available over hardware pins. PCLK2 is external.

The two input clocks (from harness and external) will be sent to a clock mux per peripheral clock controlled by a bit in a register and sent through a clock gate for each module also controlled by another register bit per module and out to the modules. If the hardware gate override pin is set, the clock gates will all turn on, bypassing the control bits and the clock mux control line will be overridden by the clock override pin

Standard Cell Test

Primary Author: Cade Breeding

Inputs

- A – Input 1 to standard cell AND gate
- B – Input 2 to standard cell AND gate
- SW[1:0] - Select line for 4:1 MUX

Outputs

- C – MUX'd output of one standard cell AND gate

Basic Components

- AND gate cell from four of the different Skywater standard cell libraries (inlcuding hd, sd, md, and hdll)
- 4:1 MUX from high density standard cell library
- Implementation Description
- Standard cell test – Test propagation delay and different libraries (hd, sd, md, hdll). 4 inputs A and B, SW[1:0], 4 AND gates from different modules share inputs and output to 4:1 mux and output to 1 pin. All pins are hardware, no registers and no SPI.

Custom Cell Test

Primary Author: Gregory Ling

Inputs

- Two inputs A and B

Outputs

- One output C

Basic Components

- One custom logic cell of our choice. Can also include a clock signal if that would be interesting.
- Implementation Description
- Three pins, two input, one output, does something with those pins, some custom 2:1 lookup table or the like. Exact specification is yet to be determined as we explore what might be possible.

Wishbone Test

Primary Author: Gregory Ling

Inputs

- PCLK1 – System Clock
- Wishbone Bus
- Backdoor SPI
- Outputs
- Wishbone Bus
- Backdoor SPI

Basic Components

- 32-bit incrementor

Implementation Description

When the wishbone bus is written, counter value is set. Counter is continuously counting PCLK1 pulses, discarding overflow. When the wishbone bus is read, the current counter value is sent.

8.4.3 Testing Results

Shift In Register

The following waveforms show the testbench results for the successful shift in operations from the previous shift in register testbench. As stated before, tests were conducted shifting in values of decimal 100, 256, 10498, and hexadecimal 0x00000000 and 0xFFFFFFFF. The expanded bit vector of o_Q demonstrates how the least significant bit is reset to 1 and propagates through the output as the enable indicator. The other propagated bits are shifted in as the shift_in task begins to shift the data contents from the I_D input, which would be the MOSI signal to the backdoor SPI module.

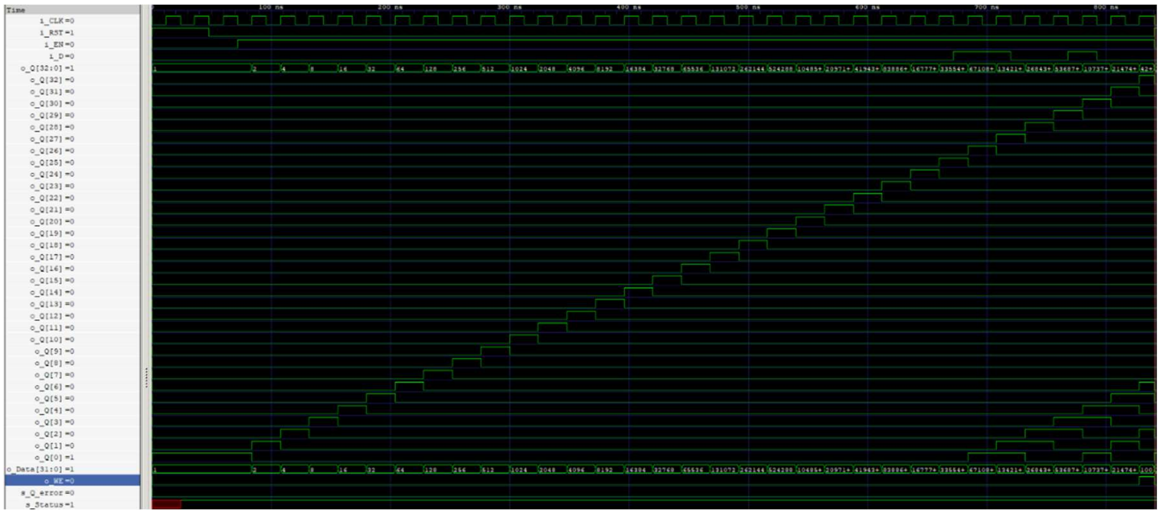


Figure 7 Shift in 100

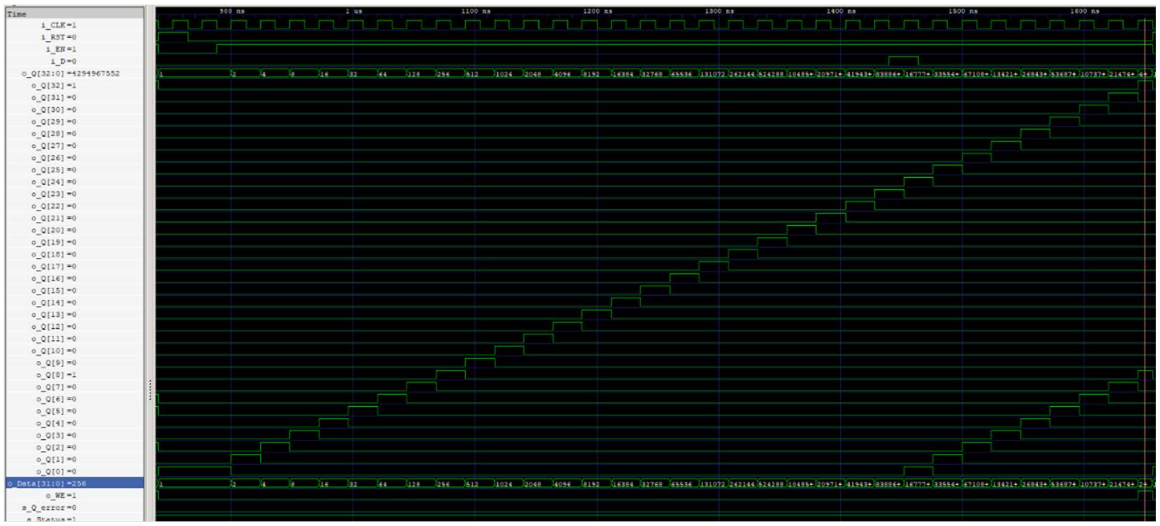


Figure 8 Shift in 256

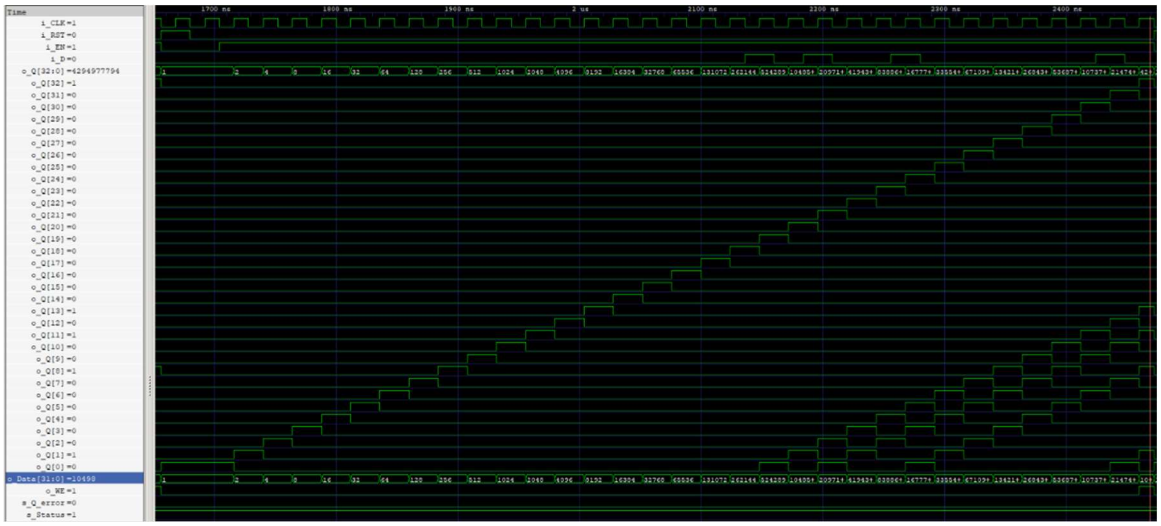


Figure 9 Shift in 10498

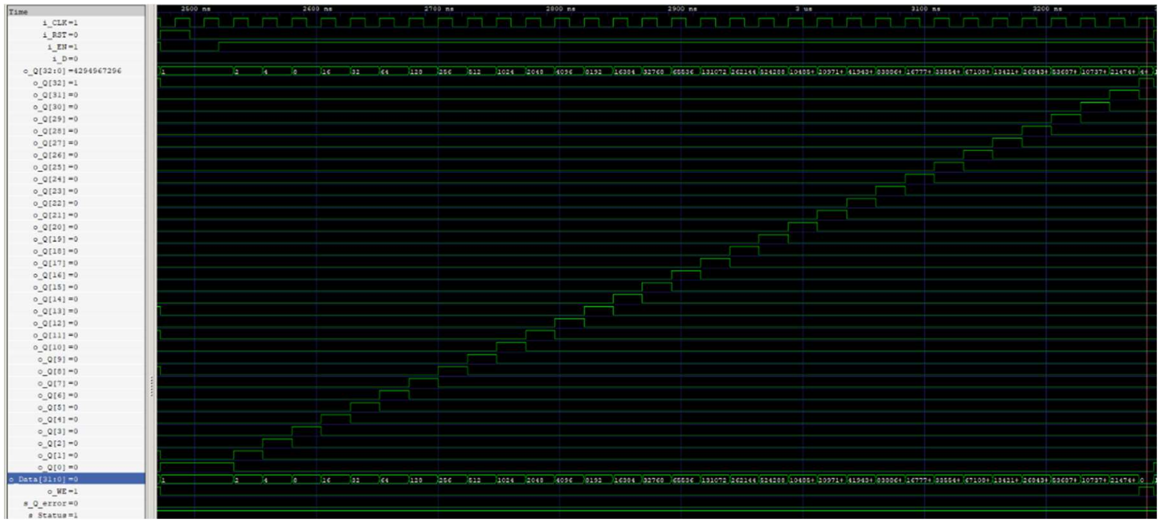


Figure 10 Shift in 0x0

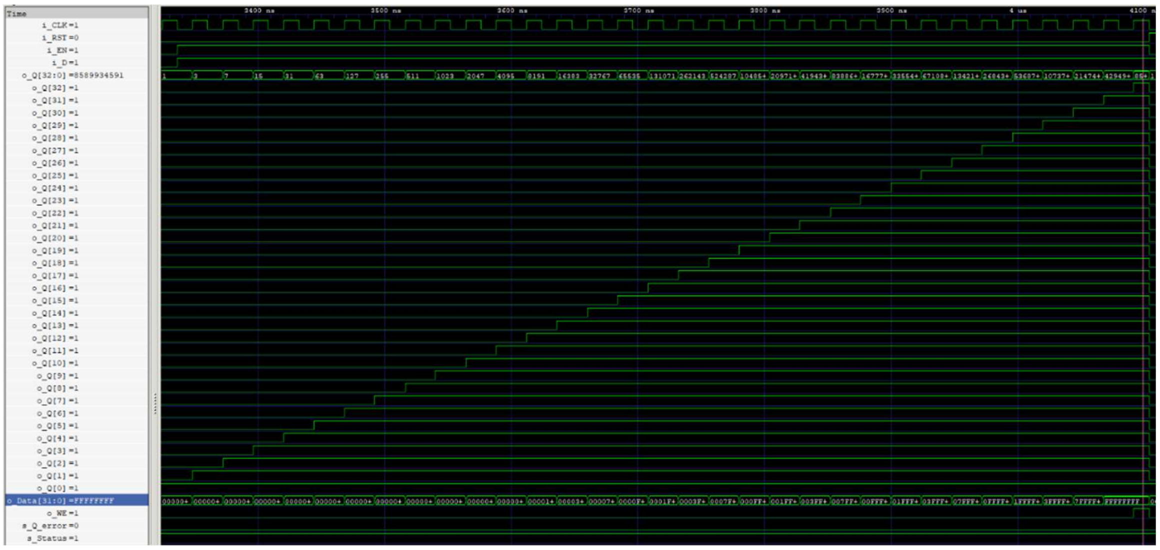


Figure 11 Shift in 0xFFFFFFFF

The second set of tests that were verified with the shift in register were the cases where the enable input was 0, to verify that no data from i_D would be shifted in. The following waveform verifies that no bits were shifted into the o_Q output register for any of the following 32 clock cycles that the enable pin was cleared to zero.

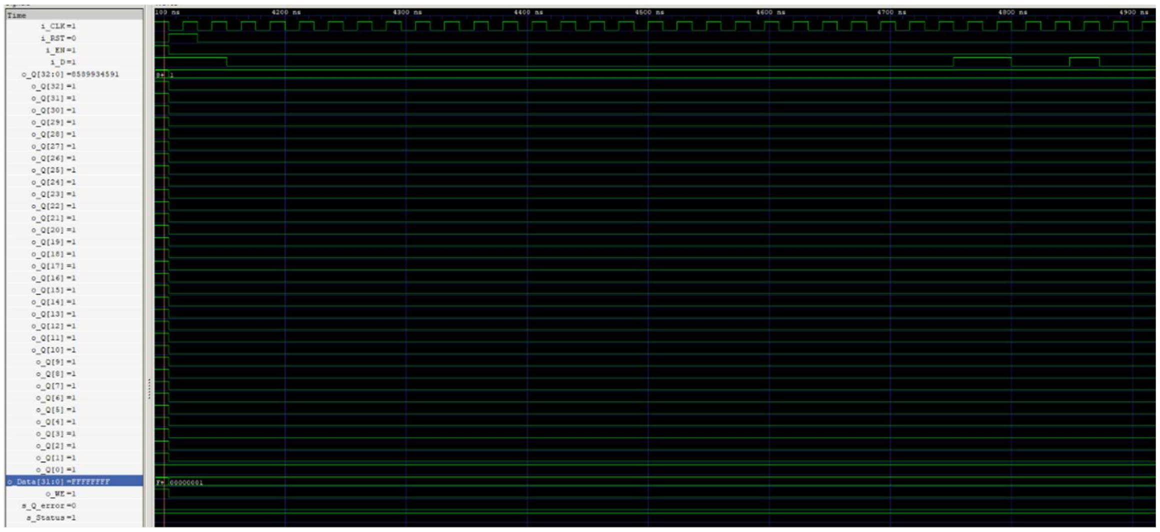


Figure 12 Shift in with enable low

Shift Out Register

The following set of waveforms depicts multiple shift out verifications with the shift out register testbench. Since the 32 bits of the i_D signal would be shifted out continuously after I_START was asserted, there were no cases to check with an enable signal during the middle of the shifting process. The following waveform verifies that i_D is properly shifted out of the serial output o_Q for values 0x64, 0x100, 0x2902, 0x0, and 0xFFFFFFFF.

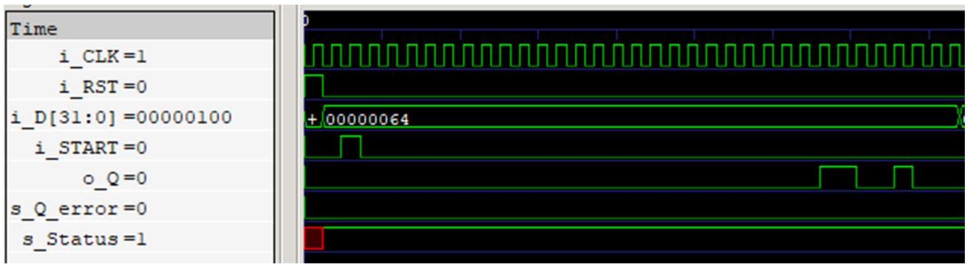


Figure 13 Shift out 0x64

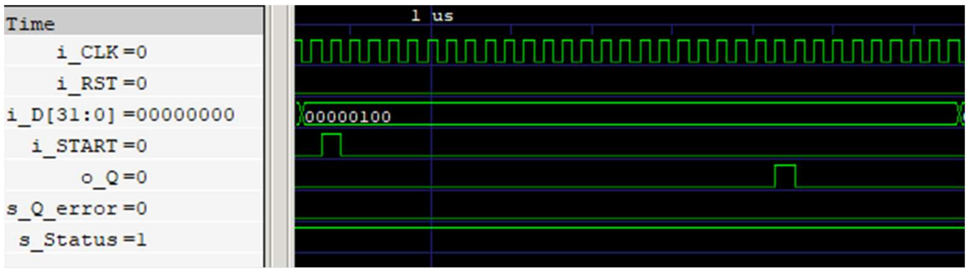


Figure 14 Shift out 0x100

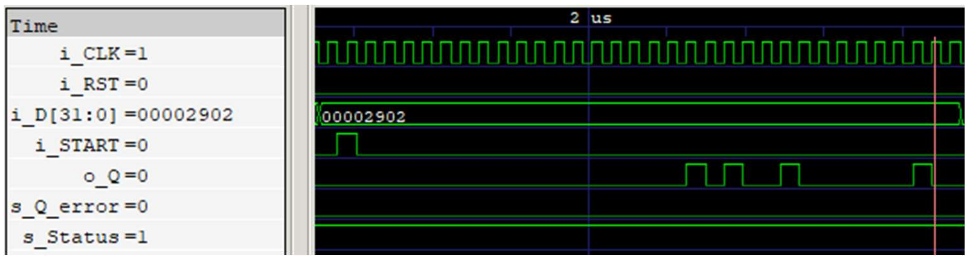


Figure 15 Shift out 0x2902

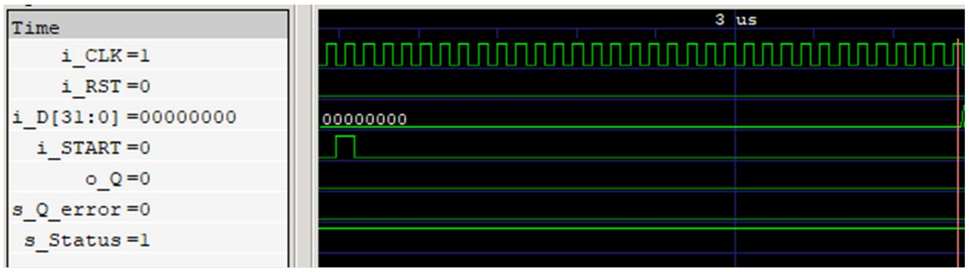


Figure 16 Shift out 0x0

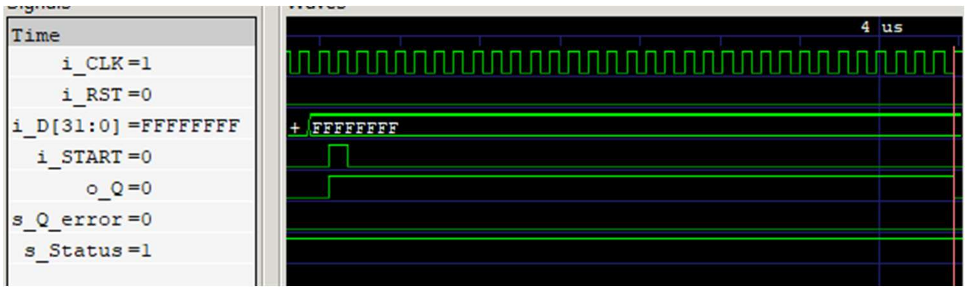


Figure 17 Shift out 0xFFFFFFFF

Adder

In the following results, the signals prev_result and wb_data_reg are added together and are output on the signal rdata. This adder module was given to us by last semester's design team to be used as an initial test to verify our open-source tool flow would run and generate a waveform output.

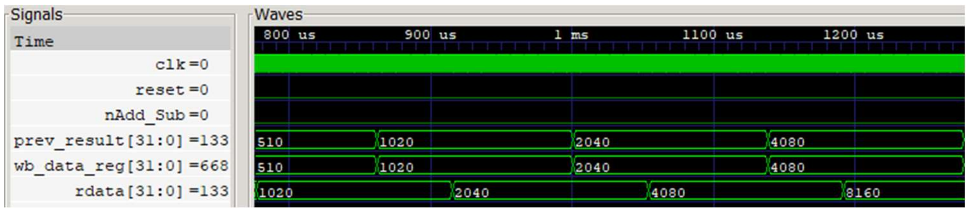


Figure 18 Adder Results

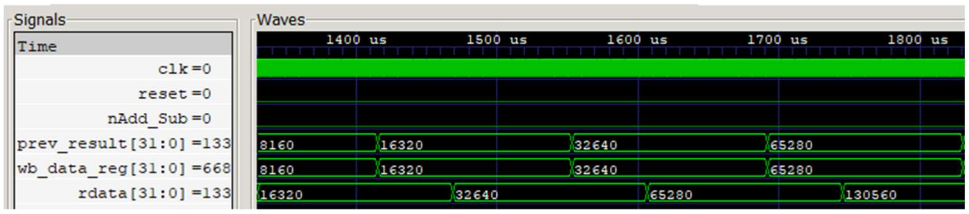


Figure 19 More Adder Results

Standard Cell Test: 2x1 MUX

The following test results describe the waveform for a SkyWater standard 2x1 MUX cell. The inputs A0 and A1 are the data lines, with S being the select line. When S is 0, A0 is routed to the output X. When S is 1, A1 is routed to output X. Due to the combinational nature of the cell, each of the 8 possible outputs are given below with each possible 3 input combination. The signal OK is used for error checking to verify the output X is correct after each combination, verifying the test and standard cell functioned as expected.

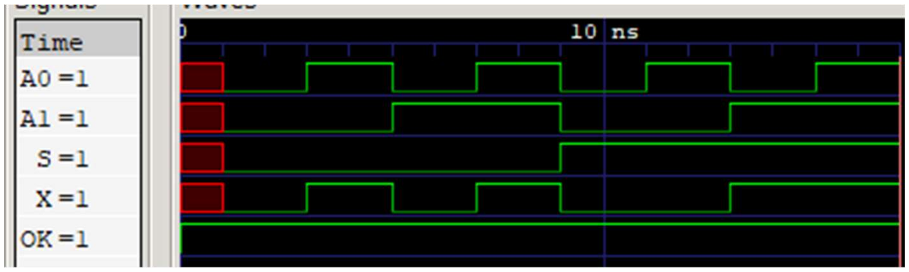


Figure 20 Standard Cell 2to1 MUX Waveforms

8.4.4 Detailed SPI Module Implementation

SPI I/O Interface:

- **MOSI**: Master out Slave In, sends serial data FROM master TO slave
- **MISO**: Master in Slave Out, sends serial data TO master FROM slave
- **BCLK**: Bus Clock, driven from master SPI module
- **SS**: Slave Select, line for master to select which slave to send data to

Serial Peripheral Interface (SPI) is a synchronous interface, meaning that data from the master or slave submodule is synchronized on either the rising or falling edge of the Master SPI's clock, BCLK. The Master Out Slave In signal, MOSI, is an output to the master SPI and an input to the slave SPI. On the other hand, MISO is an input to the master submodule and an output from the slave submodule. Due to the two separate serial data lines, the SPI protocol is full duplex, meaning data can be transmitted from both modules simultaneously. The last signal in the SPI interface is the Slave Select (SS) signal, which is an output from the master, and read by each slave SPI submodule. When the SS line is cleared to zero, the connected slave submodule will be enabled. With multiple SS select lines, it is possible to choose between communicating with multiple slave submodules. Since the master SPI submodule drives the clock and slave select pins, it is only possible for one master SPI module to be present in the system.

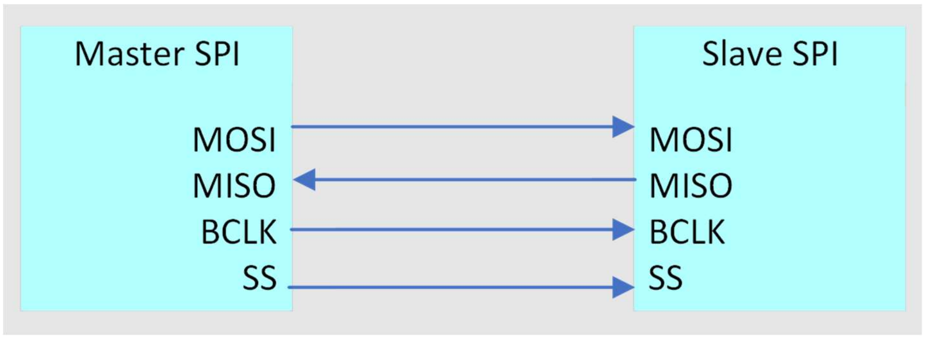


Figure 21 Master/Slave SPI Interaction

Master/Slave Clock Synchronization

One issue that we needed to consider with our Backdoor SPI module was handling metastability and clock synchronization between two separate clock sources. Since we will be supplied a clock from the external master SPI submodule, we will face the issue of metastability if unaddressed, since the clock speeds of the external master clock (BCLK) and the system clock (SYSCLK) in the user area of our chip will be different frequencies. If the edges of both clocks do not occur close enough, then timing constraints on the second clock could be violated, where there is not enough time for data to properly propagate and update before the next edge of the user area clock. Without clock synchronization, the Backdoor SPI module may not be able to properly read the address or data contents sent from the master SPI module.

To solve this issue, we decided to implement a common solution to metastability, which involves inserting two D flip flops between the two clock domains. By using the receiving clock, or the slave SPI clock SYSCLK from the user area, we can ensure that two clock cycles have to occur on the user area side before the proper flags are asserted that data is received. Instead of using more D flip flops to propagate the 7 address bits or 32 data bits from the master SPI module, we decided to instead buffer the done indicator bit that would be flagged at the end of receiving data from the master SPI module. The figure below demonstrates the connection between both SPI modules and the metastable region.

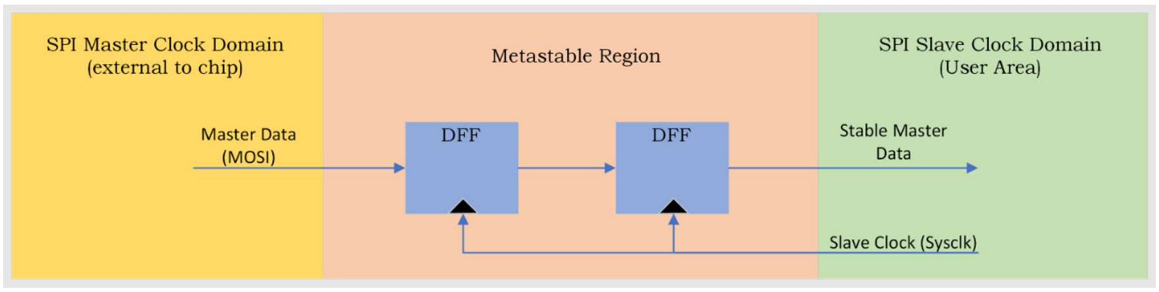


Figure 22 Clock Synchronization Example

Design Summary

Backdoor SPI Module

Inputs

- i_SYSCLK – System Clock
- i_BCLK – Bus Clock from Master
- i_SS – Bus Slave Select (Active-high reset)
- i_MOSI – Bus Master Out Slave In
- i_DATA_OUT[31:0] – Data to be sent to master SPI, loaded from other user modules

Outputs

- o_MISO – Bus Master In Slave Out
- o_ADDR[6:0] – Address to determine what the data is for
- REGISTER[6:3] - What data do you want from each module?
- MODULE[2:0] - Which module are you talking to?
- o_DATA_IN[31:0]: output data read from master SPI module, read by user modules

- o_DOUT_VALID: Flag that is asserted when DATA_OUT is ready to be read by submodules

The backdoor SPI interface will be a simple bidirectional 4-wire SPI bus on physical pins to the modules we create elsewhere on the device. The SS pin will reset the SPI module when high. The first 7 bits of transfer data from i_MOSI will be interpreted as the address from which to read and write, with the following 8th bit for the read/write identification, s_READ. At that time, s_ADDR will be set to the correct value to drive logic within the modules. It is expected that s_ADDR will drive a large multiplexer between all peripheral values into the i_DATA_OUT field which must be stable within 1 system clock cycle of ADDR or READ changing. The output of that mux will be sent in the following 32 bits to the master SPI module regardless of the s_READ bit. If s_READ is 0, the mux will be ignored, and the next 32 bits will be stored in DATA_IN. The o_DOUT_VALID flag will be high for one system clock cycle at the end of transfer during which time the data is guaranteed to be stable and the addressed module should act on the provided data.

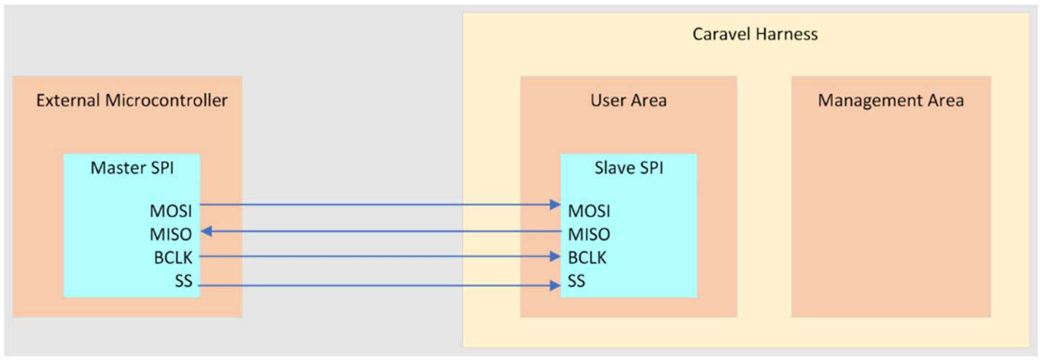


Figure 23 Backdoor SPI Top Level

The following steps outline both a read and write process involving both SPI modules. A read and write process will be initiated when the I_SS slave select input from the master SPI submodule is cleared to 0.

Data Write Transfer from MOSI:

1. Receive addressing
   - Serial shift in 7 address bits from i_MISO
   - Serial shift in   READ bit flag from i_MISO
2. Receive Data
   - Serial shift in 32 bits of data from i_MISO
3. Confirm Clock Synchronization
   - Wait 2 SYSCLK cycles to assert o_DATA_VALID flag to 1
   - Wait 1 SYSCLK cycle, clear o_DATA_VALID to 0

Data Read Transfer to MISO:

1. Receive Addressing
    - Serial shift in 7 address bits from i_MISO
    - Serial shift in 1 READ bit flag from i_MISO
2. Confirm clock synchronization
    - Wait 2 SYSCLK cycles to ensure address propagates to user modules for data
3. Send Data
    - Parallel load i_DATA_OUT from user area MUX
    - Serial Shift out 32 bits of data to o_MOSI

The following schematic depicts the proposed implementation of the Backdoor SPI module that will be placed inside the user area of our ASIC. As defined above, the schematic utilizes two shift in registers and one shift out register to handle the serial interfacing with the external master SPI module. The final bit of the shift in registers acts as an enable bit, since the shift in registers will reset the least significant bit to 1, which will stop the shift registers once the least significant bit is shifted the proper number of times for each module. The D Flip Flops are used for the clock synchronization and metastability solution. The status bits propagate through two D flip flops, and a third flip flop is included so that the status bits will only remain asserted to 1 for one SYSCLK cycle.

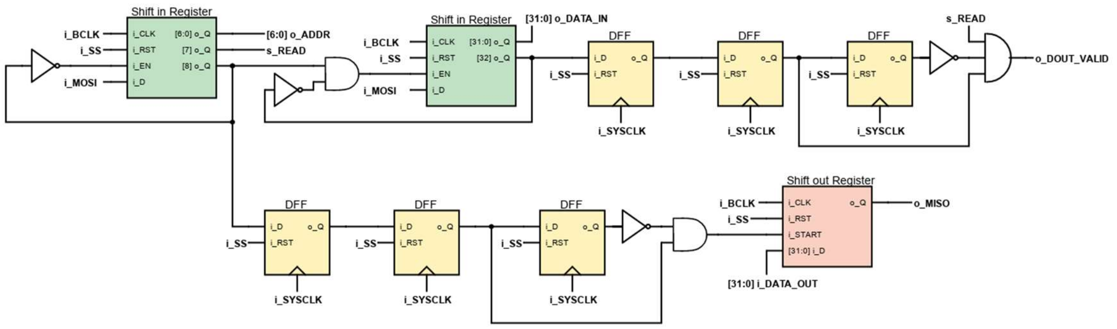


Figure 24 Backdoor SPI Schematic

The following table is an example of how the o_DOUT_VALID signal is asserted after all of the data bits from I_MOSI are received. A similar process follows for the I_START enable bit for the shift out register, while excluding the check of either a read or write.

Table 4 o_DOUT_VALID Truth Table

SYSCLK Cycle	**o_Q[32] Data Shift in Reg**	**o_Q DFF0**	**o_Q DFF1**	**o_Q DFF2**	**s_READ**	**o_DOUT_VALID**
1	0	0	0	0	1	0

2	1	0	0	0	1	0
3	1	1	0	0	1	0
4	1	1	1	0	1	1
5	1	1	1	1	1	0

Shift In Register

Inputs

- i_CLK: System Clock Input. Synchronous shift
- i_RST: Asynchronous reset
- i_EN: If asserted to 1, o_Q shifted left by 1 bit. Otherwise, no shift occurs
- i_D: Written to LSB of o_Q to shift in from MOSI

Outputs

- [DATA_WIDTH : 0] o_Q: Parallel output of DATA_WIDTH bits

Parameters

- DATA_WIDTH: number of data bits to shift in, default value 32

The shift in register designed for the Backdoor SPI module is used as a serial in, parallel output shift register to receive the incoming data from the Master SPI submodule on the MOSI input to the slave module in the user area. The shift in register will update on the positive edge of the input clock from the Backdoor SPI module, with an asynchronous reset. On the positive edge of the input clock, the output register o_Q will shift left by one bit and write i_D to the least significant bit. The shift in register will only shift left by one bit if I_EN is asserted to one on the positive edge of the next clock cycle. If the enable input is not asserted, or zero, then no shift will occur and o_Q will remain in the same state as the previous clock cycle. When the asynchronous reset is asserted, o_Q will be assigned with 1. The intent is so that the initialized least significant bit can be shifted throughout the o_Q register as 1, and act as an enable indicator outside of the shift module. The most significant bit of the output o_Q will be read as an enable bit for the shift registers, to manage the state of the incoming data. Due to this design, it enables us to create a shifting enable pin that can shift in DATA_WIDTH bits, which can vary depending on if the master SPI is sending 7 address bits or the following 32 data bits.

The following Verilog code defines the implementation of the shift in module:

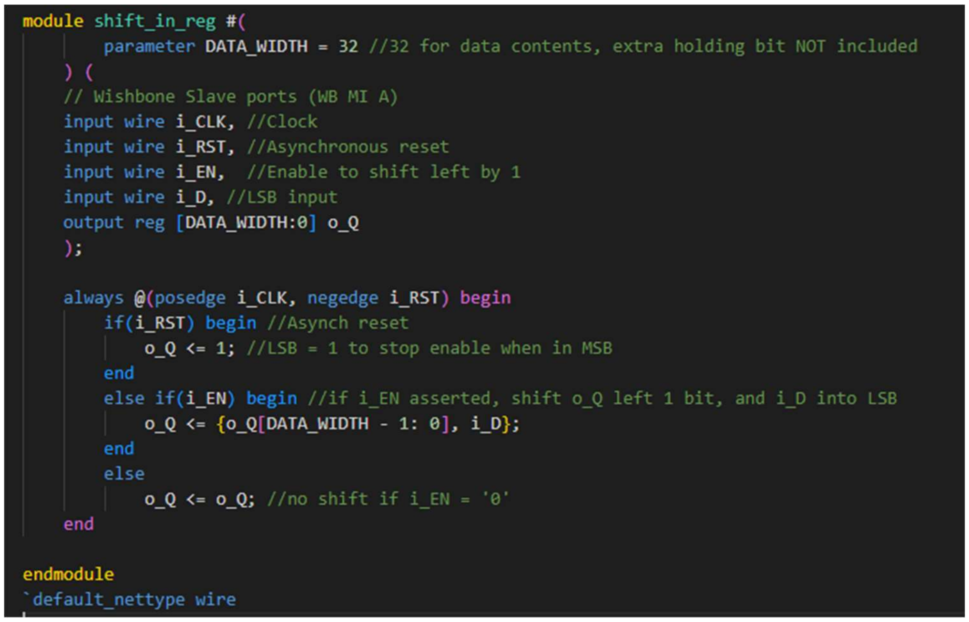

```verilog
module shift_in_reg #(
        parameter DATA_WIDTH = 32 //32 for data contents, extra holding bit NOT included
    ) (
    // Wishbone Slave ports (WB MI A)
    input wire i_CLK, //Clock
    input wire i_RST, //Asynchronous reset
    input wire i_EN,  //Enable to shift left by 1
    input wire i_D, //LSB input
    output reg [DATA_WIDTH:0] o_Q
    );

    always @(posedge i_CLK, negedge i_RST) begin
        if(i_RST) begin //Asynch reset
            o_Q <= 1; //LSB = 1 to stop enable when in MSB
        end
        else if(i_EN) begin //if i_EN asserted, shift o_Q left 1 bit, and i_D into LSB
            o_Q <= {o_Q[DATA_WIDTH - 1: 0], i_D};
        end
        else
            o_Q <= o_Q; //no shift if i_EN = '0'
    end

endmodule
`default_nettype wire
```

Figure 25 Verilog implementation of N bit shift in register, shift_in_reg.v

Shift Out Register

Inputs

- i_CLK: System Clock Input. Synchronous shift
- i_RST: Asynchronous reset
- i_START: If asserted to 1, parallel load of i_D made. Otherwise, no load
- [DATA_WIDTH – 1: 0] i_D: Parallel data load to shift out serially to MISO

Outputs

- o_Q: serial shift out intended for MISO of SPI module

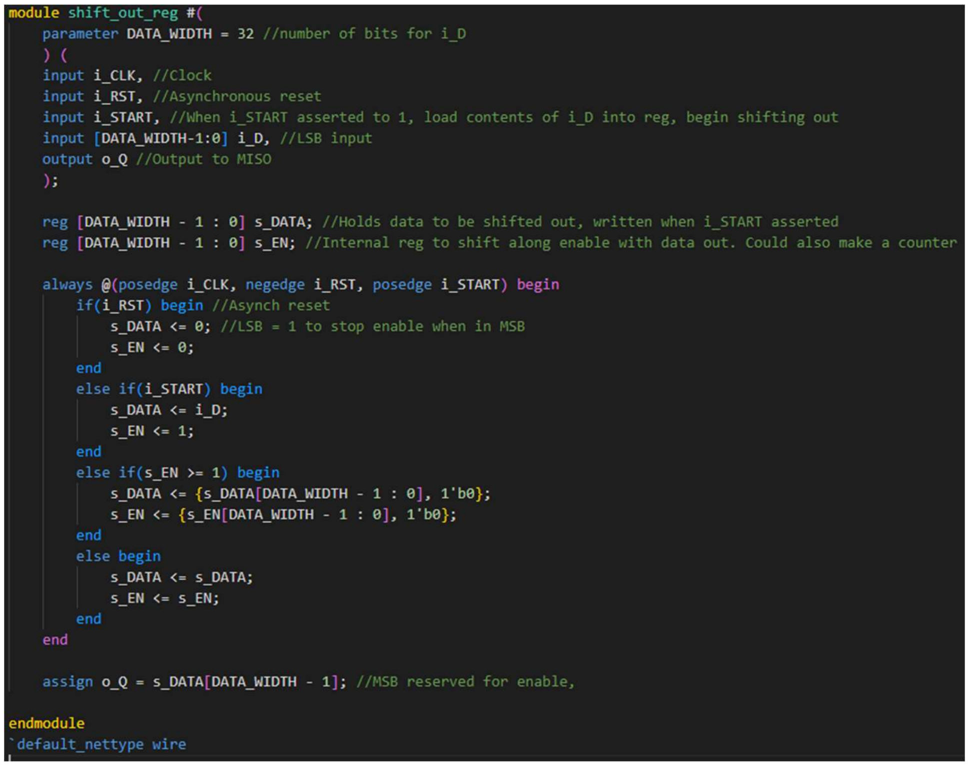
Parameters

- DATA_WIDTH: number of data bits to load and shift out

The shift out register is used to handle the MISO output from the user area to the master SPI submodule on an external microcontroller. Due to this, the input data i_D takes DATA_WIDTH bits, expecting 32, for a parallel load when i_START is asserted to 1. After I_START is asserted, every following clock cycle for DATA_WIDTH clock cycles will shift the internal register s_DATA left by 1 bit. The output o_Q is assigned to the most significant bit of s_DATA, so that the most significant bit is shifted out first, like how the most significant bit is shifted into the shift in register. When I_START is asserted, the other internal register i_EN is reset to 1. Like the shift in register, s_EN is shifted left by 1 bit alongside s_DATA, but acts as an internal enable to ensure the shift out register stops shifting data after 32 bits. Since a 0 bit is shifted into the least significant bit, this enable tracking is redundant with the current design, but helps to reduce risk if more complexity is added in the future.

The functionality between the shift in and shift out registers are very similar but switch the serial and parallel I/O between the data being loaded and the data being shifted out, to satisfy the requirements of the SPI protocol between the MOSI and MISO signal lines between the user area and the external SPI master.

The following section of Verilog code defines the implementation of the shift out register:



```verilog
module shift_out_reg #(
    parameter DATA_WIDTH = 32 //number of bits for i_D
    ) (
    input i_CLK, //Clock
    input i_RST, //Asynchronous reset
    input i_START, //When i_START asserted to 1, load contents of i_D into reg, begin shifting out
    input [DATA_WIDTH-1:0] i_D, //LSB input
    output o_Q //Output to MISO
    );

    reg [DATA_WIDTH - 1 : 0] s_DATA; //Holds data to be shifted out, written when i_START asserted
    reg [DATA_WIDTH - 1 : 0] s_EN; //Internal reg to shift along enable with data out. Could also make a counter

    always @(posedge i_CLK, negedge i_RST, posedge i_START) begin
        if(i_RST) begin //Asynch reset
            s_DATA <= 0; //LSB = 1 to stop enable when in MSB
            s_EN <= 0;
        end
        else if(i_START) begin
            s_DATA <= i_D;
            s_EN <= 1;
        end
        else if(s_EN >= 1) begin
            s_DATA <= {s_DATA[DATA_WIDTH - 1 : 0], 1'b0};
            s_EN <= {s_EN[DATA_WIDTH - 1 : 0], 1'b0};
        end
        else begin
            s_DATA <= s_DATA;
            s_EN <= s_EN;
        end
    end

    assign o_Q = s_DATA[DATA_WIDTH - 1]; //MSB reserved for enable,

endmodule
`default_nettype wire
```

Figure 26 Verilog implementation of N bit shift out register, shift_out_reg.v

Research and Planning

The following images show research and planning for the Backdoor SPI module throughout the first semester of our senior design process.

Weekly Design Meeting: 3/26/2023

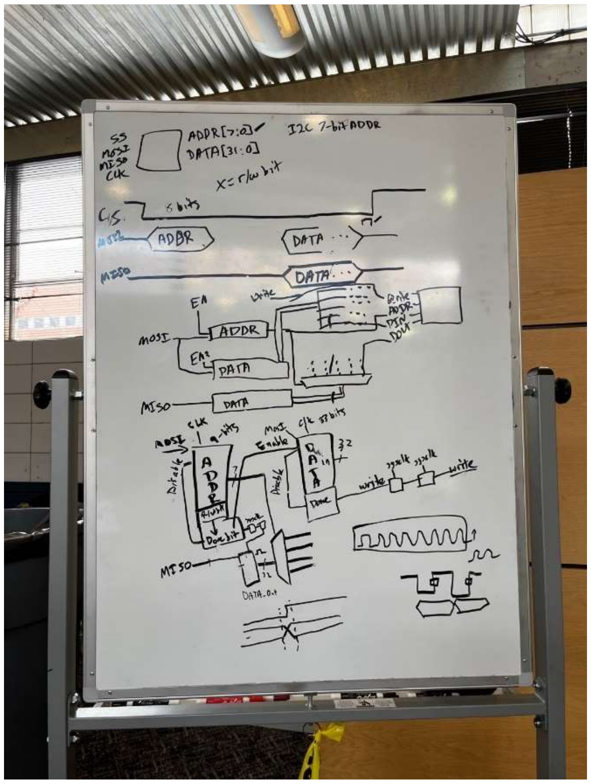


Figure 27 Initial Brainstorm

Diagram after SPI Schematic Discussion: 3/27/2023

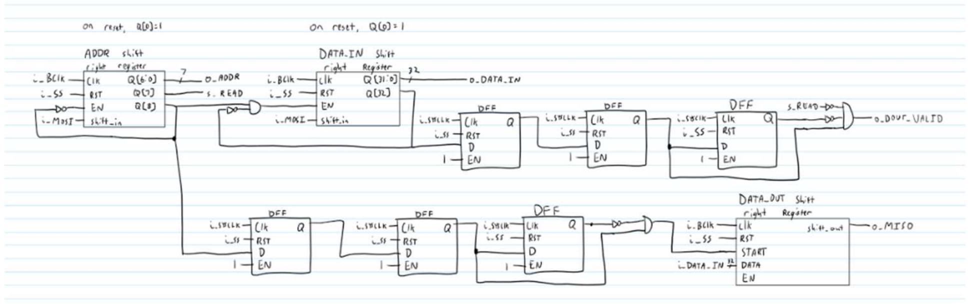
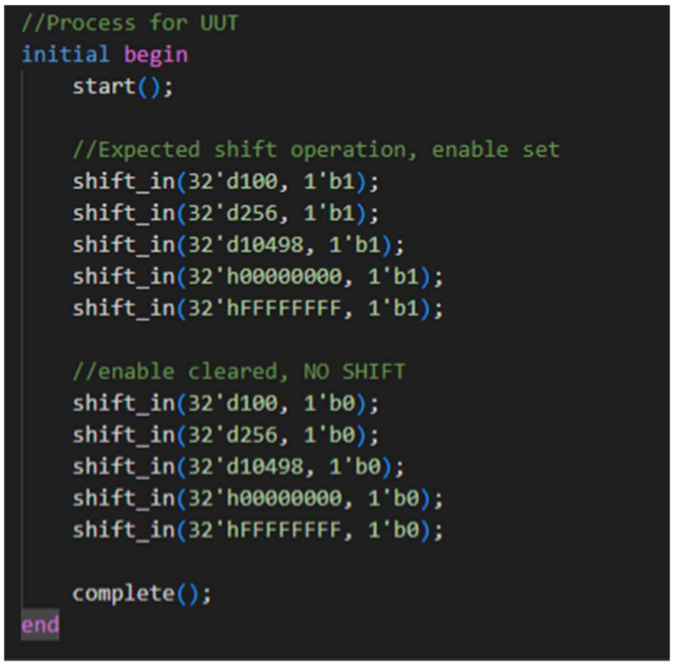


Figure 28 Rough First Schematic

Test Procedure

Shift In Register

To test the shift in register, the main requirement to fulfill was that data would be serially loaded into the output register o_Q from the input i_D. To simulate testing the data being shifted in from the master SPI module, we instantiated a shift in register with a DATA_WIDTH of 32 bits. So, this meant that for each run of our test, it would take 32 clock cycles to successfully shift in the data, with the most significant bit being shifted in first. The following initial begin block of Verilog demonstrates using tasks to automate and integrate our Verilog testbench. Through the use of tasks, we are able to make our testbench both more readable and easier to include more test cases.



```
//Process for UUT
initial begin
    start();

    //Expected shift operation, enable set
    shift_in(32'd100, 1'b1);
    shift_in(32'd256, 1'b1);
    shift_in(32'd10498, 1'b1);
    shift_in(32'h00000000, 1'b1);
    shift_in(32'hFFFFFFFF, 1'b1);

    //enable cleared, NO SHIFT
    shift_in(32'd100, 1'b0);
    shift_in(32'd256, 1'b0);
    shift_in(32'd10498, 1'b0);
    shift_in(32'h00000000, 1'b0);
    shift_in(32'hFFFFFFFF, 1'b0);

    complete();
end
```

Figure 29 Shift in tests

Multiple tasks are utilized in the shift in register testbench, including start() and reset(). The start() task resets the shift in module and ensures that the input data and enable signals are set to 0, so that they will not be driven before the first shift_in() task. The reset() task asserts the input reset for one clock cycle, so that a module reset can asynchronously occur if called in the testbench or any other task at a time. The s_status internal signal is initialized to 1 at the start of the test, and it updated to 0 only if a test case fails, indicating a failed test at the end of the testbench.

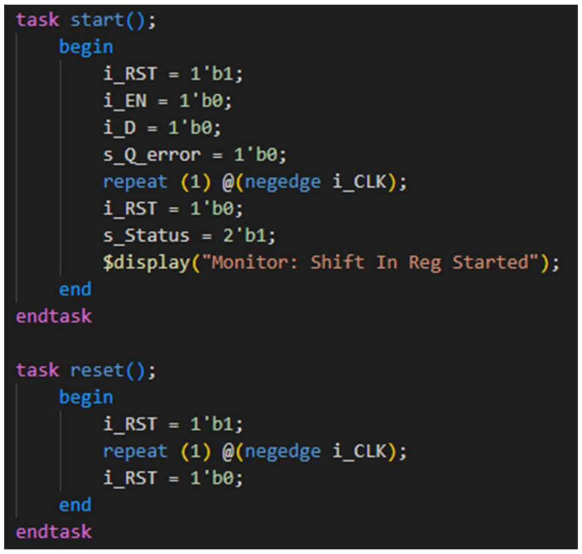
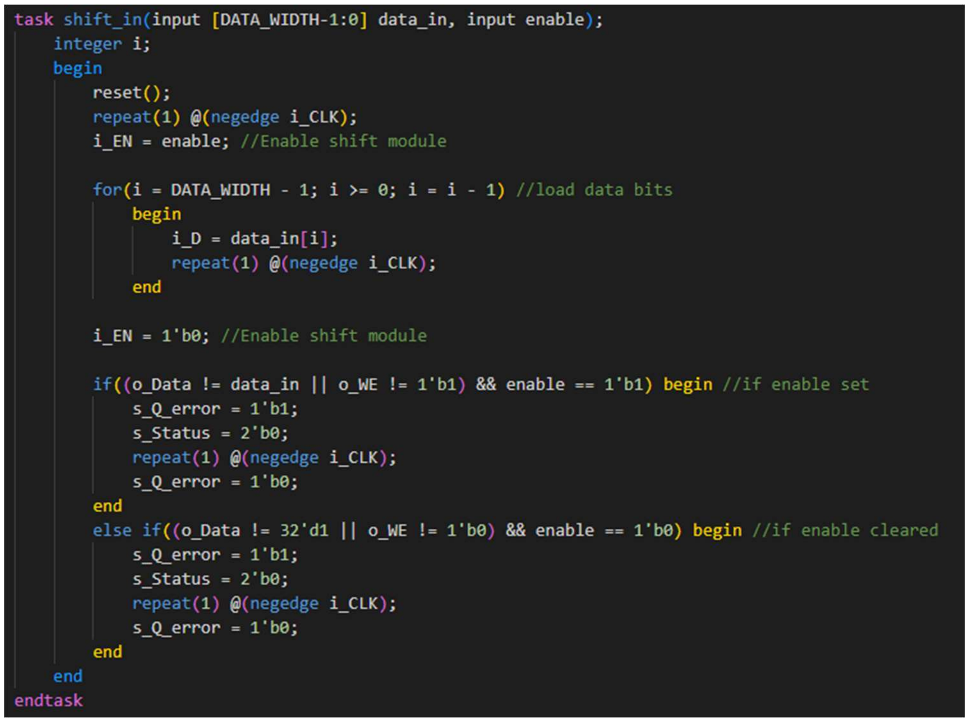

```
task start();
    begin
        i_RST = 1'b1;
        i_EN = 1'b0;
        i_D = 1'b0;
        s_Q_error = 1'b0;
        repeat (1) @(negedge i_CLK);
        i_RST = 1'b0;
        s_Status = 2'b1;
        $display("Monitor: Shift In Reg Started");
    end
endtask

task reset();
    begin
        i_RST = 1'b1;
        repeat (1) @(negedge i_CLK);
        i_RST = 1'b0;
    end
endtask
```

Figure 30 Test driver

The following task, shift_in(), is the core of the shift in register testbench. The shift_in() task takes two inputs, one for the full bit vector of the incoming data, and the other input being the enable pin. While the intent for the shift register is to route the most significant bit of the output to the enable pin outside of the module, we wanted to drive the enable pin separately to ensure full functionality for these unit tests. At the beginning of the task, the reset() task is called, since the shift in SPI module will always begin with a reset when receiving the address. After one clock cycle, the enable input is asserted, meaning that every bit after will be shifted left by 1, assigning one bit to i_D to write to the LSB of the output register o_Q. To achieve this in the shift_in() task, a for loop is used to index between the data_in input, with a one clock cycle wait in between each assignment, updated on the negative edge of the testbench and shift in clock. After 32 bits are shifted, the enable input is deasserted to 0, and error checking is computed. If the output register o_Q differs from the input data_in to the task, the error flag s_Q_error is asserted for one clock cycle, and the s_Status signal is set to 0. This ensures that the final test result can be displayed, and it is easy to interpret when an error occurs between multiple shift_in() tasks.

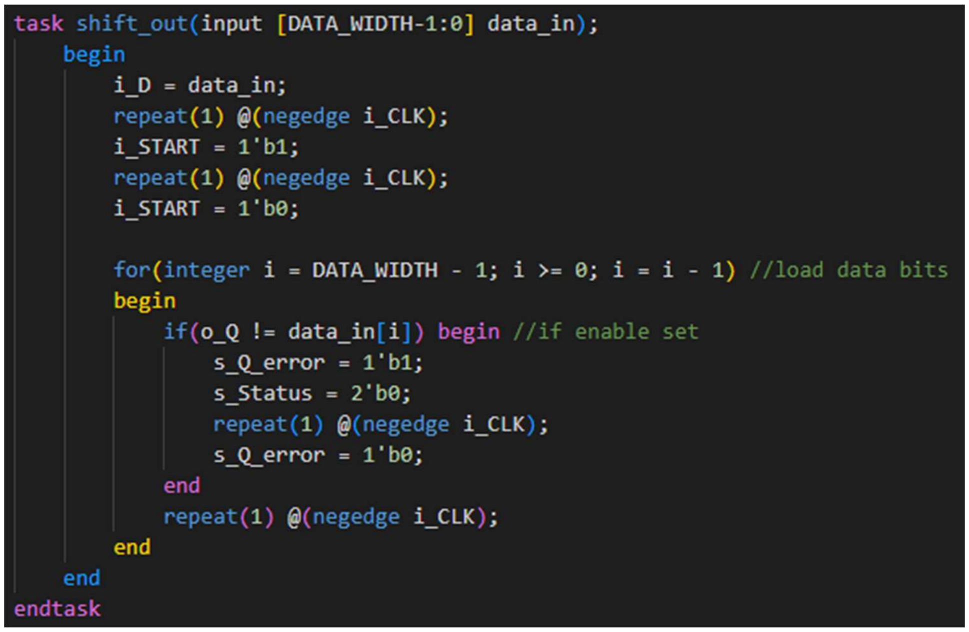


```
task shift_in(input [DATA_WIDTH-1:0] data_in, input enable);
    integer i;
    begin
        reset();
        repeat(1) @(negedge i_CLK);
        i_EN = enable; //Enable shift module

        for(i = DATA_WIDTH - 1; i >= 0; i = i - 1) //load data bits
            begin
                i_D = data_in[i];
                repeat(1) @(negedge i_CLK);
            end

        i_EN = 1'b0; //Enable shift module

        if((o_Data != data_in || o_WE != 1'b1) && enable == 1'b1) begin //if enable set
            s_Q_error = 1'b1;
            s_Status = 2'b0;
            repeat(1) @(negedge i_CLK);
            s_Q_error = 1'b0;
        end
        else if((o_Data != 32'd1 || o_WE != 1'b0) && enable == 1'b0) begin //if enable cleared
            s_Q_error = 1'b1;
            s_Status = 2'b0;
            repeat(1) @(negedge i_CLK);
            s_Q_error = 1'b0;
        end
    end
endtask
```

Figure 31 Shift in test function

Shift Out Register

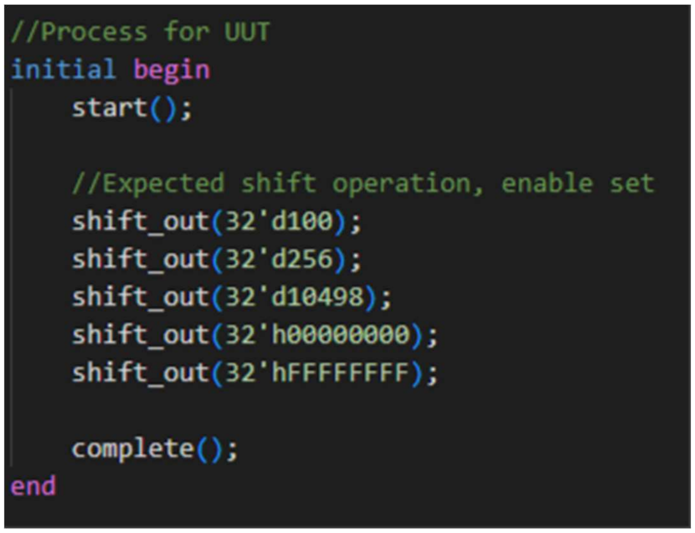
The shift out register testbench utilizes similar tasks as the shift in register but modifies the main shift_in() task as shift_out() by loading the parallel bits of data_in into i_D immediately, then asserting i_START. The following 32 clock cycles are utilized to compare the serial data being shifted out of the register from o_Q to the indexed bit from the data_in input to the task. This is very beneficial since it ensures that each bit of the clock cycle matches the expected output and raises the same s_Q_Error flag as the shift in testbench. Alongside this, the s_Status internal signal is updated to 0 on a failed comparison, indicating that the test has failed at the end of the testbench, in the complete() task.



```
task shift_out(input [DATA_WIDTH-1:0] data_in);
    begin
        i_D = data_in;
        repeat(1) @(negedge i_CLK);
        i_START = 1'b1;
        repeat(1) @(negedge i_CLK);
        i_START = 1'b0;

        for(integer i = DATA_WIDTH - 1; i >= 0; i = i - 1) //load data bits
        begin
            if(o_Q != data_in[i]) begin //if enable set
                s_Q_error = 1'b1;
                s_Status = 2'b0;
                repeat(1) @(negedge i_CLK);
                s_Q_error = 1'b0;
            end
            repeat(1) @(negedge i_CLK);
        end
    end
endtask
```

Figure 32 Shift out test function

The testbench utilizes shifting out different values of 32 bits, including the common case of multiple decimal values of 100, 256, and 10498. We also verified the shift out register was able to successfully shift out a 32-bit vector of all zeros and all ones, signified by the task calls of 0x00000000 and 0xFFFFFFFF.



```
//Process for UUT
initial begin
    start();

    //Expected shift operation, enable set
    shift_out(32'd100);
    shift_out(32'd256);
    shift_out(32'd10498);
    shift_out(32'h00000000);
    shift_out(32'hFFFFFFFF);

    complete();
end
```

Figure 33 Shift out tests