Digital ASIC Fabrication

DESIGN DOCUMENT

sddec24-12
Client & Faculty Advisor: Dr. Henry Duwe

Mitchell Driscoll
Evan Dunn
Baoshan Liang
Katie Wolf

sddec24-12@iastate.edu
sddec24-12.sd.ece.iastate.edu

Revised: December 12, 2024
Version 2.1

Executive Summary

Development Standards & Practices Used

- IEEE 1481-2019 - IEEE Standard for Integrated Circuit (IC) Open Library Architecture
- IEEE 1364-2005 - IEEE Standard for Verilog Hardware Description Language
- ISO/IEC 9899:2018 – Information technology — Programming languages — C
- WISHBONE System-on-Chip (SOC) Interconnection Architecture for Portable IP Cores

Summary of Requirements

Our chip framework will:

- Support 15-20 projects created by students
- Allow one project to be active at a time through the configuration of the management core
- Provide interconnections between user projects and chip resources
- Follow the Efabless chip guidelines and project architecture
- Pass RTL simulation, hardening, GL simulation, and MPW pre-check
- Be used by the ISU Chip Fabrication Co-curricular Team

Applicable Courses from Iowa State University Curriculum

- **CPR E 281** – Digital Logic
- **CPR E 288** – Embedded Systems I
- **CPR E 381** – Computer Organization and Assembly Level Programming
- **CPR E 488** – Embedded Systems Design
- **E E 330** – Integrated Electronics
- **E E 465** – Digital VLSI Design

New Skills & Knowledge Acquired Not Taught in Courses

- ASIC chip design and development
- Chip fabrication and tape-out process
- Open-source project architecture - Efabless, Caravel
- Open-source tools - OpenROAD, OpenLANE, KLayout, GTKWave
- Electronic design concepts - synthesis, layout, routing, static timing analysis, clock gating

Table of Contents

List of Acronyms

- ASIC – Application-Specific Integrated Circuit
- DRC – Design Rule Checking
- GPIO – General-Purpose Input/Output
- IC – Integrated Circuit
- IEEE – Institute of Electrical and Electronics Engineers
- IO – Input/Output
- IRQ – Interrupt Request Signal
- LA – Logic Analyzer
- LVS – Layout Versus Schematic
- MPW – Multi-Project Wafer
- NSPE – National Society of Professional Engineers
- PDK – Product Development Kit
- RTL – Register Transfer Level
- SoC – System-on-a-chip
- WB – Wishbone Bus

Terms and Definitions

- **Caravel Harness** – Provided chip wrapper around our design, containing the User Area and Management Core
- **Efabless** – Open-source fabrication company that will manufacture our design
- **GTKwave** – Open-source waveform viewer for viewing simulation results from VCD files
- **KLayout** – Open-source tool for viewing and editing mask layouts
- **Management Core** – Part of the Caravel Harness that contains the management utilities, including the SoC and logic analyzer probes
- **OpenROAD** – Collection of open-source tools based on OpenLANE, configured and provided by Efabless to generate production files from Verilog descriptions
- **SkyWater 130nm** – Fabrication process used by Efabless supported by the SkyWater Foundry
- **User Area** – Region inside the Caravel Harness users are allowed to modify
- **Verilog** – Hardware design language specified by IEEE Std 1364-2005
- **Wishbone Bus** – Peripheral bus used by the Management Core to communicate with peripherals in the User Area

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

Undergraduate students rarely get the opportunity to create a custom digital ASIC (Application Specific Integrated Circuit) and gain experience with chip fabrication. Chip Forge, a co-curricular team at Iowa State, provides interested students with that opportunity. However, it is infeasible to provide each group of students with their own chip due to cost and space limitations. Our project aims to build and silicon-prove a single-chip framework that will support a continuous cycle of chip designs, or "tape outs," ready for fabrication. The framework we design will provide space for multiple small projects and the ability to run each independently. The project modules will be created by the Chip Forge co-curricular team of students, ranging from freshmen to seniors. By breaking the design process into smaller, less complex subprojects, students can complete modules within a semester. Furthermore, our framework will allow multiple student projects to be fabricated on a single chip, amortizing the cost of fabrication and making it financially feasible to expand Chip Forge.

1.2 Intended Users

The Iowa State University Chip Fabrication Co-Curricular Team members will be our project's main users. This team will consist of undergraduate students, graduate students, and Electrical and Computer Engineering professors. Additionally, the open-source community will have access to our design through Efabless.

Future Students

Undergraduate students in the Chip Fabrication team, ranging from freshmen to seniors, will work on their own ASIC projects to gain chip design experience outside their classes. To make the design process more accessible and hands-on, they will need a functional, easy-to-use framework to interface with the built environment and produce their final designs, which are then printed to silicon. The fabricated chip projects can then be leveraged to qualify for extracurricular projects, internships, and career opportunities.

Professors and Educators

The professors on the Chip Fabrication team will specialize in ASIC fabrication and lead all the team members. They will need to provide hands-on experience for the ISU Chip Fabrication Co-curricular team because they want to help undergraduate students gain chip fabrication experience outside of class. They help direct the efforts of students in that they know the pitfalls and can keep a clear goal in mind. The key is a mix of hands-on guidance while letting the teams conduct their affairs.

Efabless Open-Source Community

Our project will be submitted to a public, open-source Efabless repository. Our project will be available to Efabless community members to reference in their own designs. The Efabless community will be interested in open-source ASIC development, and members will be looking for project resources, reference, and collaboration.

2 Requirements, Constraints, and Standards

2.1 Requirements & Constraints

Functional Requirements

Our project client has outlined the following functional requirements our project must fulfill:

- Framework holds up to 15 projects
- Management wrapper controls which project is active
- One project is active at a time
- Chip resources will be multiplexed between all the projects
- Projects will interface with the Wishbone bus, LA pins, and IO pins
- Design successfully passes Efabless precheck

Technical Requirements

To meet the expected functionality, our design implementation must meet the following requirements:

- Development follows the Efabless process
- Design is implemented in Verilog
- Test code is written in Verilog and C
- Final design generates a GDS2 file
- Design achieves frequency of at least 20 MHz

User-Based Requirements

To ensure proper user interaction and bring-up, our project must meet the following requirements:

- The project is fully documented on the ISU Chip Fabrication website
- Detailed descriptions of the project explain the framework architecture
- The framework is straightforward to use with minimal help or troubleshooting

Design Constraints

In addition to the requirements listed above, our project must comply with the Efabless project constraints:

- The project must use the required directory structure specified in the Caravel documentation
- Hardened project wrapper must have an area of 2.920mm x 3.520mm
- The top module must be named “user_project_wrapper”
- Pin placement and pin sizes must match the golden user_project_wrapper in the Caravel repository

2.2 Engineering Standards

- **IEEE 1481-2019 - IEEE Standard for Integrated Circuit (IC) Open Library Architecture**
  - We are creating an ASIC that should meet timing and power constraints.
- **IEEE 1364-2005 - IEEE Standard for Verilog Hardware Description Language**
  - All of our modules, as well as the project wrapper, will be implemented in Verilog.
- **ISO/IEC 9899:2018 – Information technology — Programming languages — C**
  - Our test programs for the management core will be written in C.
- **WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores**
  - The Wishbone Bus is an open-source hardware bus used for communication between different parts of an integrated circuit. Our design will use this bus to interface between the management core and user projects.

3 Project Plan

3.1 Project Management/Tracking Procedures

Our team will use a combination of agile and waterfall methodologies for project management. This allows for structured planning with clear expectations and milestones while offering flexibility. Since our project has set goals that are required before we can move on to later steps, as well as a hard deadline goal, using waterfall allows us to clearly outline a timeline to follow. We will utilize agile to adapt to unforeseen complexity or challenges as issues arise. We will meet weekly, where we can adapt our team's focus if needed.

We will track our progress through communication on Microsoft Teams and shared files on Google Drive. We will also utilize GitLab for version control of our code base and specific issue-tracking.

3.2 Task Decomposition

Our project is split up into the following tasks, which will generally be completed in sequential order:

1. Tool Setup
   a. Install and set-up open-source software and example Caravel project
   b. Run RTL simulations, harden, and run GL simulations on the example Caravel project
   c. Successfully create, simulate, and harden custom components using the tool flow
2. Design Decomposition
   a. Determine how user projects interact with Management SoC, Wishbone bus, OpenRAM
   b. Define interconnections between components and necessary control paths
   c. Harden example user projects with different configurations to understand constraints
   d. Draw out a high-level framework schematic with all subcomponents, ports, and connections

3. Create Modules
    a. Implement each module in Verilog
    b. Create a testbench that thoroughly covers the module's functionality
    c. Successfully simulate and harden each component on its own
4. Integrate Modules
    a. Review modules and test benches written by each team member
    b. Place modules in the high-level project wrapper based on the design decomposition
    c. Configure and harden the project wrapper with all components added
5. Test Overall Design
    a. Place different user projects in the framework and test that each project can function
    b. Create and run tests that ensure the framework can properly connect and activate projects
    c. Create and run tests that ensure the framework meets timing, size, and power constraints
    d. Run project through pre-check and ensure all tests pass to prepare for fabrication
6. Submit Design to Efabless
    a. Create a repository and project on Efabless's website
    b. Submit our design to the Efabless Open MPW program by the intended deadline
7. Bring-Up Documentation & Physical Testing
    a. Thoroughly document our project's design, implementation, testing, and general use
    b. Create a detailed bring-up plan for the ISU Chip Fabrication team
    c. Run pre-fabrication physical tests on the ISU Chip Fabrication FPGA board

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Our project's milestones can be broken into the main task sections listed above. Each milestone will be measured using the following metrics:

- Milestone 1: Complete Tool Setup
    - Each member can go through the RTL simulation, hardening, and GL simulation process on the example project.
    - Each member will be able to view correctly simulated waveforms in GTKWave.
    - Each member can successfully implement a new Verilog module and complete the same steps.
- Milestone 2: Complete Design Decomposition
    - Framework will fully connect each user project to the Wishbone, LA, and IO ports
    - Hardening configuration results will outline the necessary size and layout of user projects
    - The framework schematic will illustrate the full interconnections between the user projects and management wrapper
- Milestone 3: Create All Modules
    - Each implemented Verilog module will properly synthesize and function as expected

    - Each module's testbench will include base cases and edge cases for the design under test
    - Each module will pass simulation and can be hardened on its own
- Milestone 4: Integrate All Modules
    - All modules will fit into the user project wrapper in synthesis
    - Each project will be fully connected to the Management SoC
    - The project wrapper successfully synthesizes and hardens after all components are added
- Milestone 5: Test Overall Design
    - The project's resources will be multiplexed to the active project as selected in the software
    - The project will meet all timing, size, and power constraints set by Caravel for the fabrication
    - The project will successfully harden and pass all pre-check steps
- Milestone 6: Submit Design to Efabless
    - The project will be placed in a public repository on Efabless
    - The project will be fully submitted by the Efabless Fall 2024 deadline
- Milestone 7: Complete Bring-Up Documentation & Physical Testing
    - Project documentation will be easy to understand and navigate without troubleshooting
    - The bring-up plan will cover future testing and use cases for the ISU Chip Fabrication team
    - The physical chip will fully support different user projects

3.4 Project Timeline/Schedule

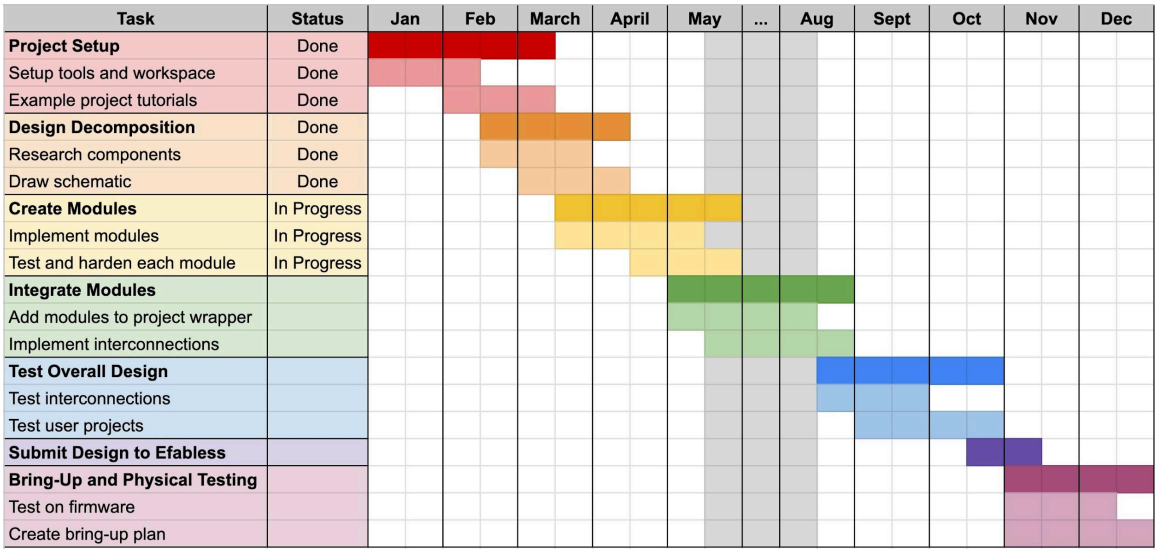

Figure 1: Project Timeline

The Gantt chart illustrates our projected timeline for this project, from start to finish. The timeline takes place over the Spring 2024 and Fall 2024 semesters and is broken down into individual weeks.

Each major milestone is highlighted in a different color, and the milestone's expected deliverable deadline is at the end of the highlighted timeframe

3.5 Risks and Risk Management/Mitigation

We have identified the following major risks and appropriate mitigations of our project:

- Risk 1: Design does not pass pre-check
  - Estimated Probability: 15%
  - Mitigation: Use simulation tools to extensively check our design against the fabrication specifications. Run DRC and LVS checks to ensure our layout meets all rules and matches the expected schematic. Hold design reviews with our advisor and mentors who may catch potential issues early.
- Risk 2: OpenRAM modules cannot be hardened or fabricated in the framework
  - Estimated Probability 50%
  - Mitigation: Exclude OpenRAM modules from the framework design or leave space for OpenRAM modules to be added in the future. Until implemented, the user projects cannot utilize OpenRAM modules. If time allows, implement DFF RAM as an alternative.
- Risk 3: Project wrapper area cannot fit framework and expected projects
  - Estimated Probability: 10%
  - Mitigation: Reduce the number of user projects the framework will support. The framework logic itself will likely not be able to be reduced due to wire spacing requirements.
- Risk 4: Error occurs during fabrication process or delivery
  - Estimated Probability: 5%
  - Mitigation: Include testing features in our design to test the fabrication process and ensure consistency in the physical chip. Plan for future redesigns or tweaks if the initial fabrication process fails.

3.6 Personnel Effort Requirements

Task	Projected Hours
Tool Setup - installing tools, running example project	20
Design Decomposition - researching components, drawing schematic	20
Create Modules - writing Verilog, simulating and hardening individual components	60
Integrate Modules - integrating modules in top-level design, hardening wrapper	40
Test Overall Design - writing and running tests covering base and edge cases	80

Submit Design to Efabless - creating a public project repository, submitting the design	10
Bring-Up Documentation - documenting a detailed bring-up plan for future users	80
Physical Testing - receiving physical chip, running user projects in the framework	40

Table 1: Effort Requirements

3.7 Other Resource Requirements

Our design will be manufactured through the Efabless ChipIgnite program. Fabricating a single chip through this program costs $9750, and the fabrication turnaround is about 4-5 months.

Our project also requires the following open-source resources and software:

- SkyWater 130nm Open-Source Process Design Kit (PDK)
- GTKWave
- KLayout

These tools are all free to use.

4 Design

4.1 Design Context

4.1.1 Broader Context

Our project addresses the lack of ASIC development opportunities for undergraduate students. By creating our framework, we can provide a cost-effective solution to allow multiple student projects to be fabricated simultaneously. This ultimately supports a continuous cycle of chip prototypes, created by students at any experience-level.

Area	Description	Examples
Public health, safety, and welfare	Our project provides practical experience and skills development for students, which is essential for building a knowledgeable workforce capable of addressing future public health and safety challenges through innovative technologies.	Students can gain experience in chip fabrication and create designs with any functionality. They can gain experience and supplement their education with a low barrier to entry.
Global, cultural, and social	By enabling students to design and fabricate chips, our project contributes to the accessibility of technology development, by lowering	The project could lead to initiatives that focus on creating affordable and accessible technology solutions for underrepresented communities

	barriers for chip design and fabrication.	or developing regions, fostering greater social and digital inclusion.
Environmental	Our project involves the design of a framework that allows multiple projects to run independently but share the same resources. This approach can lead to more efficient use of materials and energy, highlighting the importance of sustainable practices in engineering.	By creating reusable components, the project can contribute to reducing waste from excessive fabrication, promoting an efficient cycle of chip fabrication.
Economic	Our project allows multiple student projects to be fabricated at once, minimizing the cost of prototyping multiple designs. It also contributes to students' learning and development of valuable skills, which will be beneficial in the workforce.	The use of open-source work allows users to create their own ASIC designs at minimal cost.

Table 2: Project Context

4.1.2 Market Research

After investigating existing solutions for entry-level ASIC fabrication, we found that the company Tiny Tapeout is the other main competitor to Efabless. However, we found several factors about Tiny Tapeout's fabrication program that make their solution unsuitable for our project.

Tiny Tapeout's chip space is 160um x 100um, which allows for 1000 digital logic gates. This user space is extensively smaller than Efabless's, and it would not provide enough space to include the desired number of projects and interfaces that our framework will implement. Due to its limited size and interfaces, the Tiny Tapeout chip also has limited testing capabilities compared to Efabless. For these reasons, our design will use the Efabless process.

Additionally, we will be building upon the work of prior senior design teams who have developed various ASIC projects. We have access to their designs and the documentation detailing their development process. However, our project is unique because it is implementing a framework that can support multiple projects like the ones created by the senior design teams.

Prior Senior Design Teams:

- http://sdmay23-28.sd.ece.iastate.edu
- http://sddec23-08.sd.ece.iastate.edu
- https://sddec23-06.sd.ece.iastate.edu

4.1.3 Technical Complexity

Our design has multiple components and interfaces that contribute to its complexity:

- Wishbone Interface

    - We need to implement the Wishbone bus interface between the Management SoC and the user projects. This communication protocol allows data to be transmitted between the management and user area, and it will be used to send various control values to configure the framework. We must implement a way to read and store the control values through the Wishbone interface while also allowing the user projects to use the interface to access the Management SoC memory.
- External OpenRAM
    - In addition to the memory provided by the Management SoC, we planned to include additional independent memory modules in our design. This allows projects to use external memory. We must incorporate the pre-hardened, 8-bit OpenRAM modules into our design to allow projects to read and write 32-bit data. To do this, we must create a Wishbone interface between the OpenRAM modules and the user projects.
- Support of Different User Projects
    - Multiple user projects need to be able to interface with the Management SoC utilities and other chip resources. Only a single active project can send and receive data at a time. The inactive projects should be able to hold their state. The projects have independent reset signals. We are also looking into allowing the user projects to have varying frequencies, which would involve clock gating.

4.2 Design Exploration

4.2.1 Design Decisions

There were several design decisions that we had to make when planning out our framework design. Beyond the basic functionality requirements, our client allowed us to decide on the following:

1. How many projects that our framework will support
    - The total number of user projects will affect how resources are multiplexed. It also limits how big each user project can be, as we have limited space within the design area.
2. How different user projects will be activated
    - Our design should only allow one project to be “active” at a time. The active project will be reading and writing output data and utilizing memory. We must figure out how to enable and disable all the projects independently within the framework.
3. How projects will interface with the management core
    - Each project should be connected to all the chip’s resources and, when active, be able to read and write data from the management core. We must design an interface that allows all the projects to communicate independently with the management core.
4. How to incorporate OpenRAM with the user projects
    - OpenRAM is a memory module that is independent of the memory in the management core. It provides external memory to the user projects. We must decide how to connect existing OpenRAM modules to the user projects and how many of the OpenRAM modules to include in the overall design.

4.2.2 Ideation

We based our design decisions on example projects and suggestions from our client:

1. We decided to implement 15 small projects and 2 large projects. These numbers were based on the size of completed, for example, user projects. This choice may change if we run into sizing constraint issues.
2. The active user project will be selected through the management core. A select value will be written to a dedicated address in memory through software. Another option was to use physical dip switches, but we implemented project control in software for simplicity.
3. We decided to multiplex all the chip's resources to the different projects based on the select value mentioned before. This method is widely used in digital design. Our interconnection design is explained in more detail in the following sections.
4. We decided to include four 8-bit OpenRAM modules in our design. This way, 32-bit data can be written and read across the four modules. However, incorporating OpenRAM in our design requires additional ports to be added to each user project. This results in a bigger project size. For this reason, we thought of several possible solutions to minimize the size increase:
    - Add the necessary set of OpenRAM ports to every project
    - Utilize existing IO pins for OpenRAM interfacing
    - Chain the OpenRAM modules in a sequential fashion, instead of in parallel
    - Interface with the OpenRAM modules through the Wishbone protocol

4.2.3 Decision-Making and Trade-Off

To decide which method to use for OpenRAM integration, we considered the following trade-offs:

- Add the necessary set of OpenRAM ports to every project
    - **Pros:** Simplest option; does not require additional logic besides multiplexing
    - **Cons:** Results in a lot of additional ports per project; since we have 4 separate OpenRAM modules, each project will need an additional (4 * # OpenRAM ports)
- Utilize existing IO pins for OpenRAM interfacing
    - **Pros:** Does not require any additional ports to be added
    - **Cons:** Limits a project's use of IO ports; requires control module to identify which and when IO ports will be used for memory
- Chain the OpenRAM modules in a sequential fashion, instead of in parallel
    - **Pros:** Reduces number of additional ports to just (1 * # OpenRAM ports)
    - **Cons:** Increases number of cycles needed for deeper memory accesses, since each module uses 1 clock cycle; requires control module to appropriately mask address bits between the different modules
- Interface with the OpenRAM modules through the Wishbone protocol
    - **Pros:** Projects will have 1 additional set of Wishbone Master ports; no additional ports for interacting directly with OpenRAM modules
    - **Cons:** Requires an OpenRAM wrapper that maps its ports to Wishbone Slave ports; does not support parallel memory accesses

Based on these comparisons, we decided to go with the last method: using the Wishbone protocol.

We chose this option because it should be straightforward to implement and has the least significant drawbacks in size, functionality, and performance.

4.3 Final Design

4.3.1 Overview

Our design uses the open-source Caravel user project template, which can be found on GitHub. This project template is designed specifically for the chip fabrication program we are using. The Caravel chip architecture is shown below in Figure 3.

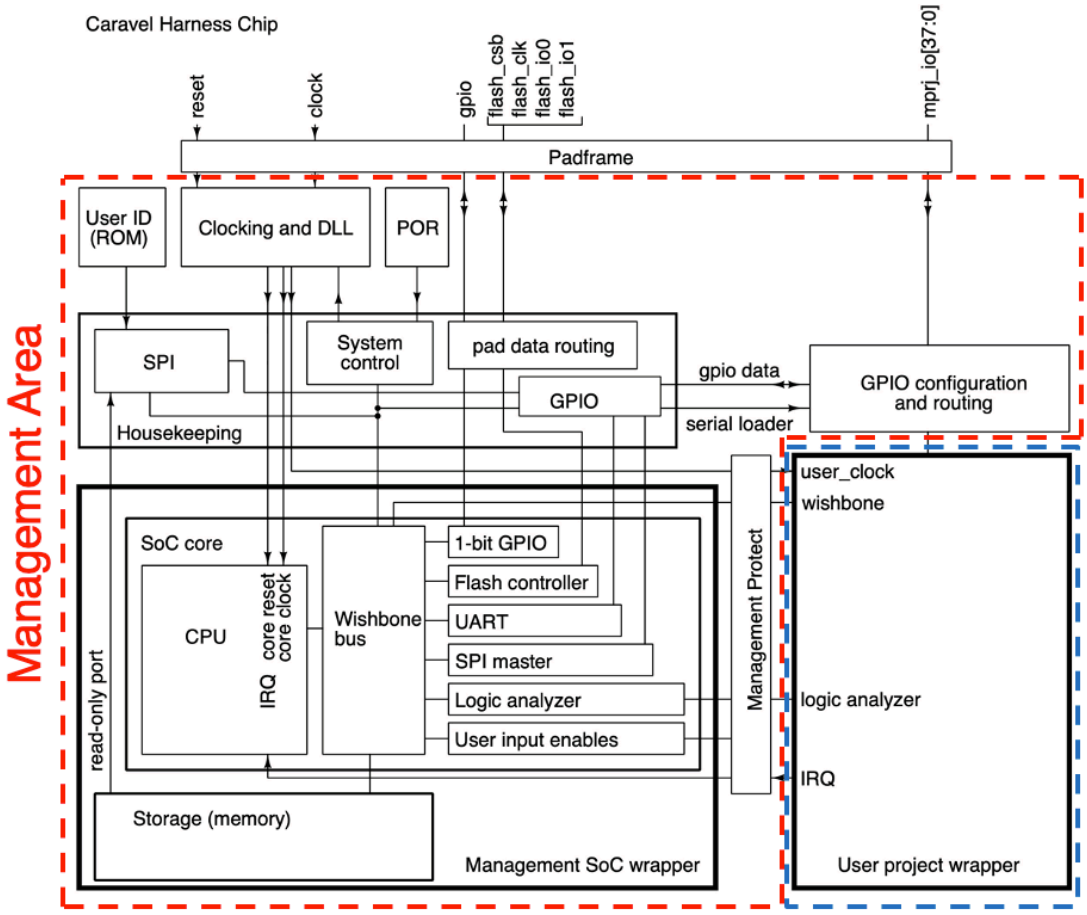

Figure 2: Caravel Chip Architecture [1]

The user project wrapper, outlined in blue in the lower right corner, is the chip's user area. This is where our framework and all user project modules will be implemented. The part outlined in red is the management core. This space contains the chip's resources, including memory, the Wishbone Master, the logic analyzer, and GPIO configuration. The management core is configured externally through software. Our framework will connect the various management core ports to the user projects, as shown in Figure 3.

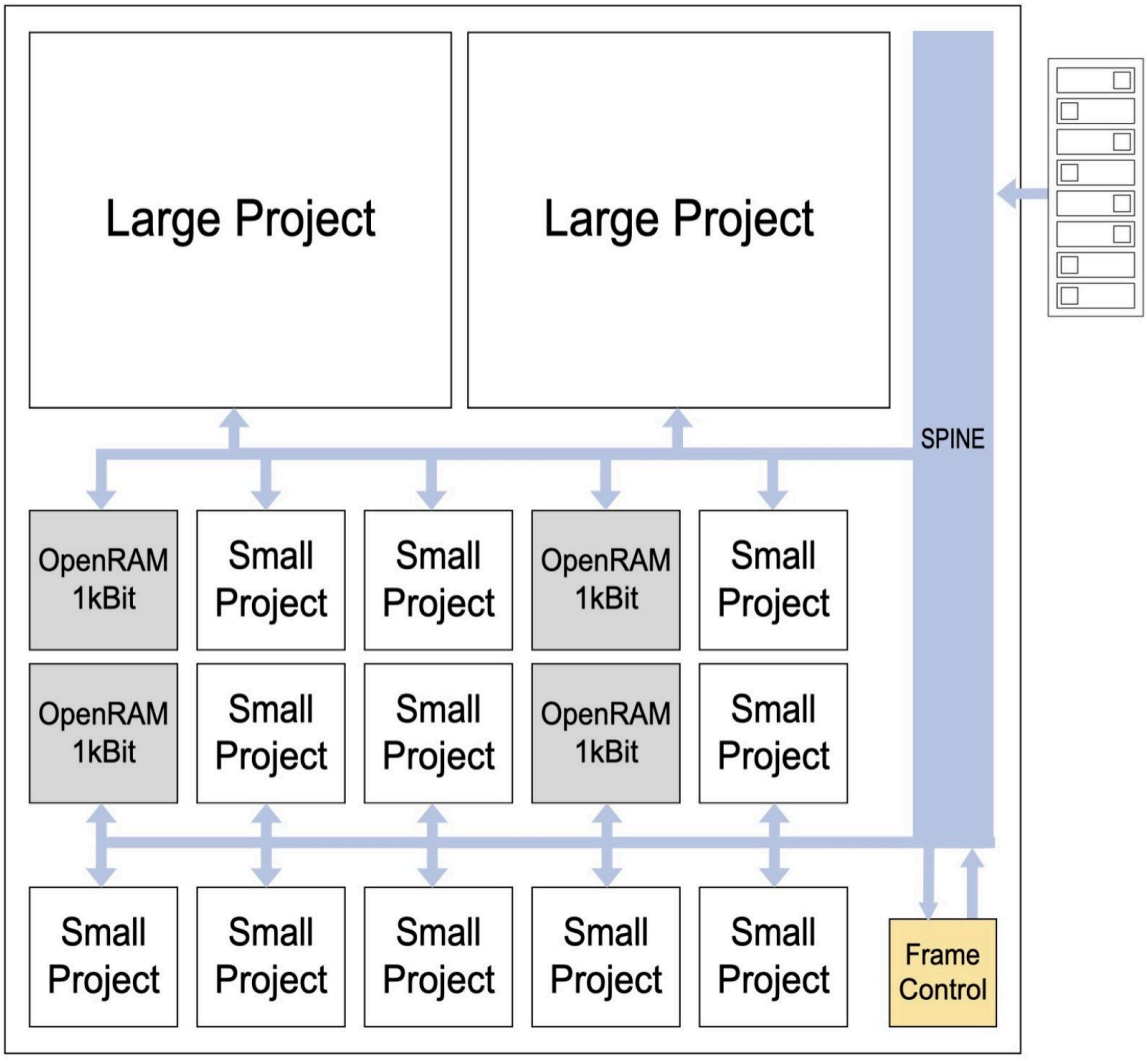

Figure 3: Framework Diagram

4.3.2 Detailed Design and Visual(s)

Figure 4 depicts the general design composition of our user project framework. Our design connects interchangeable user projects, pictured in the top half, to the resources provided by the Management Core SoC and the GPIO ports, pictured on the bottom. Only one user project will be active and communicating with the Managemen SoC at a time. The Management SoC contains the Wishbone Bus (WB) Master and the Logic Analyzer (LA) modules. Both of these are used to communicate with the user projects and transfer data, and there are separate ports for data going in and out of these modules.

Each project can be reset individually. The project reset bus signal will come from the Wishbone Bus and be stored in a control signal register. The active user project will be selected using a value outputted by the Wishbone Bus and stored in another control signal register. Each project's data output will be multiplexed based on the selected value and returned to the Management SoC. The data sent from the Management SoC to the user projects will be inputted to the active project.

Inactive projects will receive data that is all zeros. 2-1 multiplexers will select between the resource data and 0 values, and the select bits will come from a one-hot decoder.

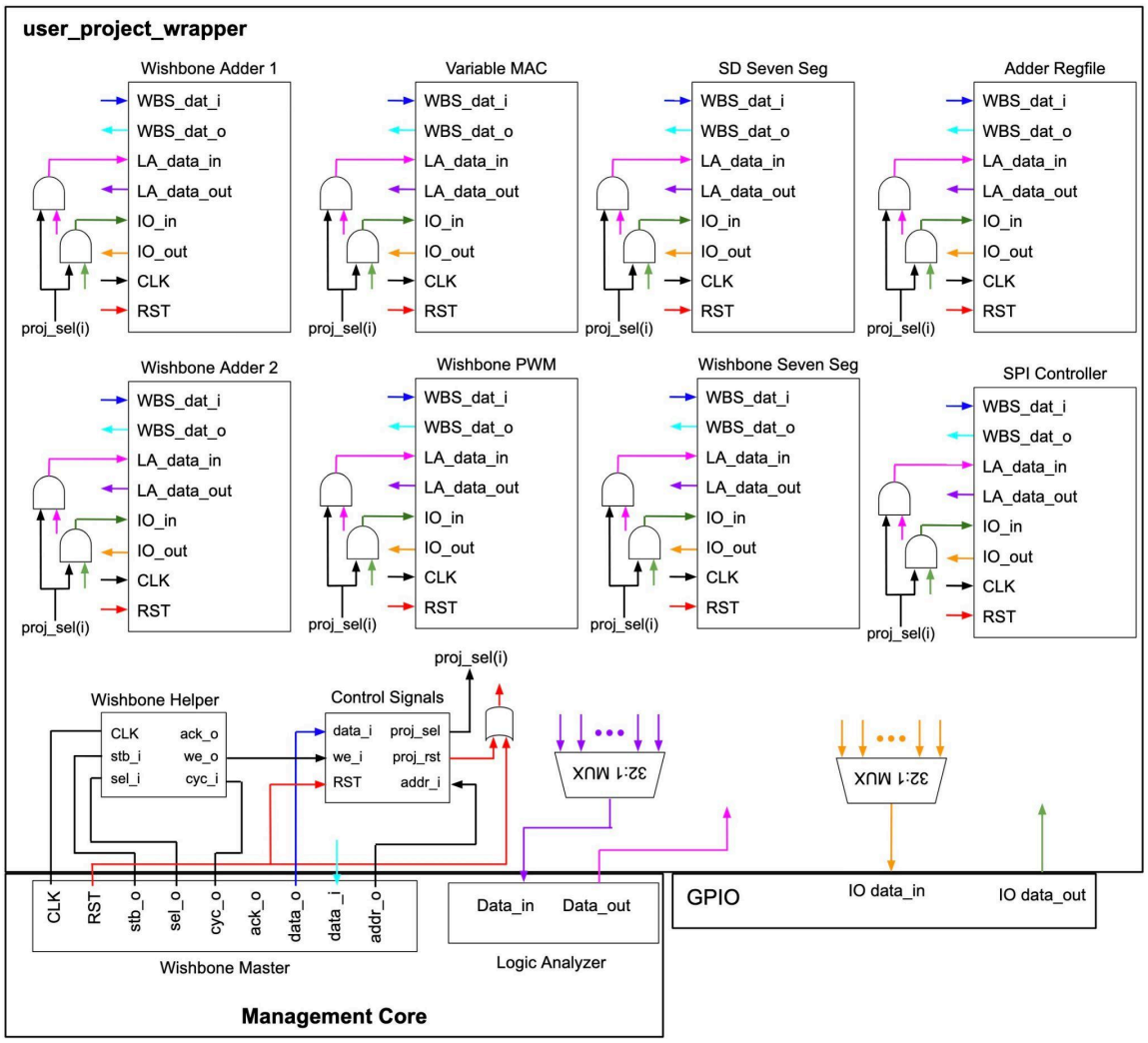

Figure 4: Design Schematic

Acronyms

- **CLK:** Clock; system clock used by the Management SoC and all projects
- **GPIO:** General-Purpose Input/Output; ports handling both incoming and outgoing digital signals
- **IO:** Input/Output; data transfer to or from the chip
- **LA:** Logic Analyzer; sends logic signals between the Management SoC and user projects
- **RST:** Reset; when high, clears all module values to 0
- **SoC:** System-on-a-chip, integrated circuit design that combines the functions of an electronic device onto a single chip
- **WB:** Wishbone Bus; connects modules to the Management SoC and handles data transfers

- **WBS:** Wishbone Bus Slave; module that responds to transactions initiated by the Wishbone Bus master in the Management SoC
- **WE:** Write enable; signal that enables writing data to the respective location

During testing, we created test modules to establish how our framework fit together. This established the spatial requirements our design would occupy. For fabrication, we required modules to include with our design, of a requisite complexity. We gathered a number of modules from undergraduate students currently in the ISU Chip Fabrication Co-curricular Team. These consisted of:

- Wishbone Adder (x2)
- Variable MAC
- SD Seven Segment Display
- Adder-Register File Datapath
- Wishbone PWM
- Wishbone Seven Segment Display
- SPI Controller

4.3.3 Functionality

With our design, users will be able to create independent ASIC projects and place them as macros in our framework. As we stated before, multiple projects can be placed in the framework at once, and each project will be able to run independently and access the management core.

After placing their project in our framework, the user will configure the management core through software to select their project as active. They will then be able to interact with their project normally, with the same functionality as if their project was the only component inside the user space.

4.3.4 Areas of Concern and Development

Throughout our brainstorming and design process, we have had regular communication with our client to ensure our design will meet user needs. However, due to the nature of the project, we have two main concerns:

- Will our solution's current level of complexity support all possible projects going forward?
- With our solution's current level of complexity, will all expected modules fit in the die area?

We have communicated these concerns with our client, and they have expressed that the total number of projects can be reduced if needed. They have also approved of our design, so we can assume that future user projects will be appropriately supported within reason.

Additionally, our client outlined stretch goals to strive for once we get further in the development process:

- Optimize power through clock gating on inactive projects
- Preserve the state of all inactive projects, rather than resetting them in between uses

4.4 Technology Considerations

Our project fully utilizes open-source tools and project architecture. Open-source software is usually free to use and modify, making it a cost-effective solution. Our project architecture is

accessible to anyone and publicly published on GitHub. Open-source projects also often have a large community of contributors that can help find and fix bugs.

However, there are also trade-offs. Open-source projects often lack official support or documentation, resulting in a learning curve for new users. Throughout our planning and design process, we have sometimes needed to search through multiple repositories and wiki pages to find the data we are looking for. Additionally, open-source software can have compatibility issues, as projects may not always have full integration with different operating systems. To combat this, we are using a Linux virtual machine that contains an installed toolflow as our development environment. This way, our team does not have to worry about environmental inconsistencies or incompatibilities.

4.5 Design Analysis

Prior to full implementation of our framework, we performed analysis on user project size limits.

Component	Run #	Changed Settings	# Ports	CLOCK_PERIOD	CLOCK_PORT	CLOCK_NET	FP_SIZING	DIE_AREA	PL_TARGET_DENSITY	Pass? (Y/N)	Errors	Notes	Date Run
adder_wrap	1	-	607	25 (40 MHz)	wb_clk_i	[blank]	absolute	0 0 700 700	0.55	Y			4/2/24
adder_wrap	2	DIE_AREA	607	25	wb_clk_i	[blank]	absolute	0 0 350 350	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (607) exceeds maximum number of available positions (424).	4/2/24
adder_wrap	3	DIE_AREA	607	25	wb_clk_i	[blank]	absolute	0 0 525 525	0.55	Y			
adder_wrap	4	DIE_AREA	607	25	wb_clk_i	[blank]	absolute	0 0 438 438	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (607) exceeds maximum number of available positions (530).	4/2/24
adder_wrap	5	DIE_AREA	607	25	wb_clk_i	[blank]	absolute	0 0 482 482	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (607) exceeds maximum number of available positions (584).	4/2/24
adder_wrap	6	DIE_AREA	607	25	wb_clk_i	[blank]	absolute	0 0 500 500	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (607) exceeds maximum number of available positions (606).	4/2/24
adder_wrap	7	DIE_AREA	607	25	wb_clk_i	[blank]	absolute	0 0 501 501	0.55	Y			4/2/24
adder_wrap	8	PL_TARGET_DENSITY	607	25	wb_clk_i	[blank]	absolute	0 0 501 501	1	Y			4/2/24
adder_wrap	9	Added WB master ports	711	25	wb_clk_i	[blank]	absolute	0 0 700 700	0.55	Y			4/9/24
adder_wrap	10	DIE_AREA	711	25	wb_clk_i	[blank]	absolute	0 0 501 501	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (711) exceeds maximum number of available positions (608).	4/9/24
adder_wrap	11	DIE_AREA	711	25	wb_clk_i	[blank]	absolute	0 0 550 550	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (711) exceeds maximum number of available positions (666).	4/9/24
adder_wrap	12	DIE_AREA	711	25	wb_clk_i	[blank]	absolute	0 0 575 575	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (711) exceeds maximum number of available positions (696).	4/9/24
adder_wrap	13	DIE_AREA	711	25	wb_clk_i	[blank]	absolute	0 0 600 600	0.55	Y			4/9/24
adder_wrap	14	DIE_AREA	711	25	wb_clk_i	[blank]	absolute	0 0 585 585	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (711) exceeds maximum number of available positions (708).	4/9/24
adder_wrap	15	DIE_AREA	711	25	wb_clk_i	[blank]	absolute	0 0 586 586	0.55	N	Failed on step 4 (Running IO placement)	[ERROR PPL-0024] Number of IO pins (711) exceeds maximum number of available positions (710).	4/9/24
adder_wrap	16	DIE_AREA	711	25	wb_clk_i	[blank]	absolute	0 0 587 587	0.55	Y			4/9/24

Figure 5: Hardening Data

Our team has researched and become familiar with the Caravel project architecture. We have begun implementing and testing the various low-level components our design uses, including a variable size register, a 2-to-1 multiplexer, and a 5-to-32 decoder. As we verify these designs through simulations, we can begin implementing the base design we outlined earlier.

One important step of the build process is hardening designs, which consists of synthesis, place and route, and error checking. Successful hardening results in generated GDSII files. While a component may pass simulation, it must also be hardened to ensure the design will work once implemented on hardware. In order to harden a design, a configuration file must be created that contains settings for various parameters, such as design size, clock frequency, and density. Part of the design process involves finding the optimal configuration settings that allow hardening to pass, while still meeting project constraints.

In order to find the best configuration, we ran various hardening configurations on an example project and recorded the results, as shown in Figure 5. As we experimented with different parameter values, we determined the various constraints we will need to consider when implementing the wrapper for the different user projects. Based on the data we have collected so far, we have found that a project area can have a minimum size of 501 µm x 501 µm while still supporting the expected

607 ports. Furthermore, if we include OpenRAM and the Wishbone OpenRAM interface on each project, this introduces additional ports and will increase the minimum hardening area. After performing further tests, we found that the minimum size with OpenRAM was 587 x 587 µm.

After beginning implementation, we also had to consider how to integrate the framework and user projects at the top-level. As discussed, submodules within a design can be hardened, creating a blackboxed macro. However, blackboxing modules comes with tradeoffs, creating several options for integrating our full design.

- Method 1: Macro-First Hardening
    - The first method involves pre-hardening all of the user logic and projects into a single macro that is then placed in the wrapper. This means there are no standard cells at the top-level and instead, one large user macro is placed. This simplifies the place and route process, which defines where modules are placed in the physical layout.
- Method 2: Full-Wrapper Flattening
    - Another method is to merge the user macro into the wrapper, so all of the logic and projects are placed directly in the wrapper and hardened all at once. This allows the user to utilize the entire wrapper area but can take additional configuration.
- Method 3: Top-Level Integration
    - The third method is to place both macros and standard cells at the top level. This means there are both hardened modules and additional logic present in the wrapper. This method is suitable when buffering is needed at the top level.

Based on these methods, we narrowed the most suitable option down to methods two and three. Initially, we were using method three. We were hardening the user projects, placing the macros in the user wrapper, and adding our framework logic in at the same level. However, this method posed some limitations. The minimum project die area with OpenRAM that we found earlier meant that in the future, we would be limited to around 40 projects, not including space left for our framework logic. Furthermore, if we wanted to re-introduce external SRAM, this would take up more area.

Based on this, we decided to look into method two, where there are no pre-hardened macros. This means that user projects do not have a minimum size, and we have the full user area to utilize. This allows us to place as many user projects and as much logic that we want into the user wrapper, and the only limit we have is the size of the wrapper itself. Additionally, we were able to talk directly with Efabless team members, and they were able to help us find the best hardening configuration to use this method. For these reasons, our framework was fully implemented directly in the user wrapper per the top-level flattening method, and we successfully hardened the full design.

5 Testing

We have a comprehensive testing plan that will thoroughly test all of our submodules and the top-level framework design. We want to ensure every component in our design is fully functional and passes hardening and pre-check in preparation for fabrication.

5.1 Unit Testing

All of the modules in our design will have one test that covers basic use and edge cases of their functionality. These tests will be written as Verilog testbenches. They will be performed by RTL and GL simulations with the OpenROAD tools. We will use GTKWave to view and verify the test results. We will test each component as they are implemented.

Modules Under Test:

- 5 to 32 Decoder
- 32 to 1 Multiplexer
- N-bit Register
- Wishbone Bus Control Module
- OpenRAM Module
- Example User Projects

We have already begun unit testing. Results can be seen in Appendix 10.1.

Due to the implementation of the framework itself, there were no independent logic components that required testing.

All user projects placed in the framework were tested independently. These tests involved RTL and GL simulations and tests run on the FPGA. These tests were completed so that each project had correct functionality before being placed and tested in the framework.

5.2 Interface Testing

We will also test interfaces between components to ensure communication between the various components of our framework. These tests will verify writing and reading values between the master and slave modules, as well as ensure that modules adhere to the bus protocols. These tests will be written as Verilog testbenches. They will be performed by RTL and GL simulations with the OpenROAD tools. We will use GTKWave to view and verify the test results.

Interfaces Under Test:

- Wishbone Bus Interface with User Projects
- Wishbone Bus Interface with the overall Framework
- GPIO Interface with User Projects
- LA Interface with User Projects

5.3 Integration Testing

After testing individual modules and interfaces, we will integrate everything into the top-level user project wrapper to create our framework design. We will ensure that all connections are implemented according to our design schematic. From there, we will undergo extensive testing of our overall framework, utilizing example user projects to place in our design. We will create tests that verify different projects can be enabled, and that all user projects can properly access the chip resources when active.

5.4 System Testing

Upon integration, we will test our framework by creating multiple testbenches that implement and activate different example user projects. We will simulate C code to program the Management SoC

to verify that the user projects can properly interface with the various chip utilities. We will also run the user project wrapper through hardening and pre-check to ensure the design can be properly synthesized. This will involve creating hardening configurations that properly set different hardening parameters for the project.

5.5 Regression Testing

To ensure we are not creating errors and breaking our existing "functional" product, we will incrementally insert pieces of our project together and run the same tests or slightly edited versions to check our results. Assuming everything operates as we intend, we will see the same results and individual stages throughout this testing. The two main stages to test our projects with and without the OpenRAM modules. Without memory should prove to be much easier, and once we have that as a base for testing we can insert multiple memoryless projects to ensure we can operate more than one project in our project. Finally, we would complete the same process with memory projects and a combination of both.

5.6 Acceptance Testing

Once we have ensured our design meets all functionality requirements, we will harden our design through the OpenROAD toolflow to ensure we meet timing and area requirements. This process includes place and route and finding the optimal hardening configuration. We will then verify our design meets all fabrication specifications by running Efabless precheck on our project. Precheck will perform additional DRC and LVS tests.

5.7 Bring-Up & Physical Testing

After passing acceptance testing and submitting our design to Efabless, we will develop a bring-up testing plan. This will include a test project template that future users will use to test that their project can be integrated into our framework. The template will include the OpenRAM modules and interface, as well as the expected connections with the rest of the chip. Users will utilize the template to ensure their design successfully interfaces with all components of our framework before integrating their project into our design.

We will also utilize the ISU Chip Fabrication FPGA board for physical tests. Our design will be flashed to the FPGA board, so we can perform physical testing before our chip finishes fabrication.

5.8 Results

Our testing results will be waveforms collected from running test benches on our components. Because we are creating the test benches, we will have expected output values when looking at the waveforms. We can tell if we meet the speed and data requirements needed to call our tests successful through these graphs. Additionally, successful acceptance testing will produce successful hardening and precheck logs that demonstrate our design meets all required checks.

Prior to submitting our design for tapeout on November 11, 2024, we successfully ran RTL and GL simulations and tests on the FPGA that verified our framework design. These tests included cases to test each core interface between the user projects and the chip resources, as well as independent project selection, reset, and functionality. Additionally, each project included in our tapeout

submission was tested within the framework by the students who authored each project. Results for these tests can be found in Appendix 10.3.

During hardening, we got timing warnings for max hold, slew, and capacitance violations. However, after careful analysis and direct communication with the Efabless team, we believe they aren't critical. We were able to determine the timing issues were caused by specific user projects, not the framework itself. We saw that if we took those user projects out of the framework and rehardened the design, the timing warnings went away. Furthermore, the paths causing these errors were between LA and IO pins, which are asynchronous interfaces. Therefore they were non-critical timing paths.

We also performed multiple iterations of chip-level static timing analysis, each time with different hardening parameters, to see their impact on the timing violations. These runs involved trying different clock frequencies and adjusting the max slew and capacitance margins. After going over these results with the Efabless team, they said our results were passable, and we submitted the iteration that minimized the timing violations.

Based on these results, we believe the risk to the functionality of the individual projects should be minimal once the chip comes back from fabrication. Additionally, it may also provide useful information for debug analysis during the bring-up process. Students in Chip Forge who will be testing our physical chip will be able to observe what timing violations may yield. If anomalous behavior does occur due, the students will be able to understand the root cause. Furthermore, as more chips are fabricated and brought up through the co-curricular, students will be able to see variation in performance between chips. Overall, the timing violations we faced gave both our team experience addressing real industry issues and it will provide future students analysis and debugging opportunities.

6 Implementation

6.1 Design Analysis

Semester 1:

Our implementation plan for next semester will cover Milestones 4-7:

- Milestone 4: Integrate All Modules
- Milestone 5: Test Overall Design
- Milestone 6: Submit Design to Efabless
- Milestone 7: Complete Bring-Up Documentation & Physical Testing

Currently, we have already begun implementing basic modules and performing unit tests. We plan to finish individual module implementation and begin integrating everything into the top-level wrapper shortly after next semester begins. Testing will take place throughout implementation until the Efabless submission deadline, in late October or early November. After submission, we will focus on developing our bring-up and physical testing plan.

Semester 2:

We fully implemented our framework with 8 fully functional user projects provided by the co-curricular team, and we submitted our design for tapeout by the Efabless deadline of November 11, 2024. Our final implemented framework matches the design outlined in section 4. It fully multiplexes the Wishbone, IO, and LA ports between all the projects, and each project can be independently selected and reset through programming the management core.

Our design has been tested through RTL, GL, and FPGA simulations. It was also verified through the Efabless hardening, precheck, and tapeout checks. These processes are all required for tapeout submission. In addition to running these checks, we were also in regular contact with members of the Efabless team. During these meetings, we were able to verify the results of our testing were reasonable. Overall, through multiple iterations of testing and design checks, we were able to verify our design is viable and was successfully submitted for tapeout.

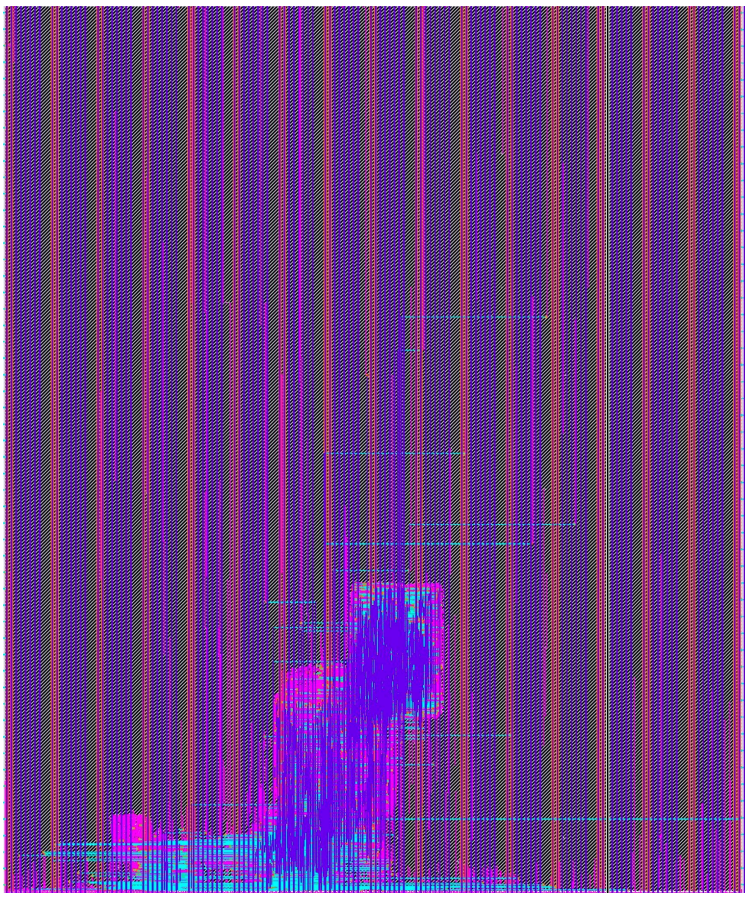

Figure 6: Final Hardened Design

7 Professional Responsibility

7.1 Areas of Responsibility

Responsibilities	Our Definition	IEEE Definition	NSPE Definition

Work Competence	Performing work that is your own and meets the standards you have set.	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts
Financial Responsibility	The product we deliver will be created using reliable components and with reliable services.	Deliver products and services of realizable value and at reasonable costs	Act for each employer or client as faithful agents or trustees
Communication Honesty	We will communicate and collaborate to provide the best product possible.	Report work truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts
Health, Safety, Well-Being	We will practice safe procedures and not cut corners to create hazards to the users or ourselves	Minimize risks to the safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public
Property Ownership	We will not steal the ideas or take responsibility for products that are not ours.	Respect the property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.
Sustainability	We will create a product that meets our standards and will withstand constant use.	Protect the environment and natural resources locally and globally.	N/A
Social Responsibility	Our product will better advance the users.	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.

Table 3: Areas of Responsibility

7.2 Project Specific Professional Responsibility Areas

Responsibilities	Relation to Our Project	Performance
Work Competence	We should hold ourselves to use our	We have been creating and using

	work and not steal from others.	our work and ideas within our project.
Financial Responsibility	We must be mindful of fabrication costs by working to meet submission deadlines.	We are working towards the Fall 2024 deadline.
Communication Honesty	We must hold ourselves accountable to communicate with each other and our client.	We have been communicating our progress to our client and sharing ideas between team members, but we could increase communications and timeliness.
Health, Safety, Well-Being	We should promote a supportive learning environment for our future users and for ourselves.	We are considering our users and the user experience throughout the design process.
Property Ownership	We must respect the ideas of our clients and our teammates.	We have shared and respected every team member's ideas and work throughout the project.
Sustainability	We should adopt energy-efficient design practices and choose sustainable materials for components.	We have make various design decisions based on performance trade-offs.
Social Responsibility	We are responsible for providing a product to enhance users' learning.	We have kept the goal of our product which is to enhance the learning capabilities of the user in mind while completing our project.

Table 4: Project-Specific Responsibility

7.3 Most Applicable Professional Responsibility Area

Our most applicable professional responsibility area is Social Responsibility.

Our team has a social responsibility to provide the best product we can to the student users to facilitate learning for future students. We plan to consider every type of user and how they might think differently. We will use these ideas to create an easy-to-use solution that allows them to learn more about ASIC development. We will also have thorough testing to ensure our product meets our standards. This approach will cover more than just our social responsibility to provide a product that facilitates the learning experience. It can also make it fun both for the users and us.

8 Conclusions

8.1 Summary of Progress

Our project was to build a chip framework that supports user ASIC projects created by students in the ISU Chip Fabrication Co-Curricular Team. For our design, we used open-source tools and project architecture provided by Efabless. Our chip framework integrated all project modules into a unified ASIC by connecting each module to the chip's GPIOs and management wrapper, or microcontroller. All of the chip's resources are multiplexed between the different projects, depending on which is active at the time. The design process involved creating a chip layout and fully functional management core, as well as configuration scripting, Verilog coding, and C programming. Through hardening tests and component simulations, we ensured that our design is functional and can be properly fabricated. We also developed a bring-up plan for future students that will use our framework.

Our group began by learning to use tools and equipment provided by our client and previous groups. This consisted of tests and examples by Caravel Efabless. After becoming familiar with the toolset we began to experiment and create our files and run our tests. This was the start of our framework, and most of our work is related to this process. We have since created a schematic for our design. This enabled basic functionality, through components such as register files, decoders, and muxes.

The biggest challenge our team faced was becoming familiar with the toolset and learning both the Verilog language, and how to integrate these modules into our design. There had been significant work done by a previous team that greatly assisted us in this process. Our next step was to determine the memory aspect of our project. There is not a lot of documentation on this, and it is up to our team to experiment and run tests to determine the best way to implement it. After many different attempts to integrate it, we consulted with the maintenance team for ChipIgnite. They gave us advice that allowed us to decisively do-away with this issue.

In the future, we will have to run tests on the completed framework to ensure that our project behaves the way it was designed. We will have to place an order for our design to be created and chips to be sent to Iowa State. Our design will then be iterated on by future students and potentially used as a part of a class.

We have now reached a finished state with regard to our original design. We submitted our design for fabrication on November 11th. We expect to receive a working copy back sometime in the next Calendar year, after our graduation. This will hopefully be a first step in pushing the FPGA extracurricular program forward at ISU, and allowing new users to further our work.

8.3 Value Provided

The value provided by our project is twofold. Because this was a novel project with no previously-established methodology, what we do is valuable not only for Iowa State, but for any future students who use this as a springboard for their careers. What we've created here will be documented, with our errors and discoveries laid out for future students to iterate upon such that this design can be further improved.

8.4 Next Steps

Our goal was to create a framework and feasibly expand the capacity of Chip Forge to create projects with student's designs. This we believe we were successful in. We submitted our design for fabrication on November 11th. It will be brought up by students in the Spring semester.

The next steps for our project are in the hands of the ISU Chip Forge co-curricular team. They will receive the physical chip on our behalf in the spring. When that happens, they may proceed to perform physical testing and usage of our chip in an academic or iterative context. Our Verilog design may be implemented into further projects and used to include undergraduate designs for fabrication. Furthermore, future senior design projects may choose to iterate based on our design and create something altogether more intricate.

9 References

Technical References:

"Caravel Harness," *Efabless*. [Online]. Available: https://caravel-harness.readthedocs.io/en/latest/. [Accessed April 16, 2024].

"Caravel Management SoC," *Efabless*. [Online]. Available: https://caravel-mgmt-soc-litex.readthedocs.io/en/latest/. [Accessed April 16, 2024].

Efabless, "Efabless/caravel_user_project," *GitHub*. [Online]. Available: https://github.com/efabless/caravel_user_project. [Accessed April 16, 2024].

"Tiny Tapeout," *Tiny Tapeout*. [Online]. Available: https://tinytapeout.com/. [Accessed April 30, 2024].

Figures:

[1] Efabless, "Efabless/caravel," *GitHub*. [Online]. Available: https://github.com/efabless/caravel. [Accessed April 16, 2024].

10 Appendices

10.1 Appendix 1 - Framework User Guide

The Chip Forge framework is able to support up to 32 separate project modules in a single Caravel user project. By multiplexing the user area's inputs and outputs to all project modules, the framework allows each project to be independently selected and run with full functionality, without affecting the other project modules.

This guide describes how to add and test projects within the framework, harden the framework, and submit the design to Efabless. For a full guide on developing and bringing up projects with Caravel, see the Chip Forge documentation here:
https://git-pages.ece.iastate.edu/isu-chip-fab/documentation/#/.

10.1.1 Adding a Project to the Framework

Project Requirements:

- Project only uses wishbone addresses **below** `0x38000000`
- Project is instantiated inside a user_project_wrapper.v file
- Project successfully hardens inside the user_project_wrapper.v file
- All source files are in the library repository

Implementing Your Project Inside the Framework:

1. Ensure all project source files are in the /library/[PROJECT_NAME]/verilog/rtl directory
2. Ensure there is an example user_project_wrapper.v file in /library/[PROJECT_NAME]/verilog/dv/[MODULE_NAME]
3. Checkout the framework GitLab repo. Make sure to do a recursive checkout, so the library submodule is also downloaded.

   ```
   git clone --recursive git@git.ece.iastate.edu:sd/sddec24-12.git
   ```

4. Copy the user_project_wrapper.v from /library/[PROJECT_NAME]/verilog/dv/[MODULE_NAME] to /verilog/rtl/projects/
5. Change the copied user_project_wrapper.v file name AND module name from "user_project_wrapper" to a new, unique name for the project module, ex. "wishbone_adder_project"
6. In /openlane/user_project_wrapper/config.json, add the relative path of all project source files (in /library/[PROJECT_NAME]/verilog/dv/[MODULE_NAME] AND /verilog/rtl/projects) to the VERILOG_FILES parameter.

   ```
   "dir::../../library/[PROJECT_NAME]/verilog/rtl/[SUBMODULE_NAME]",
   …
   "dir::../../library/[PROJECT_NAME]/verilog/rtl/[MODULE_NAME]",
   "dir::../../verilog/rtl/projects/[PROJECT_MODULE]"
   ```

7. Add an instantiation of your /verilog/rtl/projects/[PROJECT_MODULE] to /verilog/rtl/user_project_wrapper above the CONTROL LOGIC comment header

    a. In the generate loop before the CONTROL LOGIC comment header, increment the initial i value in the for loop. Make note of the **old** i value.

```
// before
for (i = 8; i <= 31; i = i + 1) begin

// after
for (i = 9; i <= 31; i = i + 1) begin
```

    b. For each port in the module instantiation, pass in the following signals. For all arrays, set the index to the **old** i value. (In the example above, index=8)

Port	**Signal to pass in**
wb_clk_i	wb_clk_i
wb_rst_i	wb_rst_i_s[index]
wbs_cyc_i	wbs_cyc_i_s[index]
wbs_stb_i	wbs_stb_i_s[index]
wbs_we_i	wbs_we_i_s[index]
wbs_sel_i	wbs_sel_i_s[index]
wbs_adr_i	wbs_adr_i_s[index]
wbs_dat_i	wbs_dat_i_s[index]
wbs_ack_o	wbs_ack_o_bus[index]
wbs_dat_o	wbs_dat_o_bus[index]
la_data_in	la_data_in_s[index]
la_data_out	la_data_out_bus[index]
la_oenb	la_oenb_s[index]
io_in	io_in_s[index]
io_out	io_out_bus[index]
io_oeb	io_oeb_bus[index]
user_clock2	user_clock2
user_irq	user_irq_bus[index]

Note: It may be easiest to copy a previous project module instantiation and update the module name and index.

After completing these steps, your project should be successfully instantiated inside the framework and ready for testing within the framework.

10.1.2 Testing Projects Within the Framework

Any existing testbenches and test C code written for individual projects may be reused to test that project within the framework with slight modifications:

1. Copy the existing library/[PROJECT_NAME/verilog/dv/[TEST_NAME] folder into /verilog/dv. Rename the test directory and testbench if desired.
2. Delete the user_project_wrapper.v file from the dv/[TEST_NAME].
3. In the C code file, add the following defines at the top of the file

```
#define PROJ_SELECT (*(volatile uint32_t *)0x38000000)
#define PROJ_RESET (*(volatile uint32_t *)0x38000004)
```

4. In the main test method, before any test cases, write the project's index to the PROJ_SELECT register. Reset the project by writing the same value to PROJ_RESET if desired, then clear the reset by writing oxFFFFFFFF.

```
PROJ_SELECT = 0x00000001; // select proj1
PROJ_RESET = 0x00000001; // reset proj1
PROJ_RESET = 0xFFFFFFFF; // clear reset
```

   The project's index will be the same value set in step 7b under "Adding Project to the Framework"

5. To deselect all projects, write oxFFFFFFFF to PROJ_SELECT.

The test may now be run as either an RTL or GL simulation with the normal commands:

```
make verify-TEST_NAME-rtl
make verify-TEST_NAME-gl
```

10.1.3 Hardening the Framework

The framework is hardened without any pre-hardened submodules. The hardening configuration file for the user_project_wrapper should be located at openlane/user_project_wrapper/config.json. The following configuration file should be set as follows:

All source files for *every* project must be listed under VERILOG_FILES, like the following:

```
"VERILOG_FILES": [
        "dir::../../verilog/rtl/defines.v",
        "dir::../../verilog/rtl/projects/adder_regfile_project.v",
        "dir::../../verilog/rtl/projects/sdmay25_17_project.v",
```

```
        "dir::../../verilog/rtl/projects/spi_controller_project.v",
        "dir::../../verilog/rtl/projects/wishbone_adder_project.v",
        "dir::../../verilog/rtl/projects/wishbone_pwm_project.v",
        "dir::../../verilog/rtl/projects/variable_precision_mac_project.v",
        "dir::../../verilog/rtl/projects/wishbone_seven_seg_project.v",
        "dir::../../library/adder_regfile_datapath/verilog/rtl/add_sub.v",
        "dir::../../library/adder_regfile_datapath/verilog/rtl/datapath1.v",
        "dir::../../library/adder_regfile_datapath/verilog/rtl/regfile.v",
        "dir::../../verilog/rtl/user_project_wrapper.v"
    ],
```

Additionally, use these settings as a reference for hardening the wrapper:

```
      "BASE_SDC_FILE": "dir::base_user_project_wrapper.sdc",
      "CLOCK_PERIOD": 25,
      "CLOCK_PORT": "wb_clk_i",

      "___0": "Set to number of cores to use for routing, dependent on build
machine",
      "ROUTING_CORES": 16,
      "KLAYOUT_XOR_THREADS": 16,
      "KLAYOUT_DRC_THREADS": 16,

      "___1": "Disable to speed up builds, test with these enabled before
submission",
      "RUN_KLAYOUT_XOR": 1,
      "RUN_KLAYOUT_DRC": 1,

      "PL_TARGET_DENSITY": 0.3,
      "FP_CORE_UTIL": 45,
      "IO_SYNC": 1,
      "MAX_TRANSITION_CONSTRAINT": 1.0,
      "MAX_FANOUT_CONSTRAINT": 16,
      "PL_RESIZER_SETUP_SLACK_MARGIN": 0.4,
      "GLB_RESIZER_SETUP_SLACK_MARGIN": 0.2,
      "GLB_RESIZER_HOLD_SLACK_MARGIN": 0.2,
      "PL_RESIZER_HOLD_SLACK_MARGIN": 0.4,
      "SYNTH_BUFFERING": 0,
      "RUN_HEURISTIC_DIODE_INSERTION": 1,
      "HEURISTIC_ANTENNA_THRESHOLD": 110,

      "___2": "Change synth strategy if timing errors are not being
resolved",
      "SYNTH_STRATEGY": "AREA 0",

      "RUN_LINTER": 1,
      "PL_RANDOM_GLB_PLACEMENT": 0,
      "SYNTH_ELABORATE_ONLY": 0,
      "PL_RESIZER_DESIGN_OPTIMIZATIONS": 1,
      "PL_RESIZER_TIMING_OPTIMIZATIONS": 1,
      "GLB_RESIZER_DESIGN_OPTIMIZATIONS": 1,
      "GLB_RESIZER_TIMING_OPTIMIZATIONS": 1,
```

```
      "PL_RESIZER_BUFFER_INPUT_PORTS": 1,
      "FP_PDN_ENABLE_RAILS": 1,
      "GRT_REPAIR_ANTENNAS": 1,
      "RUN_FILL_INSERTION": 1,
      "RUN_TAP_DECAP_INSERTION": 1,
      "FP_PDN_CHECK_NODES": 1,
      "RUN_CTS": 1,
      "RUN_CVC": 1,
      "MAGIC_DEF_LABELS": 0,
      "QUIT_ON_SYNTH_CHECKS": 0,
      "FP_PDN_VPITCH": 180,
      "FP_PDN_HPITCH": 180,
      "FP_PDN_VOFFSET": 5,
      "FP_PDN_HOFFSET": 5,
      "MAGIC_ZEROIZE_ORIGIN": 0,
      "FP_SIZING": "absolute",
      "UNIT": 2.4,
      "FP_IO_VEXTEND": "expr::2 * $UNIT",
      "FP_IO_HEXTEND": "expr::2 * $UNIT",
      "FP_IO_VLENGTH": "expr::$UNIT",
      "FP_IO_HLENGTH": "expr::$UNIT",
      "FP_IO_VTHICKNESS_MULT": 4,
      "FP_IO_HTHICKNESS_MULT": 4,
      "FP_PDN_CORE_RING": 1,
      "FP_PDN_CORE_RING_VWIDTH": 3.1,
      "FP_PDN_CORE_RING_HWIDTH": 3.1,
      "FP_PDN_CORE_RING_VOFFSET": 12.45,
      "FP_PDN_CORE_RING_HOFFSET": 12.45,
      "FP_PDN_CORE_RING_VSPACING": 1.7,
      "FP_PDN_CORE_RING_HSPACING": 1.7,
      "FP_PDN_VWIDTH": 3.1,
      "FP_PDN_HWIDTH": 3.1,
      "FP_PDN_VSPACING": "expr::(5 * $FP_PDN_CORE_RING_VWIDTH)",
      "FP_PDN_HSPACING": "expr::(5 * $FP_PDN_CORE_RING_HWIDTH)",
      "VDD_NETS": [
      "vccd1",
      "vccd2",
      "vdda1",
      "vdda2"
      ],
      "GND_NETS": [
      "vssd1",
      "vssd2",
      "vssa1",
      "vssa2"
      ],
      "SYNTH_USE_PG_PINS_DEFINES": "USE_POWER_PINS",
      "pdk::sky130*": {
      "RT_MAX_LAYER": "met4",
      "DIE_AREA": "0 0 2920 3520",
      "FP_DEF_TEMPLATE":
"dir::../../openlane/user_project_wrapper/fixed_dont_change/user_project_wra
pper.def"
```

```
        }
}
```

10.1.4 Submitting to Efabless

In order to submit a design for tapeout, all source files must be committed and pushed to an Efabless Git repository. Due to the Efabless servers and toolflow setup, the Efabless Git repository cannot contain any Git submodules. Pushing large file sizes may also cause corruption.

The following example steps can be used to properly push the framework from an Iowa State GitLab repository to an Efabless repository:

```
git clone git@git.ece.iastate.edu:sd/sddec24-12 sddec
cd sddec
git checkout tapeout
git submodule update --init
cd library/
git fetch # Verify library is latest commit
cd ..
git log
make setup
make compress
cd ..
cp sddec/verilog/gl/user_project_wrapper.v.gz .
gunzip user_project_wrapper.v.gz
cat user_project_wrapper.v
cd sddec
git log

# Delete library submodule references
rm .gitmodules
rm library/.git
rm -rf .git

# Remove repository history
git init
git add -A
git status
git commit -m "Final Tapeout"
git remote add origin
ssh://git@repositories.efabless.com/duwe/ChipForgeFrame.git
git branch
git push --force origin master:main
cd ..
git clone ssh://git@repositories.efabless.com/duwe/ChipForgeFrame.git tmp
cd tmp
cd verilog/
cd gl
gunzip user_project_wrapper.v.gz
cat user_project_wrapper.v
```

10.2 Appendix 2 - Initial Design

10.2.1 Requirements

Functional Requirements

Our project client has outlined the following functional requirements our project must fulfill:

- Framework holds 15-20 projects, including small and large projects
- Management wrapper controls which project is active
- One project is active at a time
- Chip resources will be multiplexed between all the projects
- Projects will interface with the Wishbone bus, LA pins, and IO pins
- The framework will include external OpenRAM modules
- Design successfully passes Efabless precheck

Technical Requirements

To meet the expected functionality, our design implementation must meet the following requirements:

- The development follows the Efabless process
- Design is implemented in Verilog
- Test code is written in Verilog and C
- The final design generates a GDS2 file
- The design achieves a frequency of at least 20 MHz

User-Based Requirements

To ensure proper user interaction and bring-up, our project must meet the following requirements:

- The project is fully documented on the ISU Chip Fabrication website
- Detailed descriptions of the project explain the framework architecture
- The framework is straightforward to use with minimal help or troubleshooting

10.2.2 Detailed Design and Visual(s)

Figure 7 depicts the initial design composition of our user project framework. Our design connects small, interchangeable user projects, pictured on the right, to the resources provided by the Management Core SoC, pictured on the left, and the GPIO ports. Only one user project will be active and communicating with the Managemen SoC at a time. The Management SoC contains the Wishbone Bus (WB) Master and the Logic Analyzer (LA) modules. Both of these are used to communicate with the user projects and transfer data, and there are separate ports for data going in and out of these modules.

Each project can be reset individually. The project reset bus signal will come from the Wishbone Bus and be stored in the RST register. The active user project will be selected using a value outputted by the Wishbone Bus and stored in the SEL register. Each project's data output will be multiplexed based on the selected value and returned to the Management SoC. The data sent from the Management SoC to the small projects will be inputted to the active small project. Inactive projects will receive data that is all zeros. 2-1 multiplexers will select between the resource data and 0 values, and the select bits will come from a one-hot decoder.

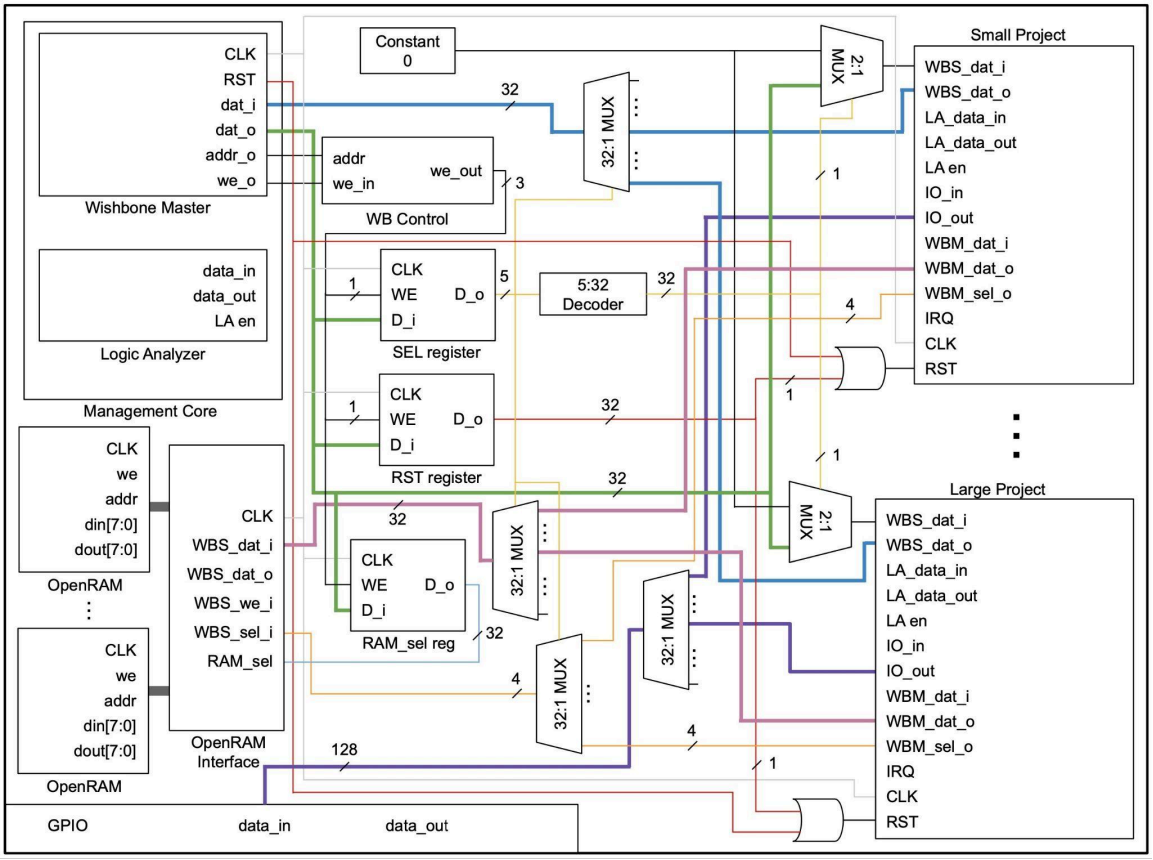

Figure 7: Initial Design Schematic

Acronyms

- **CLK:** Clock; system clock used by the Management SoC and all projects
- **GPIO:** General-Purpose Input/Output; ports handling both incoming and outgoing digital signals
- **IO:** Input/Output; data transfer to or from the chip
- **IRQ:** Interrupt Request Signal; designates peripheral devices/processes that are ready within a module to be serviced by the Management SoC
- **LA:** Logic Analyzer; sends logic signals between the Management SoC and user projects
- **RST:** Reset; when high, clears all module values to 0
- **SoC:** System-on-a-chip, integrated circuit design that combines the functions of an electronic device onto a single chip
- **WB:** Wishbone Bus; connects modules to the Management SoC and handles data transfers
- **WBS:** Wishbone Bus Slave; module that responds to transactions initiated by the Wishbone Bus master in the Management SoC
- **WE:** Write enable; signal that enables writing data to the respective location

10.2.3 Simulation Waveforms

Based on our initial design, our framework included several structural logic modules that were tested independently. The test results are shown below.

We have finished implementation and unit testing for the following modules:

- 32-to-1 decoder
- N-bit register
- Wishbone control module

The waveform results for these unit tests are shown in Figures 8-10.

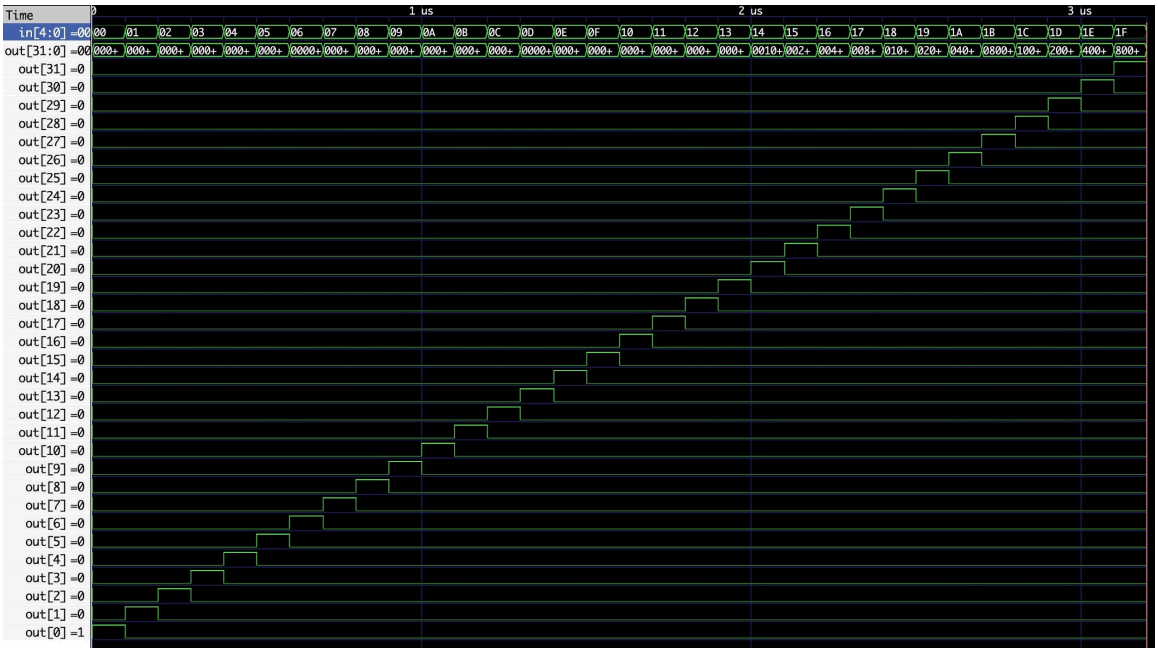

Figure 8: 32-to-1 Decoder

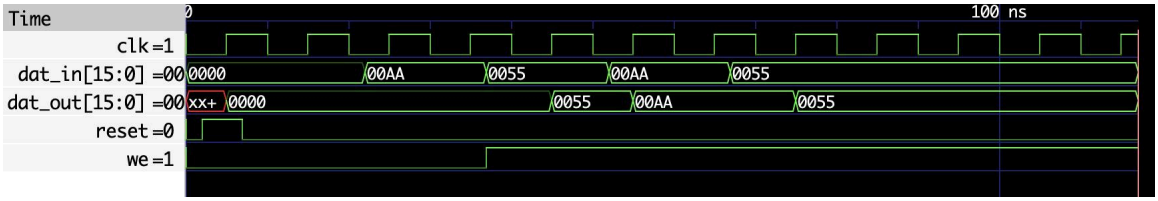

Figure 9: N-Bit Register

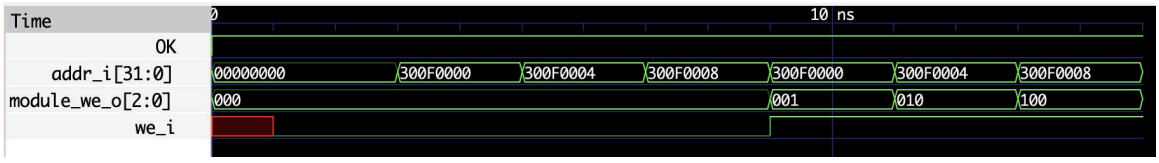

Figure 10: Wishbone Control Module

10.3 Appendix 3 - Testing Results

To test the functionality of our framework, we ran a test that selected the two different adder projects, ran independent test cases on each, and reset each independently. The waveform results are shown in Figure 11.

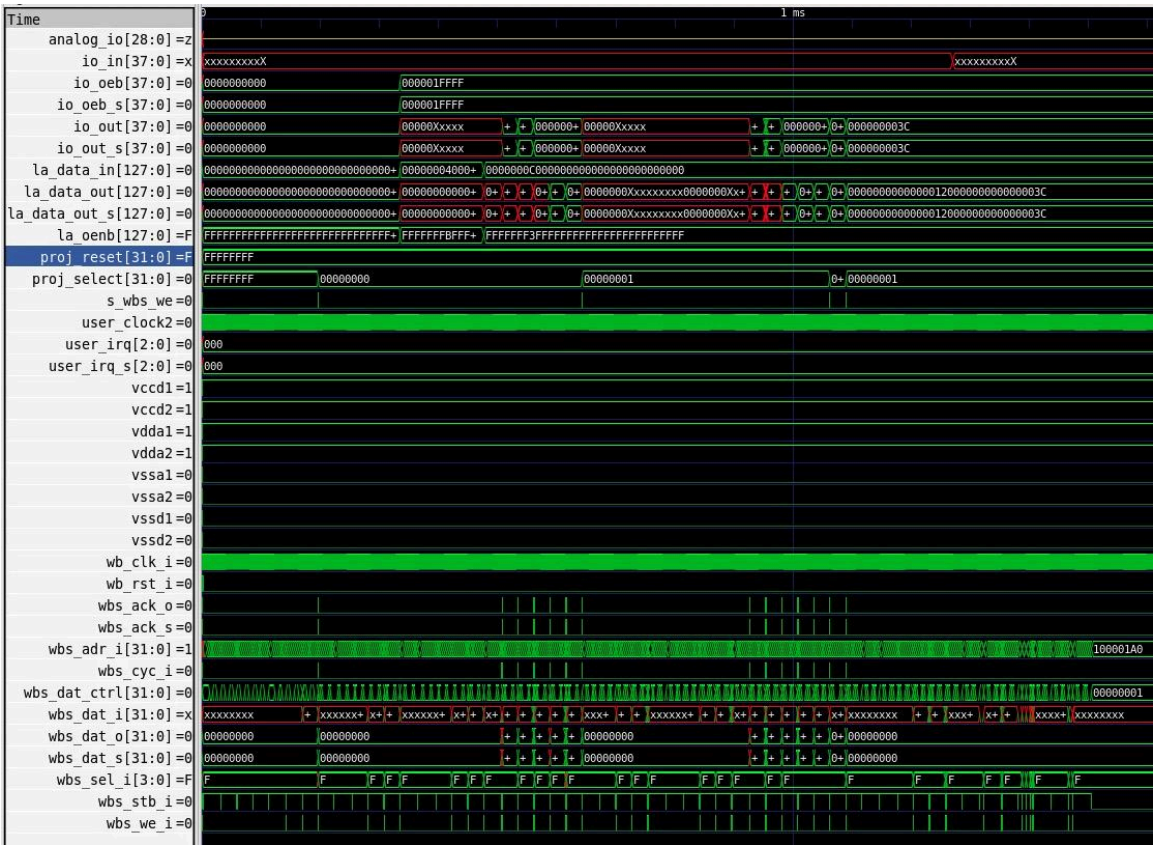

Figure 11: Adder Projects in Framework

This test was run as both an RTL and GL simulation, and both produced correct results.

Other projects within our framework could not be verified through basic simulation, so these projects were tested on the FPGA. The results for the senior design seven segment display project are shown in Figure 12.

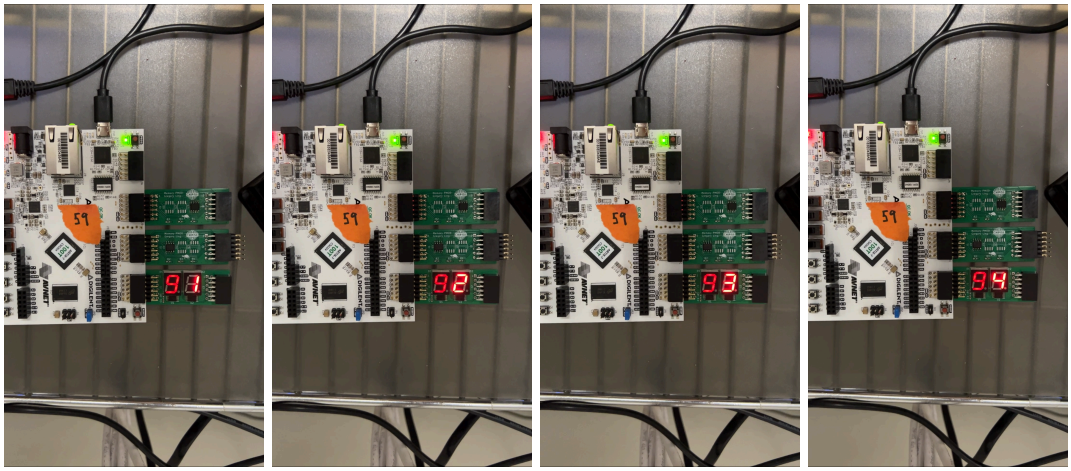

Figure 12: Seven Segment Display in Framework

These tests verified the interfaces between the projects and the Wishbone bus, LA pins, and IO pins, as well as project selection and reset, were functioning as expected.

10.4 Appendix 4 - Team

10.4.1 Team Members

- Mitchell Driscoll
- Evan Dunn
- Baoshan Liang
- Katie Wolf

10.4.2 Skill Sets Covered by the Team

- Mitchell – Circuit board design, code debugging
- Evan – Verilog development
- Baoshan – Verilog code debugging
- Katie – System design, Verilog development

10.4.3 Project Management Style Adopted by the team

Combination of Waterfall and Agile

10.4.4 Initial Project Management Roles

- Mitchell - Component Implementation
- Evan - Researcher, Design Testing
- Baoshan - Design Testing
- Katie - System and Component Design

10.4.5 Team Contract

Team Members:

1) Mitchell Driscoll 2) Evan Dunn

3) Boashan Liang 4) Katie Wolf

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
    - Tuesday 4-5 pm with Dr. Duwe
    - Thursday 2-3 pm with just team members
    - As needed digitally when something comes up.
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    - Discord and Teams
3. Decision-making policy (e.g., consensus, majority vote):
    - Consensus, with potential veto with cause.
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    - Use GitHub and Google Drive to take notes and share files
    - Share links and resources through Discord

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    - All members are present and on time for meetings
    - Let others know ahead of time if you will be late or absent
    - Exceptions permitted in regard to unexpected delays, sicknesses, etc
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
    - Meet deadlines as best as you can
    - All members contribute to team assignments
    - Submissions and contributions should be submitted in a timely manner unless otherwise excused via aforementioned exceptions.
3. Expected level of communication with other team members:
    - Regular communication throughout the week
    - Let others know immediately if issues or concerns come up
    - Respond to messages within 24 hours
4. Expected level of commitment to team decisions and tasks:
    - Commit to decisions decided as a team
    - Voice concerns as soon as possible
    - Each member should commit at least a significant portion of their time to the active development of the project.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
    - Evan - team organization
    - Mitchell - schematic design
    - Baoshan - design testing
    - Katie - schematic and component design

2. Strategies for supporting and guiding the work of all team members:
    - Gentle reminders, along with active guidance regarding development and subject matter. A clear statement of expectations when it comes to weekly and monthly progress.
    - Excessively plan every facet of the project so that there are no surprises and everyone can fully commit.
3. Strategies for recognizing the contributions of all team members:
    - Rigorously document individual and group accomplishments throughout the Semester.
    - Clear documentation and acknowledgment of past accomplishments. Focus on the project as a whole. with support from others.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
    - Evan - Computer Engineering Major, Digital design, “Industry contacts.”
    - Mitchell - Electrical Engineering major, Schematic and PCB design
    - Baoshan - Electrical Engineering major, Schematic and layout, DRC and LVS check
    - Katie - Computer Engineering major, computer architecture, digital design
2. Strategies for encouraging and supporting contributions and ideas from all team members:
    - Asking everyone for ideas
    - Looking into all the ideas that are given
    - Discuss ideas as necessary to obtain consensus
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment obstructs their opportunity or ability to contribute?)
    - Be honest and communicate early on
    - Get the entire team’s input on issues
    - Consult faculty advisor and/or professor

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
    - Meet the client’s expected deliverables given in the project documentation.
    - Establish early on team expectations and finish ahead of deadlines to maximize testing.
    - Establish a unified level of knowledge regarding the subject material.
2. Strategies for planning and assigning individual and teamwork:
    - Assign tasks based on team members’ expertise and availability
    - Make sure all members have around the same amount of responsibility
    - Use our suggested communication systems to inform the group when doing solo work.
3. Strategies for keeping on task:
    - Meet regularly and review upcoming tasks and deadlines
    - Check in on each other and hold each other accountable
    - Request review by a peer or sponsor if any systemic or personal problems arise.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
    - Talk to the other team members
    - Review the contract and expectations
    - Involve higher authority when necessary.
    - Mandatory disembowelment.
2. What will your team do if the infractions continue?
    - Talk to Professor Shannon and Professor Fila
    - Discuss further measures.
    - Repeat the process of disembowelment - just for good measure.

---

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) Mitchell Driscoll DATE: 1/30/24

2) Evan Dunn DATE: 1/30/24

3) Baoshan Liang DATE: 1/30/24

4) Katie Wolf DATE: 1/30/24