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 Accelerator Lead

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

Revised: 12/2/2023 Revision: 2.1

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

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 because 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:

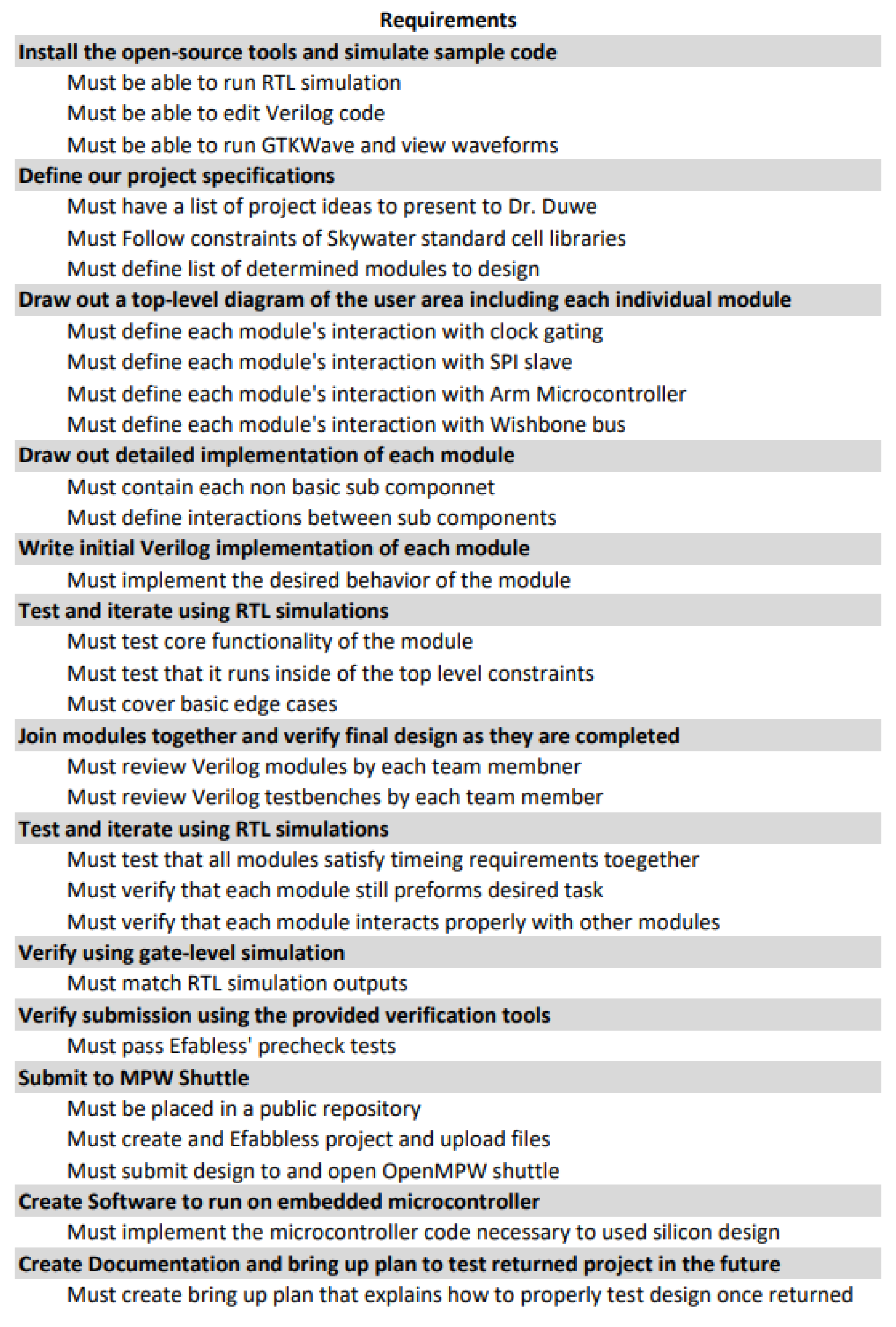
1. **Install the open-source tools and simulate sample code**
    a. Build a sample user project with the Caravel tool flow
    b. Complete a Verilog simulation of a sample user project
    c. View the output waveforms of a sample user project in GTKWave
2. **Define our project specifications**
    a. 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

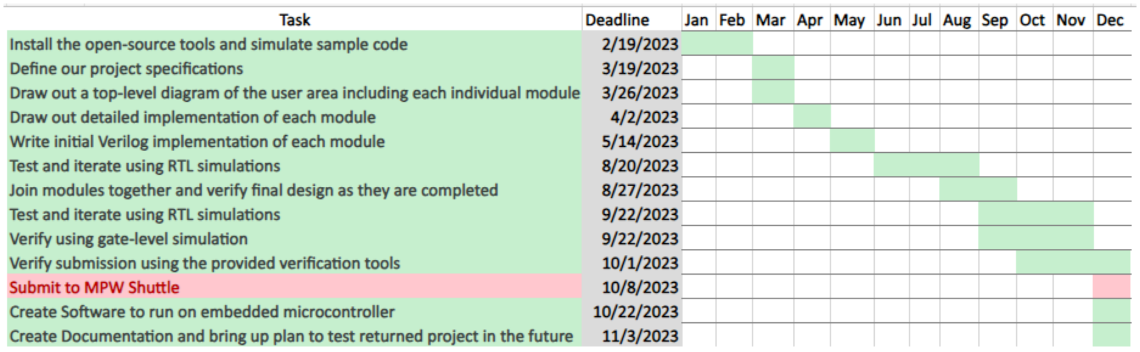
Due to the nature of our project, we rely on functional requirements, and our milestones are based on qualitative data. Our progress milestones are defined as a functional module or subunit of a module that has been tested and verified independently from the rest of the design which will provide small 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	🟩	🟩										
Define our project specifications	3/19/2023			🟩									
Draw out a top-level diagram of the user area including each individual module	3/26/2023			🟩									
Draw out detailed implementation of each module	4/2/2023				🟩								
Write initial Verilog implementation of each module	5/14/2023					🟩							
Test and iterate using RTL simulations	8/20/2023						🟩	🟩	🟩				
Join modules together and verify final design as they are completed	8/27/2023								🟩	🟩			
Test and iterate using RTL simulations	9/22/2023									🟩	🟩	🟩	
Verify using gate-level simulation	9/22/2023									🟩	🟩	🟩	
Verify submission using the provided verification tools	10/1/2023										🟩	🟩	🟩
Submit to MPW Shuttle	10/8/2023												🟥
Create Software to run on embedded microcontroller	10/22/2023												🟩
Create Documentation and bring up plan to test returned project in the future	11/3/2023												🟩

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.

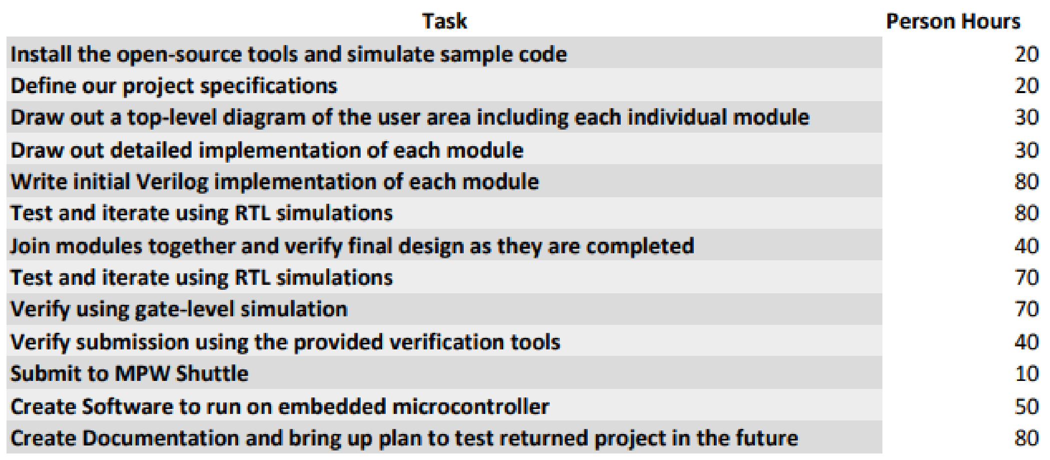
These risks will affect the outcome of our project, but not the viability of our product as a test of the system. If a submission does not open, our design will be left in a submittable form to be submitted whenever submissions do open, and the usefulness of our project will not be diminished. If our 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
- Magic
- XSchem
- 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.

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 Accelerator Module**
   - This module will require the development of a large convolution network accelerator 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 Accelerator Module for the system that was previously designed for a microcontroller?

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 Accelerator 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 an accelerator for a Weiner Filter that uses a direct convolution to apply the filter to the incoming audio all in the time domain.
- The second design idea was to implement an accelerator for 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 create an accelerator for 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 accelerator for 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 accelerator 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 Accelerator (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

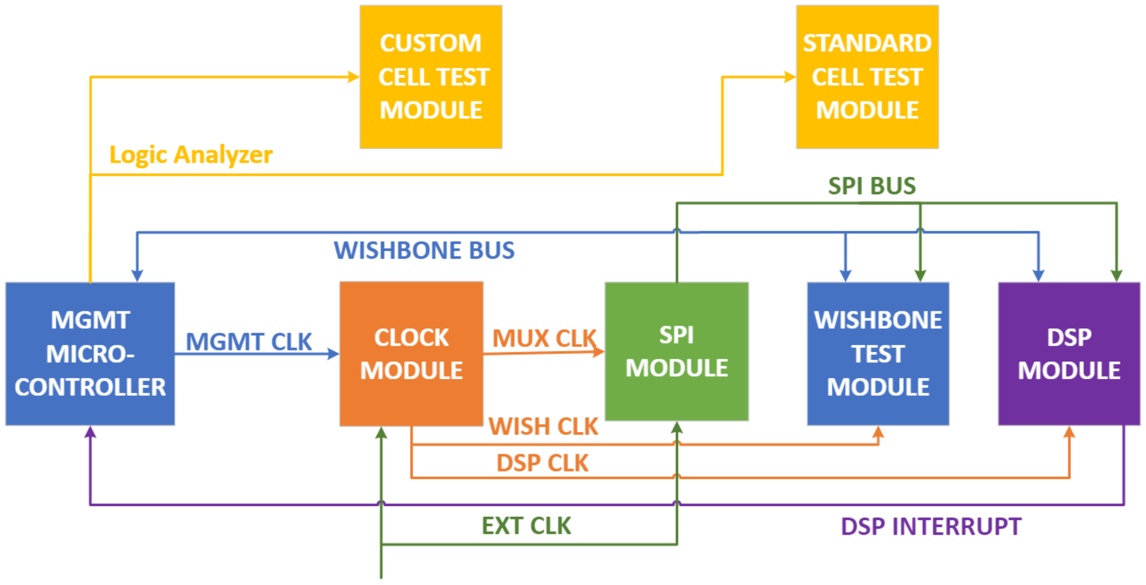


Figure 4 Caravel User Area Module Diagram

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 Accelerator module implements an accelerator for 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 Accelerator 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 convolutional accelerator 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.

Results for each phase of testing follow below in the Appendix 8.4.6.

5.1 Unit Testing

Each module will have a single Verilog testbench 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. Their waveforms will be verified by using GTKWave, alongside automatic unit case verifications inside of the Verilog testbenches. Verilog Tasks will be utilized to drive inputs to each Design Under Test (DUT) based on a set of input conditions dependent on each submodule design. Both the RTL and GL test will perform functional Verilog, with the GL tests using the synthesized design with functional SkyWater PDK cells in an expanded netlist, from our functional Verilog designs.

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. Tasks are created to drive 32-bit data words over both the SPI and Wishbone buses for easier top-level validation of other submodule designs that utilize both communication interfaces. The Wishbone bus will be individually validated with the Wishbone Test Submodule design, while the SPI interface will be verified using the Backdoor SPI submodule unit tests.

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.

Each module will be validated with the previously designed Verilog tasks which automate much of the SPI and Wishbone communication buses. Each of our designed submodules will be instantiated under the top-level user_project_wrapper Verilog instance and will be validated first with RTL and GL simulations first, like the unit level testing done for each submodule. After that, SDF simulations were completed with a final hardened design, which allowed us to validate our design with real timing requirements instead of ideal functional simulations. To complete the SDF simulations, we were required to successfully harden our full design, so this step came very late in our testing process.

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.

This process followed very closely our integration testing, with the added bonus that we could use C code to verify our design. The downside to this was that it took a very long time to complete but would better model our final test plan by being more aligned with our bring-up plan, which a future team could use to validate our chip once it is fabricated and shipped.

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.

To ensure that changes were easy to make as our design changed over the course of our project, we use layers of abstraction and modularity with our Verilog designs and pin assignments to ensure that as little amount of work as needed had to be done to refactor or alter our code. This was required multiple times, for example when we had to redo our pin assignments for the IO and LA pins, or when we had to move the memory module for the DSP submodule from inside another Verilog module to the top level wrapper.

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.

After our design is fully hardened, we can perform signoff with the SPF waveform tests, which will utilize our fully laid out design for validation with real timing constraints placed on our design.

Finally, we ran our final user area wrapper against the eFabless precheck that is built into the provided GitHub repository that we are working out of. The eFabless precheck includes multiple checks, including DRC and LVS checks which analyze our final hardened design. The precheck also includes checks to ensure that we modified the README file, containing a unique design that differs from the default caravel repository, and that we successfully initialized the GPIO pins.

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

At the end of the first semester, we had 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 the Appendix below.

By the end of the second semester, we were able to perform RTL and GL tests on more submodule designs, and run RTL, GL, and SDF simulations on our final hardened design, which encompassed all 6 of our submodule designs under one Verilog module, which would be fabricated for the user area of our chip. However, the SDF simulation failed to complete due to what appear to be issues in the management core wrapper provided by eFabless which were unable to determine a workaround for. This step occurred late in our process since we had many difficulties hardening our design, which blocked us from running SPF simulations early on. We were also required to change where we placed the memory module for the DSP design, which affected our signal assignments in our Verilog testbenches. Results can be seen in the Appendix 8.4.6 below.

6 Implementation

6.1 Standard Cell Test

Semester 1:

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 the Appendix below.

Semester 2:

Write the Verilog for the and gate that is going to be tested and then verify through simulation that it is working as intended.

We have written Verilog for the standard cell test which instantiates a SkyWater 2 input AND gate using a Behavioral Verilog design. It can be validated after synthesis that a single AND gate is wired to the desired LA pins.

6.2 Custom Cell Test

Semester 1:

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.

Semester 2:

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.

Designing a custom cell turned out to be a significantly harder challenge than expected. The tools were very hard to work with, and DRC/LVS issues plagued us constantly. We started with what we considered a simple project – creating a D flip-flop – but ended up reducing it down to a single NAND gate because the DFF was too large to count as a `cell` in the hardening process (See the User Guide for the difference between a cell and a block).

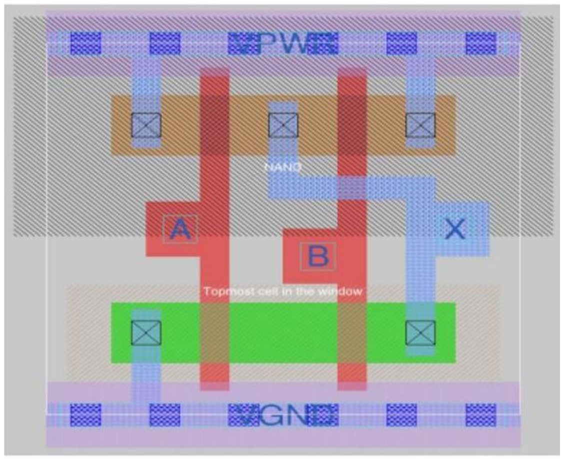


Figure 5 Custom NAND Gate Cell

6.3 Wishbone Test

Semester 1:

The Wishbone Bus test has 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.

Semester 2:

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.

We designed a wishbone bus test to verify the functionality of the primary communication busses. This continually counts based on the input clock (which due to the clock gating module below can be diverted to an LA probe for testing), and can be set or accessed via the wishbone bus or the backdoor SPI module. It works as expected in the C integration tests and also provides the count value via 32 LA probes for further testing.

6.4 Clock Gating

Semester 1:

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

Semester 2:

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.

The clock gating system consists of three clkmux cells taken from the HDLL library. The current version of the PDK does not contain these cells, so we imported them manually from the old library. The first clkmux muxes the primary user project clock (clkmux_CLK) between the wishbone bus clock (the primary clock from the management SOC), and an external IO pin for external clocking if needed. The second clkmux muxes the wishbone test's clock between the project clock (clkmux_CLK) and an LA probe to allow us to override the clock, single-step the design, and turn off the clock independently for the wishbone test. The third clkmux does the same for the DSP accelerator's input clock so each can be driven individually via LA pins or turned off if needed.

6.5 Backdoor SPI

Semester 1:

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 the Appendix below. 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 the Appendix, describing design decisions surrounding the shift registers, clock synchronization, and the external master SPI interface.

Semester 2:

In the second semester, all of the Verilog modules for the Backdoor SPI design have been completed and verified with both RTL and GL simulations using Verilog testbenches. We were able to successfully integrate the Backdoor SPI design into the top-level wrapper and verify that the SPI communication worked successfully by driving IO pins for communications with other submodule designs. Results can be seen in the Appendix below.

6.6 Voice Road Noise Isolation Accelerator (DSP)

Semester 1:

The Voice Road Noise Isolation Accelerator 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.

Semester 2:

In the second semester the first thing that was done was implementing the different submodules in the design. That consisted of designing the mac unit and the counter unit. Each of these components were then tested using a Verilog test bench to ensure that they were functioning properly. Next the main state machine was implemented using the submodules. This module was created in a way to contain all the logic of the design while providing ports to connect the external memory to. The final Verilog implementation was to wire the memory together with the final module inside of the project wrapper. Once all the Verilog was implemented the whole system was tested with a basic Verilog test bench. Once it was verified to be functional the whole module was then hardened, and the same tests could be run on the gate level. Finally, once the module was verified individually on the gate level it was tested in the top-level test to verify that the module is fully functional.

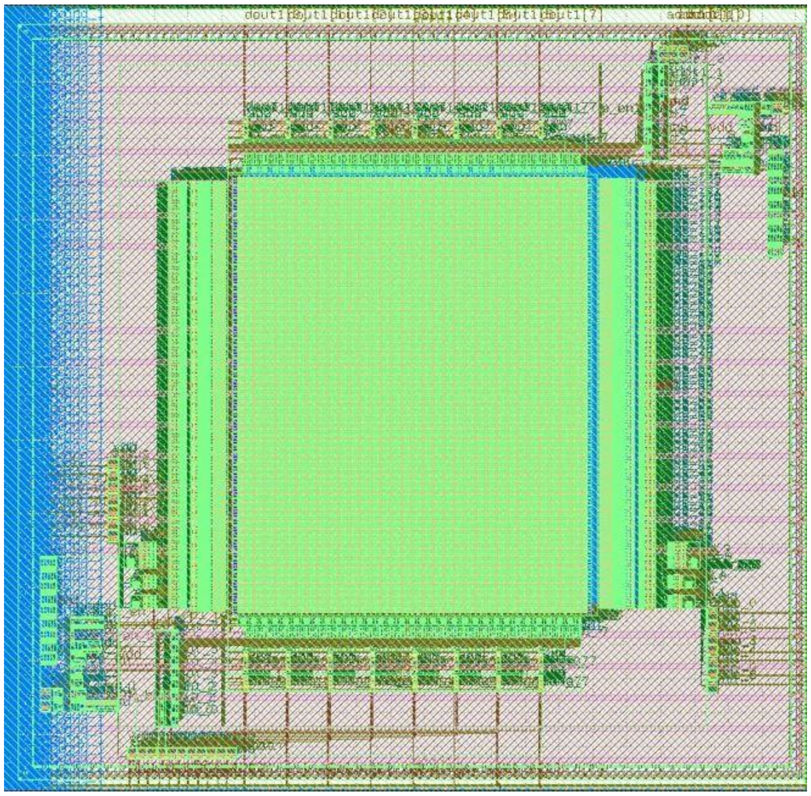


Figure 6 Implemented SRAM

6.7 Submission

Semester 2:

Once all the modules had completed their individual tests the integration process began. Each of the modules were compiled into one large system and routed together. With the fully integrated design complete the top-level system tests were written and applied to the design. The design has been tested and verified to fulfil the requirements of the project design and was run through the eFabless precheck process. The design passes the precheck process and is ready to be submitted to the next available Open MPW Shuttle.

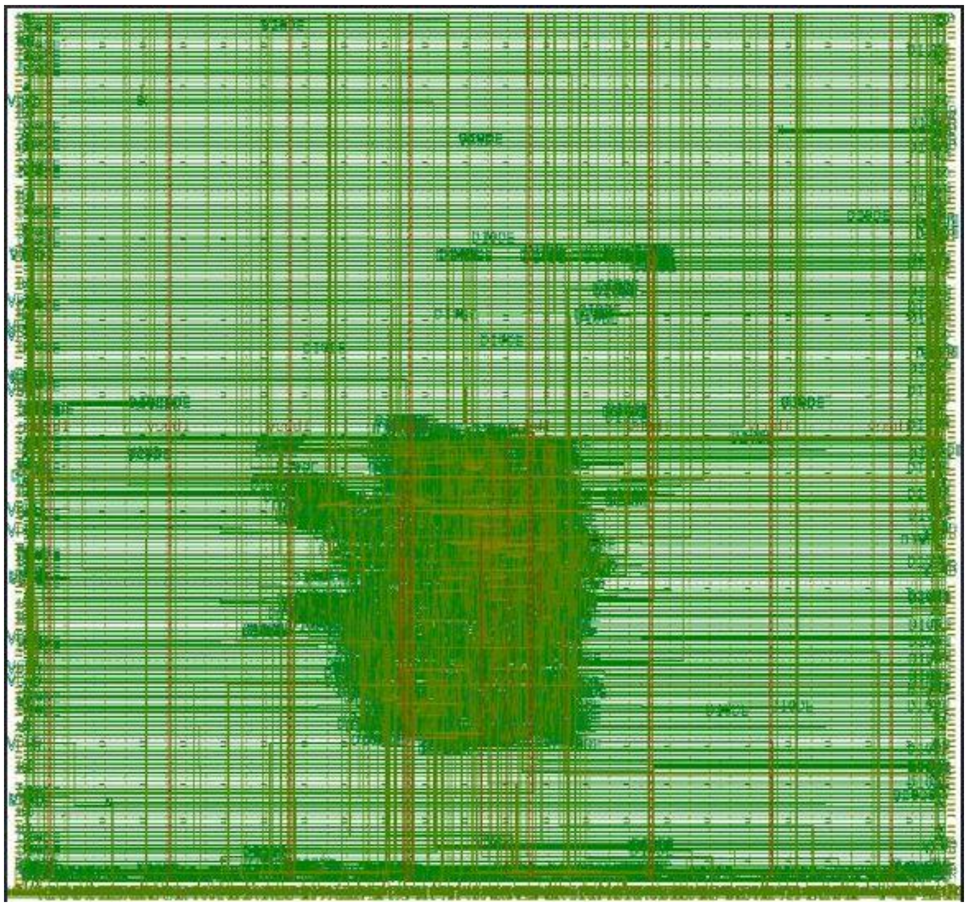


Figure 7 Implemented Behavioral Verilog

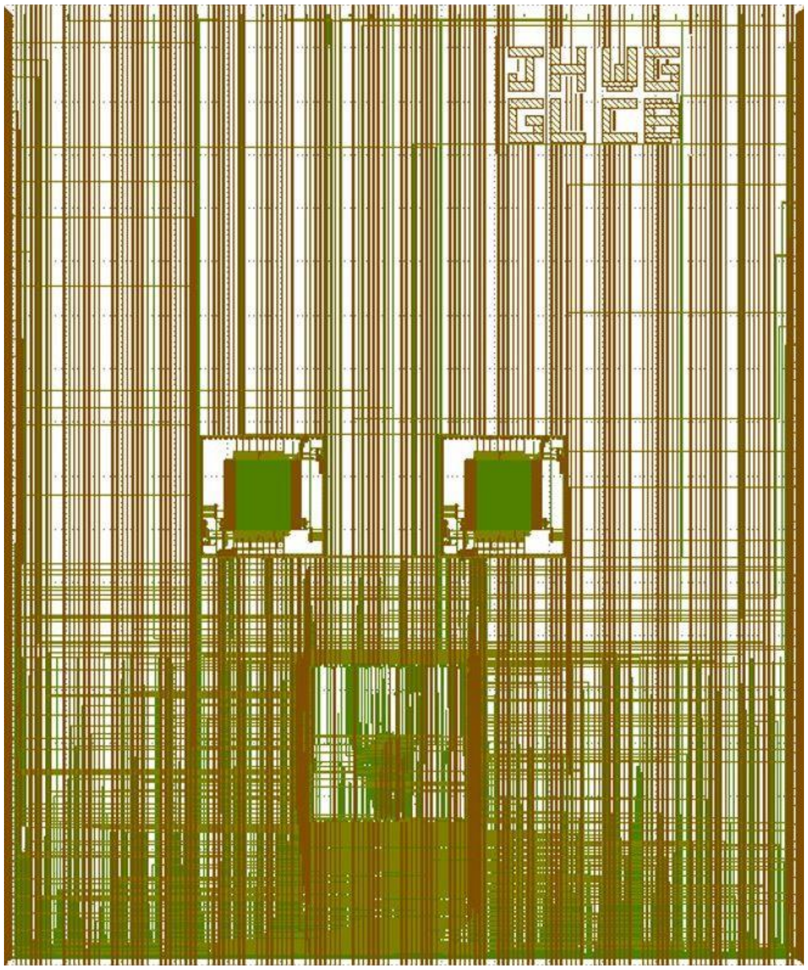
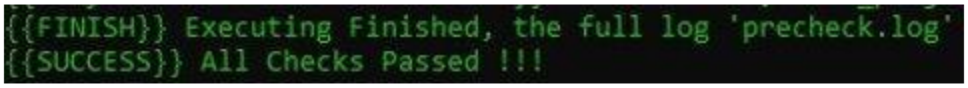


Figure 8 Final Hardened Design



```
{{FINISH}} Executing Finished, the full log 'precheck.log'
{{SUCCESS}} All Checks Passed !!!
```

Figure 9 Precheck Passes

6.8 Bring-up Plan

Semester 2:

Once the design passed precheck and was ready for submission, work could 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. For our submitted bring up plan deliverable, refer below to the Appendix 8.4.2

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

Semester 1:

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 the Appendix below.

Semester 2:

As of the end of our second semester in senior design, we have successfully completed all the deliverables of our project. We have created a functional ASIC design that implements all the modules that we have designed. Each of the modules passes the necessary functional tests to verify that they meet our requirements. Each of the modules are integrated into the top-level design and more functional tests were used to verify the system. The fully tested system was then put through the OpenMPW precheck test to ensure that it meets all the eFabless requirements for manufacturing. Along with the completed design we were able to create a bring up plan for the design that will guide future groups in testing the manufactured product. Finally, we created a caravel user guide that describes all the new technologies that we implemented in our design

8.2 Conclusion

For our project, we have created a modular digital ASIC design using open-source tooling and an OpenMPW Shuttle submission through eFabless. By utilizing tools such as OpenRoad, Magic, XSchem, and more, we have pushed the capabilities and knowledge base for creating a full digital ASIC design within our team and for future senior design teams at Iowa State University. After passing all functional requirements for each submodule, and all integration tests, we were able to successfully harden our design and pass the submission precheck, which will enable our design to be submitted for fabrication in the future. We also have successfully generated a bring up plan with comprehensive firmware code for ASIC validation and pairing documentation for a user to test the chip without design experience. Finally, we were able to provide documentation for how to use the Caravel toolflow in the Caravel User Guide, to outline both the standard workflow, alongside using more unique tools such as the custom cell or OpenRAM tools for other unique design possibilities for future groups.

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 Caravel User Guide

Resources

Getting Started

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

OpenMPW: https://efabless.com/kb-articles/creating-your-first-open-mpw-or-chipignite-project

Simulation: https://caravel-user-project.readthedocs.io/en/latest/#running-full-chip-simulation

Tools

KLayout: https://www.klayout.de/build.html

Docker Desktop: https://www.docker.com/products/docker-desktop/

GTKWave: https://sourceforge.net/projects/gtkwave/files/gtkwave-3.3.100-bin-win64/

Magic: http://opencircuitdesign.com/magic/

Reference

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

Harness: https://caravel-harness.readthedocs.io/en/latest/

MGMT SoC: https://caravel-mgmt-soc-litex.readthedocs.io/en/latest/

Wishbone Bus: https://cdn.opencores.org/downloads/wbspec_b4.pdf

PDK DRC Rules: https://skywater-pdk.readthedocs.io/en/main/rules/periphery.html

OpenRAM: https://vlsi.jp/OpenMPWSRAM_eng.html#using-sram-with-openmpw

Bring-up: https://github.com/efabless/caravel_board/tree/main

Project Setup

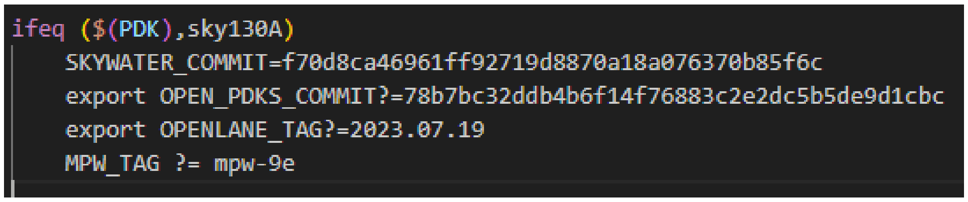
Clone the required caravel_user_project repo from caravel github:
https://github.com/efabless/caravel_user_project

Before running any simulations or hardening, the following three commands must be run in the root of the caravel repository:

```
export OPENLANE_ROOT=$(pwd)/dependencies/openlane_src

export PDK_ROOT=$(pwd)/dependencies/pdks

export PDK=sky130A
```

NOTE: This must be done EVERY TIME anything is run, NOT just on setup for the export commands.

After the root paths are set, we can now build our project. Please ensure that the git commit tags in the root Makefile (seen below) are UP TO DATE with the most recent MPW shuttle branch for the required project submission.



```
ifeq ($(PDK),sky130A)
    SKYWATER_COMMIT=f70d8ca46961ff92719d8870a18a076370b85f6c
    export OPEN_PDKS_COMMIT?=78b7bc32ddb4b6f14f76883c2e2dc5b5de9d1cbc
    export OPENLANE_TAG?=2023.07.19
    MPW_TAG ?= mpw-9e
```

Figure 10 Skywater130A MPW9 Git Commit Tags

Run ‘make setup’ to install the following:

- Skywater pdk (make pdk-with-volare)
- Openlane (make openlane)
- Submission precheck (make precheck)

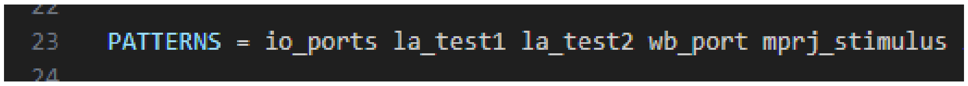
While other dependencies and checks are made in make setup, all the required builds will occur here, ensuring you DO NOT need to run any other make command to build the project. If there is an issue with your project build, it is encouraged to delete the dependencies folder and rerun make setup. If that does not work, delete the cloned repsository and start fresh with another ‘make setup’, ensuring the most up to date commit tags are given in the Makefile, as pulled from the user_project repo.

RTL Simulations

To setup a Verilog testbench in the caravel repository, follow the steps below:

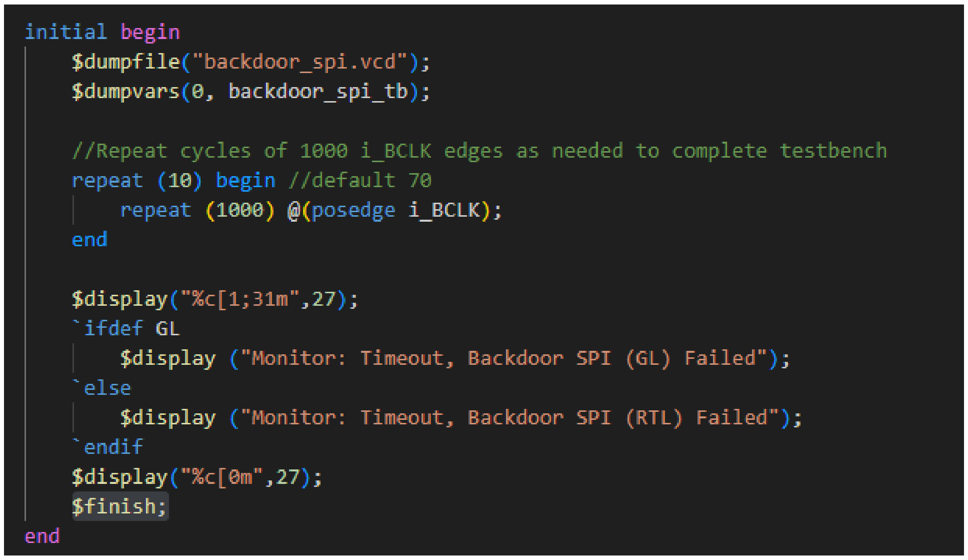
1. /verilog/dv/Makefile

    a. Add MODULE name to PATTERNS list in makefile
2. /verilog/dv
    a. Copy an existing testcase folder
    b. Rename the new folder to MODULE
3. /verilog/dv/MODULE
    a. Rename the copied testbench verilog file and c file to MODULE
    b. Delete all C code from Module.c if only running RTL simulation
4. /verilog/dv/MODULE/MODULE_tb.v
    a. Set name for .vcd file by editing $dumpfile("MODULE.vcd");
    b. Set name for dumpvars by editing $dumpvars(0, MODULE_tb)
    c. Add cmd displays with $display("STRING") in the testbench if desired



```
PATTERNS = io_ports la_test1 la_test2 wb_port mprj_stimulus
```

Figure 11 /verilog/dv/Makefile PATTERNS

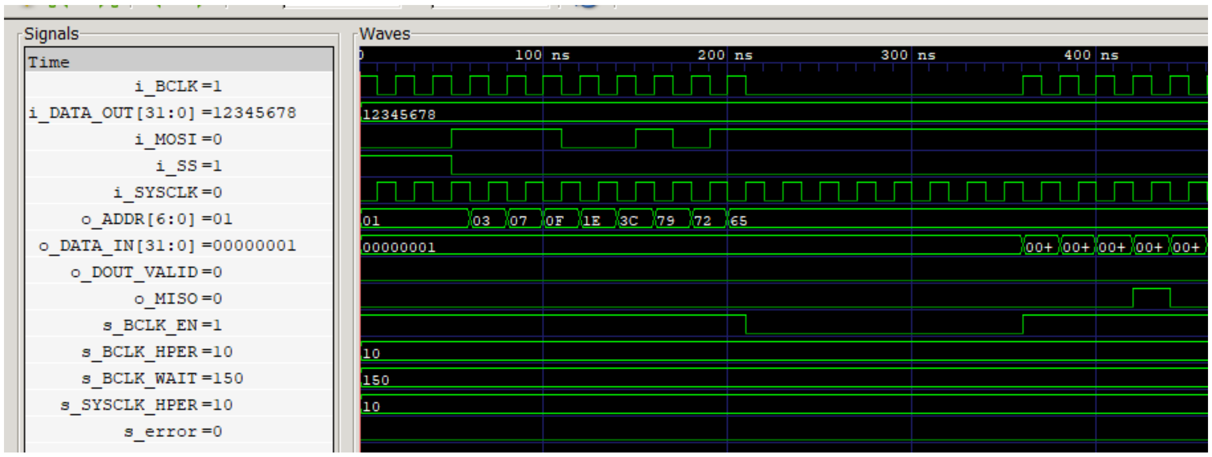


```
initial begin
    $dumpfile("backdoor_spi.vcd");
    $dumpvars(0, backdoor_spi_tb);

    //Repeat cycles of 1000 i_BCLK edges as needed to complete testbench
    repeat (10) begin //default 70
        repeat (1000) @(posedge i_BCLK);
    end

    $display("%c[1;31m",27);
    `ifdef GL
        $display ("Monitor: Timeout, Backdoor SPI (GL) Failed");
    `else
        $display ("Monitor: Timeout, Backdoor SPI (RTL) Failed");
    `endif
    $display("%c[0m",27);
    $finish;
end
```

Figure 12 /verilog/dv/MODULE/MODULE_tb.v dumpfile, dumpvars, display examples

To run a RTL level simulation from root: "make verify-<module>-rtl"

After the RTL simulation is run, you can view the waveform results using the Open-Source tool GTKWave. After opening GTKWave, select File, Open New Tab, and then navigate to the /verilog/dv/MODULE path. Inside of your module's dv folder, a .vcd file will have been generated, as specified with the $dumpfile command and name in your testbench. With the vcd file opened, signals can now be appended to be viewed.



```
Time
i_BCLK=1
i_DATA_OUT[31:0]=12345678
i_MOSI=0
i_SS=1
i_SYSCLK=0
o_ADDR[6:0]=01
o_DATA_IN[31:0]=00000001
o_DOUT_VALID=0
o_MISO=0
s_BCLK_EN=1
s_BCLK_HPER=10
s_BCLK_WAIT=150
s_SYSCLK_HPER=10
s_error=0
```

Figure 13 RTL Simulation GTKWave Sample

Module Hardening

Functional modules are hardened under the /openlane/ folder in the Caravel repository. The supplied top level design user_project_wrapper comes with a set of requirements to pass the eFabless precheck and fabrication and should not be edited. The only Verilog files to add to the top-level module are the top-level functional Verilog files or hardened macros, as referenced below.

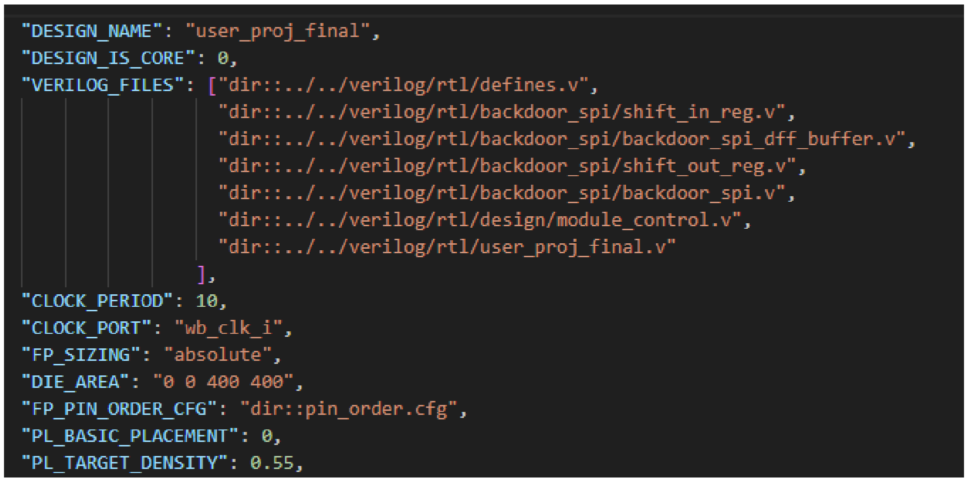
It is encouraged to harden your own macros individually, to verify both that they can harden on their own and pass through synthesis, but also in case you want to pass in your hardened design as a drop in macro, which can reduce the amount of resynthesizing of the top-level wrapper.

To setup a new module for hardening, follow the steps below:

1. /openlane/
    a. Copy an existing sample hardening configuration under /openlane/ that is NOT user_proj_wrapper
    b. Rename the copied config folder to the same MODULE name as your rtl and dv design
2. /openlane/MODULE
    a. Edit config.json with Desired parameters below
    b. Delete pin_order.cfg if not specifying north/east/south/west side for pin placement
    c. Delete macro.cfg if not placing pre-hardened macro into hardened design

The following parameters inside config.json SHOULD be changed, IF NOT user_project_wrapper:

1. **DESIGN_NAME**: name of functional verilog module
2. **VERILOG_FILES**: list of related Verilog files, including submodules instantiated in hardened design
3. **CLOCK_PORT**: Clock input to module for timing analysis
4. **CLOCK_NET**: Clock Net to module for timing analysis
5. **FP_SIZING**: Determines how length/width of module is determined
    a. Set to "Absolute" for a fixed sized based on DIE_AREA
    b. Set to "Relative" for optimized size. NOTE: Does not work for designs small enough if you have too many I/o ports
    c. Default is "Relative"
6. **DIE_AREA**: Determines length and width of hardened module if FP_SIZING set to Absolute
    a. Set to "x0 y0 x1 y1", or "0 0 1000 1000" for area corners
    b. Unit is µm
7. **PL_TARGET_DENSITY**: percentage from 0 to 1 of how DENSE cells are in area
    a. Most designs harden up to around 0.6
    b. 1 = closely dense, 0 = widely spread

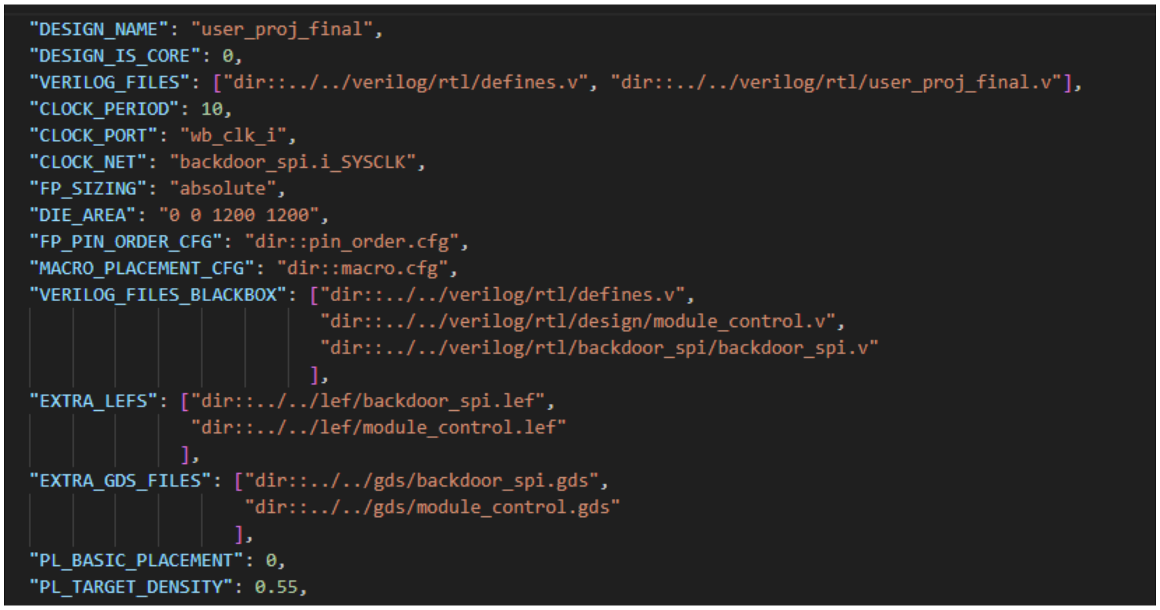


```
"DESIGN_NAME": "user_proj_final",
"DESIGN_IS_CORE": 0,
"VERILOG_FILES": ["dir::../../verilog/rtl/defines.v",
                  "dir::../../verilog/rtl/backdoor_spi/shift_in_reg.v",
                  "dir::../../verilog/rtl/backdoor_spi/backdoor_spi_dff_buffer.v",
                  "dir::../../verilog/rtl/backdoor_spi/shift_out_reg.v",
                  "dir::../../verilog/rtl/backdoor_spi/backdoor_spi.v",
                  "dir::../../verilog/rtl/design/module_control.v",
                  "dir::../../verilog/rtl/user_proj_final.v"
                 ],
"CLOCK_PERIOD": 10,
"CLOCK_PORT": "wb_clk_i",
"FP_SIZING": "absolute",
"DIE_AREA": "0 0 400 400",
"FP_PIN_ORDER_CFG": "dir::pin_order.cfg",
"PL_BASIC_PLACEMENT": 0,
"PL_TARGET_DENSITY": 0.55,
```

Figure 14 Sample config.json WITHOUT pre-hardened macros

If you are using pre-hardened modules, you should also edit the following config lines:

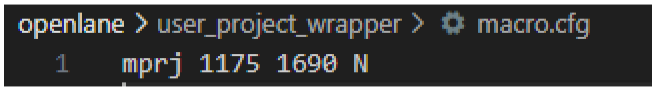
1. **VERILOG_FILES_BLACKBOX**: List each pre-hardened verilog design
2. **EXTRA_LEFS**: List each pre-hardened LEF file /lef/
3. **EXTRA_LIBS**: List each pre-hardened GDS file under /lib/
4. **EXTRA_GDS_FILES**: List each pre-hardened GDS file under /gds/
5. **MACRO_PLACEMENT_CFG**: Set to "dir::macro.cfg" for macro placement



```
"DESIGN_NAME": "user_proj_final",
"DESIGN_IS_CORE": 0,
"VERILOG_FILES": ["dir::../../verilog/rtl/defines.v", "dir::../../verilog/rtl/user_proj_final.v"],
"CLOCK_PERIOD": 10,
"CLOCK_PORT": "wb_clk_i",
"CLOCK_NET": "backdoor_spi.i_SYSCLK",
"FP_SIZING": "absolute",
"DIE_AREA": "0 0 1200 1200",
"FP_PIN_ORDER_CFG": "dir::pin_order.cfg",
"MACRO_PLACEMENT_CFG": "dir::macro.cfg",
"VERILOG_FILES_BLACKBOX": ["dir::../../verilog/rtl/defines.v",
                           "dir::../../verilog/rtl/design/module_control.v",
                           "dir::../../verilog/rtl/backdoor_spi/backdoor_spi.v"
                          ],
"EXTRA_LEFS": ["dir::../../lef/backdoor_spi.lef",
               "dir::../../lef/module_control.lef"
              ],
"EXTRA_GDS_FILES": ["dir::../../gds/backdoor_spi.gds",
                    "dir::../../gds/module_control.gds"
                   ],
"PL_BASIC_PLACEMENT": 0,
"PL_TARGET_DENSITY": 0.55,
```

Figure 15 Sample config.json WITH pre-hardened macros

NOTE: If you are using pre-hardened macros, you MUST specify macro placement in a macro.cfg config file under the folder for your openlane hardening config.



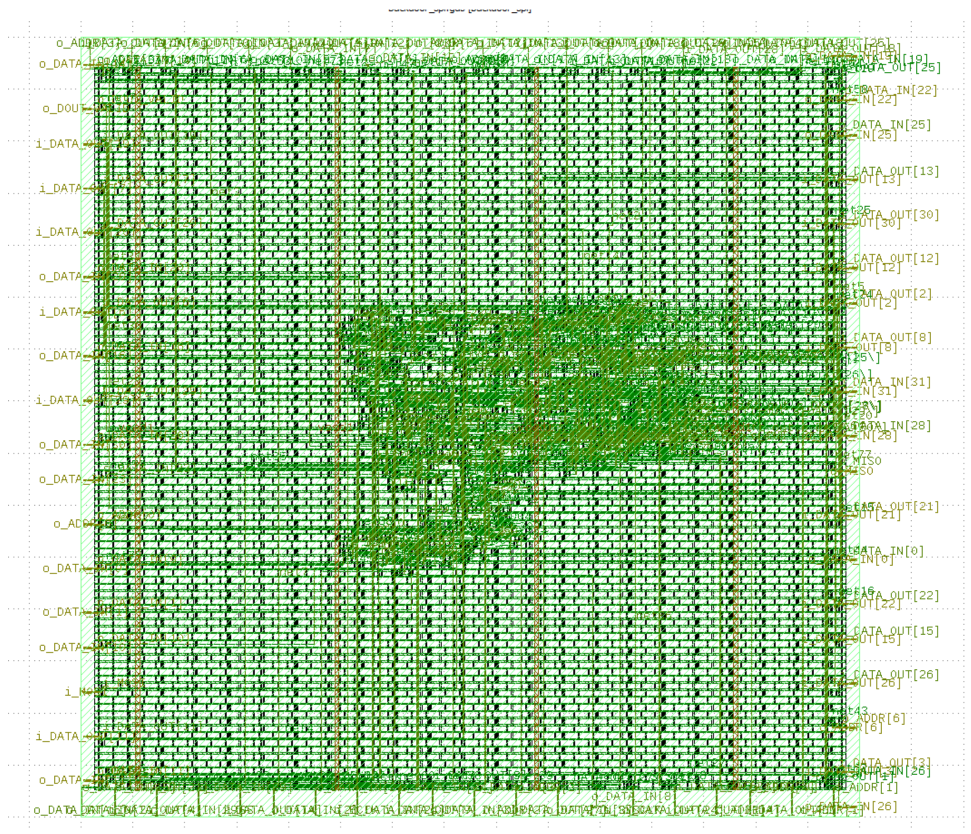
openlane > user_project_wrapper > macro.cfg

```
mprj 1175 1690 N
```

Figure 16 Sample /openlane/user_project_wrapper/macro.cfg

Useful Links: https://github.com/The-OpenROAD-Project/OpenLane/blob/master/docs/source/reference/configuration.md

The generated GDS files can be viewed in KLayout. After opening KLayout, select File, Open, and then navigate to the caravel/gds/MODULE.gds to open the hardened macro.



Sample GDS View in KLayout

RTL Design

A functional RTL design can be written in Verilog and placed under the /verilog/ folder in the caravel repository. Inside of this folder, the following actions should be made to set up a new RTL design:

1. /verilog/rtl
   a. Add the new Verilog designs, can also be in own folder path under /verilog/rtl
2. /verilog/includes/includes.gl.caravel_user_project
   a. Add "-v $(USER_PROJECT_VERILOG)/gl/design/MODULE.v"
3. /verilog/includes/includes.gl+sdf.caravel_user_project
   a. Add "$USER_PROJECT_VERILOG/gl/design/MODULE.v"
4. /verilog/includes/includes.rtl.caravel_user_project
   a. Add "-v $(USER_PROJECT_VERILOG)/rtl/design/MODULE.v"

NOTE: For each of the verilog files under /verilog/includes, there MUST be an empty line after the last entry, or the make files for the caravel repository will break.

Gate Level Simulations

Gate Level simulations are very easy to run if your design has followed the above steps for RTL implementation, RTL simulation, and Module Hardening. At this point, the only thing left to do is run the make verify command with the gl tag, as seen below.

NOTE: To run gate level simulations, your design must be successfully hardened. This is because a netlist is generated based on the SkyWater PDK Standard Cell Library, where each of the functional logic gates are pulled from.

To run a GL level simulation from root: "make verify-<module>-gl"

GTKWave can be used to verify your gate level design also, which can be used with the vcd file GL-MODULE.vcd.

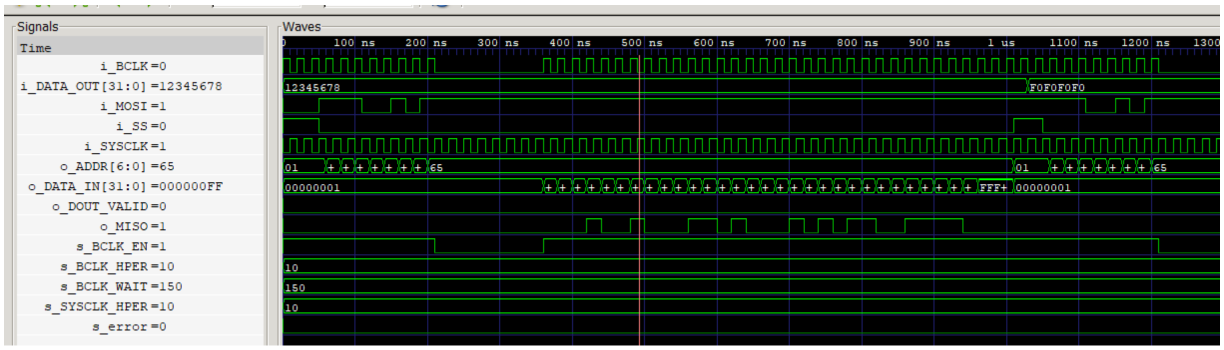


Figure 17 GL Simulation GTKWave Sample

Custom Cell Layout – Magic

Magic

Rationale: Instead of trying to draw the custom cell directly in magic which seemed very imprecise drawing each rectangle by hand, I opted to write a tcl script which would draw rectangles at specific coordinates. In retrospect, the .mag format is simple enough, I could probably have just written the .mag file by hand. However, then the tcl script got too unwieldy, so I created a python script to generate the .tcl file so I could use variables to define the box coordinates easier. This will probably make someone used to TCL cringe, but it worked. Magic must also be built from source, APT version is too old (https://github.com/RTimothyEdwards/magic)

Resources

https://isn.ucsd.edu/courses/beng207/lectures/Tim_Edwards_2021_slides.pdf

https://skywater-pdk.readthedocs.io/en/main/rules/periphery.html#npc

https://skywater-pdk.readthedocs.io/en/main/rules/layers.html

Generation flow

1. Run `make setup` to fetch the pdk
2. Copy dependencies/pdks/sky130A/libs.tech/magic/sky130A.magicrc to `.magicrc` in the cell generation working directory (`cell_gen`).
3. Run the python script to generate the .tcl file: `python3 NAND.py`
4. Open Magic by `cd`ing into cell_gen and running `magic`. Ensure the PDK environment variables are set.
5. Run the tcl script: `source NAND.tcl`. Make sure you're completely zoomed out when you run that, sometimes if you have zoomed into a certain portion of the design, only that part of the design actually gets erased correctly.
6. Update the JSON file in openlane (see below), and create a blackbox verilog file (see Creating the Blackbox below)
7. Run the flow (`make user_proj_final`). The GDS and LEF files were automatically inserted into openlane, and the .mag file has been saved for future reference.

LEF Class

There are two main LEF classes, BLOCK and CELL. The custom cell was designated as a CELL, so it will be placed in a row automatically by the detailed placer as expected. BLOCK is used for hardened macros (SRAM, user_proj_final, etc) to designate it as a block that should not be placed in a row. There are size constraints on a CELL that are not present in a BLOCK (must fit in an existing row), but BLOCKS cannot be packed as densely.

Creating the Blackbox

A blackbox verilog file is required to represent the custom cell. The inputs do not all have to be used, but it must have all four power pins (and the LEF/GDS should have all four power pins as well). These pin names must match exactly the names in the LEF/GDS file. A template from our NAND gate is below. One very important feature is the `/// sta-blackbox` which identifies this as a black box for static timing analysis and LVS check. List this in the openlane JSON file with the gds/lef files (VERILOG_FILES_BLACKBOX, EXTRA_LEFS, and EXTRA_GDS_FILES).

```
`default_nettype none

/// sta-blackbox

`celldefine
module NAND (
   output X,
   input A,
   input B,
   `ifdef USE_POWER_PINS
      inout VPWR,
      inout VGND,
      inout VPB,
      inout VNB
   `endif
);
   assign X = ~(A & B);

endmodule
`endcelldefine

`default_nettype wire
```

Tech Issues

The provided SKY130A tech file has an issue that makes all custom cells fail DRC. By DRC rules, two diff layers must have a gap of 27μm. NSDM/PSDM layers must have either no gap or a gap larger than some amount. In the standard cells, the NSDM/PSDM layers are 13.5μm from the end of the DIFF layers, but in the current SKY130A tech file, the NSDM/PSDM layers are auto-generated as a bounding box with a border of 12.5μm. Therefore, if you put the DIFF layers close enough for the PSDM layers to have no gap, the DIFF layers are too close, but if you put the DIFF layers further away, there is a gap in the NSDM layers. To fix this, edit the sky130A tech file in magic to change the 125's to 135's. Because this is re-generated after make setup, you will have to change this whenever using magic after make setup has been run.

templayer basePSDM pdiffres,mvpdiffres	templayer basePSDM pdiffres,mvpdiffres
grow 15	grow 25
or xhrpoly,uhrpoly,xpc	or xhrpoly,uhrpoly,xpc
grow 110	grow 110
bloat-or allpactivetap * 125	bloat-or allpactivetap * 135
allnactivenontap 0	allnactivenontap 0
bloat-or allpactivenontap * 125	bloat-or allpactivenontap * 135
allnactivetap 0	allnactivetap 0
templayer baseNSDM ndiffres,mvndiffres	templayer baseNSDM ndiffres,mvndiffres
grow 125	grow 135
bloat-or allnactivetap * 125	bloat-or allnactivetap * 135
allpactivenontap 0	allpactivenontap 0
bloat-or allnactivenontap * 125	bloat-or allnactivenontap * 135
allpactivetap 0	allpactivetap 0

Custom Cell Layout – XSchem

XSchem

XSchem is a schematic capture tool which will use the base models in the sky130PDK to allow you to simulate a custom cell from basic building blocks. To use XSchem, install from https://xschem.sourceforge.io/stefan/index.html, determine a working directory, and copy the dependencies/pdks/sky130A/libs.tech/xschem/xschemrc file into `.xschemrc`. Ensure the PDK_ROOT environment variable is set, then run `xschem` from the folder with `.xschemrc`. This will load the sky130A PDK into XSchem.

Useful Keyboard Shortcuts

U = Undo

Shift + U = Redo

Shift + I = Insert

C = Copy

M = Move

W = Wire

Shift + W = Snap Wire

Resources on Generating Simulations

https://xschem.sourceforge.io/stefan/xschem_man/graphs.html

https://xschem.sourceforge.io/stefan/xschem_man/tutorial_run_simulation.html

NGSpice Manual

https://ngspice.sourceforge.io/docs/ngspice-41-manual.pdf

SKY130 Inverter Reference

http://web02.gonzaga.edu/faculty/talarico/vlsi/xschemTut.html

Running a simulation

Generate a netlist by clicking the Netlist button in the upper right corner, or press `n`.

Run the simulation by clicking the Simulate button in the upper right corner, or run ngspice manually.

Load the waveform by holding ctrl and clicking the Load Waves button to load the waves from the .raw file produced by the simulation.

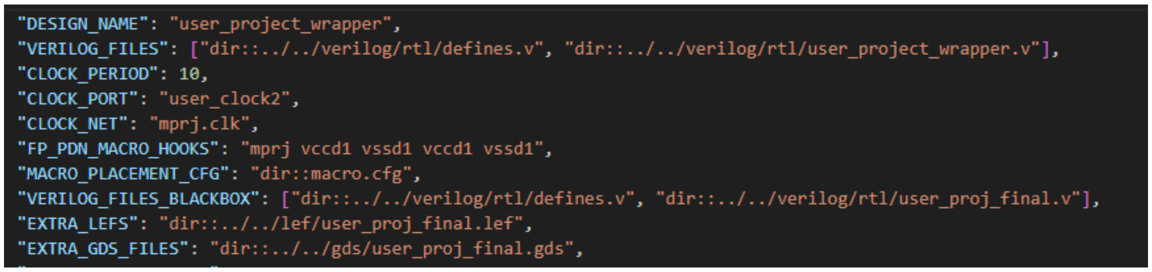
Double-click the graph body to change the nets shown. Digital mode will stack the graphs as separate waveforms instead of overlapping in space.

Top Level Wrapper Design

Once all the individual functional designs are complete, you will need to instantiate your design inside of the user_project_wrapper Verilog file. It is recommended that you create a wrapper module to be instantiated inside of the user_project_wrapper, so that you can harden this design WITHOUT the set requirements of the user_project_wrapper, which is used as the top level design for the submission and precheck.

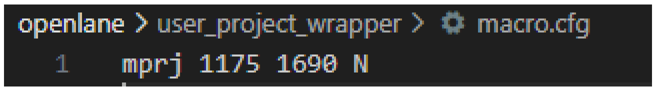
The user_project_wrapper has its own specific set of hardening configurations set up under the path /openlane/user_project_wrapper. Changing this should be taken with care, since this is used to generate the submission precheck and is used as the top level result for your fabricated design. What you SHOULD edit is the following:

1. /openlane/user_project_wrapper/config.json
    a. If NOT using pre-hardened macro:
        i. VERILOG_FILES: include top level module design in wrapper
    b. If using pre-hardened macro:
        i. VERILOG_FILES_BLACKBOX: include top level module design in wrapper
        ii. EXTRA_LEFS: include top level module design in wrapper
        iii. EXTRA_GDS_FILES: include top level module design in wrapper
2. /openlane/user_project_wrapper/macro.cfg
    a. If using pre-hardened macro:
        i. Specify center location of prehardened macro



```
"DESIGN_NAME": "user_project_wrapper",
"VERILOG_FILES": ["dir::../../verilog/rtl/defines.v", "dir::../../verilog/rtl/user_project_wrapper.v"],
"CLOCK_PERIOD": 10,
"CLOCK_PORT": "user_clock2",
"CLOCK_NET": "mprj.clk",
"FP_PDN_MACRO_HOOKS": "mprj vccd1 vssd1 vccd1 vssd1",
"MACRO_PLACEMENT_CFG": "dir::macro.cfg",
"VERILOG_FILES_BLACKBOX": ["dir::../../verilog/rtl/defines.v", "dir::../../verilog/rtl/user_proj_final.v"],
"EXTRA_LEFS": "dir::../../lef/user_proj_final.lef",
"EXTRA_GDS_FILES": "dir::../../gds/user_proj_final.gds",
```

Figure 18 Sample /openlane/user_project_wrapper/config.json



```
mprj 1175 1690 N
```

Figure 19 Sample /openlane/user_project_wrapper/macro.cfg

SRAM Usage and Integration

eFabless SRAM

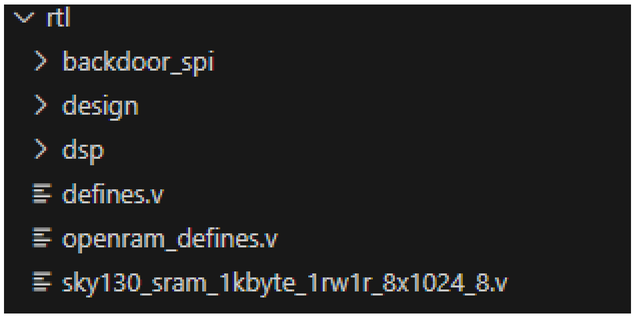
Utilizing onboard SRAM in the user area can have massive benefits over the DRAM integrated with the management microcontroller. The provider for the memory designs used in the eFabless OpenPDK is another open-source project called OpenRAM. This open-source project is a memory compiler that is capable of building memory macros that you can utilize in your designs. The sddec23-06 did not do a lot of work with the OpenRAM project other than using their precompiled macros. There is a great deal of customization that is available in the project and is something worth looking at in the future.

Sourcing the memory design

Now that you have decided to use SRAM in your design the first choice that you must make is where to source your prehardened SRAM macro. There are three primary locations where you can get these designs from. The first is to use the actual OpenRAM tool to generate your own custom design. This method will give you the most flexibility as it will allow you to directly create what you need for your specific design. The only drawback to this method is that it is another tool that you must learn to utilize the memory. The second location is in the OpenPDK that is downloaded when you setup your project. The PDK has four simple designs that you can utilize in your project. This is the most convenient method of getting memory into your project as it is included in the PDK already. However, the one downside to this method is that your memory selection is limited along with bugs existing in PDK version; some of the memories have been bugged for a while and it does not appear to be a priority for eFabless to push out the fixed designs by the OpenRAM team. The third and final way to source your memory designs is to use the designs in the second OpenRAM test chip https://github.com/VLSIDA/openram_testchip2. This was a test chip designed by the creators of OpenRAM and implements eleven different memory designs. This was the best method that I found for sourcing memory designs as it provided a wide range of designs to choose from. These designs also have the benefit of being designed and verified by the creators of the open-source project; the designers also manually went through the designs by hand to ensure that they would pass future drc and other hardening checks.

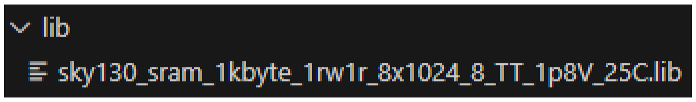
Bringing the files into your design

Once you have selected your design the next step is to bring all the needed files int your project. The four files that you need are the .gds .lef. .lib and .v files. They should all be place in the corresponding folders with the Verilog file placed in your rtl folder.



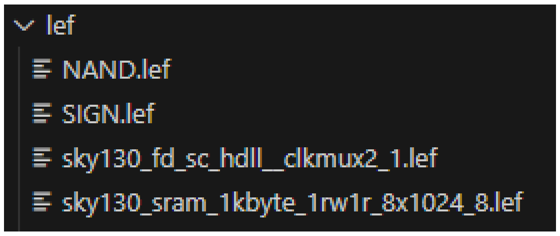
```
⌄ rtl
  › backdoor_spi
  › design
  › dsp
  defines.v
  openram_defines.v
  sky130_sram_1kbyte_1rw1r_8x1024_8.v
```

Figure 20 RTL location



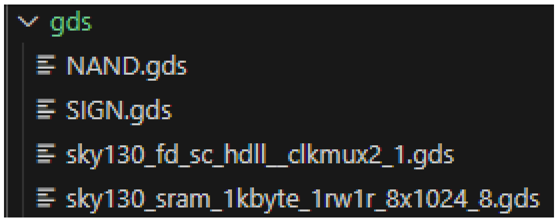
```
lib
  sky130_sram_1kbyte_1rw1r_8x1024_8_TT_1p8V_25C.lib
```

Figure 21 Lib location



```
lef
  NAND.lef
  SIGN.lef
  sky130_fd_sc_hdll__clkmux2_1.lef
  sky130_sram_1kbyte_1rw1r_8x1024_8.lef
```

Figure 22 Lef location



```
gds
  NAND.gds
  SIGN.gds
  sky130_fd_sc_hdll__clkmux2_1.gds
  sky130_sram_1kbyte_1rw1r_8x1024_8.gds
```

Figure 23 Gds location

With all these folders added to your design you are ready to start integrating the memory into your rtl design.

RTL design

Implementing and using the memory is straight forward using the functional model provided in the design Verilog file. The first thing you must do though while designing with the memory is to place it in the user_project_wrapper and to interact with it there. The reasoning behind this will be explained in the hardening section as that is the time where the SRAM location matters. With the memory placed in the correct location you can start to hook up ports to the design. The first two that need to be connected are the power and ground connections called vccd1 and vssd1. These are important connections and need to be connected to either the vccd1 and vssd1 or vccd2 and vssd2 provided by the wrapper. The next ports to hook up are the clock ports to your desired clock signal. From there the next port to hook up is the chip select line to enable the memory. The next port to connect is the write enable port that will control whether the memory is reading from or writing to memory. From there we can hook up the address port to your address driver. The last two ports to hook up are the data in and data out ports that provide your data connection to the memory.

One niche thing about the memory is that it will have a three-cycle delay after the address is read in before the data is stored in the memory or able to be read from the memory. You must keep this in mind while designing with the memory. With this caveat in mind and having the memory ports hooked up you can implement it in your design however you see fit.

Hardening the SRAM

Once you have completed your rtl design with memory the next step is to go through the hardening process. The first step in this process is to modify / replace the config.json file for the user_project_wrapper hardening process. I would suggest that you use the config.json file from either sddec23-06's repo or from the second OpenRAM test chip https://github.com/VLSIDA/openram_testchip2. The reason for this is because there are multiple settings that need to be configured so that the memory will properly harden. During our initial attempts we were struggling to get the memory to properly hook up to the power rails. This issue was caused by the fact that we were instantiating the memory inside of another macro. This was problematic as the SRAM needs access to the top metal layer met5 for its power connections. When the macro was connected to the met5 layer it was able to be placed into the user_project_wrapper.

The main items that need to be set in the file are FP_PDN_CHECK_NODES, FP_PDN_ENABLE_RAILS, RUN_FILL_INSERTION, and RUN_TAP_DECAP_INSERTION. With all of these sets the memory should be able to properly harden and you should be able to see in the created gds file that the power rings of the memory module are connected to top met5 power layer.

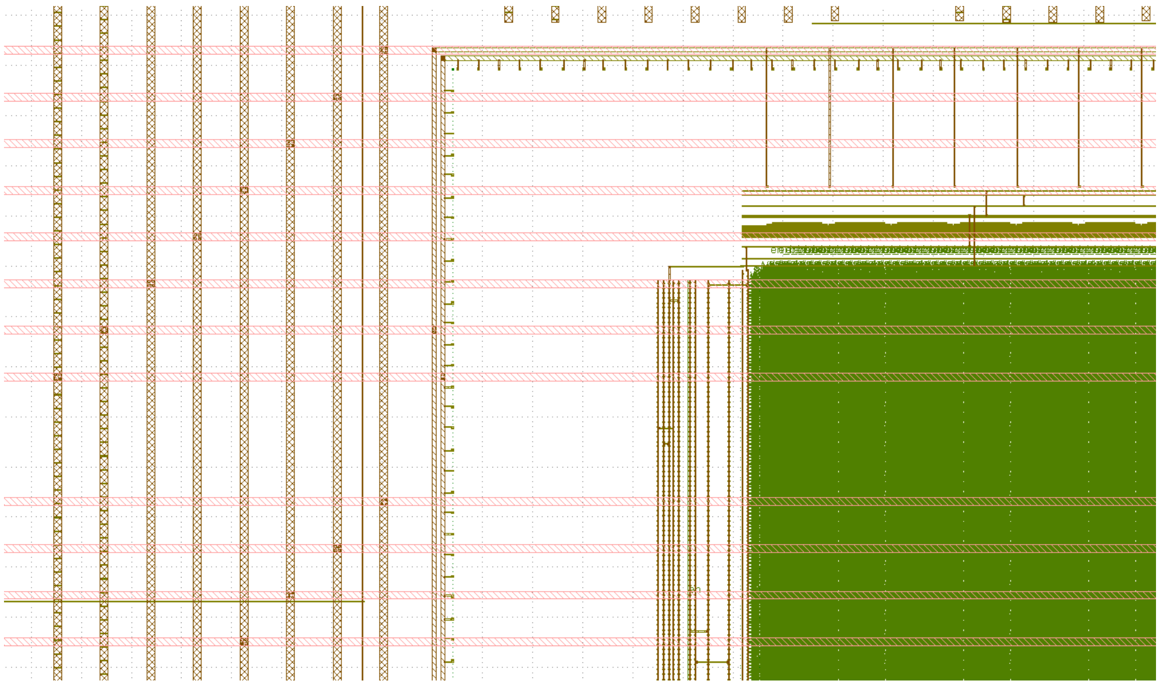


Figure 24 Power Rail Connection

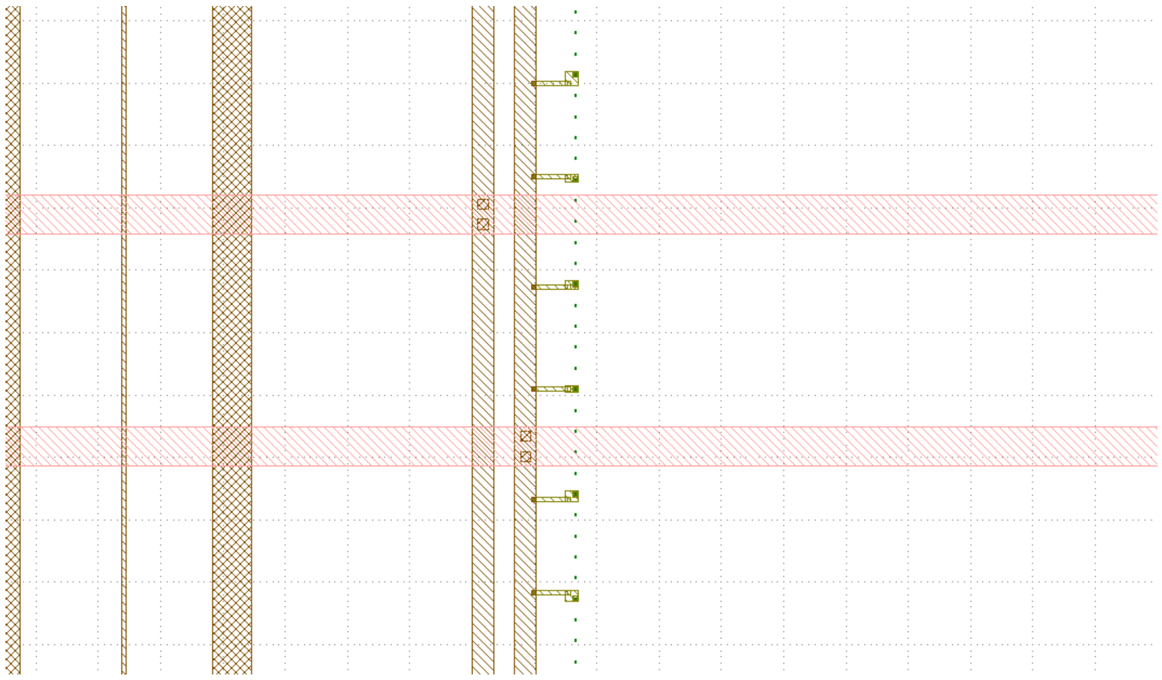


Figure 25 Zoomed In Power Rail Connection.

EFabless Submission Precheck

The MPW Precheck is a SEPERATE Github repository that can be cloned using the root makefile in the caravel repository. This is used to compare the hardened result of user_project_wrapper, our top-level module, against multiple different requirements for project submission. As long as the design fits all functional requirements, AND passes this precheck, a design is ready for submission.

To clone the MPW precheck github repository: ‘make precheck’

To run the MPW Precheck: ‘make run-precheck'

NOTE: The top-level wrapper user_project_wrapper must be hardened with ‘make user_project_wrapper’ before a precheck can occur.

Modifying project to pass precheck parameters:

1. Documentation Pass
    a. The root README.md file must be modified to ensure documentation has been updated

   b. For a minimum pass, delete all existing contents and add custom header title
2. GPIO Pass
   a. Update GPIO INIT config from INVALID for following file:
      i. sddec23-06/verilog/rtl/user_defines.v
   b. Can set to either inputs, outputs, bidirectional, or analog pins

Troubleshooting Notes

LINTER: Mixing positional and .*/named instantiation connection

- `ifdef USE_POWER_PINS must be the first items in the module port list, for some reason it gets angry if you put the power pins last.

Supported Verilog (* attributes *)

- https://github.com/The-OpenROAD-Project/yosys/blob/master/README.md#verilog-attributes-and-non-standard-features

What is maglef vs lef?

- Best guess is according to https://github.com/The-OpenROAD-Project/OpenLane/issues/1067, some of the older PDK's had a set of LEF issues, maglef is LEF files re-exported using Magic to hopefully fix most of those issues.

8.4.2 Bring up Plan

Overview

This is a growing document which includes portions of and builds off a previous senior design team's work, sdmay23-28. This document includes a basic Nucleo test plan, and expands to fit our specific design.

Purpose

The purpose of this document is to guide the students on how to test Efabless project from our team. This document provides firmware examples, flash programming and diagnostic tools for testing Open MPW and chipIgnite projects using Caravel. It also provides schematics, layout and gerber files for PCB evaluation and breakout boards.

Development Board Description:

The larger green board below is the Caravel Development Board, which contains:

- Reset button
- USB to micro-B programming port
- 38 GPIO ports
- External Reset
- External power connections
- M.2 Edge connector for Caravel Breakout Board
- Jumper connectors to determine power sources

The Caravel Development Board contains two 28 pin connectors on both sides of the board, which including 38 GPIO pins, power connections, and an external clock connection. These GPIO pins can be used to interface with our Backdoor SPI interface.

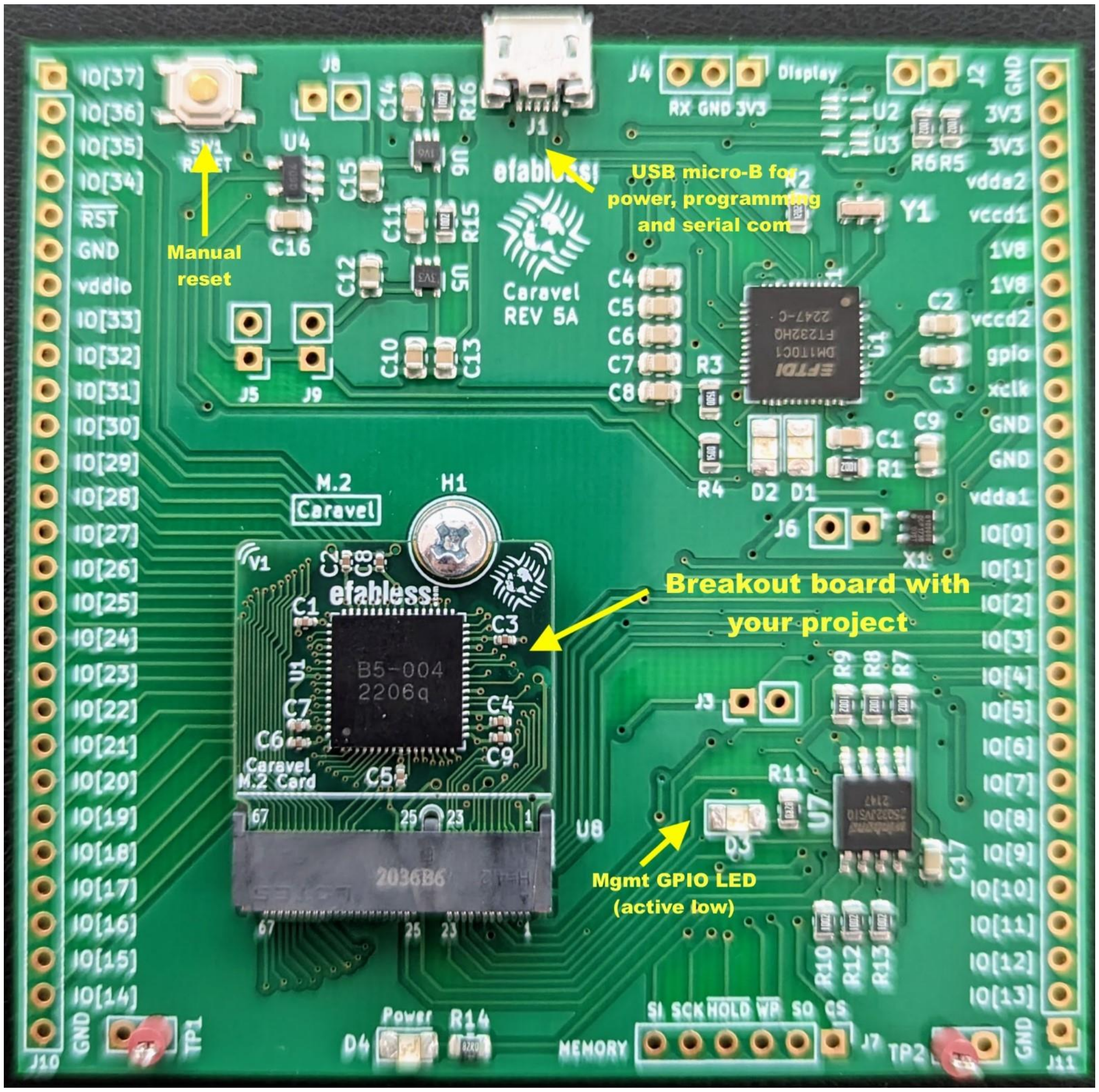

Manual reset
USB micro-B for power, programming and serial com
Breakout board with your project
Mgmt GPIO LED (active low)

Figure 26 Efabless Development Board and Breakout Board[1]

I/O Connections

Notice:

- The clock is driven by X1 with a frequency of 10MHz. To drive the clock with custom frequency, disable X1 through installing J6 and use the external pin for xclk
- The voltage regulator U5 and U6 supply 1.8V and 3.3V through J8 and J9. The traces have to be cut if they are supplied externally.
- vccd1 is connected to 1.8V through J3. The trace has to be cut if it is supplied externally
- vddio is connected to 3.3V through J5. The trace has to be cut if it is supplied externally

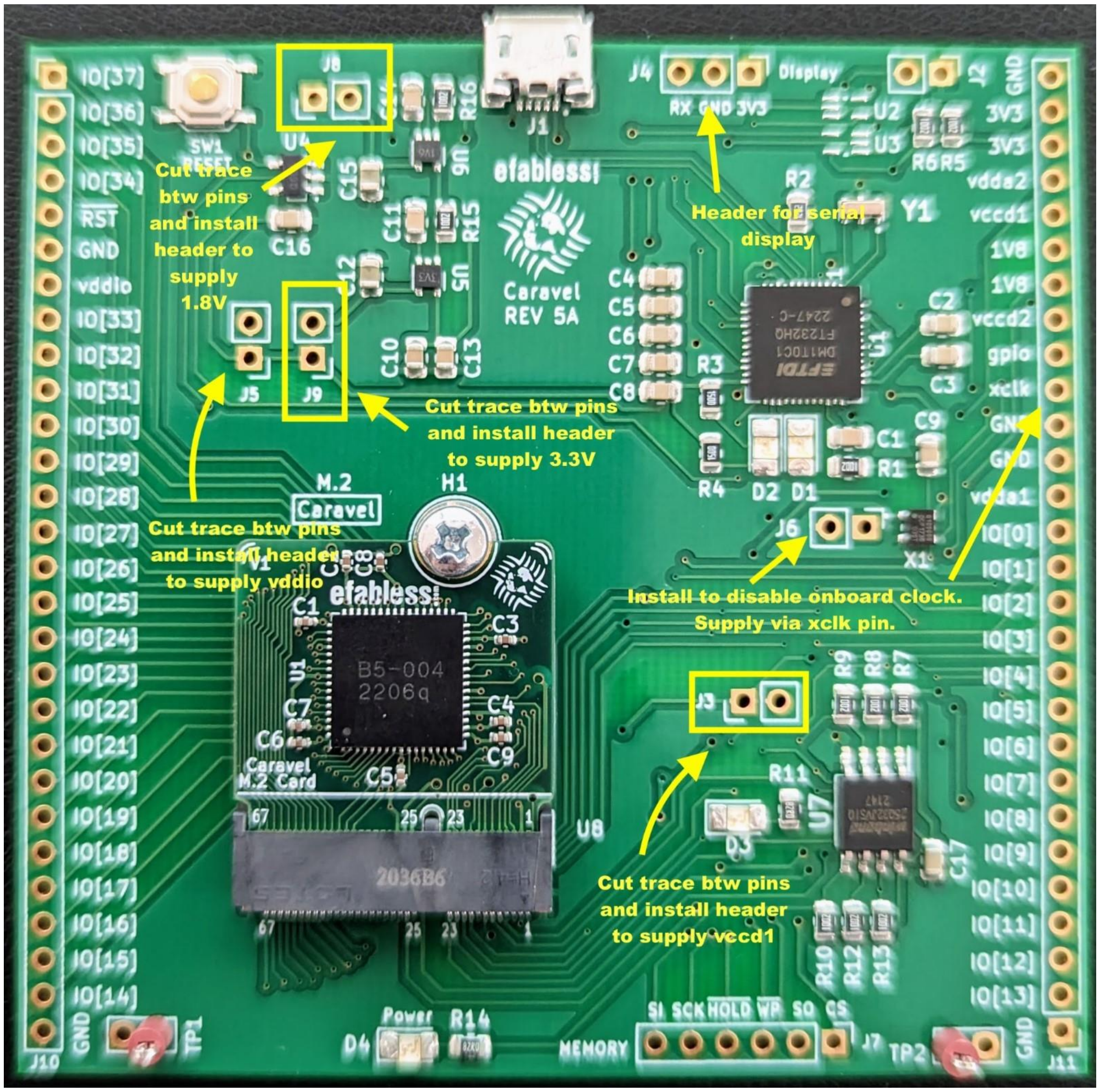

Cut trace btw pins and install header to supply 1.8V
Header for serial display
Cut trace btw pins and install header to supply 3.3V
Cut trace btw pins and install header to supply vddio
Install to disable onboard clock. Supply via xclk pin.
Cut trace btw pins and install header to supply vccd1
J8
J5
J9
J6
J3
J4
Caravel REV 5A
efabless

Figure 27 Efabless Development Board Jumper Labels[1]

Hardware Test

Test Procedure:

1. Set multimeter to Continuity mode and ensure there are no short circuits between 3.3V and GND
2. Set multimeter to Continuity mode and ensure there are no short circuits between 1.8V and GND
3. Insert USB to Micro-B connector into both the users laptop and the J1 connector on the top of the Evaluation board to supply power
4. Set multimeter to DC Voltage mode and verify J8 reads 1.8V between J8 and GND
5. Set multimeter to DC Voltage mode and verify J9 reads 3.3V between J9 and GND

Software Setup

Setup Procedure:

1. Install the Caravel Breakout Board onto the Evaluation Board
    a. Remove the screw (M1) to add/remove the breakout board module
    b. Insert the Caravel Breakout Board into a ZIF connector at a 45 degree angle
    c. Press the Caravel Breakout Board down so that it is flush with the Development Board, and turn the screw (M1) down
2. Connect the Master SPI IO pins to J11
    a. SPI Master Clock: IO[5]
    b. Slave Select Reset: IO[6]
    c. Master Out Slave In (MOSI): IO[7]
    d. Master In Slave Out (MISO): IO[8]
3. Connect the DSP Accelerator IO pins to J11
    a. DSP Weight ACK: IO[9]
    b. DSP Data ACK: IO[10]
    c. DSP Convolution ACK: IO[11]
4. Insert USB to Micro-B connector into both the users laptop and the J1 connector on the top of the Evaluation board

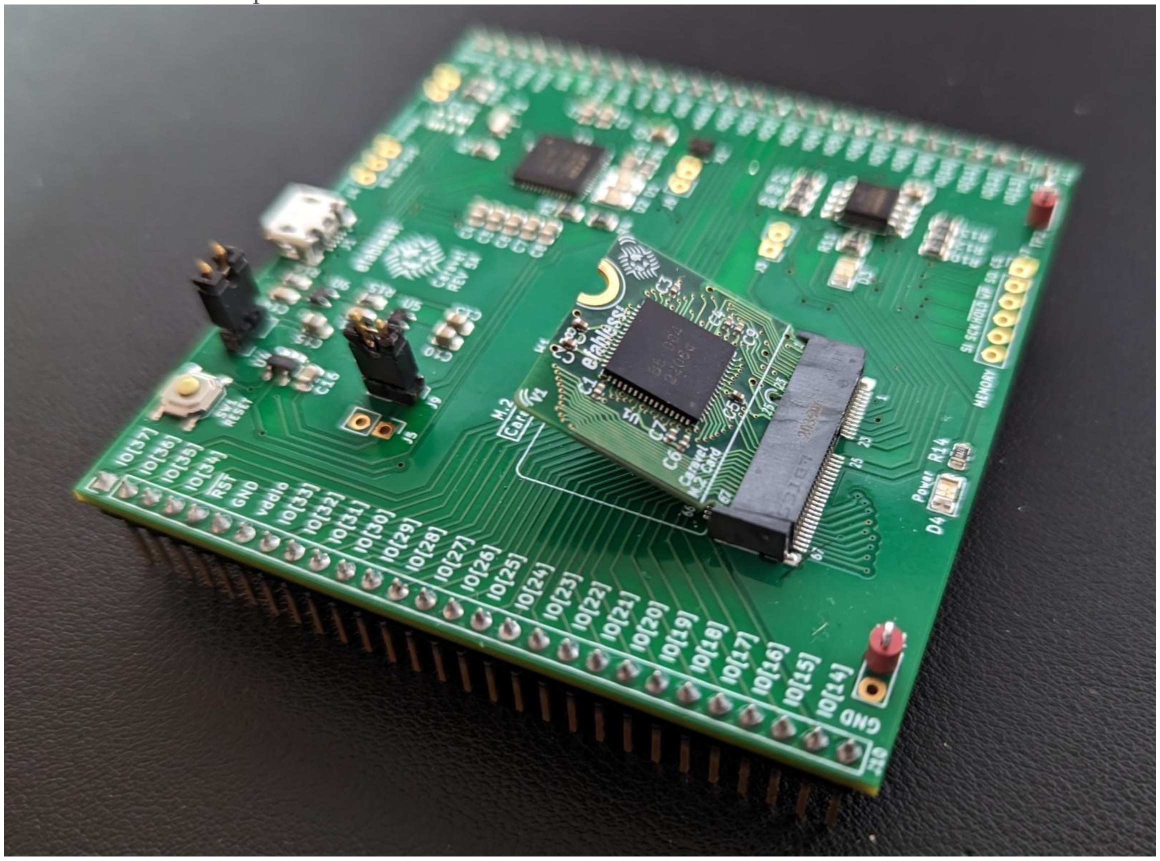


Figure 28 Efabless Development Board Installation[1]

Software Test

We have not yet received a Nucleo to test with, so these instructions will require some debugging with reference to the Nucleo board and its interface to the management SoC.
Use https://github.com/efabless/caravel_board/tree/main/firmware/chipignite#readme to program the board. This repository contains makefiles and instructions for interfacing with the board.

1. Use the template above to run the basic blink test. Follow the README instructions, running `make flash` in the blink folder. Make sure the GPIOs blink as expected from that program before continuing.
2. Copy the blink program and rename it to c_test. Copy the c_test.c file from verilog/dv/c_test in the project repository. This file contains example code for every module and examples of using every module from the management SoC. You will need to edit the makefile and remove isr.c from the list of source files as c_test provides its own interrupt handler.
3. Use the Nucleo programming guide to use the clkmux with the EXTCLK input pin to try clocking the wishbone bus externally. Verify using a simple program in C on the management SoC.
4. Use the Nucleo programming guide to interface with the user project via SPI to the wishbone test, then the DSP module.
5. Verify the clock frequency of the user project, and increase until the DSP module fails to function to determine the physical maximum clock frequency.
6. Reference the Design Document and experiment with any other experimental questions you may have.

Module Definitions

For a full description of how each evaluated module should function, please reference Appendix 8.4.4. For a detailed description of the Backdoor SPI interface, please reference Appendix 8.4.7.

Reference

[1] Efabless. (n.d.). Efabless/caravel_board. GitHub. Retrieved April 17, 2023, from https://github.com/efabless/caravel_board

8.4.3 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 team 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

8.4.4 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.

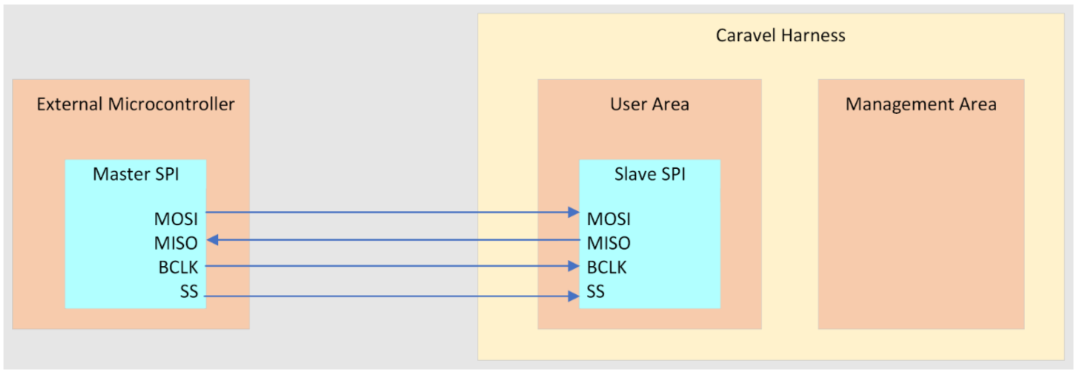


Figure 29 External SPI Connection

Voice Road Noise Isolation Accelerator (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
- 8 bit multiplier
- 32 bit accumulator

Implementation Description

The Voice Road Noise Isolation Accelerator (DSP) module will implement an accelerator for 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 for the accelerator. 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 accelerator 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.

In order for this module to be able to run in real time we had to ensure that our chip speed and design of module were compatible and capable of running in real time. The primary requirement for this to work is that it must be able to handle the required sampling rate for the human voice. This is a sampling rate of 8 kHz. With that in mind we next looked at the operation that we needed to accelerate. This operation was a 1-D convolution that was 1024 values long. This means that with our chosen implementation of 1 processing unit it will take 1024 cycles for the operation to complete. Finally, we can multiple our sampling rate but the number of cycles per operation to get our required speed of 8.2 MHz. After working with the chip we decided on a clock speed of 10 MHz that would give us enough cushion for the other modules to operate in the chip.

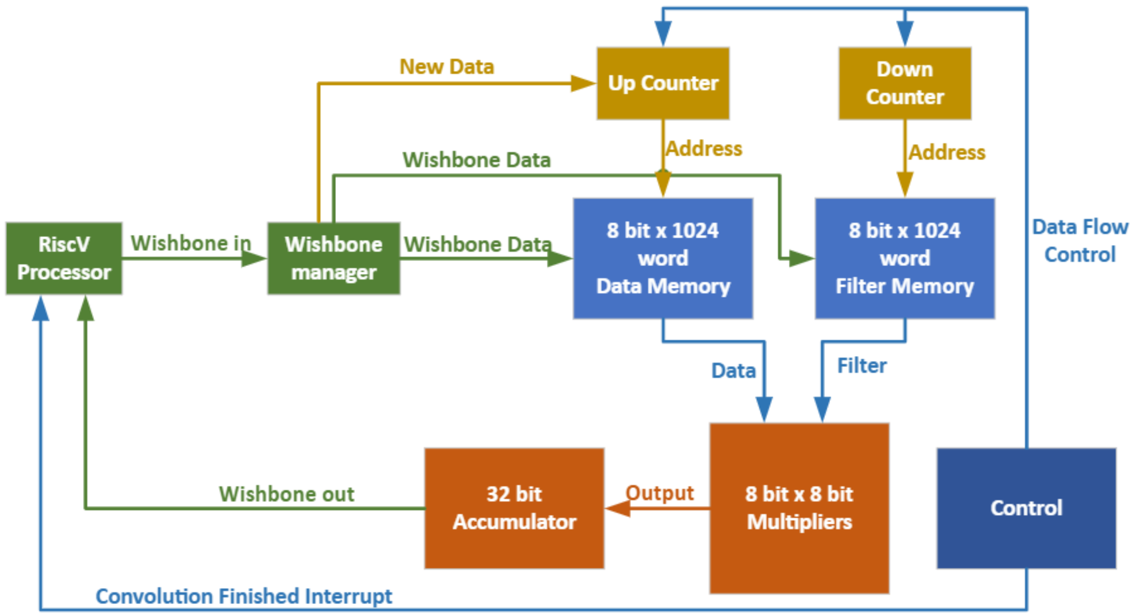


Figure 30 Voice Road Noise Component Diagram

Clock Gating Module

Primary Author: Cade Breeding

Inputs

- MGMT_CLK – Management Clock
- EXTCLK – External Clock
- Wishbone Test Clock Input
- Wishbone Test Select
- DSP Clock Input
- DSP Select

Outputs

- Wishbone Test Clock
- DSP Clock

Basic Components

- Three clkmux instances

Implementation Description

- This consists of three clkmux instances. The first will multiplex between the clock provided by the built in management SoC and an external clock on an IO pin to provide a system clock to the user project. There are also two other clkmux instances which will multiplex between the system clock and an individual LA probe pin to allow each module to be clocked manually (or disconnected from a clock altogether). This allows for clock multiplexing by switching between clock sources and clock gating by connecting the modules to either an unchanging LA probe pin or the external clock and driving the external clock pin to a constant value.

Standard Cell Test

Primary Author: Cade Breeding

Inputs

- A – Input 1 to standard cell AND gate
- B – Input 2 to standard cell AND gate

Outputs

- C – output of one standard cell AND gate

Basic Components

- AND gate cell from the primary Skywater standard cell library

Implementation Description

- Test propagation delay2 4 inputs A and B, 1 AND gate and output to 1 pin. All pins are LA probes, 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 NAND cell.

Implementation Description

- Three pins, two inputs, one output of a NAND operation between the two input pins.

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.

8.4.5 Top Level Pin Assignments

Table 4 Logic Analyzer Pins

Logic Analyzer					
Port Type	**Bits**	**Index**	**Module**	**Signal**	**Notes**
Output	[31:0]	la0	WB Test	wb_dat_o	WB Test Count Value
	[63:32]	la1	DSP	o_DSP_data	DSP Output
	64	la2:0	std cell AND	X	Output AND
	65	la2:1	custom Cell	X	Custom NAND out
	66	la2:2	DSP	dsp_weight_ack	DSP Weight Ack
	67	la2:3	DSP	dsp_data_ack	DSP Data Ack
	68	la2:4	DSP	dsp_conv_ack	DSP Conv Ack
	69	la2:5	Clk Mux	clkmux_clk_s	MUXed clock output
Input	70	la2:6	std cell AND	A	AND op 1
	71	la2:7	std cell AND	B	AND op 2
	72	la2:8	clk mux 1	S	Sel Line for clk mux
	73	la2:9	clk mux 2	A1	Alt Clock for WB
	74	la2:10	clk mux 2	S	Sel Line for WB CLK
	75	la2:11	clk mux 3	A1	Alt Clock for DSP
	76	la2:12	clk mux 3	S	Sel Line for DSP CLK
	77	la2:13	custom Cell	A	Custom NAND Op 1
	78	la2:14	custom Cell	B	Custom NAND Op 2
	79	la2:15	DSP	i_RST	Reset for DSP
N/A	[89:80]	[89:80]	Unused		
Input	[127:90]	[127:90]	IO_oeb		

Table 5 IO Pins

IO				
Port Type	**Bits**	**Module**	**Signal**	**Notes**
N/A	0	TEST	N/A	Firmware Test Start/Stop
N/A	1	TEST	N/A	Firmware Test Pass/Fail
N/A	[4:2]	Unused		
In	5	SPI	i_BCLK	Master Clock
In	6	SPI	i_SS	Slave Select Reset
In	7	SPI	i_MOSI	Master Out Slave In
Out	8	SPI	o_MISO	Master In Slave Out
Out	9	DSP	dsp_weight_ack	DSP Weight Ack
Out	10	DSP	dsp_data_ack	DSP Data Ack
Out	11	DSP	dsp_conv_ack	DSP Conv Ack
N/A	[37:12]	Unused		

Table 6 Interrupts

Interrupts			
Line	**Module**	**Signal**	**Notes**
0	DSP	dsp_conv_ack	DSP convolution complete
1	IRQ	wb_module_we_s[2]	Manual IRQ test
2	Unused		

8.4.6 Testing Results

Backdoor SPI

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.

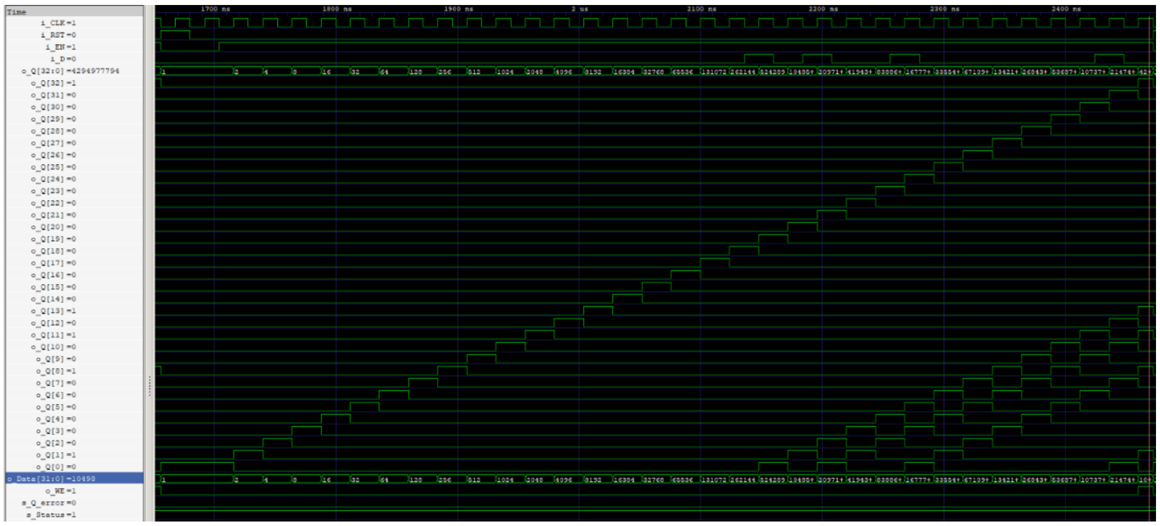


Figure 31 Shift in 10498

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.

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 0x12345678 and 0xF0F0F0F0.

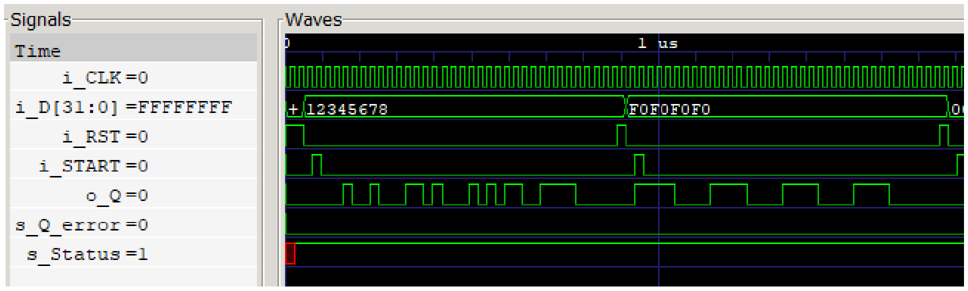


Figure 32 Shift Out Test

DFF Buffer

The following waveform depicts the DFF Buffer module to be placed inside of the external SPI module, to support clock synchronization between the system clock in the user area and the external master SPI bus clock. The first test case was when the enable bit was set to 1, and the data input I_D was set to 1 also. This mimics the expected output after the shift in register finishes receiving data, since the new bit is updated from 0 to 1 for the DFF buffer, leading to propagating 1's throughout the flip flops. A similar test was made with an input of i_D = 1'b0, to verify that no 1's would shift in when not expected. The same set of tests were run with the enable bit cleared, to ensure no bits would be shifted into the buffer. While this case will not apply to our module, it was important to check to ensure functionality since it could be implemented in the future.

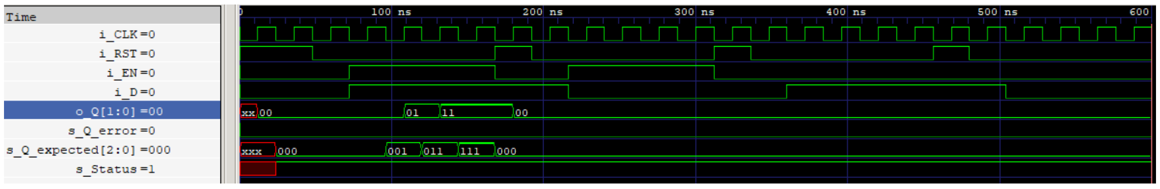


Figure 33 DFF Buffer Waveform Outputs

External SPI Top Level Module

The screenshot below shows a sample read task in the backdoor_spi testbench module. This task begins by shifting in the address field for 8 I_BCLK cycles. Then, a delay is asserted to assure that 2 SYSCLK cycles in the user area can occur for clock synchronization. Next, data is read from I_DATA_IN and is shifted out over o_MISO. After 32 BCLK cycles, the data is validated with the automatic error checking in the read() task.

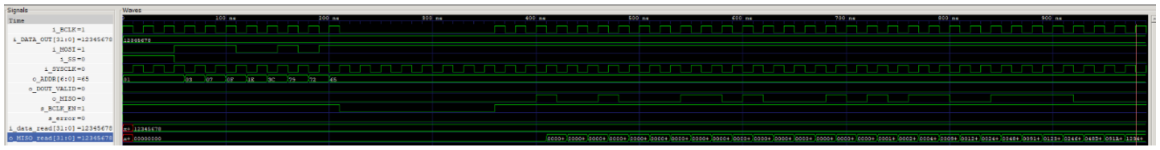


Figure 34 Sample Master SPI Read Transaction

The screenshot below shows a sample write task in the backdoor_spi testbench module. At the beginning of the task the first 8 cycles of I_BCLK are used to shift in the address bits using I_MOSI. This bit field can be seen changing and validated from o_ADDR. After this, data is written for 32 BCLK cycles, where data is shifted in and o_DATA_IN is updated. After waiting 2 SYSCLK cycles for the data to be stable in the user area BCLK domain, o_DOUT_VALID is asserted so that the respective module based on the 7 bit address field could write the 32-bit incoming word.

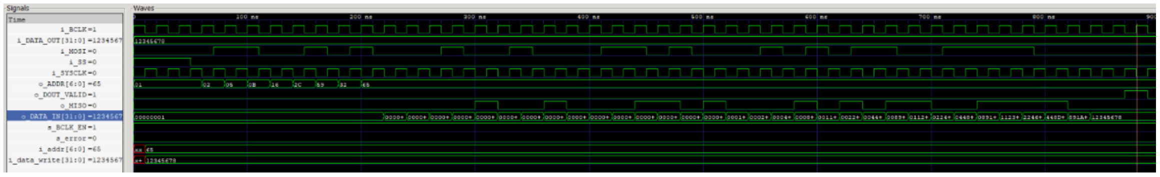


Figure 35 Sample Master SPI Write Transaction

External SPI with Control Bus

The example below shows a Master SPI read transaction with the module_control design wired with the backdoor_spi module. The process follows as listed above, with the exception that the data being read is multiplexed from one of four module data signals, listed as the bottom waveforms s_MODULE_DATA_N. Based on the received address o_ADDR, one of four of these words would be indexed to be shifted out of the user area and to the master SPI interface. Since the address 01 was received, s_MODULE_DATA_0 would become the input for I_MODULE_DATA in the backdoor_spi module, and in turn be shifted out to the Master SPI interface after loaded. By monitoring o_MISO and counting the shifted bits with error validation in the write task, we can verify both that s_MODULE_DATA_0 is loaded into the backdoor_spi module, and that it is shifted out to the master SPI interface.

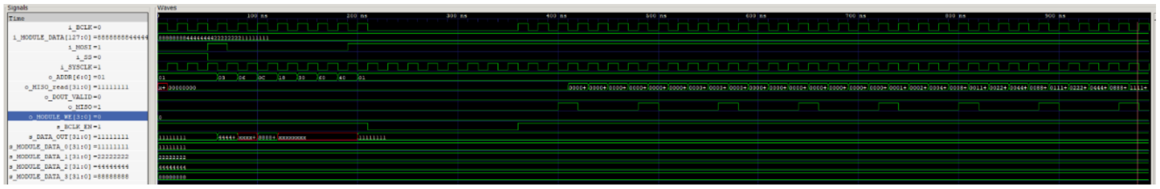

Signals
Waves
Time
i_BCLK=0
i_MODULE_DATA[127:0]=888888884444
i_MOSI=1
i_SS=0
i_SYSCLK=1
o_ADDR[6:0]=01
o_MISO_read[31:0]=11111111
o_DOUT_VALID=0
o_MISO=1
o_MODULE_WE[3:0]=0
s_BCLK_EN=1
s_DATA_OUT[31:0]=11111111
s_MODULE_DATA_0[31:0]=11111111
s_MODULE_DATA_1[31:0]=22222222
s_MODULE_DATA_2[31:0]=44444444
s_MODULE_DATA_3[31:0]=88888888

Figure 36 Master SPI Read with Module Control

Top Level RTL Tests

Top Level RTL tests were run on the integrated top level wrapper module user_project_wrapper, which housed all our modular designs in conjunction with IO and LA port connections.

The standard cell test demonstrates functionality of a 2 input AND gate with inputs A and B for all 4 potential input combinations, with output X. As expected, the output X is asserted to 1 only when A and B both are 1.

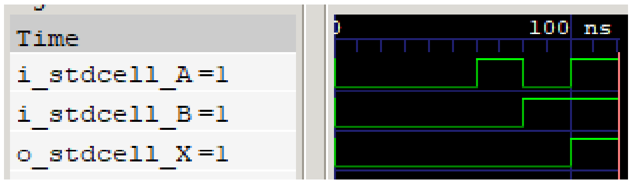


Figure 37 User Project Wrapper – RTL Standard Cell Waveform Results

The custom cell test demonstrates functionality of a 2 input NAND gate with inputs A and B for all 4 potential input combinations, with output X. As expected, the output X is asserted to 0 only when A and B both are 1.

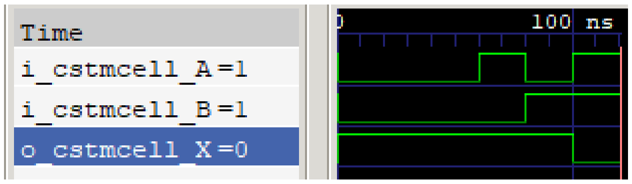


Figure 38 User Project Wrapper – RTL Custom Cell Waveform Results

The Clock MUX waveform results below show how one clock MUX module can select between the SPI Master clock labeled I_BCLK and the provided wishbone bus clock wb_clk_i

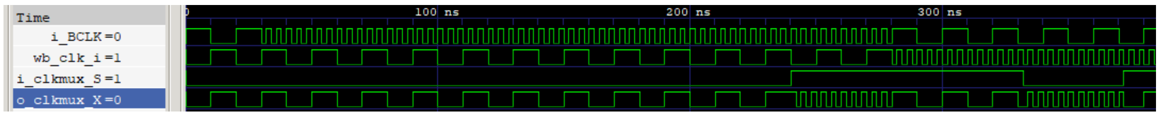


Figure 39 User Project Wrapper – Clock MUX Waveform Results

The results below show a sample SPI Write transaction into the user area design for our chip from an external master SPI interface. We were able to verify that the intended 32 bit word was written from the Master SPI over the I_MOSI I/O pin, and that the data was received and a write enable was asserted for the addressed module for 1 clock cycle.

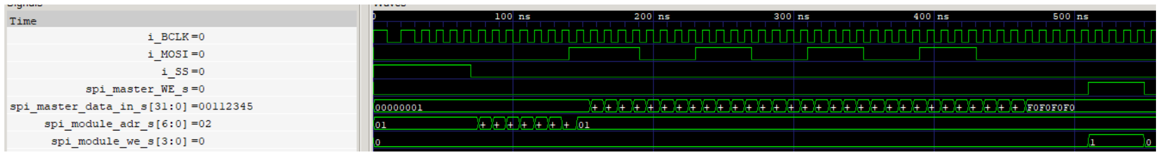


Figure 40 User Project Wrapper – Backdoor SPI Write Waveform Results 1

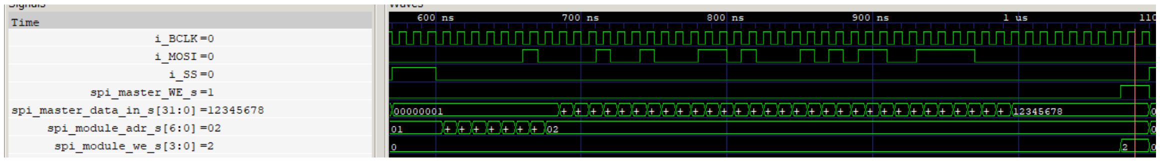


Figure 41 User Project Wrapper – Backdoor SPI Write Waveform Results 2

The below waveform verifies that the Wishbone Test Module can be written to using the Backdoor SPI module and an external interface. After the Wishbone Test Module is written to over SPI, the counter value is set and begins counting from the new value.

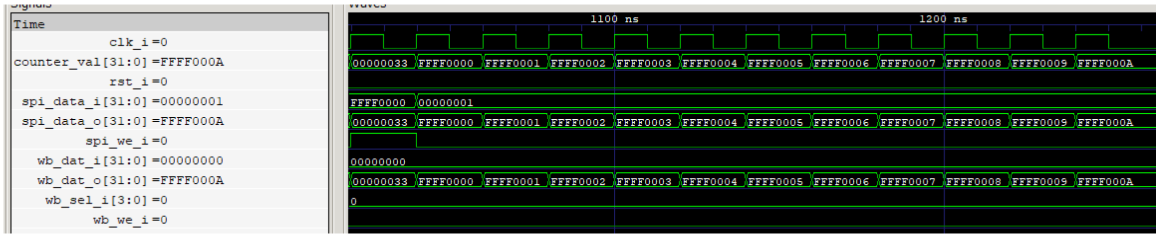
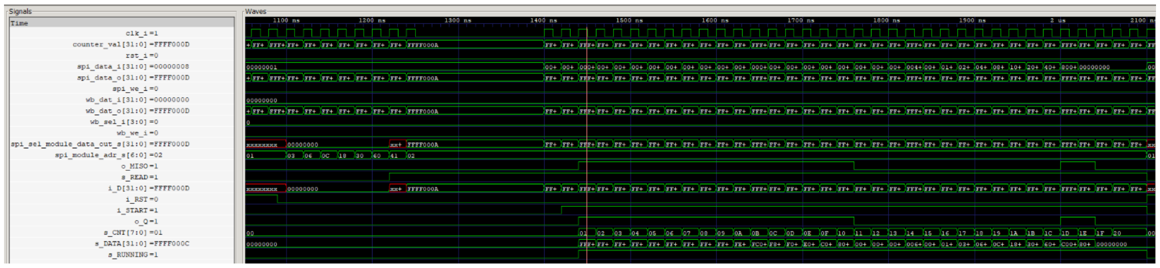


Figure 42 User Project Wrapper – Wishbone Test SPI Write Waveform Results

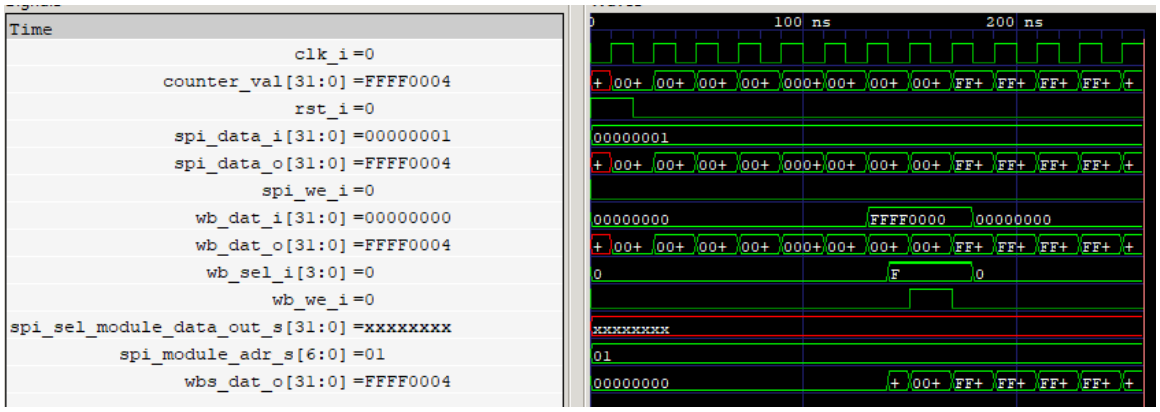
The test below demonstrates reading the Wishbone Test Module counter value over the external SPI communication bus. It can be verified that the selected address is correct and the intended counter value is read.



```
Time
                                  clk_i =1
              counter_val[31:0] =FFFF000D
                                  rst_i =0
               spi_data_i[31:0] =00000008
               spi_data_o[31:0] =FFFF000D
                               spi_we_i =0
                 wb_dat_i[31:0] =00000000
                 wb_dat_o[31:0] =FFFF000D
                   wb_sel_i[3:0] =0
                                wb_we_i =0
spi_sel_module_data_out_s[31:0] =FFFF000D
            spi_module_adr_s[6:0] =02
                                 o_MISO =1
                                 s_READ =1
                        i_D[31:0] =FFFF000D
                                  i_RST =0
                                i_START =1
                                    o_Q =1
                        s_CNT[7:0] =01
                     s_DATA[31:0] =FFFF000C
                              s_RUNNING =1
```

Figure 43 User Project Wrapper – Wishbone Test SPI Read Waveform Results

The waveforms below demonstrate writing and reading to the Wishbone Test Module over the wishbone communication bus. This verifies that we can functionally read and write to the wishbone test module over both the wishbone and SPI communication busses.



```
Time
                                  clk_i =0
              counter_val[31:0] =FFFF0004
                                  rst_i =0
               spi_data_i[31:0] =00000001
               spi_data_o[31:0] =FFFF0004
                               spi_we_i =0
                 wb_dat_i[31:0] =00000000
                 wb_dat_o[31:0] =FFFF0004
                   wb_sel_i[3:0] =0
                                wb_we_i =0
spi_sel_module_data_out_s[31:0] =xxxxxxxx
            spi_module_adr_s[6:0] =01
                wbs_dat_o[31:0] =FFFF0004
```

Figure 44 User Project Wrapper – Wishbone Test Wishbone Waveform Results

Top Level C Firmware Tests

We designed a C firmware test running on the simulated RISCV processor in the management SOC to verify every module. This test program is designed to run as part of the bring-up plan to verify every part of the design and work as an example use case for every module and a starting point for debugging. This program verifies the standard cell, custom cell, clock muxing, wishbone test, DSP accelerator, and interrupt handling.

The test starts by raising GPIO0.

The two simple cells are checked via the LA probes to verify the full truth table for each cell.

The wishbone test first checks the ability to reset the counter and that the counter increments automatically. Then it verifies loading a specific value into the counter. Each of these are verified via the LA probe pins and reading via the wishbone bus.

To test the clock muxing with the wishbone test, the clock to the wishbone test is redirected to an LA probe (which breaks the wishbone bus communication as you cannot toggle an LA probe to clock the module while a wishbone read/write is occurring, so it uses LA probes to verify). Then it toggles the LA probe clock to ensure the wishbone test counts by exactly one per clock.

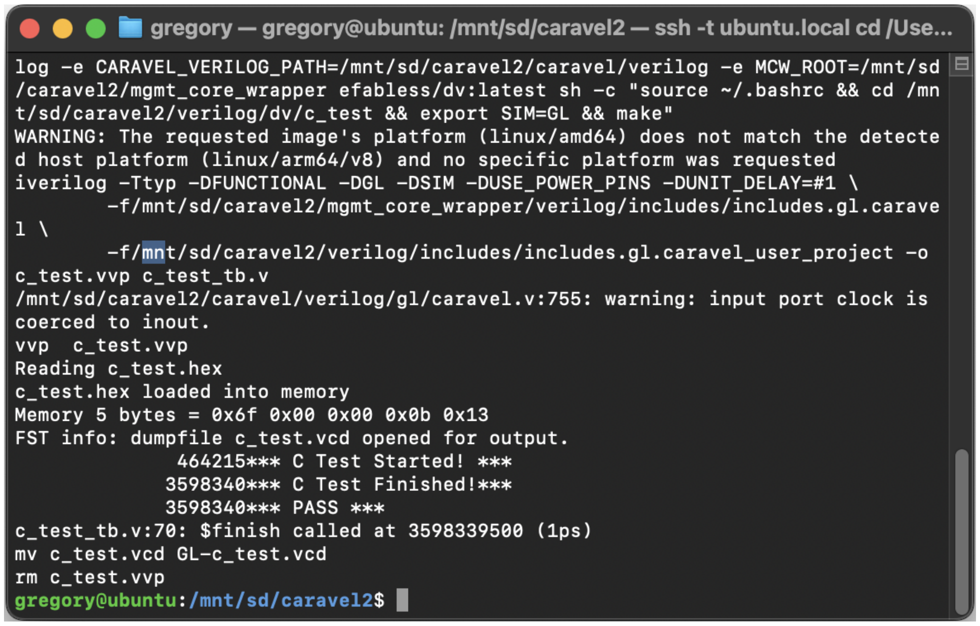
Interrupts are verified by writing to the interrupt wishbone address range which triggers irq1 and provides a simpler interrupt test than the entire DSP module for debugging interrupts themselves. See the MGMT SoC docs for more information on how the interrupts behave.

The DSP module is tested by writing an incrementing value to every weight from 0 to 1024 (mod 255), then writing 1's to every data value. A convolution is triggered by pushing a 1 into the data buffer, and the result is verified. The first test is polling the conv_ack LA probe to see when the convolution is completed as this is the simpler method. The second test pushes a 2 into the data buffer, causing another convolution to start. An interrupt is enabled, and when the interrupt triggers, the interrupt will set irq0_ack to trigger the DSP test to continue. This verifies the convolution output changes correctly when the data changes and also that interrupts work as expected with the DSP module. A third test for good measure repeats the second but passes in a 3 instead of a 2. The output of the convolution verified over the wishbone bus and the LA probes.

Finally, the test passes by outputting high on GPIO1 and lowering GPIO0.

Results

We ran the C test via `make verify-c_test-rtl` and `make verify-c_test-gl` to run the RTL-level and gate level verification tests. Both passed the C test indicating that both the verlog and the hardened gate level design pass the integration tests, giving us a high confidence that this design will work.



```
gregory — gregory@ubuntu: /mnt/sd/caravel2 — ssh -t ubuntu.local cd /Use...
log -e CARAVEL_VERILOG_PATH=/mnt/sd/caravel2/caravel/verilog -e MCW_ROOT=/mnt/sd
/caravel2/mgmt_core_wrapper efabless/dv:latest sh -c "source ~/.bashrc && cd /mn
t/sd/caravel2/verilog/dv/c_test && export SIM=GL && make"
WARNING: The requested image's platform (linux/amd64) does not match the detecte
d host platform (linux/arm64/v8) and no specific platform was requested
iverilog -Ttyp -DFUNCTIONAL -DGL -DSIM -DUSE_POWER_PINS -DUNIT_DELAY=#1 \
        -f/mnt/sd/caravel2/mgmt_core_wrapper/verilog/includes/includes.gl.carave
l \
        -f/mnt/sd/caravel2/verilog/includes/includes.gl.caravel_user_project -o
c_test.vvp c_test_tb.v
/mnt/sd/caravel2/caravel/verilog/gl/caravel.v:755: warning: input port clock is
coerced to inout.
vvp  c_test.vvp
Reading c_test.hex
c_test.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
FST info: dumpfile c_test.vcd opened for output.
              464215*** C Test Started! ***
             3598340*** C Test Finished!***
             3598340*** PASS ***
c_test_tb.v:70: $finish called at 3598339500 (1ps)
mv c_test.vcd GL-c_test.vcd
rm c_test.vvp
gregory@ubuntu:/mnt/sd/caravel2$
```

Figure 45 Gate Level C Tests

8.4.7 Detailed SPI Module Implementation

IEE SPI Standards: https://ieeexplore.ieee.org/document/9227527

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.

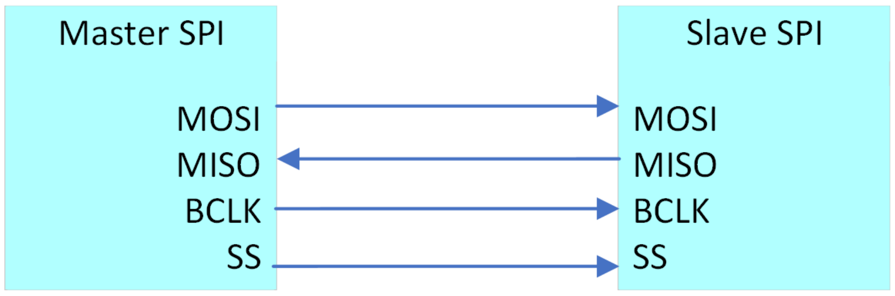


Figure 46 SPI Protocol

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.

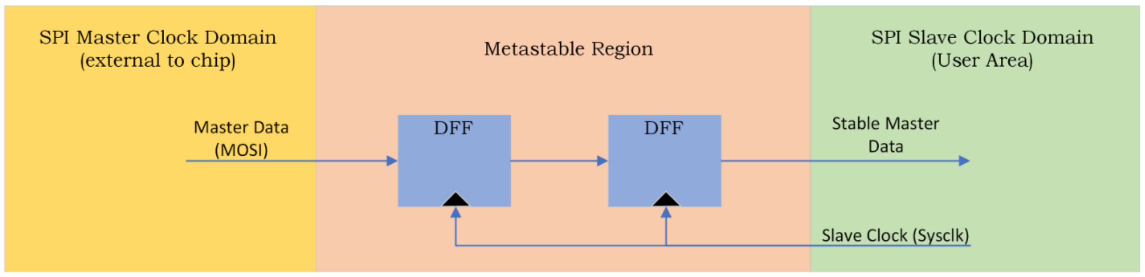


Figure 47 Clock Synchronization

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 in shift out register

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.

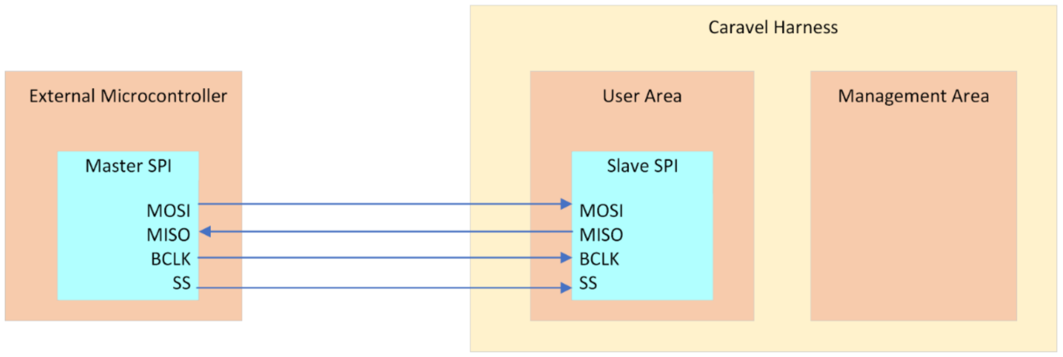


Figure 48 Master SPI Interface

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
    a. Serial shift in READ bit flag from i_ MOSI
    b. Serial shift in 7 address bits from i_ MOSI, MSB first
2. Receive Data
    a. Serial shift in 32 bits of data from i_MOSI, MSB first
3. Confirm Clock Synchronization

  a. Wait 2 SYSCLK cycles to assert o_DOUT_VALID flag
  b. Wait 1 SYSCLK cycle, Clear o_DOUT_VALID to 0

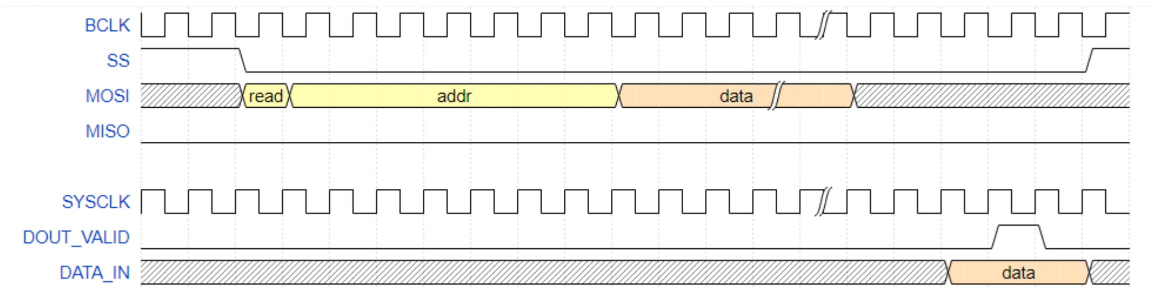


Figure 49 Data Write SPI Protocol

Data Read Transfer to MISO:

1. Receive Addressing
   a. Serial shift in 1 READ bit flag from i_ MOSI
   b. Serial shift in 7 address bits from i_ MOSI
2. Confirm clock synchronization
   a. Stop BCLK for atleast 10 us
   b. Wait 2 SYSCLK cycles to ensure address propagates to user modules for data
   c. Parallel load i_DATA_OUT from user area MUX
3. Send Data
   a. Re-enable BCLK
   b. Serial Shift out 32 bits of data to o_MOSI, scan on negedge BCLK

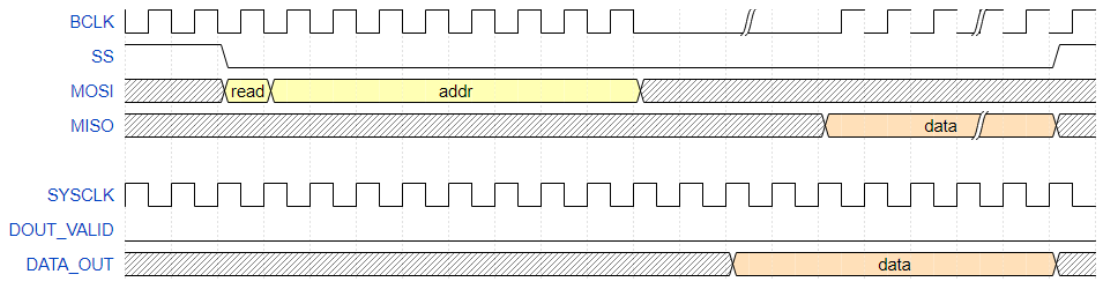


Figure 50 Data Read SPI Protocol

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.

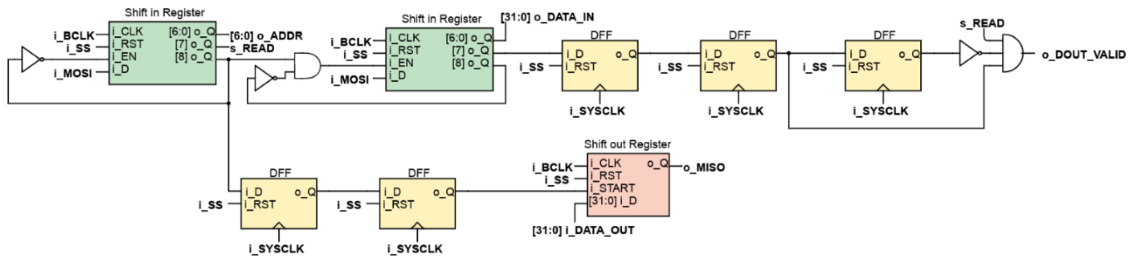


Figure 51 Backdoor SPI Top Level Diagram

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 7 SPI Ack

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

Communication Bus Control (module_control.v)

Inputs

- We_I: Write enable control, dictated by o_DOUT_VALID from top level backdoor spi
- [2:0] Addr_I: Module address field from master SPI (3 bits of 8 address space)
- [127:0] Module_data_I: 32 bit word for each indexed module to send TO master SPI

Outputs

- [31:0] Data_o: outgoing data FROM master SPI to indexed module, determined by addr_i
- [3:0] Module_we_o: One hot decoded write enable control for each module using SPI

Parameters

- N_MODULES: Number of modules to index between for data/addressing (default 4)

This module is used BOTH for the SPI and Wishbone interfaces to all our module designs. The module_control design will take a write enable input we_I, a module address addr_I, and a 32-bit word input for each module, for data to be written from the user area modules, module_data_I. For the SPI interface, the we_I input will be connected to the output o_DOUT_VALID from the backdoor_spi module, to decode and write data from the master SPI outside of the user area. By using the addr_I module address field, we can determine which module to write to, which is one hot decoded with the output module_we_o. On the inverse side, the input module_data_I will take a 32-bit word from each module and be multiplexed by the module address space determined by the master SPI module. The multiplexed output is represented as data_o, which is 1 32-bit word, determined by the referenced address.

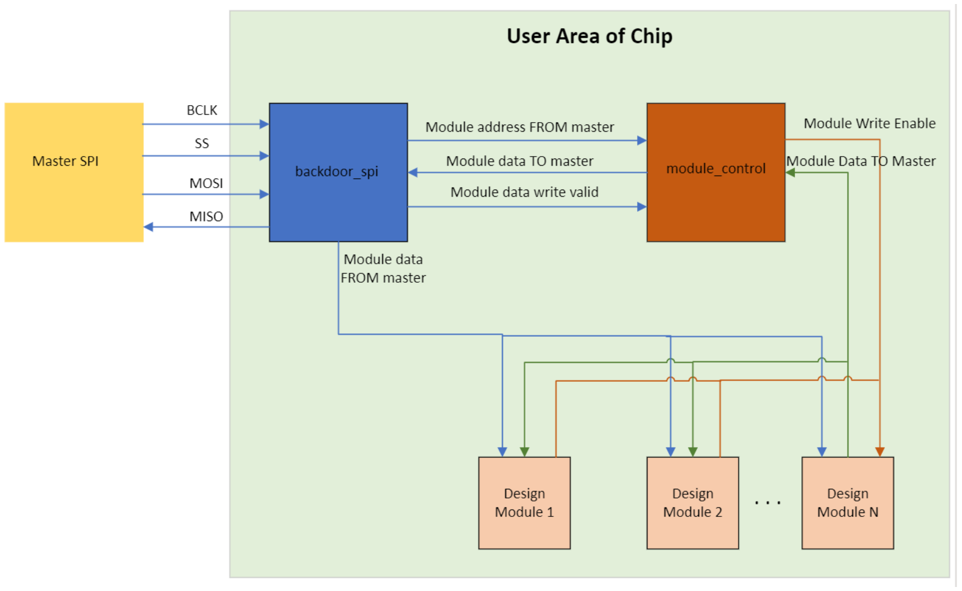


Figure 52 Backdoor SPI integration to Module Control Decoding

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.

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. The shift_out process can ONLY begin after I_RST is asserted and deasserted again, being dependent on the Master Slave Select I_SS.

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.

DFF Buffer

Inputs

- i_CLK: synchronous clock input
- i_RST: Asynchronous reset
- i_EN: If asserted to 1, i_D shifted in to LSB of internal buffer reg
- i_D: Data to be shifted in

Outputs

- [1:0] o_Q: output of two most significant DFF's

Parameters

- BUFFER_WIDTH: number of i_CLK cycles to buffer (default 2)

The DFF Buffer module was created to house the three D Flip Flops used after the data shift in register, and before the data shift out register. In our top-level diagram, this module does not exist, and instead if depicted as three D Flip Flops in a row. Similar to the shift registers, the inputs include a synchronous clock, with an asynchronous reset. While in the full implementation the enable input is always set to 1, it was included to ensure our module could easily be modified at the top level if the logic ended up changing. The input data i_D would be shifted to the least significant D flip flop in the reg, like the shift in register. Unlike the shift register, the output o_Q only outputs the two final D flip flops, to handle the clock synchronization flag that is used for the shift out register and data out valid flag to be read by the user modules. The following figure references the Verilog implementation for the DFF buffer to be placed in the external SPI module.

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

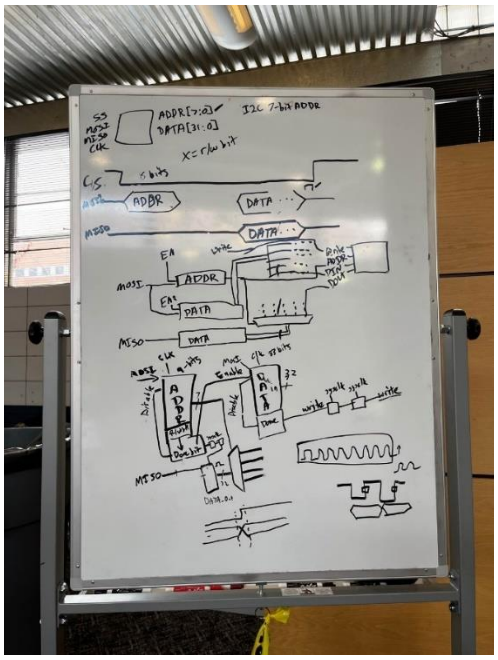


Figure 53 Backdoor SPI Planning 1

Diagram after SPI Schematic Discussion: 3/27/2023

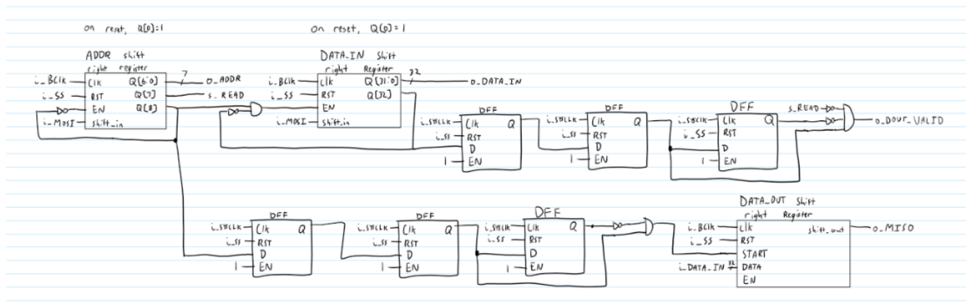


Figure 54 Backdoor SPI Planning 2

Communication Bus Controller Discussion: 8/27/2023

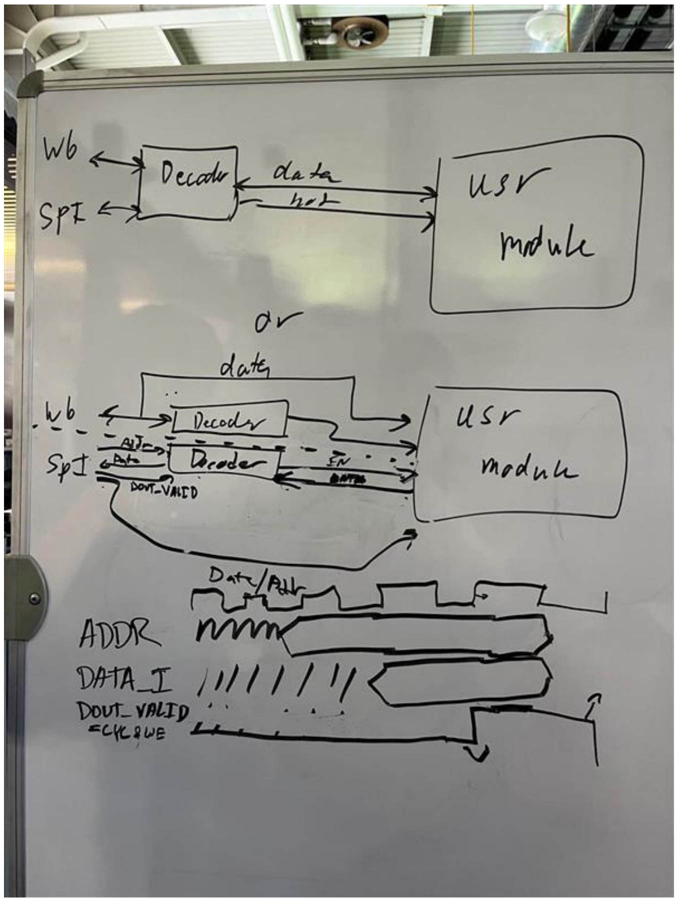


Figure 55 Backdoor SPI Planning 3

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.

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.

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.

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.

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.

DFF Buffer

Since the DFF Buffer module is a modified version of the shift in register, I began by copying the testbench for the shift in register mentioned before. The goal of this testbench was to verify if the module would update based on if the enable pin was set or not, and if the output for the two last D flip flops would be correct after N clock cycle buffers. To ensure this would occur, error checking was included after every clock cycle, with an internal signal **s_Q_expected**, that would be shifted in the testbench exclusive to the unit under test. An example of the task used can be seen below:

External SPI Top Level Module

To test the backdoor_spi module, tasks were created to create a Bus Functional Model for the Master SPI driver, to simulate read and write transactions. We also created test conditions to test the backdoor_spi module under varying clock speeds for both the Master SPI clock BCLK and the user area clock SYSCLK. Both the read and write tasks began by sending the address bit field to the backdoor_spi module. The write task would then send 32 bits of data, wait a set amount of time, and then verify that the incoming data to the backdoor_spi module was shifted in properly from the Master SPI BFM. The read task would deviate from this, instead waiting for a sent amount of time after the address bits were sent, and then counting the bits shifted out of MISO to verify data was sent TO the Master SPI BFM properly. Automated error checking was included in both tasks for better modularity and faster validation.

External SPI with Control Bus

The final testbench designed utilized both the backdoor_spi and the module_control designs together, as they will be instantiated in our final design to utilize communication between an external Master SPI module and the user modules in the caravel wrapper of our ASIC design. The key addition to validate was the decoded write enable signals going to each module, that were dependent on the module address field of the 7-bit address from Master SPI. We also included a MUX based on the same module address field to determine which of the 32-bit words from each module to be read for the Master SPI interface. The same read and write tasks from the backdoor_spi module were used, with additional error checking in each task.