

Open SI ASIC Chip Design

DESIGN DOCUMENT

Sdmay23-28

Client and Advisor: Dr. Henry Duwe

Tyler Green, Katherine Gisi, Aaron Sledge, William Zogg, Fulai Zhu

<https://sdmay23-28.sd.ece.iastate.edu>

sdmay23-28@iastate.edu

1 Team

1.1 PROJECT MANAGEMENT STYLE ADOPTED BY THE TEAM

Our group completed our project using the agile project management style. One major component of our project was doing many iterations on several pieces of the project, with emphasis on both design and testing. Due to the needs of the Caravel project (overall size, ability to harden, etc.), going back and reworking components was a natural part of our process. We will also be sought input from our client over the course of the project because of our goal of developing a curriculum around chip design for new students in CprE. This management style worked relatively well for us.

1.2 INITIAL PROJECT MANAGEMENT ROLES

- Katherine Gisi: Spokeswoman, SNN neuron design
- William Zogg: SNN neuron Design
- Tyler Green: SNN Design/Implementation, Software tool research, Environment setup
- Aaron Sledge: Software tool research & Documentation
- Fulai Zhu: Documentation

2 Introduction

2.1 PROBLEM STATEMENT

Iowa State University's ECpE department does not offer a full scope chip design curriculum that allows for students to design a chip, get it fabricated, and bring up their own design. This is an issue especially with the increasing demand for people in the workforce that can design complex chips. Our group is aiding in laying down the groundwork for getting a chip design curriculum, that encompasses the full scope of the process, started at ISU. We have undergone the design process and documented our experiences in Open-Source ASIC creation.

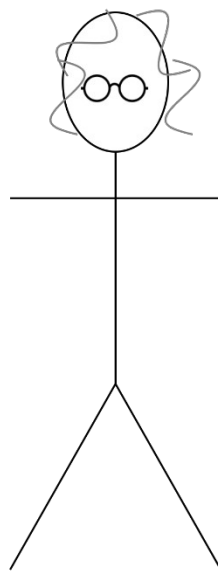
2.2 INTENDED USERS AND USES

The general user groups of this project are threefold. The present users are those that will be continuing to build the infrastructure surrounding open-source ASIC bring-up at Iowa State University. These include our senior design team, future senior design teams, and our client who is a professor interested in building a cocurricular in line with this

project. The future users will be those students of that potential cocurricular. Lastly, the broader users will be the open-source silicon community.

The present users who are partnering with our team to build infrastructure at ISU are described as outlined below.

- **Key Characteristics:** Interested in chip design, wants a working chip at the end of the day, looking to learn and share knowledge, have a desire to show their aptitude for chip design for future work-related opportunities.
- **Needs:** Documentation on processes involved, documentation on how to operate open-source design tools, solutions to potential problems, example projects to build that knowledge base.
- **Benefits from this project:** Knowledge gained from chip design process, knowledge of open-source design tools, ability to teach others, have a novel project to present to employers for future work-related opportunities.



Dr. Duwe Client

Demographic:

Age 30, Male, Professor at ISU, Resides in Ames IA

Hobbies & Interests:

Chip Fabrication, IOT, Teaching

Work Motivations:

Intellectual curiosity, developing a cocurricular, investing in students

Personality & Emotions:

Curious, smart, big picture thinker without loosing the small stuff

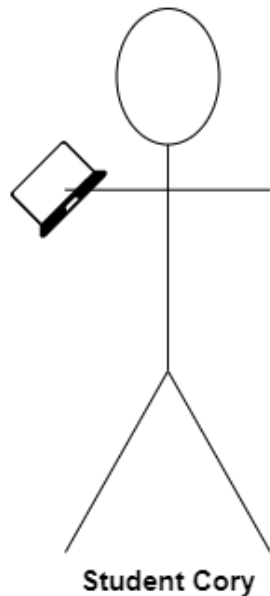
Values:

Student experiences, family

The future users who may be involved in a class based on the work done by senior design projects such as this one can be characterized similarly.

- **Key Characteristics:** ECpE students, enrolled in a chip design course, want an end product/interested in something to hold in their hand. Additionally, future students of CprE 487/587 want the best class experience and labs possible.
- **Needs:** A relatively smooth class experience, examples to draw from, modularity of process flow. For 487/587 students, a working, reliable spiking neural network ASIC that can be used for a class lab.

- Benefits from this project: Completed design example, documentation for course work, knowledge given to the professor who may teach this course. For 487/587 students, a new piece of hardware to further the lab experiences possible for the course, have a novel project to present to employers for future work-related opportunities.



Demographic:

Age 18, Male, Computer Engineering, Senior, Resides in the Social West, Single

Hobbies & Interests:

VLSI, programming, outdoors, music

Work Motivations:

Complete major requirements, design a chip, build resume, build connection with project mentor.

Personality & Emotions:

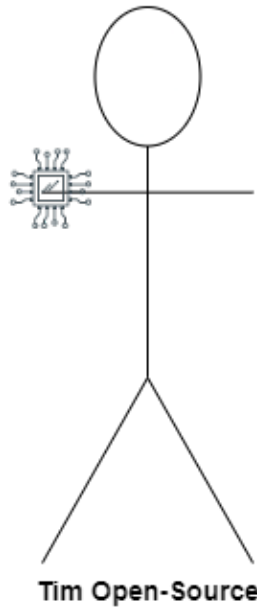
Driven, doesn't sweat the small stuff, will lead if need be.

Values:

Developing an end product, graduating with a good grade, learning about ASIC design

The open-source community is a user by nature of this project. Open-source silicon is a relatively new field, and the program that will be participated in for the fabrication of this chip was created to build the OS ecosystem in general. The attributes of the community as listed below.

- Key Characteristics: Participating in similar projects, may or may not have previous chip design experience, possibly looking to improve upon early iterations of a design
- Needs: New projects released and developed using fully open-source tools and information, well-documented instructions on how to use the product
- Benefits from this project: The open-source community will gain a new resource for a spiking neural network ASIC, as well as documentation and code for continuing/modifying an existing design

**Demographic:**

Age 40, Male, Electrical Engineer Hobbyist, 2 kids, Researches at University, Resides in Austin TX.

Hobbies & Interests:

Open Source ASICs, Family time, programming, PCB design

Work Motivations:

Researches in this area, intellectual curiosity, career growth, believes OS mission.

Personality & Emotions:

Curious, introverted, do-it-yourself kind of guy.

Values:

Knowledge sharing, investing in the net generation, ideation.

2.3 REQUIREMENTS & CONSTRAINTS

Physical Requirements

- Our logic must fit within the space provided by eFabless, specifically in a 10 square mm area (constraint)

Environment Requirements

- The project must be posted on a git-compatible repo and be publicly accessible and contain the following items:
 - The Caravel test harness with the SNN occupying the user space for the layout.
 - The GDSII in a compressed format
 - A makefile with targets to compress/uncompress, and clean the project
 - All source code required to generate the GDSII including any third-party components
 - The makefile should also include verify target to execute a test verification suite for the design
 - LICENSE files for the top-level project as well as each of the third-party components used
- Project is designed utilizing open-source tools (constraint)
- Project should pass checks provided in caravel harness GitHub repository before submitting to eFabless (constraint)

Design and Functionality Requirements

- Logic represents at least one neuron in a Spiking Neural Network
- The project must be targeted on the currently supported SkyWater Open PDK for the 130nm process
- Projects must use a common test harness and padframe based on the Caravel repo.
- Design uses a digital configuration
- Design has some reasonable amount of novelty
- Design must successfully pass the Open MPW precheck tool, including LVS and DRC clean using the referenced versions of OpenLane flow
- Design should implement and pass a simulation test bench for their design integrated into Caravel
- Functionality of fabricated design is software testable

Documentation Requirements

- Repository should have a README file
- Create a bring up plan for future students to use to bring up our chip design in the caravel infrastructure.
- Create a tutorial document explaining all open-source tools necessary for the Efabless process.

2.4 ENGINEERING STANDARDS & PRACTICES USED

IEEE 1076.4-1995 - VITAL ASIC Modeling Specification: Since our group is designing an ASIC according to a preset standard, we are therefore able to clearly communicate all aspects of our design to other engineers as necessary.

IEEE 1500-2022 - Testability Method for Embedded Core-based Integrated Circuits: Our group is creating an ASIC that will be tested, and those tests will follow a testing standard that will allow for others who come across our open-source design to easily understand what it is that we are doing. And validate the testing that we have done on our design.

IEEE 1801-2018 - Design and Verification of Low-Power, Energy-Aware Electronic Systems: This standard applies to our design because it will allow for others to analyze our chip timing and power consistently across a broad set of electric design automation.

3 Design

3.1 DESIGN CONTEXT

3.1.1 Broader Context

Spiking Neural Networks (SNN) is a third-generation network, it can be applied in many areas, such as image processing, voice recognition, and human-like computing (AI). Compared to other generation networks, SNN has advantages of higher energy efficiency, higher area efficiency, efficiency on chip learning and more fault tolerance. The disadvantages of SNN are training is difficult, the accuracy does not match the other generation networks, and programming frameworks are still in their infancy.

Our project is to make an open-source SNN in an application specific integrated circuit (ASIC), with the main point of our project being that we can provide others with our solution and document the process to help others complete their own ASIC. The communities that our design will be for will be for those looking to further the open-source design of digital chips. Specifically, those who are a part of the slack community looking to add to the amount of open-source chip designs that exist. This project will fulfill the needs of all who are looking to accelerate the growth and complexity of digital chip ideas and projects alike.

Area	Description	Examples
------	-------------	----------

<p>Public health, safety, and welfare</p>	<p>How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or may be indirectly affected (e.g., solution is implemented in their communities)</p> <p>Our project won't necessarily have a large effect on the public's health. But it does influence the public's welfare & quality of life. Because as digital chip designs grow more complex at a faster rate there will be more complex and useful applications in the electronics that are used by the public.</p>	<p>Increasing/reducing exposure to pollutants and other harmful substances, increasing/reducing safety risks, increasing/reducing job opportunities</p> <p>If our design is used for purposes related to health research/studies, the results of those may have a positive impact on public health, which would be indirectly caused by our design.</p>
<p>Global, cultural, and social</p>	<p>The open-source culture is one of openness, growth, and collaboration. Our project is part of an open-source initiative for silicon design. It will accordingly reflect the values of the community as we aim to create a foundation for future groups whether in our University or the open-source silicon community at large.</p>	<p>This project as a spiking neural network accelerator can be used as a component of an artificial intelligence system. AI comes with its own set of challenges and potential violations of community ethics.</p>
<p>Environmental</p>	<p>What environmental impact might your project have? This can include indirect effects, such as deforestation or unsustainable practices related to materials manufacture or procurement.</p> <p>The materials that are used to fabricate our chip are natural resources. The mining of those and the fabrication process itself is often unsustainable and has an environmental impact in the waste product that is disposed of.</p>	<p>Increasing/decreasing energy usage from nonrenewable sources, increasing/decreasing usage/production of non-recyclable materials</p> <p>Our project is to fabricate the chip, so the usage of nonrenewable sources and non-recyclable materials such as silicon, photoresist or other chemical products</p>

		will increase.
Economic	<p>What economic impact might your project have? This can include the financial viability of your product within your team or company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.</p> <p>In the long term, open-source chip design has the potential to disrupt the industry and pave the way for fast paced innovation and economic growth in that area. Cost of devices to consumers could be reduced. In the short term, a curriculum at Iowa State for chip design will create more learned future engineers and give a broader base for students to draw off and implement their unique, creative ideas.</p>	<p>Product needs to remain affordable for target users, product creates or diminishes opportunities for economic advancement, high development cost creates risk for organization</p> <p>Our final product, a fabricated SNN neuron chip design, will be free and publicly available for other’s use. It will need to be documented well.</p>

3.1.2 Prior Work/Solutions

Current Literature

One of the literature pieces that we are using as a basis for our project is called “An Adaptive Memory Management Strategy Towards Energy Efficient Machine Inference in Event-Driven Neuromorphic Accelerators”. Now we are primarily using this literature piece to get a general idea of how SNN will likely need to be laid out as well as some of the high-level component ideas that will be needed for our SNN. The document can be found below:

An Adaptive Memory Management Strategy Towards Energy Efficient Machine Inference in Event-Driven Neuromorphic Accelerators: <https://par.nsf.gov/servlets/purl/10113424>

Advantages:

- Don’t have to start designing SNN from scratch
- Gives an idea of some of the high-level components necessary to construct circuits
- Gives an idea of general functionality of the high-level components that it does describe

Disadvantages:

- The version of their SNN is more Bio based and therefore some of the components in there do not apply to our SNN

Relevant Projects

This project was a continuation of an cocurricular development by the client. As such, a previous Iowa State senior design team performed a similar project and has provided resources to this team to continue developing a knowledge base. The work of the previous senior design team can be found as described in their website linked below.

Website: <http://sddec22-17.sd.ece.iastate.edu/>

This team will be following their personal advice, environment set-up documentation, and design flow. Our project will be different from theirs in functionality and structure. While their team completed the design of a bitcoin mining chip, our team will be creating a SNN accelerator.

Related Models

FPGA implementation is a close relative to ASIC design. To guide our design process on SNN acceleration, our team will be gleaning information from related FPGA projects. Our project is fundamentally different as the functionality is built into the circuit before the chip is created, but much of the circuit creation is similar. To get a grasp of the inputs, outputs, and steps required in an SNN, the paper “FPGA implementation of Spiking Neural Networks” was referenced.

Rosado-Muñoz, A., Bataller-Mompeán, M., & Guerrero-Martínez, J. (2016, April 21). FPGA implementation of spiking neural networks. IFAC Proceedings Volumes. Retrieved October 21, 2022, from

https://www.sciencedirect.com/science/article/pii/S1474667015404562?ref=pdf_download&fr=RR-2&rr=75dc73ab0b3c8114

In breaking down the design of our neural network, logic structures were created. The work “Design of Various Logic Gates in Neural Networks” provided a starting place for our design.

Yellamraju, S. (2013, December 23). Design of Various Logic Gates in Neural nNetworks. ResearchGate. Retrieved October 21, 2022, from

https://www.researchgate.net/publication/259990623_Design_of_Various_Logic_Gates_in_Neural_Networks

3.2 DESIGN EXPLORATION DURING 491

3.2.1 Design Decisions

The first decision we made is to make updates and iterations to user adder example and make changes to user adder example, by trying and adding components and change the functionality of components within the user adder example to gain more experience and familiarity with the design environment.

The second decision we made is to create an initial design sketch and design flow chart. This is important because we are laying the groundwork for creating high-level diagrams of circuits, identifying what components need to be present in each high-level section of the design, and creating K-Map for each component to identify gates that will be used.

The third decision we made is to add test block, the benefit of adding test block is it able to test our codes after finishing the test code and ensure it tests the desired criteria. If components don't behave as designed, then we will repeat the part-by-part block design then test the components again. The test block will save us a lot of time and improve fault tolerance.

3.2.2 Ideation

Each member individually investigated possible applications to use in our design. After brainstorming on our own, we compiled a list of possibilities that we could suggest to our client. The options we considered were

5. RSA encryption/decryption
6. Image reconstruction
7. Fingerprint reader accelerator
8. Digital encoder
9. Spiking neural network

To decide which application to use, we considered the requirements of our project, as well as where our own skills exist as a team. Our design requirements specify that the design must fit on the chip and should have some level of novelty. This eliminates fingerprint reading, RSA, image reconstruction, and digital encoder. With the remaining application of spiking neural network, we decided to select this since it would meet the requirements of the design and is something we believe we can implement.

3.2.3 Decision-Making and Trade-Off

Our group agreed to use spiking neural networks for the following reasons:

- It is a design which we will most likely be able to fit in our user area of the chip

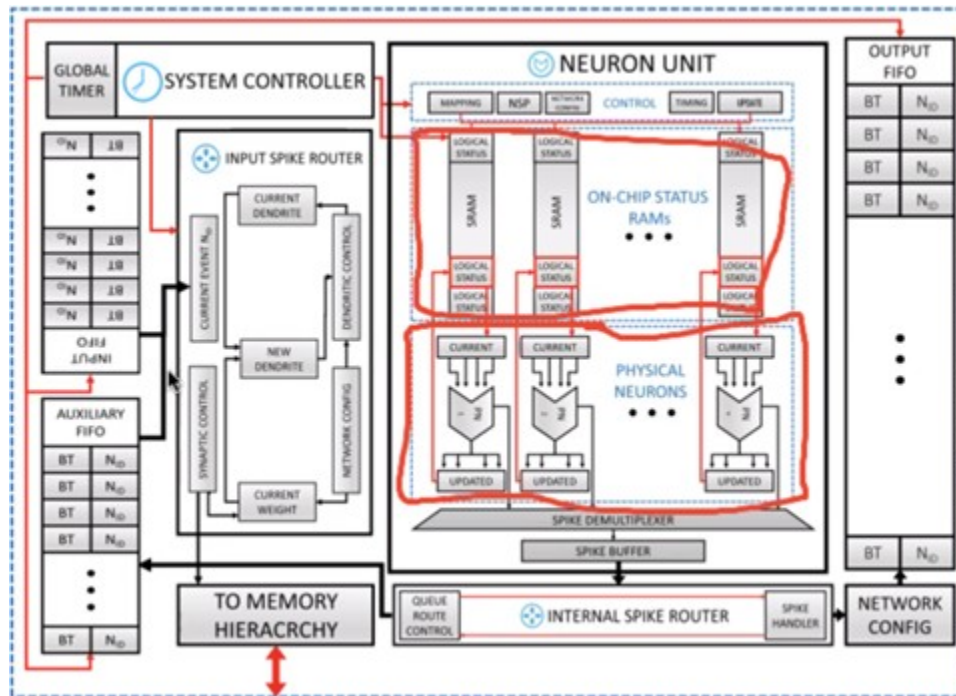
- Our client has an interest in this area of technology
- It has a level of novelty that will help in our selection for the efabless program
- It has elements that would benefit from the acceleration of an application specific integrated circuit (ASIC)

After identifying these reasons, our group selected spiking neural networks as our application for design.

3.3 PROPOSED DESIGN IN 491

3.3.1 Overview

The 3rd generation of neural networks, spiking neural networks, aims to bridge the gap between neuroscience and machine learning, using biologically realistic models of neurons to carry out computation.



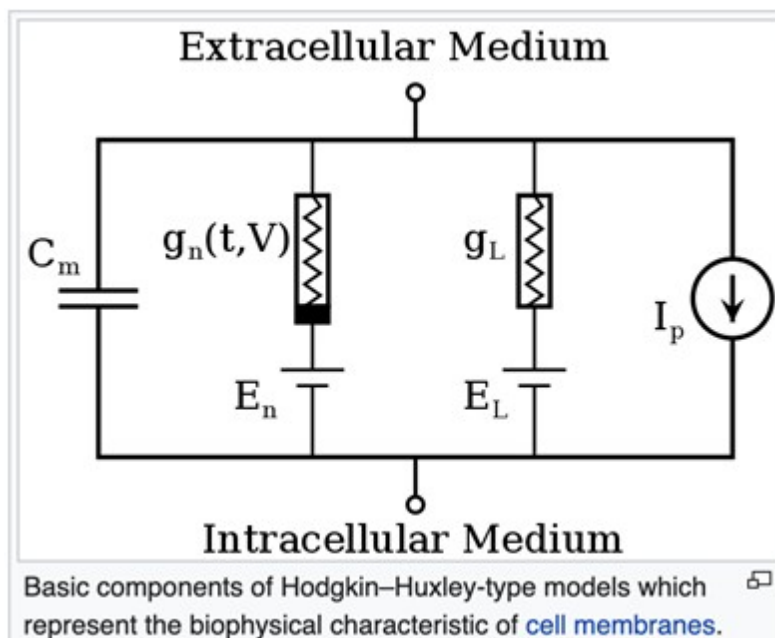
3.3.2 Detailed Design and Visual(s)

SNN tries to more closely mimic a biological neural network. Therefore, instead of working with continuously changing in time values used in ANN, SNN operates with discrete events that occur at certain points of time. SNN receives a series of spikes as input and produces a series of spikes as the output (a series of spikes is usually referred to as spike trains).

The general design descriptions:

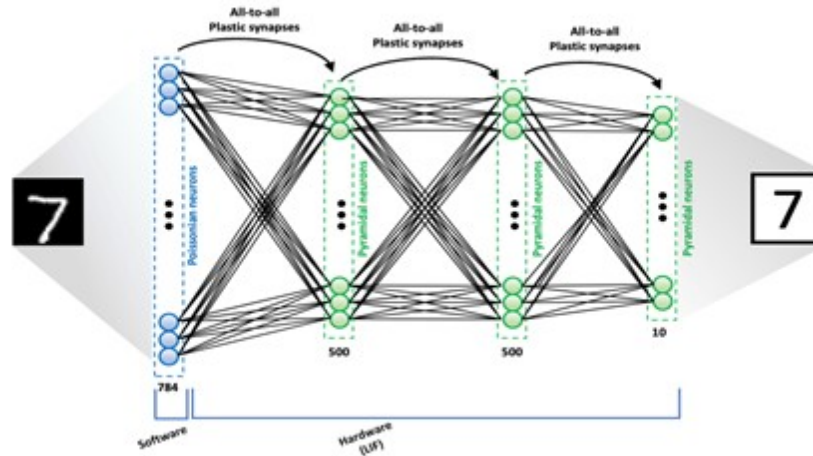
1. At every moment of time each neuron has some value that is analogous to the electrical potential of biological neurons;
2. The value in a neuron can change based on the mathematical model of a neuron, for example, if a neuron receives a spike from the upstream neuron, the value might increase or decrease;
3. If the value in a neuron exceeds some threshold, the neuron will send a single impulse to each downstream neuron connected to the initial one;
4. After this, the value of the neuron will instantly drop below its average. Thus, the neuron will experience the analog of a biological neuron's refractory period. Over time the value of the neuron will smoothly return to its average.

To build SNN, first we need SNN neuron models. SNN neurons are built on the mathematical descriptions of biological neurons. There are a lot of models that can be used. For example, Hodgkin–Huxley model, or conductance-based model, is a mathematical model that describes how action potentials in neurons are initiated and propagated. It is a set of nonlinear differential equations that approximate the electrical characteristics of excitable cells such as neurons and muscle cells. It is a continuous-time dynamical system.



Also, SNN architectures is an important part of the design, there are three types of architectures, and we use Feedforward Neural Network which is a classical NN

architecture that is widely used across all industries. In such an architecture the data is transmitted strictly in one direction – from inputs to outputs, there are no cycles, and processing can take place over many hidden layers.



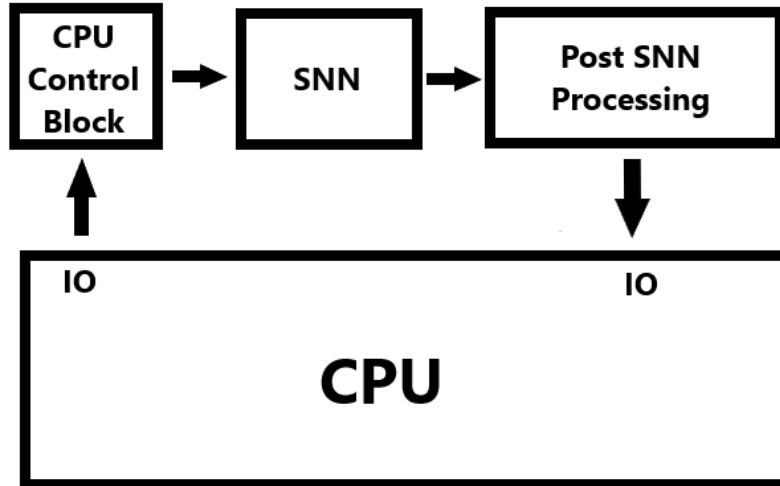
3.3.3 Functionality

Our design will provide customers with complete SNN design plans and drawings, as well as resources about programming. Users can get these resources for free when they need them and can use these resources to improve their own programs. For example, if a university needs to offer a course on SNN, then they can directly use our design to schedule the lab. Our design will provide each stage of SNN design including methods and processes. Our designs can save a lot of time and provide different ideas for those who need them.



3.4 DESIGN ANALYSIS

We are now starting to draw logical circuits and SNN hardware design, we think our design is feasible because the technology used in each step is essential, and as each step is completed, we can fabricate the SNN chip successfully. We are working on the hardware of the SNN as expected part of the construction, our plan is to complete the hardware design first and then solve the programming problem, this is the most reasonable plan for our overall design, because the hardware design is the cornerstone of the SNN chip design, and then all the processes must be worked on this basis.



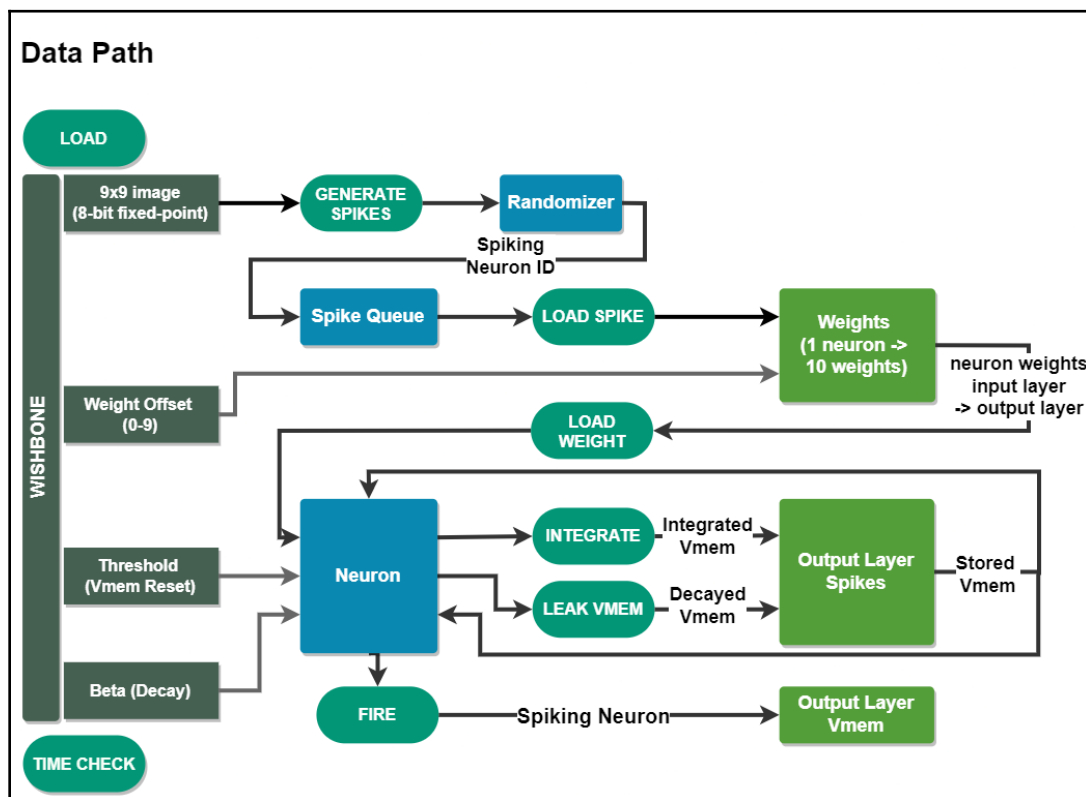
This is our proposed design and flow at a high level. The data from the CPU will be gotten from the SRAM through an I/O. It will arrive first at a control block which will ‘distribute’ it accordingly. We need this block to interface with the built-in features of the Caravel harness that all the MPW design submissions sit on. Additionally it will allow the data to flow more efficiently. Then the neuron/MAC hardware accelerator will be fed the data and complete the calculations according to the model we choose and is in line with the MNIST data set. Last the output of the neuron will be restructured and stored according to a predefined function block.

4. Implementation

Since the end of 491 our design changes have been focused on refinements to the dataflow of the system and the creation of control logic for the system.

4.1 DATA FLOW

Starting with dataflow, we've previously described the operation of each neuron, but we've made refinements to other building blocks that our neuron will need to communicate with. The below figure presents these changes. Our image was reduced to a 9x9 version with 8-bit values so that it could fit within our user area. To prepare the data for neuron processing, we've added other units including a randomizer module and a spike queue. The randomizer uses a pseudo-random algorithm in conjunction with the image values to generate the spikes for the current timestep. Non-zero timesteps and their corresponding identifier (neuron ID) are stored in the spike queue to be processed by the neuron.



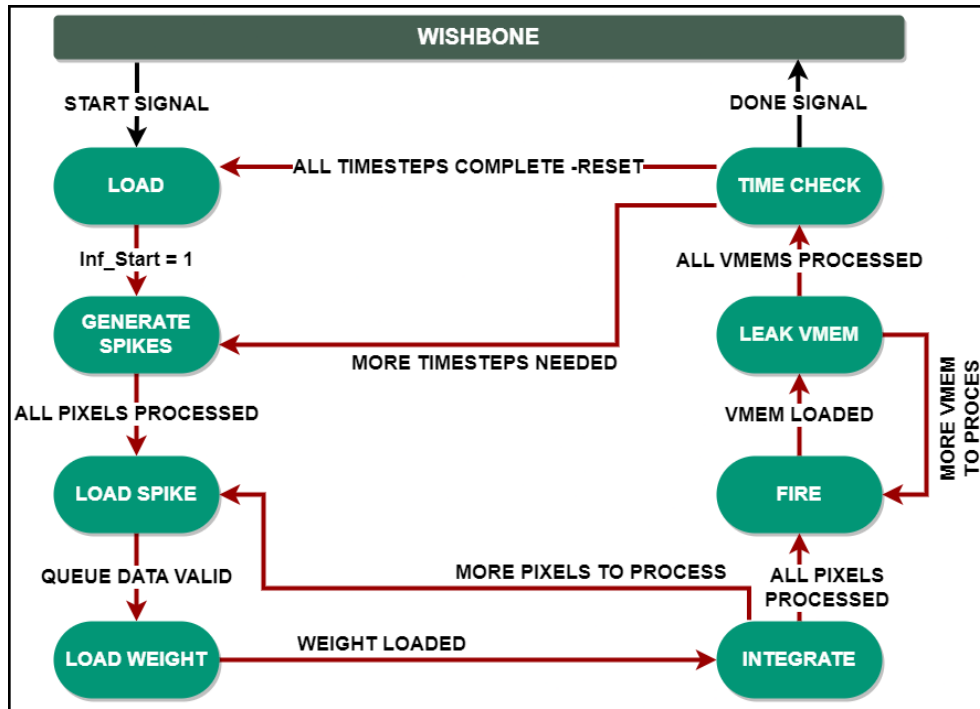
4.2 CONTROL LOGIC

The main design additions made since the end of 491 is our control logic. Without the proper control, our hardware units will not be able to carry out their respective tasks. However, we must also be cautious of deadlocks that can cause our control logic to become “stuck”.

The control logic is implemented using an 8-state FSM with the following states: LOAD, GENERATE SPIKES, LOAD SPIKE, LOAD WEIGHT, INTEGRATE, FIRE, LEAK VMEM, and TIME CHECK. Here is a brief description of state alongside the conditions that result in a state change:

- **State**

- **LOAD:** The initial state of the entire system and default after a reset. This state loads each data structure needed for an inference including the image and weights. We move forward to GENERATE SPIKES upon receiving a start signal from the wishbone (CPU).
- **GENERATE SPIKES:** Generate spikes using the randomizer unit and store them in the spike queue. Move to LOAD SPIKE when every input neuron (pixel) has moved through the randomizer.
- **LOAD SPIKE:** Access the queue to load a spike and continue to LOAD WEIGHT when the queue signals valid output.
- **LOAD WEIGHT:** Load the respective weight and the output neuron’s current vmem which will be processed by the neuron asynchronously using the integrate mode of operation. When the weight is finished loading, we move to the INTEGRATE stage.
- **INTEGRATE:** Take the output vmem from the neuron and store it. If we still have more spikes to process, then we move back to LOAD SPIKE. Otherwise, each spike has been utilized and we can move to the FIRE state.
- **FIRE:** Process an output neuron for whether or not it has fired and move forward to LEAK VMEM. If the neuron spiked, we increment its respective register.
- **LEAK VMEM:** If the neuron did not spike, we apply the leak operation to it. When all output neurons have been processed, move forward to TIME CHECK. Otherwise, we go back to FIRE and process the next neuron.
- **TIME CHECK:** Here we check if we have processed all timesteps. If so, raise the done signal, reset the system, and go back to LOAD. Otherwise, we transition to GENERATE SPIKES to create spikes for the next timestep.



5. Testing

5.1 UNIT TESTING

The main unit that we tested for our design is the Verilog code for our neuron model. We tested it to make sure our design is working the way it is supposed to by constructing a test bench in Linux. It used a combination of C code to set up the virtual hardware that was provided to us as well as Verilog code to simulate the inputs and outputs to our neuron model.

The tools that we used were open-source Linux based software tools, it contained programs such as Icarus Verilog as well as docker set up in its environment. All of that along with a self-made test benches which, as mentioned before, were made with a combination of C based code files and Verilog based code files.

5.2 INTEGRATION TESTING

The integration of our design was tested as follows. There were two major areas for integration testing along the way, integrating the block level components to the top-level design, and integrating the design into the harness.

Block to Top Testing:

To test each block we analyzed their outputs. We used GTKWave for viewing waveforms and Verilator for simulating system Verilog. For each level, block, mid, and top, we wrote

a number of test benches to verify that our block components were working as they should.

Design to Harness Integration Testing

To test our design integration into the provided Caravel harness we ended up using the test benches given to us. These test benches ensured that things such as logic analyzer connections, GPIO connections etc were all correct and working properly. Our design was able to pass each of those tests and operate correctly

5.3 ACCEPTANCE TESTING

The acceptability of our design if for the most part tested by the Efabless provided prechecks. These are the tests done before the submission is allowed onto the shuttle and to verify that all the components of the design are working in tandem. Our design was able to pass the necessary pre-check test verification that has to be passed before submission of the project. Now we are not able to submit our design to Efabless as of now since they have not opened up submission for a new shuttle just yet. But our design passing the given pre-check means that as soon as Efabless does open up submission for the next shuttle our design can be submitted to them right away.

5.6 INTERNAL TESTING

In the case of the documentation that our client asked to create, for example the Caravel Tutorial Document in appendix 2, we were only able to test it by performing internal testing. For example: one of the members in our group had to go through the trouble of deleting WSL and everything on it and re-downloading all of the software tools and setting up the environment once more. He used the tutorial document since he did not remember how to do so and expressed his thoughts on the document so that the creator of those sections in the document could make improvements where the group thought improvements could be made.

5.7 HARDWARE AND SOFTWARE TESTING

Proper hardware testing is essential for ensuring the proper functioning and reliability of a board. The purpose of the hardware testing is to outline some steps that can be taken to test the connectivity, integrity, and functionality of the Caravel Nucleo board's components, as well as to ensure that the power rails have the correct voltage levels. By following these testing plans, any issues with the board can be identified and addressed promptly, leading to better performance and improved user experience.

The purpose of the software test is to provide instructions for running diagnostic software to characterize timing failure patterns between GPIO pads on Caravel for

related shuttles. This diagnostic software is essential to ensure that the Caravel part under the test is functioning correctly.

5.8 SNN CHIP TESTING

this bring-up test plan outlines the crucial steps necessary to validate the functionality and performance of the NMNIST-based Spiking Neural Network (SNN) chip in the Efabless program. The plan includes test objectives to ensure functional correctness, performance, and interface compatibility, as well as a test setup that utilizes simulation tools, firmware, and test scripts. A variety of test cases are included, such as power-on self-test, input/output interface tests, basic functionality tests, integration tests, and performance tests. The plan also outlines the necessary test inputs and expected outputs, including classification results and debug and diagnostic information. The deliverables of the plan include a timeline for executing the test plan, test reports, and debug and optimization logs to document the results of each test case and any necessary troubleshooting steps taken during testing. Overall, this bring-up test plan is a critical step in ensuring the proper functioning and performance of the SNN chip.

5.9 TESTING RESULTS

Our test results prove that our product is usable and does its job correctly. Every part of our design was tested and gave results to prove proper functionality along with meeting our requirements. Our design is from hardware to software, which is very useful and efficient because once we have completed the hardware design and testing, there will be a lot of room for trial and error in the code part.

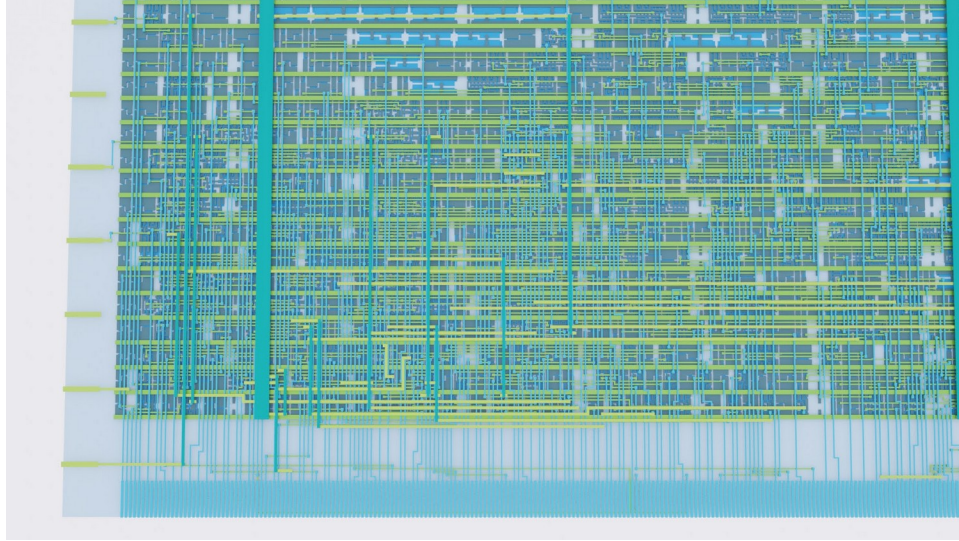
6. Project Outcomes

In designing a digital neuron ASIC, there were a couple major outcomes which will be detailed in the following sections

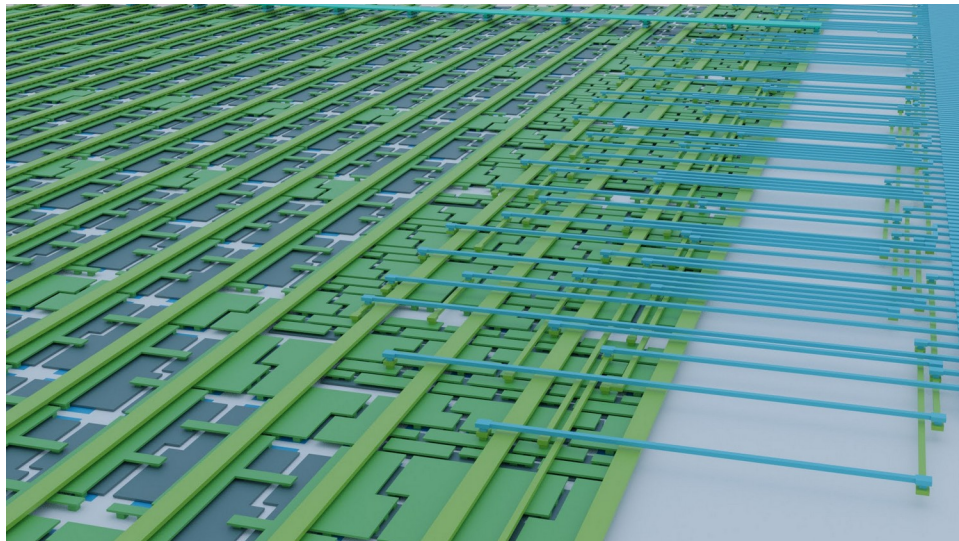
6.1 SUBMISSION READY DESIGN

As mentioned in section 5.3 our design was tested and passed all prechecks. From this the design was deemed to be submission ready for the next open multi project wafer run.

Included are renderings of parts of the design.



Above: Top view of control logic. Below: Single Neuron Unit interacting with the I/Os



6.2 BRING-UP PLAN

A bring up plan was written for use when physical chip is delivered. At the time of writing this document, it is unsure when there will be another shuttle run. However, future team will have the ease of referencing this document to use our wafer to identify MNIST digits. See Appendix 1: User Manual for the full bring-up plan.

6.3 GITLAB REPOSITORY

The code for the project is contained within a public GitLab repository. The project serves the Open-Source community by taking its place in the ecosystem of designs to build from and inspire new ideas. The link to the repository is:

<https://git.ece.iastate.edu/sd/sdmay23-28>

6.4 DOCUMENTATION

Throughout the process of design, each step was documented in detail. It was tested according to section 5.6. The result is 38+ pages of documentation detailing:

- Environment Set-Up
- Getting Started
- Submission
- Daughter & Carrier Board
- Wishbone Bus
- Interrupts
- Logic Analyzer
- GPI
- ETC...

Additionally, various trouble-shooting sections are included based on this team's experiences. The documentation is included in Appendix 2.

7. Closing Material

7.1 DISCUSSION

The main result of our project is the final product we fabricated can satisfy all the requirements and work with general accuracy and efficiency.

Potential Applications for this project include:

- Basis for SNN accelerator unit
- Contribution to the Open-Source Silicon Community
- Inspire and empower students in the field of digital design
- Documentation for more teams and future classes

7.2 CONCLUSION

There is great potential for open-source ASICs in education and industry. This project provides a basis for future projects whether students at ISU or individuals in the Open-source SI ecosystem.

7.3 REFERENCES

1. Saha, Saunak, et al. "An Adaptive Memory Management Strategy towards Energy Efficient Machine Inference in Event-Driven Neuromorphic Accelerators." *2019 IEEE 30th*

International Conference on Application-Specific Systems, Architectures and Processors (ASAP), 2019, <https://doi.org/10.1109/asap.2019.000-2>.

2. "Digital Simulation of Spiking Neural Networks." *Pulsed Neural Networks*, 1998, <https://doi.org/10.7551/mitpress/5704.003.0014>.
3. "IEEE Standard VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification," in *IEEE Std 1076.4-1995* , vol., no., pp.1-96, 17 May 1996, doi: 10.1109/IEEESTD.1996.80811.
4. "IEEE Standard Testability Method for Embedded Core-based Integrated Circuits," in IEEE Std 1500-2022 (Revision of IEEE Std 1500-2005) , vol., no., pp.1-168, 12 Oct. 2022, doi: 10.1109/IEEESTD.2022.9916221.
5. "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems," in IEEE Std 1801-2018 , vol., no., pp.1-548, 29 March 2019, doi: 10.1109/IEEESTD.2019.8686430.
6. Efabless. (n.d.). *Efabless/caravel_board*. GitHub. Retrieved April 17, 2023, from https://github.com/efabless/caravel_board

Appendix

1. User Manual – Bring Up Plan

EE 492 Project Test Plan

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.

Components:

NUCLEO-F746ZG or NUCLEO-F413ZH

Caravel Nucleo Hat

One or more Caravel breakout boards with a Caravel part installed

Two jumpers for J8 & J9

USB micro-B to USB-A cable

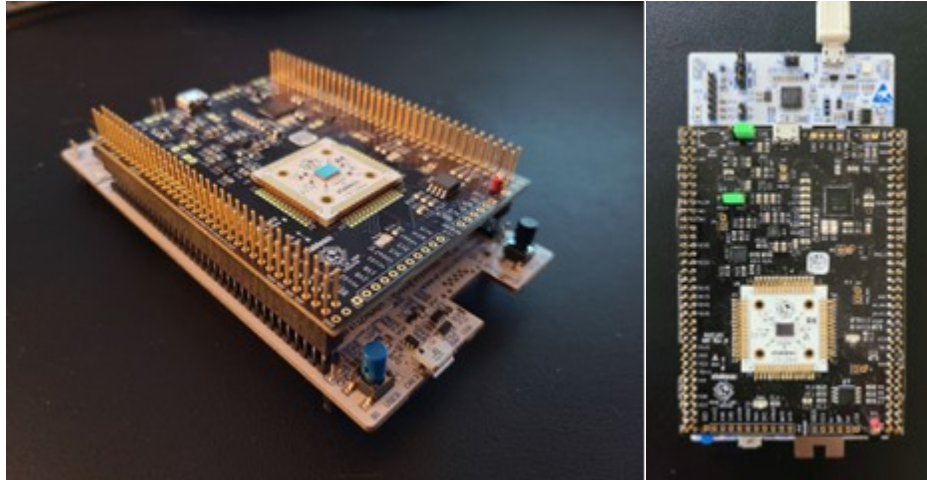


Figure 1: Efabless Caravel Board^[1]

Description:

The bottom white board is Caravel Nucleo Board: F746ZG, which provide an affordable and flexible way for users to try out new concepts and build prototypes.

The black board on middle is a custom design from efabless, referred as the "Caravel Nucleo Hat", which provides a diagnostic software for characterizing timing failure patterns between GPIO pads on Caravel for the related shuttles.

The smaller yellow breakout on top (which should be white for most people) is referred as the "Caravel breakout". The fuction of this board is hold our SNN chip for testing.

Prepare:

The purpose of this step is to connect each components together amnd make the boards ready to test. There are some details need to notice during the assembling.

1. Install the jumpers on J8 and J9 in the 'HAT' position to enable the board to be powered by the Nucleo.
2. Plug the Caravel Nucleo Hat in Nucleo board pins
 - The USB on the hat should face the ST-LINK breakoff board on Nucleo and away from the push buttons on Nucleo
 - IMPORTANT: the FlexyPin socket allows you to swap breakout boards with different parts. You do not need to solder any pins.
 - Be careful not to bend a pin when inserting the breakout board. If one of the pins bends, use needle-nose pliers to re-straighten it.
 - When pressing the Caravel Hat board on the pin headers of the Nucleo, only press far enough to engage the pins. If you press to far, you can short the Flexy pins under the board against jumpers on the Nucleo.
3. Install a Caravel Breakout board into the socket on the Caravel Hat board
 - The Efabless logo should face the USB connector on the Hat
4. Connect the USB cable from the connector CN1 on the Nucleo to your workstation / laptop.
5. Connect a second USB cable from your desktop / workstation from connector CN13 on the opposite side of the Nucleo board from the ST-LINK breakaway board.
 - This port presents a mountable volume for the Flash filesystem on Nucleo and is how the software and firmware files are copied on to Nucleo. It is also used to retrieve the gpio_config_def.py file after the diagnostic completes.

Installation:

The purpose of this step is to complete the test background setting.

1. Install the required tools including mpremove, mpy-cross and rshell. The diagnostic runs on a customized Micropython image on the Nucleo board. The Nucleo firmware image, diagnostic software and Makefile targets for installing and running the routines are located in the firmware_vex/nucleo directory in the caravel_board repo.
 - mpremove is used for connecting the Micropython
 - mpy-cross is a cross compiler for Micropython that compiles a python file into a binary format which can be run in micropython. It is used here to reduce the size of the files because the size of the flash on the Nucleo board is limited on some models.
2. You will also need to install the stlink tools for your client. These are required to flash Micropython firmware on the Nucleo board.
3. After you made both USB connections, you will need to find the path for the Flash volume.
 - On MacOS, it should be located at //Volumes/PYBFLASH.
 - On Ubuntu, it should be mounted at /media/<userid>/PYBFLASH.
 - You will need to export FLASH=<path> or set the path in the Makefile at the top of the file.
4. It will be required to update the diagnostic software to get the latest enhancements and bug fixes. You can find the model of the Nucleo board on a label in the lower left corner of the Nucleo board opposite the ST-LINK breakaway board (F746ZG or F746ZH). Run one of the following make targets based on the model of your Nucleo board.

5. You can also just recompile and copy the files onto the flash by running on of the following make targets. After the flash completes, check the version of the software.

Test Process:

I. Hardware Test:

Input: Check that all pins are connected correctly according to the board schematic. Test each pin individually for connectivity and short circuits

Expected Output: All pins should be connected correctly with no short circuits or open connections.

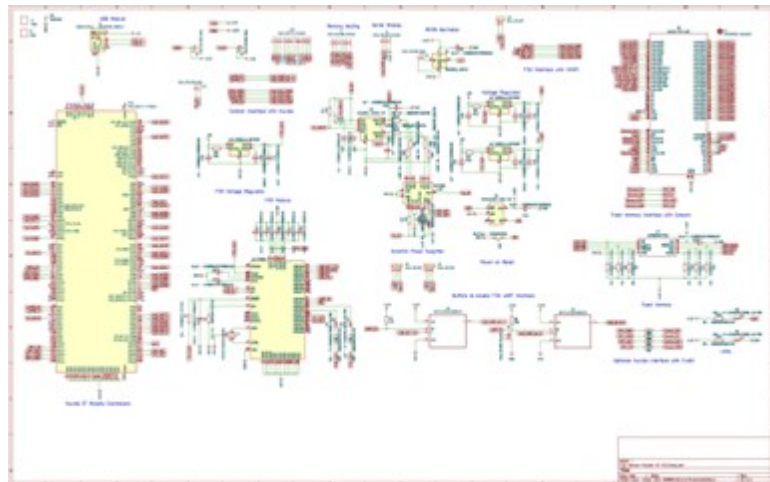


Figure 2: The Schematic of Caravel Nucleo Board^[1]

Notice:

- The clock is driven by X1 with a frequency of 10MHz. To drive the clock with custom frequency, disable X1 through 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
- To check if all pins are connected correctly, the following test plan can be developed:
 1. Check the schematic of the board to identify the expected connectivity of the pins.
 2. Using a multimeter, verify the connectivity of each pin against the expected schematic connectivity.
 3. Verify the connectivity of all power and ground pins.
 4. Check the resistance of the pins to identify any potential short circuits or open connections.
 5. Verify the signal integrity of the pins by sending test signals and measuring the output.
- To test that all parts of the component on the board are working well, the following test plan can be developed:
 1. Check the datasheet of each component to identify the key parameters that need to be tested.
 2. Develop test cases to verify the correct functioning of each component, based on their respective specifications.

3. Use appropriate test equipment to measure and record the values of the parameters for each component.
 4. Compare the measured values with the expected values from the datasheet to identify any discrepancies.
 5. Define the acceptable tolerance range for each parameter and establish the pass/fail criteria accordingly.
 6. Conduct additional tests, if necessary, to validate the correct functioning of each component.
- To test that all power rails have the correct voltage levels within the specified tolerance range, the following test plan can be developed:
 1. Identify all the power rails on the board.
 2. Using a multimeter, measure the voltage levels of each power rail.
 3. Compare the measured voltage levels with the expected voltage levels as specified in the datasheet.
 4. Define the acceptable tolerance range for the voltage levels based on the specifications.
 5. Establish the pass/fail criteria based on the measured voltage levels and the acceptable tolerance range.
 6. Conduct additional tests, if necessary, to verify the correct functioning of the power rails.

II. **Software Test (The reseou**

- To run the diagnostic, enter the following commands. The command depends on whether your project uses Caravel or Caravan (for analog). The PART variable is an ID for the part you are testing defined by you. It will be recorded in the output of the test for future reference.

```
cd caravel_board/firmware_vex/nucleo

## for digital (or analog) projects using Caravel & user_project_wrapper
make run PART=<part id>

## for analog projects using Caravan & user_analog_project_wrapper
make run_analog PART=<part id>
```

Figure 3: Test Code for Caravel

The test will begin with the green LED on the Nucleo flashing 5 times. When the test concludes, the green and red leds will be as follows:

GREEN	RED	STATUS
2 short + 4 long	off	Full Success - BOTH IO chains configured successfully
2 long	2 short	Partial Success - LOW IO chains configured successfully
4 long	2 short	Partial Success - HIGH IO chains configured successfully
off	2 short + 4 long	Failed - BOTH IO chains failed to configured fully
off	solid	Test failed to complete

Table 1: Test Result Chart

Type Ctrl-C to exit the test diagnostic once it completes. If the test is completed for the part, run the `make get_config` to retrieve the configuration file. The file will indicate the IO that was successfully configured. Successfully configured IO can be used for this part for firmware routines.

- The following will run a sanity check test using the `gpio_config_def.py` produced from the diagnostic above. The `gpio_config_def.py` file is stored from the `make get_config` run above and local on your desktop.

To run the sanity check:

```
cd caravel_board/firmware_vex/nucleo
make sanity_check FILE=gpio_config_def.py
```

Figure 4: Sanity Check Test

- Before using IO, you need to call `configure_io()`.
- You should leave the Caravel Nucleo board mounted and powered by Nucleo. The timing failure pattern specified in the `gpio_config_def.py` is sensitive to the capacitance loading on the pins that is present when the Caravel board is mounted on the Nucleo and may not be the same when the board is detached.
- You can flash and run your firmware with the Caravel Hat mounted on the Nucleo by running: `make clean flash_nucleo`.
- **GPIO TEST FILE**

gpio_config_io.py:

https://github.com/efabless/caravel_board/blob/main/firmware_vex/gpio_test/

[gpio_config_io.py](#)

gpio_test.c:

https://github.com/efabless/caravel_board/blob/main/firmware_vex/gpio_test/

[gpio_test.c](#)

io_config.c:

https://github.com/efabless/caravel_board/blob/main/firmware_vex/io_config/

[io_config.c](#)

III. SNN Chip Test

This is a bring-up test plan for our NMNIST-based Spiking Neural Network (SNN) chip in the Efabless program which is crucial to validate its functionality and performance.

1. Test objectives:

- Functional correctness: Ensure that the SNN chip correctly implements the NMNIST-based neural network and can process input data as intended.
- Performance: Validate that the chip meets the target performance metrics (e.g., classification accuracy, latency, power consumption) for the NMNIST dataset.
- Interface compatibility: Confirm that the chip interfaces correctly with external components, such as memory and peripherals.

2. Test setup:

- Simulation tools:

- Icarus Verilog: Used to simulate RTL (Register Transfer Level) designs
- GTKwave: Used to view the waveform output generated by simulation
- Firmware:
 - Caravel harness repo: Provides makefiles that handle the compilation of CPU software to be run on a simulated CPU along with the rest of the user project wrapper.
- Test scripts:
 - Python: A popular scripting language used for automation and testing.
 - JMeter: An open-source load testing tool used for testing the performance and scalability of web applications.
 - Postman: A tool used for testing and debugging APIs.

3. **Test cases:**

- Power-on self-test (POST): Verify that the chip initializes correctly upon power-up and performs any necessary self-tests.
- Input/output (I/O) interface tests: Validate the proper functioning of all I/O interfaces, including communication protocols and data transfers with external components.
- Basic functionality tests: Test individual components and sub-blocks of the SNN chip to ensure that they perform as intended (e.g., neuron models, synapses, learning rules).
- Integration tests: Confirm that the SNN chip components work together correctly as a complete system when processing MNIST data.
- Performance tests: Evaluate the chip's performance against predefined performance metrics (e.g., classification accuracy, latency, power consumption).

4. **Test inputs:**

- Preprocessed MNIST dataset: Use the MNIST dataset after applying the necessary preprocessing steps, such as converting images to spike trains or other suitable formats for SNN processing.

5. **Expected outputs:**

- Classification results: The chip should produce correct classification outputs for the input MNIST dataset with an accuracy comparable to that of the target performance metrics.
- Debug and diagnostic information: Collect relevant debug and diagnostic data during testing to aid in troubleshooting and optimizing the chip's performance.

6. Test deliverables:

- Develop a timeline for executing the test plan, including milestones for completing individual test cases and achieving overall objectives
- Test reports: Document the results of each test case, including pass/fail criteria, observed outputs, and any issues encountered.
- Debug and optimization logs: Record any necessary debugging and optimization steps taken during testing, along with their outcomes.

Simulation code:

```
/*  
* SPDX-FileCopyrightText: 2020 Efabless Corporation  
*  
* Licensed under the Apache License, Version 2.0 (the "License");  
* you may not use this file except in compliance with the License.  
* You may obtain a copy of the License at  
*  
* http://www.apache.org/licenses/LICENSE-2.0  
*  
* Unless required by applicable law or agreed to in writing, software  
* distributed under the License is distributed on an "AS IS" BASIS,  
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
* See the License for the specific language governing permissions and  
* limitations under the License.  
* SPDX-License-Identifier: Apache-2.0  
*/  
  
// This include is relative to $CARAVEL_PATH (see Makefile)  
#include <defs.h>
```

```
#include <stub.c>
```

```
// -----
```

```
/*
```

```
    MPRJ Logic Analyzer Test:
```

- Observes counter value through LA probes [31:0]
- Sets counter initial value through LA probes [63:32]
- Flags when counter value exceeds 500 through the management SoC gpio
- Outputs message to the UART when the test concludes successfully

```
*/
```

```
void main()
```

```
{
```

```
    int j;
```

```
    /* Set up the housekeeping SPI to be connected internally so      */
```

```
    /* that external pin changes don't affect it.                    */
```

```
    // reg_spi_enable = 1;
```

```
    // reg_spimaster_cs = 0x00000;
```

```
    // reg_spimaster_control = 0x0801;
```

```
    // reg_spimaster_control = 0xa002; // Enable, prescaler = 2,
```

```
        // connect to housekeeping SPI
```

```
    // Connect the housekeeping SPI to the SPI master
```

```
    // so that the CSB line is not left floating. This allows
```

// all of the GPIO pins to be used for user functions.

// The upper GPIO pins are configured to be output

// and accessible to the management SoC.

// Used to flad the start/end of a test

// The lower GPIO pins are configured to be output

// and accessible to the user project. They show

// the project count value, although this test is

// designed to read the project count through the

// logic analyzer probes.

// I/O 6 is configured for the UART Tx line

reg_mprj_io_31 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_30 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_29 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_28 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_27 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_26 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_25 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_24 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_23 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_22 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_21 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_20 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_19 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_18 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_17 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_16 = GPIO_MODE_MGMT_STD_OUTPUT;

```
reg_mprj_io_15 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_14 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_13 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_12 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_11 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_10 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_9 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_8 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_7 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_5 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_4 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_3 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_2 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_1 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_0 = GPIO_MODE_USER_STD_OUTPUT;
```

```
reg_mprj_io_6 = GPIO_MODE_MGMT_STD_OUTPUT;
```

```
// Set UART clock to 64 kbaud (enable before I/O configuration)
```

```
// reg_uart_clkdiv = 625;
```

```
reg_uart_enable = 1;
```

```
// Now, apply the configuration
```

```
reg_mprj_xfer = 1;
```

```
while (reg_mprj_xfer == 1);
```

```
// Configure LA probes [31:0], [127:64] as inputs to the cpu
```

```
// Configure LA probes [63:32] as outputs from the cpu
```

```
reg_la0_oenb = reg_la0_iena = 0x00000000; // [31:0]
```

```

reg_la1_oenb = reg_la1_iena = 0xFFFFFFFF; // [63:32]
reg_la2_oenb = reg_la2_iena = 0x00000000; // [95:64]
reg_la3_oenb = reg_la3_iena = 0x00000000; // [127:96]

// Flag start of the test
reg_mprj_datal = 0xAB400000;

// Set Counter value to zero through LA probes [63:32]
reg_la1_data = 0x00000000;

// Configure LA probes from [63:32] as inputs to disable counter write
reg_la1_oenb = reg_la1_iena = 0x00000000;

while (1) {
    if (reg_la0_data_in > 0x1F4) {
        reg_mprj_datal = 0xAB410000;
        break;
    }
}

print("\n");
print("Monitor: Test 1 Passed\n\n"); // Makes simulation very long!
reg_mprj_datal = 0xAB510000;
}

```

IV. Reference

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

2. Caravel Tutorial Document

1. Overview of the Efabless Process Steps

The Caravel project is a long project that will take time and dedication to complete. In this section an overview of the step-by-step process for the caravel project will be explained from start to finish.

- 1) The very first thing that will need to be done is to familiarize yourself with the Caravel harness.
 - a) You will want to study all of the components that make up the Caravel harness and their functionalities. This can be done by reading through the following website that contains information about the user harness.
 - b) Link: [Efabless Caravel "harness" SoC — Caravel Harness documentation \(caravel-harness.readthedocs.io\)](https://caravel-harness.readthedocs.io)

- 2) You will have to download all necessary efabless tools and familiarize yourself with how they work.
 - a) how to do so is explained in the sections below.

- 3) You will need an open-source Git repository.
 - a) This is to store the caravel project as well as give detailed documentation on what your project does and how it works in the form of a README.
 - b) If you are part of a senior design team the ETG will set up your repository for you, but it will be up to you to update your repository as necessary.

- 4) Next you will need to come up with a chip design idea that you believe will be able to fit within the user area of the Caravel harness slowly making a high level RTL design of your project Idea to implement into the user area.

- 5) After coming up with an idea and creating a viable RTL for it you will need to slowly add all of the components/modules that comprise your design idea into the user area
 - a) Be sure to set up test benches testing each module individually to check for functionality before implementing them into the caravel harness.
 - b) Then slowly integrate each module into the caravel harness setting up test benches to check for functionality between each of the modules you have as you implement them all into the user area.

- 6) After successfully integrating your design into the user area next you will need to harden your design inside of your linux environment.
 - a) Explanation on how to do so is in section 4.3.

- 7) Once you have successfully hardened your design you will need to instantiate it in the project wrapper.

- 8) After you have instantiated your design inside of the wrapper you will harden the project wrapper.
 - a) Explanation on how to do so is in section 4.4.

- 9) After hardening and creating the new project wrapper you will test its functionality using the test benches given in the caravel repository as well as any personal test benches you design for your project to ensure that it is working properly.
 - a) Explanation on how to do so is in section 4.5.

- 10) Once your project has passed all test benches the last thing to do is to run the final checks on it. Your project must pass these prechecks before you attempt to submit your design to Efabless
 - a) Explanation on how to do so is in section 4.6.

- 11) After your project has passed the final prechecks you will submit your project to efabless. Keep in mind that it may take several hours for your design to be fully

submitted to eFabless due to the final tests that will need to be performed on your project submission on eFabless submission page.

- a) Explanation on how to submit your project and what needs to be done is in section 8.

12) The next thing that needs to be done is to prepare a bring-up plan for your project. You will need to come up with steps to test the physical PCB board & all components on it, as well as testing your overall project's functionality. Essentially coming up with tests that have hardware and software aspects to it to ensure that everything is functioning properly.

- a) To understand what to expect to be on the PCB boards that you will receive you may review all of section 9 of this document.

13) Eventually you will get your project design back in the form of a die soldered to a PCB board. You will obtain a carrier board as well as a daughter board. The only thing left to do is to execute your bring-up plan to see how well your project turned out.

2. Windows Subsystem for Linux Installation (WSL2) on Windows 11

Source: [Install WSL | Microsoft Learn](#)

2.1 OPENING POWER SHELL

First open PowerShell on your windows computer, you can do so by clicking in the search bar at the bottom of your windows interface circled in green as seen in figure 1.



Figure 1. Windows search bar

A search window, as displayed in figure 2 should appear.

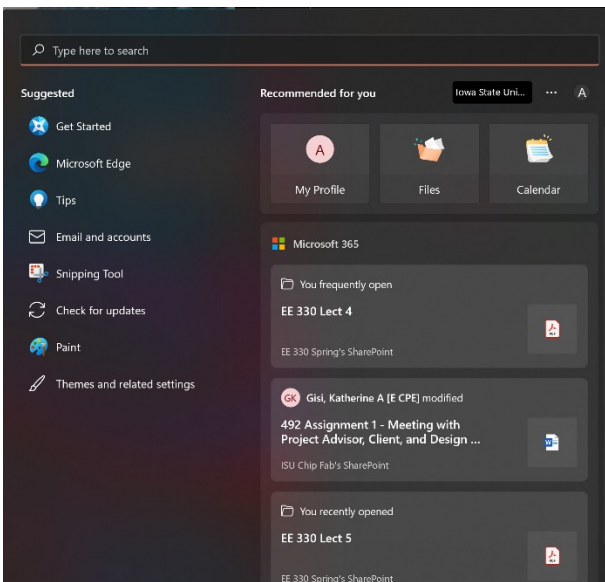


Figure 2. Windows search window

Afterwards type in PowerShell into your search window and open it as seen in figure 3.

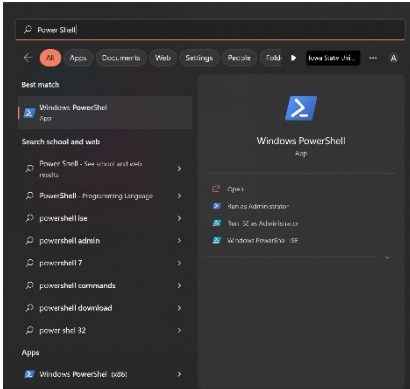


Figure 3. PowerShell App

2.2 INSTALLING WSL2

Once PowerShell is up and running type in the following command into the command line and run it refer to figure 4.

Command: `wsl --install`

This will enable all the necessary features in order to run Linux on your machine. It will also install the Ubuntu distribution of Linux on your machine. You will be prompted to create a username and password for Ubuntu, it is very important that you remember your password as you will need it every time you open Ubuntu and attempt to run the first command.



Figure 4. Installation of wsl through PowerShell

3. Setting Up Dependencies for the Caravel Repository

3.1 UPDATING UBUNTU TO LATEST VERSION

The first thing that will need to be done is to make sure that your system is running on the latest version of Ubuntu. To do so you will have to open up Ubuntu by typing it in the same search bar the same way that you opened PowerShell. After doing so type the following command into the command line and run it. Before Ubuntu executes the command, you may be prompted to type in your password that you created in the previous step. Refer to figure 5.

Command: `sudo apt update`



Figure 5. Ubuntu

As can be seen in figure 5 you will be prompted to type in your password before Ubuntu executes the above-mentioned command. It is important to note that as you type in your password Ubuntu will not display any of the characters that you type in, this is for security reasons. Understand that Ubuntu is receiving the characters that you are typing there is nothing wrong with your terminal.

After typing in the correct password Ubuntu will execute the command that you entered prior to it prompting you with a request for your password as can be seen in figure 6.



Figure 6. Execution of first command

After Ubuntu has executed the “`sudo apt update`” command in the second to last line in figure 6 it will tell you the number of packages you are able to upgrade as well as how to view the list of upgradable packages if needed.

Type the following command into the command line and run it. This command will upgrade all of the packages listed from the prior command. You will be promoted with a question asking if you wish to continue where you can enter "Y" for yes and "n" for no, refer to figure 7.

Command: `sudo apt upgrade`



Figure 7. Ubuntu executing `sudo apt upgrade`

Enter in 'Y' and Ubuntu will upgrade your packages. Depending on the number of packages that need to be upgraded it could take Ubuntu anywhere from a few seconds to a few minutes to finish upgrading the packages. Once Ubuntu has finished it will display "done" in the second to last line, indicating that the upgrading process was completed and successful.

3.2 INSTALLING DOCKER IN UBUNTU

Source: [Install Docker Engine on Ubuntu | Docker Documentation](#)
[Docker for WSL](#)

First you must be in your home directory, to do so enter the following command into the command line and run it. This will return you to your home directory.

Command: `cd`

Afterwards you will have to update the apt package index, enter in the following command in the command line and run it. The output should look like figure 8 below.

Command: `sudo apt-get update`



Figure 8. Output of `sudo apt-get update` command

If Ubuntu is unable to run the previous command then your default unmask could be incorrectly configured. If that is the case, then type in the following commands in the command line and run each one separately.

Command: `sudo chmod a+r /etc/spt/keyrings/docker .gpg`

Command: `sudo apt-get update`

Next you will have to install the Docker Engine, container, and Docker Compose. Type in the following command in the command line and run it. The following command will install the latest version of docker on your machine.

Command: `sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin`

To verify that docker installed correctly type in the following command line and run it. The output should look like figure 8 down below, if it does then you have successfully downloaded docker into your Linux environment.

Command: `sudo docker run hello-world`



Figure 9. Correct Docker Installation Output

3.3 TROUBLESHOOTING

Based on past experience, WSL may or may not work easily with docker. The recommended way to use WSL and docker is to install Docker Desktop for Windows, then enable a setting to connect to WSL; however, there have also been issues where Docker Desktop does not start correctly on Windows either. To avoid using Docker Desktop, you can try following the normal install steps for a traditional Ubuntu installation. This has its own issues, specifically with GPG key errors and the daemon not starting correctly by default.

GPG key error:

`sudo apt-get update` may fail with an error about gpg keys. To fix this, you can run the following from the directory `"/etc/apt/keyrings"`:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -  
o /etc/apt/keyrings/docker.gpg
```

Then try `sudo apt-get update` again`

Docker daemon not running:

The docker daemon may not start after installing docker, so `sudo docker run hello-world` give an issue with not being able to communicate with the docker daemon. To solve this, try running sudo dockerd &`. This runs the command in the background, but the output will still be sent to the terminal. You should still be able to run sudo docker run hello-world`. If that does not work, then enter in the command below:`

Command: `sudo service docker start`

Your daemon error should be fixed by now.

3.4 INSTALLING PYTHON 3 WITH PIP

Source: [How to Install Python Pip on Ubuntu 20.04 | Linuxize](#)

Installing Python 3 with Pip is the final dependency needed in order to use the caravel (repository). Type the following command into the command line. This will install the latest version of Python 3 along with PIP at the same time.

Command: `Sudo apt install python3-pip`

After running the previous command type the following command in the command line to verify that the installation executed correctly.

Command: `pip3 --version`

The output should look like figure 9 down below, the versions number may vary.



Figure 10. Pip and Python version check output

3.5 INSTALLING KLAYOUT (OPTIONAL)

All though this software tool is not required in order to be able to use the caravel environment it is a nice visual aid that will help with understanding how much space your design occupies in the user project area. Type and run the following commands:

1. `sudo apt update`
2. `sudo apt upgrade`
3. `sudo apt -y install klayout`

It should only take your terminal a few minutes at most to download the klayout software tool. Again it is not necessary but can prove to be extremely useful when it comes to understanding how much of the user area space is taken up by your design.

4. Getting Started with the Caravel Repo

Source: [caravel_user_project/quickstart.rst at main · efabless/caravel_user_project · GitHub](#)

Before you are able to do anything with the Caravel repository you must first set up the local environment inside of Linux inside of your project repository.

4.1 CLONING YOUR PROJECT REPOSITORY INTO LINUX

First you must clone your project repository given to you by the ETG into Linux. Type the following command into your command line and run it.

Command: `git clone <your repo URL>`

Upon running that command, you may be prompted to give your username and password in order to clone your repo. The username should be your school email and the password should be your school password. After entering that Linux should clone your repository without issue.

4.2 SETTING UP LOCAL CARAVEL ENVIRONMENT

The first thing that you will need to do inside of Linux is to enter into your project repository. Type the following command into your command line and run it to enter your project repository.

Command: `cd ~/<your project name>`

Next you will need to run the following commands one at a time.

Commands:

1. `mkdir dependencies` (you only need to do this once)
2. `export OPENLANE_ROOT=$(pwd)/dependencies/openlane_src`
3. `export PDK_ROOT=$(pwd)/dependencies/pdks`
4. Make setup

If you are running into an error with the communication with daemon from docker please refer to the trouble shooting section under installing docker in Ubuntu.

Commands 2 & 3 will have to be exported every time that you start a new shell (every time you reopen your terminal). After the fourth command is executed, the following tools will be installed:

- Caravel_lite
- Management core for simulation
- Openlane to harden your design
- Pdk

These installations should only take a few minutes to complete.

4.3 BEFORE HARDENING USER PROJECT

Source: [MPW-6 Walkthrough - YouTube](#) (17:20 & 21:36)

Before you go to harden your project there are a few files that have to be modified so that the make command knows what to harden. The first file that has to be changed is in the user project example directory and is called the config.tcl file. You may find and change the config file under `caravel_user_project/openlane/user_proj_example/config.tcl` or you may make your own directory inside of openlane and create your own config file. Below I will show you how to modify the config file that is already present in the directory that I described above.

1. In the config.tcl file you will change the "DESIGN_NAME" section to the name of your project, so change line 21 by replacing `user_proj_example` with your project name.
2. Next you will have to change line 25, changing the `user_proj_example.v` file to your projects .v file or files depending on how many different Verilog scripts your project has.
3. If you have an issue with hardening your project you may have to play around with lines 34 and 39. For line 34 only change the right two numbers for the die area. So only change the 900 and or the 600. And for line 39 you must choose a number that is no greater than 1. But to keep the number as low as possible is desirable as you may receive a number of errors or violations for trying to make the placement target density too high.



Figure 11. config.tcl file for user project example

4.4 HARDENING USER PROJECT TO TEST OPEN LANE SETUP: EXAMPLE - USER_ADDER

Testing your openlane set up is essential due to the fact that you must be able to harden your design before you are able to do anything else regarding too project submission to Efabless. Type and run the following command:

Command: `make user_adder`

Your terminal will then run through 48 steps, by the end of the process the output of your terminal should look the same as figure 12 below.



Figure 12. Output of hardening of user adder successfully

As can be seen in figure 12 there are two warnings given, however you can ignore these warnings for now because they will not hinder you any time soon.

4.5 BEFORE HARDENING THE USER PROJECT WRAPPER

Before you go on to harden your project inside of the project wrapper you once again have to change the contents of the corresponding config file or create your own. For this example, I will describe how to change the config file that is already present in the caravel user project. The location of the config.tcl file is in `caravel_user_project/openlane/user_proj_wrapper/config.tcl`.

1. You will have to edit line 42 changing the mprj clock to whatever the name is of the clock that you made for your project.
2. Change lines 57,60, & 63 to your projects corresponding file types.



Figure 13. Config file fore user project wrapper

4.6 CREATING USER PROJECT WRAPPER

Before you are able to run the final checks you must first take your harden design and place it inside of the caravel project wrapper. After doing so type and run the following command to create the user project wrapper inside of your environment.

Command: `make user_project_wrapper`



Figure 14. Output of making the user wrapper

4.7 RUNNING TEST BENCH SIMULATIONS ON YOUR DESIGN

After hardening your design inside of the user project wrapper there are only a few more things that will need to be done before your design is ready for submission to Efabless. First you will be instructed on how to run personal test benches for the user adder project so that you can familiarize yourself with how the output of the tests should look like inside of the terminal. However, the example will only use personal test benches because the user adder project will fail all other test benches that your project must pass before submission to Efabless. The other test benches that your project will need to pass are for the io ports, mprj, the logic analyzer, and wishbone bus which will be addressed after the example with the user adder project. Your project will need to be tested at both rtl and gl and passed at both levels before your design is ready for submission.

4.8 Example

There are two personal test benches to run on the user adder, the commands for each of those tests are as follows:

Command: `make verify-user_adder_test1-<rtl or gl>`

The output for the previous command should look similar to figure 15 down below



Figure 15. Output of `make verify-user_adder_test1`

Command: `make verify-user_adder2-<rtl or gl>`

The output for the previous command should look similar to figure 16 down below



Figure 16. Output of `make verify-user_adder_test2`

There are a number of other tests that your design will have to pass before submission. However, you will not be able to use the user adder project to run the tests down below. The tests will time out if you try to run them using the user adder project and fail the test.

To run each test individually you will have to enter in one of the following commands:

Command: `make verify-io_ports-<rtl or gl>`

Command: `make verify-mprj_stimulus-<rtl or gl>`

Command: `make verify-la_test1-<rtl or gl>`

Command: `make verify-la_test2-<rtl or gl>`

Command: `make verify-wb_port -<rtl or gl>`

4.9 RUNNING FINAL CHECKS ON YOUR PROJECT BEFORE SUBMISSION

The final thing that you will have to do is to pass all of the prechecks set out by Caravel. There are 13 in total that your design will have to pass before your project is ready for submission to caravel:

1. License check: caravel will check your directory and make sure that there are no prohibited license files within it.

2. Make file: caravel will check your make file in your directory and ensure that it is valid.

3. Default: caravel will check to see if you have edited the 'README.md' file to ensure that you do not maintain the default 'README.md' file that comes with the caravel repository.

4. Documentation: caravel will check to see if there is appropriate documentation in your directory.

5. Consistency: Caravel will check the following.

A) Hierarchy - Module user_project_wrapper is instantiated in caravel.

B) Complexity - Netlist caravel contains at least 8 instances.

C) Modeling - Netlist caravel is structural.

D) Submodule Hooks - All module ports for user_project_wrapper are correctly connected in the top-level netlist caravel.

E) Power Connections - All instances in caravel are connected to power.

F) Ports - Netlist user_project_wrapper ports match the golden wrapper ports.

G) Complexity - Netlist user_project_wrapper contains at least 1 instance.

H) Modeling - Netlist user_project_wrapper is structural.

I) Layout - The GDS layout for user_project_wrapper matches the provided structural netlist.

J) Power Connections - All instances in user_project_wrapper are connected to power.

K) Port Types - Netlist user_project_wrapper port types match the golden wrapper port types.

6. XOR: Checks for total XOR differences if looking for more information on this test check the file in the directory that the pre-check suggests.
7. Magic DRC: Checks for any violations in the design rule check
8. Klayout FEOL: Checks for DRC violations
9. Klayout BEOL: Checks for DRC violations
10. Klayout Offgrid: Checks for DRC violations
11. Klayout Metal Minimum Clear Area Density: Checks for DRC violations
12. Klayout Pin Label Purposes Overlapping Drawing: Checks for DRC violations
13. Klayout ZeroArea: Checks for DRC violations

Each one of these checks will have to be passed before you are able to submit your project to Efabless. In order to execute these tests, enter the following commands into your terminal:

Commands: make precheck (you only need to do this once)

Commands: make run-precheck

If your project passes all of the prechecks the output in your terminal should appear similar to figure 17 below.



Figure 17. Output of make run-percheck

After this step is complete you must make sure that your repository is up to date and that you push all necessary files or edits to previous file to your repo. You now should have a solid base of understanding of the caravel repository, how to operate it, as well as what will need to be done for your project.

5. Getting started with the Wishbone Bus

5.1 WHAT IS THE WISHBONE BUS


The wishbone bus is a 32-bit bus connecting the user project wrapper with the management SoC. It includes the following user project wrapper ports:

Port Description	Port name
Clock	wb_clk_i
Reset	wb_rst_i
Strobe	wbs_stb_i
Cycle	wbs_cyc_i
Write enable	wbs_we_i
Select	wbs_sel_i
Data in (to user area)	wbs_dat_i
Data out (to SoC)	wbs_dat_o
Address	wb_adr_i
Acknowledgement	wbs_ack_o

Note: Addresses must be in 4-byte increments

5.2 USING THE WISHBONE BUS FROM THE MANAGEMENT SoC

In a caravel testbench, the management SoC can communicate using the wishbone using the following steps:

1. Enable wishbone: `reg_wb_enable = 1;`
2. Write to bus: `<32-bit address> = <32-bit data>;` (the user area may respond to any address from `0x3000_0000` to `0x7FFF_FFFF`) 
3. Read from bus: `uint32_t in_data = <32-bit address>;`

5.3 WRITING TO THE USER AREA

The user area can respond to a write request using the following process:



where `'valid = wbs_cyc_i && wbs_stb_i'`.

You should check if the address on the wishbone matches your component's address before reading/writing/acknowledging the data.

5.4 READING FROM THE USER AREA

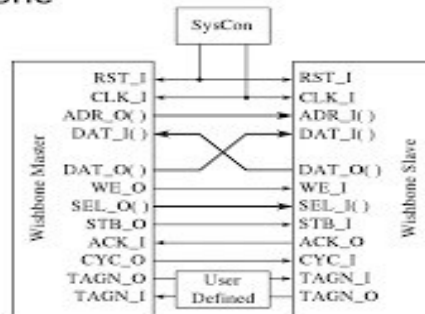
The user area can respond to a read request using the following process:



The caravel harness github repository includes the `user_proj_example`, which has some interaction with the wishbone, as well as some tests in the design verification (DV) verilog directory.

Video: [Caravel Wishbone bus demo - YouTube](#)

What is Wishbone



[https://en.wikipedia.org/wiki/Wishbone_\(computer_bus\)](https://en.wikipedia.org/wiki/Wishbone_(computer_bus))

Necessary Repositories:

1. [GitHub - mattvenn/wishbone_buttons_leds at caravel](#)
2. [caravel_user_project/README.md at wishbone-demo · mattvenn/caravel_user_project · GitHub](#)

6. Getting started with the Logic Analyzer

6.1 WHAT IS THE LOGIC ANALYZER

The logic analyzer is a 128-bit port going to/from the user area and management SoC. It allows the SoC to "probe" any desired signals from the user area after fabrication (but they cannot be changed once fabricated, so choose wisely). This is useful for debugging signals during the bring-up process, possibly to discover bugs in the hardware, or to verify that they are working correctly.

6.2 HOW TO USE THE LOGIC ANALYZER IN THE USER AREA

The logic analyzer can be used by connecting the 128 bits to any of the desired signals in your design. This can be seen in the user_proj_example caravel module.



Here, the logic analyzer is used to set the counter value. It takes the or-reduction of la_write (or'ing all bits together, |x operator) and if it is 1, sets the count to the la_write and'ed with la_input.

6.3 HOW TO USE THE LOGIC ANALYZER FROM THE MANAGEMENT SOC

For details on how to use the logic analyzer from the SoC, refer to the la_test1 dv directory in the base caravel repository.



First, the logic analyzer probes are configured to be inputs or outputs, then the data is written (or read) from the data register.

7. How to use interrupts

7.1 WHAT IS AN INTERRUPT

An interrupt is a signal from the hardware that pauses the current processor execution and begins execution somewhere else. This is commonly used to avoid blocking code, where the CPU continuously polls the hardware for data. Instead, the hardware can alert the CPU when a condition is met, such as a timer reaching a specified value.

7.2 HOW TO TRIGGER AN INTERRUPT FROM THE USER AREA

An interrupt can be triggered from the 3 irq bits available to the user area. Each of these bits corresponds to an interrupt. The interrupt should be high for one clock cycle, and low otherwise.

7.3 HOW TO RECEIVE AN INTERRUPT IN THE SoC

Interrupts can be enabled in the SoC in the following way:



The `irq_vex.h` header adds the functions/macros for `irq_setmask`, `irq_setie`, etc. The code starts by setting the mask to 0 and enabling interrupt 1. The external flag variable is the variable used by the interrupt handler. This handler is part of the existing simulation infrastructure, so it is not necessary to write it. The interrupt handler must be placed at a specific location in the SoC's program memory, so it is simpler to use the example provided by the harness repository by default. The second `irq_setmask()` sets the correct bit for the irq we are triggering in hardware. In this case, it is the 0th interrupt (irq bit 0 in the user area). Next, the `req_user_irq_enable` register should have the corresponding bit set for the interrupt you are using. Finally, the `req_userX_irq_en` must be set to 1 (where X is the irq index). In simulation, the flag variable will be set to 1 when the interrupt is triggered.

8. How to use klayout to view GDS files

Klayout can be a very useful tool when it comes to seeing the cells that you have added to the user project wrapper. So, in this section of the tutorial document, you will be instructed on how to view the project wrappers as well as how to interpret the cells. Klayout is only useful after you have hardened your design inside of the user_project_wrapper. The first thing you need to do is to enter the directory of your project where you hardened your design. Then type in the following command:

Command: `find -name '*klayout.gds'`

This command will find all of the files within the directory you are in that contain the "klayout.gds" attachment. The output will look similar but not the same as figure 18 down below.



Figure 18. Finding all klayout gds files

As can be seen from figure '#' above you are able to look at just the user area or the user project wrapper as a whole. The file names starting with **./user_project_wrapper** are the GDS files that contain your project in the user area inside of the user project wrapper. The file names that start with **./user_adder** are the GDS files that will show the user area with the user adder inside of it. For either one of the two the way to tell which file is the most recent is by looking at the numbers in the files. Reference figure 19 below for which numbers to look at.



Figure 19. Display of date and time GDS file was created

The meaning behind what each number represents is listed below:

1. 23 – stands for the year the GDS file was made
2. 02 – stands for the month the GDS file was made
3. 16 – stands for the day the GDS file was made
4. 15 – stands for the hour the GDS file was made in military time
5. 05 – stands for the minute the GDS file was made

Highlight and copy the most recent version of the file you made, or whichever version you want to view inside of klayout. Use the following command and paste the file name that you previously copied and want to view it inside of klayout.

Command: `klayout < Name of GDS File >`



Figure 20. Display of Klayout command with desired file

Once you enter the command a window should pop up with the layout of the caravel harness. See figure 21 below for reference.



Figure 21. Display of user project wrapper in klayout

This is the view of the unlabeled layout of the caravel harness. If you would instead like for a less cluttered version of the layout with labels, then you should clone the following repository into your (Insert directory name) directory. Once that is done your layout for the caravel harness will look more like figure 22 down below.



Figure 22. Display of filtered klayout user project wrapper

9. How to submit your project design to Efabless

Source: [MPW-6 Walkthrough - YouTube](#) - 26 min till the end

After your project design has passed all necessary tests and checks in your environment the next step is to submit your design to Efabless. You will want to head to the following link and look for the next available MPW shuttle submission.

Link: [Efabless](#)



Figure 23. Efabless MPW submission page

You will need to fill out the form displayed in figure (#), the only thing that you have to make sure to do is that when you copy and paste the URL to your git repository in the 'GIT URL' is to add the '.git' at the end of it. The 'Version' & 'Tags' section do not need to be filled out for your first submission. Once done hit save, then you will be taken to the following page displayed in figure 24 below.



Figure 24. Efabless MPW workspace on submission page

Be sure to Navigate to the Workspace section in this page. Then you will need to click the 'Git Pull' button. The time it takes for the submission page to pull your repository will vary depending on how many people are trying to do the same thing at the same time. But once the page has successfully pulled your repository the page will notify you that it has successfully pulled your repository.

Once that is finished the next step is to click on the MPW Precheck button and give it a 'Job Name' any name will do. It will take roughly 5 minutes for the test to complete but once it does and it succeeds your workspace window should appear like figure 25 below.



Figure 25. Image of Successful MPW Precheck

After you have successfully passed the 'MPW Precheck' the last thing that you will have to do is pass the Tapeout. Click on the 'Tapeout' button and the following pop-up window will appear as displayed in figure 26 below.

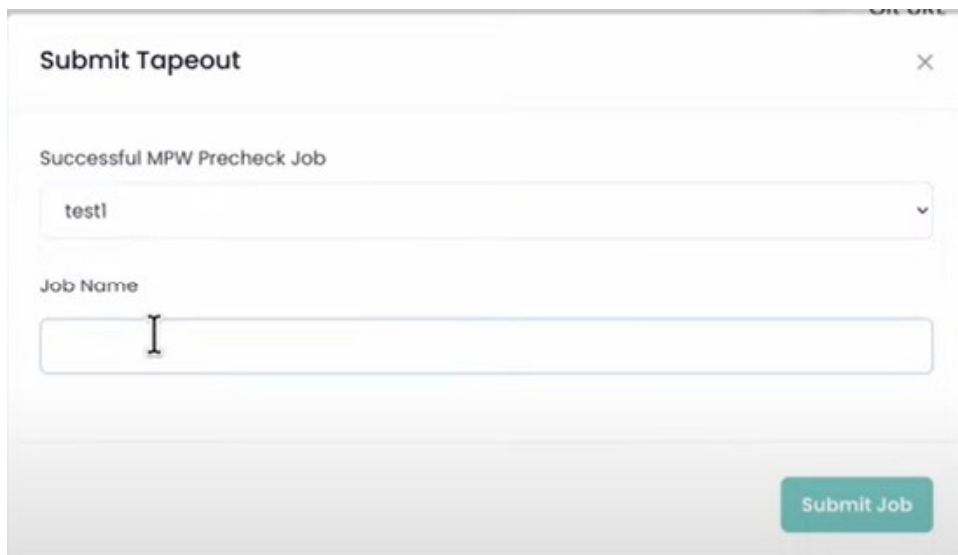


Figure 26. Image of Tapeout pop up window

You will then need to select the 'MPW Precheck' job name that you created and enter in a new Job name for the Tapeout then click 'Submit Job'. Now this test will take somewhere around an hour to complete. But once the Tapeout has finished you have successfully submitted your design to Efabless for the MPW submission.

10. Caravel PCB daughter & carrier boards you receive from Efabless

10.1. INTRODUCTION

The efabless caravel_board is an open-source development platform for designing and testing custom System on Chips (SoCs). It is designed to be flexible, customizable, and accessible to a wide range of users, from hobbyists to professional designers. The caravel_board is built on top of the efabless OpenLANE tool flow, which provides a complete open-source RTL to GDSII flow for chip design.

10.2.0. OVERVIEW OF CARAVEL_BOARD

The caravel_board is a small form factor board that measures 85mm x 54mm. It consists of several components that are organized into logical sections on the board. These sections include the power management section, the FPGA section, the GPIO section, and the JTAG section.

10.2.1. POWER MANAGEMENT SECTION

The power management section is responsible for providing power to the various components on the board. It consists of a 3.3V regulator, a 1.8V regulator, and a power-on reset circuit. The regulators ensure that the various components on the board receive the correct voltage levels, while the power-on reset circuit ensures that the board is properly initialized when power is applied.

10.2.2. FPGA SECTION

The FPGA section is the heart of the caravel_board. It consists of an FPGA chip and several components that support its operation. The FPGA chip is a Lattice iCE40HX8K FPGA, which provides 7680 logic cells, 128Kbits of

embedded RAM, and 4 PLLs. The FPGA is connected to the rest of the board through several interfaces, including SPI, I2C, and GPIO.

10.2.3. GPIO SECTION

The GPIO section provides a set of general-purpose input/output (GPIO) pins that can be used to interface with external devices. The GPIO pins are connected to the FPGA and can be programmed to perform various functions, such as controlling LEDs, reading buttons, or communicating with other devices.

10.2.4. JTAG SECTION

The JTAG section provides a JTAG interface for programming the FPGA and debugging the board. The JTAG interface is connected to the FPGA and can be used to program the FPGA with new firmware, or to test and debug custom logic on the board.

10.3.0. OVERVIEW OF CARAVEL_NUCLEO

The caravel_Nucleo is an accessory board that is used in conjunction with the caravel_board. It provides additional functionality that is necessary for programming and testing the caravel_board. The caravel_Nucleo board consists of several components, including an ST-Link debugger, a USB interface, and several buttons and LEDs.

10.3.1. ST-LINK DEBUGGER

The ST-Link debugger is used to program the caravel_board with new firmware, and to debug custom logic on the board. It is connected to the JTAG interface on the caravel_board and communicates with the board through the USB interface.

10.3.2. USB INTERFACE

The USB interface provides a way to communicate with the caravel_board and to transfer data to and from the board. It is connected to the ST-Link debugger and can be used to program the board with new firmware, or to read and write data to the board.

10.3.3. BUTTONS AND LEDs

The buttons and LEDs on the caravel_Nucleo board are used for testing and debugging the caravel_board. The buttons can be used to trigger various events on the board, while the LEDs can be used to indicate the status of various components on the board.

10.4.0 HOW THE PARTS ON THE BOARD CONNECT TO EACH OTHER?

The various parts on the caravel_board and caravel_Nucleo are connected to each other through a series of connectors and traces on the board.

10.4.1. FPGA CONNECTIONS

The FPGA on the caravel_board is connected to several components on the board through a series of connectors and traces. The FPGA is connected to the power management section, which provides the necessary power to the FPGA. It is also connected to the GPIO section, which provides a set of GPIO pins that can be used to interface with external devices. The FPGA is also connected to the JTAG section, which provides a JTAG interface for programming the FPGA and debugging the board.

10.4.2. POWER MANAGEMENT CONNECTIONS

The power management section on the `caravel_board` is connected to the various components on the board through a series of connectors and traces. The 3.3V regulator is connected to the FPGA and the GPIO section, while the 1.8V regulator is connected to the FPGA. The power-on reset circuit is connected to the FPGA and the JTAG section.

10.4.3. GPIO CONNECTIONS

The GPIO section on the `caravel_board` is connected to the FPGA through a series of connectors and traces. The GPIO pins are used to interface with external devices and can be programmed to perform various functions, such as controlling LEDs, reading buttons, or communicating with other devices.

10.4.4. JTAG CONNECTIONS

The JTAG section on the `caravel_board` is connected to the FPGA through a series of connectors and traces. The JTAG interface is used to program the FPGA with new firmware, or to test and debug custom logic on the board.

10.4.5. ST-LINK CONNECTIONS

The ST-Link debugger on the `caravel_Nucleo` board is connected to the JTAG interface on the `caravel_board` through a series of connectors and traces. The ST-Link communicates with the `caravel_board` through the USB interface, which is also connected to the `caravel_Nucleo` board.

10.4.6. BUTTON AND LED CONNECTIONS

The buttons and LEDs on the `caravel_Nucleo` board are connected to the `caravel_board` through a series of connectors and traces. The buttons can be

used to trigger various events on the board, while the LEDs can be used to indicate the status of various components on the board.

10.5. HOW THE CARAVEL_BOARD CONNECTS TO THE USER AREA AND HOW TO TEST IT?

The `caravel_board` can be used to design and test custom SoCs. The user area is where the custom logic is implemented and tested. The user area is a part of the FPGA chip and is programmed with custom firmware using the JTAG interface and the ST-Link debugger.

To connect to the user-area, the user can use the available pads on the `caravel_board`. The user can also add additional pads if necessary. The user can then use the open-source tools provided by eFabless to design and test custom logic for the user-area.

To test the user area, the user can write custom Verilog code to implement the desired functionality and then use the open-source tools provided by eFabless to synthesize, place, and route the design. Once the design is complete, it can be programmed onto the `caravel_board` using the ST-Link debugger and the JTAG interface.

Furthermore, to test the user area, the user can then interact with the custom logic using the GPIO pins on the `caravel_board`. The GPIO pins can be programmed to perform various functions, such as controlling LEDs, reading buttons, or communicating with other devices. The user can also use the JTAG interface and the ST-Link debugger to debug the custom logic and to monitor its behavior.

In short, to test the user-area, the user can use the following steps:

- 1 Design custom logic using the open-source tools provided by eFabless, such as OpenLANE RTL to GDSII flow.
- 2 Synthesize the design using the open-source Yosys synthesis tool.

- 3 Place and route the design using the OpenLANE tool.
- 4 Generate the GDSII layout file for the design.
- 5 Create a new firmware for the caravel_board that includes the custom logic.
- 6 Program the caravel_board with the new firmware using the JTAG interface on the caravel_Nucleo board.
- 7 Test the custom logic on the caravel_board using the appropriate testbench or software

The open-source tools and documentation provided by efabless make it easy for users to customize and extend the caravel_board for their specific needs. The caravel_Nucleo board provides the necessary interfaces for programming and testing the caravel_board, and the power management circuitry ensures that the board is powered properly during testing.

10.6. CONCLUSION

In conclusion, the efabless caravel_board is an open-source platform that provides an excellent opportunity for learning and experimenting with ASIC/SOC development. It includes all the essential components required for a complete system-on-a-chip design, such as the FPGA, RAM, flash memory, voltage regulators, and various input/output interfaces. The board's open-source nature enables the community to contribute and improve the design, making it accessible and affordable for hobbyists and professionals alike.

The caravel_board design flow uses open-source tools such as the Yosys synthesis tool and the OpenLANE flow for design automation. It simplifies the design process, enabling users to customize the platform to their needs and interests. With caravel_board, users can create custom designs and test them in a real-world environment without requiring significant financial investment.

At last, the eFabless Caravel Board is a valuable platform for anyone interested in learning or experimenting with ASIC/SOC development. Its open-source nature, affordability, and community support make it an ideal choice for hobbyists and professionals alike. It provides an excellent opportunity for users to experiment with custom designs and develop their skills in FPGA development.

10.7. FULL FORM OF ALL ABBREVIATIONS

ASIC - Application-Specific Integrated Circuit

SOC - System-on-a-Chip

FPGA - Field-Programmable Gate Array

PDK - Process Design Kit

RAM - Random Access Memory

HDL - Hardware Description Language

I/O - Input/Output

PCB - Printed Circuit Board

USB - Universal Serial Bus

JTAG - Joint Test Action Group

GPIO - General Purpose Input/Output

GDSII - short for Graphic Data System II

LED - Light Emitting Diode

OpenLANE - Open-Source Digital ASIC Implementation Flow

RTL - Register Transfer Level

RISC-V - Reduced Instruction Set Computing - Five

11. References

- 1 Caravel Board on GitHub. https://github.com/efabless/caravel_board
- 2 Caravel Documentation. <https://caravel-design.readthedocs.io/en/latest/>
- 3 Introduction to the eFabless Caravel Platform. <https://www.youtube.com/watch?v=xj9KYWz1H7k>
- 4 eFabless Caravel Board: The Best Place to Start Learning About Chip Design. <https://www.electronicsforu.com/electronics-projects/hardware-diy/efabless-caravel-board-best-place-start-learning-chip-design>

- 5 The Caravel Open Source ASIC Design Flow.
<https://www.allaboutcircuits.com/technical-articles/the-caravel-open-source-asic-design-flow/>
- 6 An Introduction to ASIC Development with Caravel.
<https://www.hackster.io/news/an-introduction-to-asic-development-with-caravel-396a20c8b33a>
- 7 Caravel: An Open Source PDK and ASIC/SOC Development Platform.
<https://efabless.com/news/2019/8/7/caravel-an-open-source-pdk-and-asicsoc-development-platform>
- 8 Design Your Own Chip: The efabless Open-Source Caravel Platform.
<https://www.iotforall.com/design-your-own-chip-the-efabless-open-source-caravel-platform/>
- 9 ASICs and FPGAs and SoCs, Oh My! A Guide to Silicon on Chip.
<https://www.eetimes.com/asics-and-fpgas-and-socs-oh-my-a-guide-to-silicon-on-chip/>
- 10 Open Source FPGA Toolchain.
https://en.wikipedia.org/wiki/Open_Source_FPGA_Toolchain