ReRAM Compute ASIC Fabrication

Design Document

sddec23-08

Client: Dr. Duwe
Adviser: Dr. Wang
Team Members: Joshua Thater, Aiden Petersen, Matthew Ottersen, Regassa Dukele
sddec23-08@iastate.edu
https://sddec23-08.sd.ece.iastate.edu/

Revised: 12/02/2023

Table of Contents

1 Team 3

1.1 Team Members 3

1.2 Required Skill Sets for Your Project 3

1.3 Skill Sets covered by the Team 3

1.4 Project Management Style Adopted by the team 3

1.5 Initial Project Management Roles 3

2 Introduction 4

2.1 Problem Statement 4

2.2 Requirements & Constraints 4

2.3 Engineering Standards 5

2.4 Intended Users and Uses 5

3 Design 6

3.1 Design Context 6

3.1.1 Broader Context 6

3.1.2 User Needs 7

3.1.3 Prior Work/Solutions 7

3.1.4 Technical Complexity 8

3.2 Design Exploration 9

3.2.1 Design Decisions 9

3.2.2 Ideation 9

3.2.3 Decision-Making and Trade-Off 10

3.3 Proposed Design During 491 10

3.3.1 Design Visual and Description 10

3.4 How the Design Changed During 492 13

3.4.1 Design Visual and Description 13

3.4.2 Functionality 18

3.4.3 Areas of Concern and Development 19

3.5 Technology Considerations 19

3.6 Design Analysis 20

3.7 Design Plan 20

4 Implementation 21

4.1 Implementation of Devices 21

4.2 Implementation Issues 26

5 Testing 26

5.1 Unit Testing 26

5.2 Interface Testing 27

5.3 Integration Testing 27

5.4 System Testing 28

5.5 Regression Testing 28

5.6 Acceptance Testing 28

5.7 Issues With ReRAM Simulation........ 29
5.8 Results........ 29
6 Project Outcomes........ 30
6.1 Final 8x8 Compute Crossbar........ 31
6.2 GitHub Repository........ 31
6.3 Post-Fabrication Bring-up Plan........ 31
6.4 Tutorial Documentation on Analog Design Flow + Other Documentation........ 32
7 Closing Material........ 32
7.1 Discussion........ 32
7.2 Conclusion........ 32
7.3 References........ 33
8 Appendices........ 33
8.1 Appendix 1 - Operation Manual........ 33
8.1.1 Pre-Fabrication Environment Setup........ 33
8.1.2 Pre-Fabrication Tool Setup........ 39
8.1.3 GitHub Repository Cloning........ 51
8.1.4 Post-Fabrication Bring-up Plan........ 51
8.2 Team Contract........ 69

1 Team

1.1 Team Members

- Joshua Thater
- Aiden Petersen
- Matthew Ottersen
- Regassa Dukele

1.2 Required Skill Sets for Your Project

- Willingness to learn and work together
- Strong persistence through challenging work
- Schematic editor
- Layout editor
- DevOps
- MCU programming
- PCB design

1.3 Skill Sets covered by the Team

- Whole team
- Whole team
- Joshua Thater, Matt Ottersen
- Joshua Thater, Matt Ottersen
- Aiden Petersen
- Aiden Petersen
- Joshua Thater

1.4 Project Management Style Adopted by the team

- We are going to use The Agile method with short 1 week sprints.

1.5 Initial Project Management Roles

- Joshua Thater: Software tool researcher, mixed-signal designer
- Aiden Petersen: DevOps and digital designer
- Regassa Dukele: VLSI designer
- Matt Ottersen: VLSI designer

2 Introduction

2.1 Problem Statement

Ideal:

- Iowa State University would have access to fabricated ReRAM chips for research purposes
- Iowa State University would have institutional knowledge of how the analog design flow works for the Skywater 130nm process

Reality:

- It is difficult to get any fabricated chip, especially one with ReRAM, because of how new of a technology it is.
- Iowa State University has never produced a fabricated analog chip on the Skywater 130nm processed

Consequences:

- It's difficult to do novel research that involves ReRAM because of the lack of fabricated chips.
- It's very difficult for students and faculty to create analog chips using the Skywater 130nm process because of the poor public documentation and the lack of internal documentation.

Proposal:

- Use eFabless's MPW shuttle program to submit a ReRAM chip proposal.
    - If it gets approved, it would give us access to fabricated ReRAM chips
    - Along the way, we would document our workflow, contributing to ISU's internal knowledge of open-source analog chip fabrication in the SkyWater 130nm process.

2.2 Requirements & Constraints

Functional Requirements (Specification)

- Create an eFabless MPW submission that contains a ReRAM crossbar that can be interacted with through the caravel harness.
- Crossbar must output higher whenever a single storage cell is in a LRS (low resistive state).
- Design must fit with the user project area (10 mm^2)
- MPW Submission must pass eFabless pre-check.
- The crossbar will be interacted with through the wishbone bus, which is then turned into analog signals using a DAC and read back into the harness using an ADC.
- The ReRAM crossbar will be functional and able to perform multiple and accumulate operations.
- The ReRAM crossbar's inputs and weights will be assignable.
- Periphery analog circuitry must be designed to support crossbar computations. Some components that may be built include:
    - 1-bit ADC
    - 1-bit DAC

        - Transimpedance amplifier
        - 1T1R storage cell
        - ReRAM crossbar
        - MUXs
- Analog circuitry must be functional both at schematic and post-layout R+C extracted versions.

Non-Functional Requirements

- Document tools and tool flow necessary in the open-source analog design flow through eFabless.
- Create tutorial documentation on the open-source analog design flow. Include:
    - Environment setup
    - Required tools
    - How the tools interact with each other
    - How the tools integrate with the SkyWater 130 nm process
- Create a detailed post-fabrication bring-up plan that details how to test the chip and all of the different components located on it
- Create drivers that provide an API for programmers to interact with the crossbar.

2.3 Engineering Standards

IEEE 1481-2019 - IEEE Standard for Integrated Circuit (IC) Open Library Architecture (OLA): This standard fits our project since it goes over ways for our integrated circuit to be analyzed for timing and power consumption across a set of design automation (EDA) tools.

IEEE 1076.4-2000 - IEEE Standard VITAL ASIC Modeling Specification: This standard applies to our project since we will be designing an ASIC chip, and we will need it to test it with highly accurate and efficient simulation models.

IEEE 1149.4-2010 - IEEE Standard for a Mixed-Signal Test Bus: This standard will fit our project since it will house both analog and digital components, and we will have to thoroughly test all components both independently and together.

IEEE 1364-2005 - IEEE Standard for Verilog Hardware Description Language: This standard fits our project since we will have to write Verilog code that will communicate between the wrapper and the analog part of our project

2.4 Intended Users and Uses

The main intended user for the results of our project will be Professor Duwe and Professor Wang. They first want to test the potential computing power of a ReRAM crossbar. They also want to use our experiences and knowledge of our project to help future students design ASIC chips through the eFabless ecosystem, whether it be for research or in pursuit of creating a club for chip fabrication.

Another user would be potential future students who are interested in analog/mixed-signal chip fabrication. It is our hope that they will be able to use our project as either an example or a jumping-off point for their own design. With the help of our documentation and written experiences, they should be able to hit the ground running when they want to work on fabricating their own chip through the eFabless process.

3 Design

3.1 Design Context

3.1.1 Broader Context

ReRAM is a non-volatile RAM that has the potential to perform computations in the analog domain. This can be beneficial as this requires less data movement and less power consumption as compared to typical computations done in the digital domain. Our project is to silicon prove a ReRAM crossbar using the SkyWater 130 nm process. We are hoping to tape-out this design and get it fabricated so that Professor Duwe and Professor Wang can evaluate the crossbar for computational potential. Outside of the two Professors, we are also designing for future students interested in open-source ASIC chip fabrication, as our project will help pave the way as an example. With our design, we are hoping that Iowa State will be able to create a curriculum or a club that is focused on the fabrication of chips. This will help the societal need of young engineers looking to get trained in the process of designing and fabricating ASIC chips.

Relevant considerations related to our project:

Area	**Description**	**Examples**
Public health, safety, and welfare	**This project does not have a huge impact on public health, safety, or welfare. However, ASIC chips will continue to have important uses in many applications around the world that do improve these areas.**	**Our design will be added to the open-source hardware in the eFabless program, which may lead to inspire or help other engineers learn how to design ASIC chips.**
Global, cultural, and social	**As this is an open-source project, we are following the core ideals of open-source development. These include transparency, collaboration, and continual growth. As such, our project will follow these ideals and will help pave the way for other engineers to contribute to the open-source community.**	**This project will add to the open-source hardware repository on the eFabless website. Our design may be used as a template for future engineers, whether inside this university or outside, as a potential template for future ReRAM crossbar designs.**
Environmental	**ReRAM has the potential to consume much less power than its digital counterparts. There are many**	**Our project will be implementing a basic ReRAM crossbar. This ReRAM will have**

	companies that are looking at ReRAM and trying to gauge whether or not it will be beneficial to their designs in the future. If this design is adopted, then there may be a substantial improvement in power consumption for RAM throughout the world.	**the hopes of utilizing much less power compared to other RAM types of similar size and computational power.**
Economic	**As this is an open-source project, all of the tools are completely free to use. Along with this, there are similar designs that have been created through this open-source workflow as well. These designs may lead to faster innovation and may lead to companies feeling the need to innovate their designs as well.**	**As we are designing our project through this open-source workflow, all of our designs and process will be free to view and use for future use.**

Table 1: Broader Context

3.1.2 User Needs

Professor Duwe is interested in building up ISUs in house capabilities with open-source ASIC chip design. He currently has other teams in the past go through the digital design flow, but he has yet to have a team map out the analog design flow. He has tasked us with figuring out this process flow and documenting it out for future use here at ISU with potential students and researchers.

Professor Wang is currently doing research with ReRAM, and he wants to test the SkyWater 130 nm process ReRAM for computational potential and fabrication. He needs us to create this ReRAM crossbar through the eFabless program so that he can evaluate its potential in his future research.

Future students interested in ASIC chip design need a way to understand how to create chips through the open-source workflow because learning how to tape out chips before entering the industry is an important skill to have.

3.1.3 Prior Work/Solutions

There are only a couple of similar designs on the market, but they are proprietary and may not allow for the types of testing that our client wants. There have been numerous research studies and papers into ReRAM, but they often don’t focus on the periphery analog circuitry that is needed to do ReRAM computations. They also don’t talk about the fabrication process of these designs. So, there is not a lot of outside documentation we can utilize for how to approach designing a ReRAM compute crossbar much less attempting to actually fabricate it. That means that much of our design is going to be left to us to figure out how to do it based on our understanding of ReRAM computation and the analog circuitry that would be needed to support it.

We have also looked through passed MPW submissions that contained ReRAM modules. Many of these submissions claim to have said ReRAM modules in them, but after cloning the repositories and checking the files, very few actually include ReRAM at all. There are a few that do include some ReRAM modules, but they are untested, and we don't actually know if they would work on a fabricated chip. They also seemed to not have a way to test the ReRAM module at the schematic level, so while there were some designs that included ReRAM, we decided it would be best to come up with our own design based on concept sketches we have found through a few research papers.

One of the research papers that we used to research ReRAM computation and the basic structure of the periphery analog circuitry:

[1] A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea (South), 2016, pp. 14-26, doi: 10.1109/ISCA.2016.12. https://ieeexplore.ieee.org/document/7551379

Another research paper that was used to aid our research into ReRAM computation can be found here:

[2] T. Chou, W. Tang, J. Botimer, and Z. Zhang, Electrical Engineering and computer science, https://web.eecs.umich.edu/~zhengya/papers/chou_micro19.pdf.

3.1.4 Technical Complexity

The design consists of both digital and analog circuit design. The components of the design include:

Resistive Ram Module

- This requires research into how resistive ram works
- Figuring out how to implement ReRAM into our design
- How to simulate this module by itself and with other components

ADC and DAC

- We will need to design Op-Amps with our preferred performance characteristics that will amplify signals and work as voltage shifters
- Knowledge of how to design ADC and DAC for specific applications
- How to integrate these designs together to work with the ReRAM crossbar

Use of open source tools

- Ability to learn, setup, and troubleshoot software/process flow that has little to no documentation

Creating a layout for the circuit that passes LVS and DRC

- This will require design knowledge and skills for ASIC chip design

One challenge is the theoretical implementation of ReRAM. This will require research of how ReRAM works and then designing a circuit that allows a system to ReRAM effectively. Then, the next challenge comes through having to bring that design to silicon by using an open-source design process. This will be complex as the open-source analog process flow we are using has very little documentation on how to bring design through the process.

3.2 Design Exploration

3.2.1 Design Decisions

Our design has many circuit components in it, and all of these components have multiple ways to be implemented. One of the key design decisions we made was what type of DAC would be used to pipe the digital voltages (high or low) to analog voltages so that the ReRAM crossbar would compute in the analog domain. We landed on doing a simple DAC architecture of a buffer, which is just 2 inverters cascaded together. This will work for our design is it will allow us to take in a voltage and shift it to whatever voltage is supplied to the VDD of the buffer.

Another key design decision we made was the architecture of the ADC. Just like with the DAC, we need a way to convert between analog and digital realms. For this project, we need a way to convert the analog computations back to a digital value to be read. The next 2 sections detail our design decision for this crucial component.

One last key design decision we made was how we were going to load weights onto the ReRAM cells. We have decided that we are going to use the logic analyzer pins provided to us on the Caravel Harness. However, these pins only output 0 V or 1.8 V, while our ReRAM crossbar will need varying voltages. To get these varying voltages we will use level shifters and our DAC. The level shifters, which will be voltage dividers, will take the power supply voltage and reduce it down to the needed voltages to the VDDs of DAC that will then feed into the ReRAM crossbar.

3.2.2 Ideation.

For data converter architectures, there are many options to choose from. Specifically for ADC converters, there are almost too many options to choose from. Each of these options comes with its own benefits but also its own drawbacks. One ADC may be faster but less accurate, while another may be able to do more complex conversions. So, picking the architecture for our ADC was important as we wanted it to be able to best match the needs of our project. From our research, we found that there were five main architectures that are widely used. These architectures were:

- Successive Approximation (SAR) ADC
- Dual-Slope ADC
- Pipelined ADC
- Flash ADC
- Delta-Sigma ADC

3.2.3 Decision-Making and Trade-Off

In order to choose which architecture to use, we weighed the pros and cons of each. After some research, we created this table:

ADC Architecture	Pros	Cons
SAR	Handles waveshapes very well, takes very high sample rate, high resolutions	Can be slow, larger/more complex design at low resolutions
Dual-Slope	Very accurate, and has high-resolution	Slow sample rate and speed, can be hard to implement
Pipelined	Very fast speeds, good resolution	Inherent latency due to architecture
Flash	The fastest speed, no latency, easy to implement at "1-bit" level	Circuit gets much bigger with resolution increases, limited to 8-bit resolution
Delta-Sigma	Very high resolution, reduces quantization noise	Limited sample rate, does not handle unnatural waves

Table 2: Decision Making for ADC Architecture

As can be seen from this table, each architecture comes with its own pros and cons. For our project, we are only using a 1-bit ADC, so we do not care about resolution. We also want something that is relatively small and easy to implement, as this circuit will be copied and pasted for the 8 columns that our crossbar will have. Because of all of this, we went with the flash ADC. While it can get larger with more bits, since we are only using it for 1-bit conversions, this will not be a concern. We also chose this one because of its fast speed, low latency in conversions, and because it is relatively simple to implement in a 1-bit form.

We also needed to choose from which type of ReRAM cell we were going to use. We decided on using 1T1R (1 transistor 1 resistor) since it had the most information available about it in the research papers we used to base our designs off of, and it suited our needs. There were also 1 transistor multiple resistors, 1 selector 1 resistor, and 1 diode 1 resistor options, but we decided not to use these.

3.3 Proposed Design During 491

3.3.1 Design Visual and Description

The figure below shows a basic ReRAM crossbar with analog circuitry around it that allows for a digital value to be input and a digital value to be output from the analog computation.

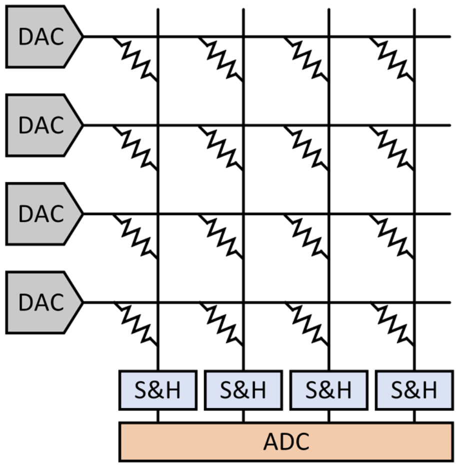
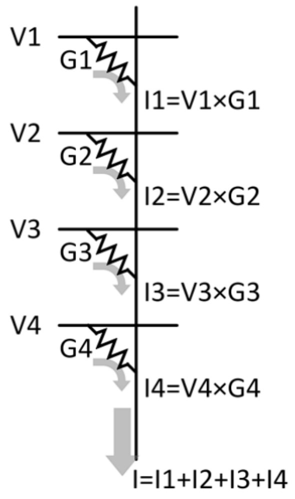


Figure 1: ReRAM Crossbar High-Level Diagram [2]

The figure above shows the basic top-level schematic of how we will create our design to test the ReRAM crossbar for computational potential. First, we input digital values into the DAC. As they go through the DAC, the digital values are converted into analog voltages, which go through the ReRAM crossbar. At each column of the crossbar, current will accumulate, depending on the state of the crossbar and the digital values inputted. At each column, a transimpedance amplifier will be placed so that we can convert the current back to a voltage. Once that voltage is obtained, it will go through a sample and hold circuitry (S&H) and then be converted back to a digital value through a 1-bit ADC. This digital value is then inputted back into the Caravel harness to be read. This is called a MAC operation, and a simplified version of it is shown below. The input voltage is multiplied by the conductance of the ReRAM which gets accumulated on the column.



$$I = \begin{bmatrix} V_1 & V_2 & V_3 & V_4 \end{bmatrix} \times \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

Figure 2: MAC Operation [2]

The conductance of a ReRAM cell is changed depending on whether it is in an LRS (low resistive state) or if it is in an HRS (high resistive state). If it is in an LRS, it can be interpreted as a "digital 1" as the cell will be on. If it is in an HRS, it can be interpreted as a "digital 0" as the cell will be off. By setting these different cells to different states, we can perform complex computations. The figure below shows an illustrative way to understand this idea.

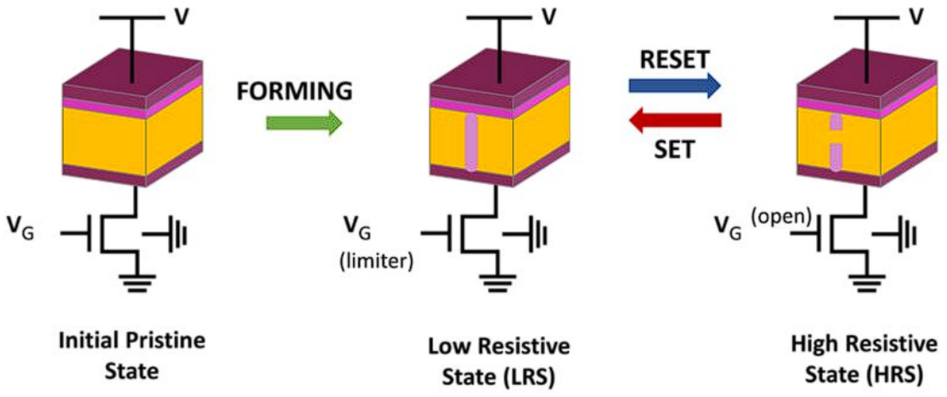


Figure 3: ReRAM States [3]

The figure below shows what one cell of the ReRAM (1T1R cell) crossbar will look like.

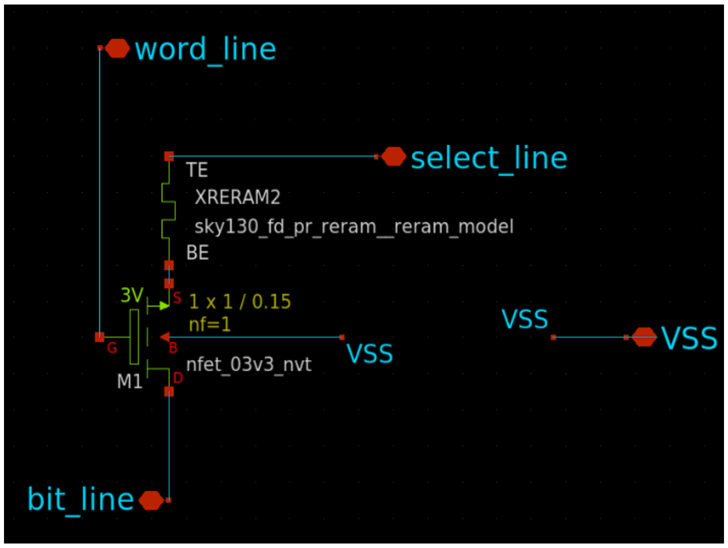


Figure 4: 1T1R Cell

As is seen in the figure, the cell consists of the ReRAM and a transistor. Thus, it is named a 1T1R cell. The gate of the transistor is going to be attached to the logic analyzer of the Caravel harness, which will write a weight to it. In the overall circuit, we will write the weights for an entire row of these cells at once. The select_line will be what is inputted into the cell. So, that means that the output of the 1-bit DAC will go into the select_line. Finally, the output of the cell will be the bit_line which will accumulate current. These currents will be added to the currents of other cells along that column that will then be converted to a voltage.

The figure below shows our whole system and how it will be integrated with the Caravel harness infrastructure:

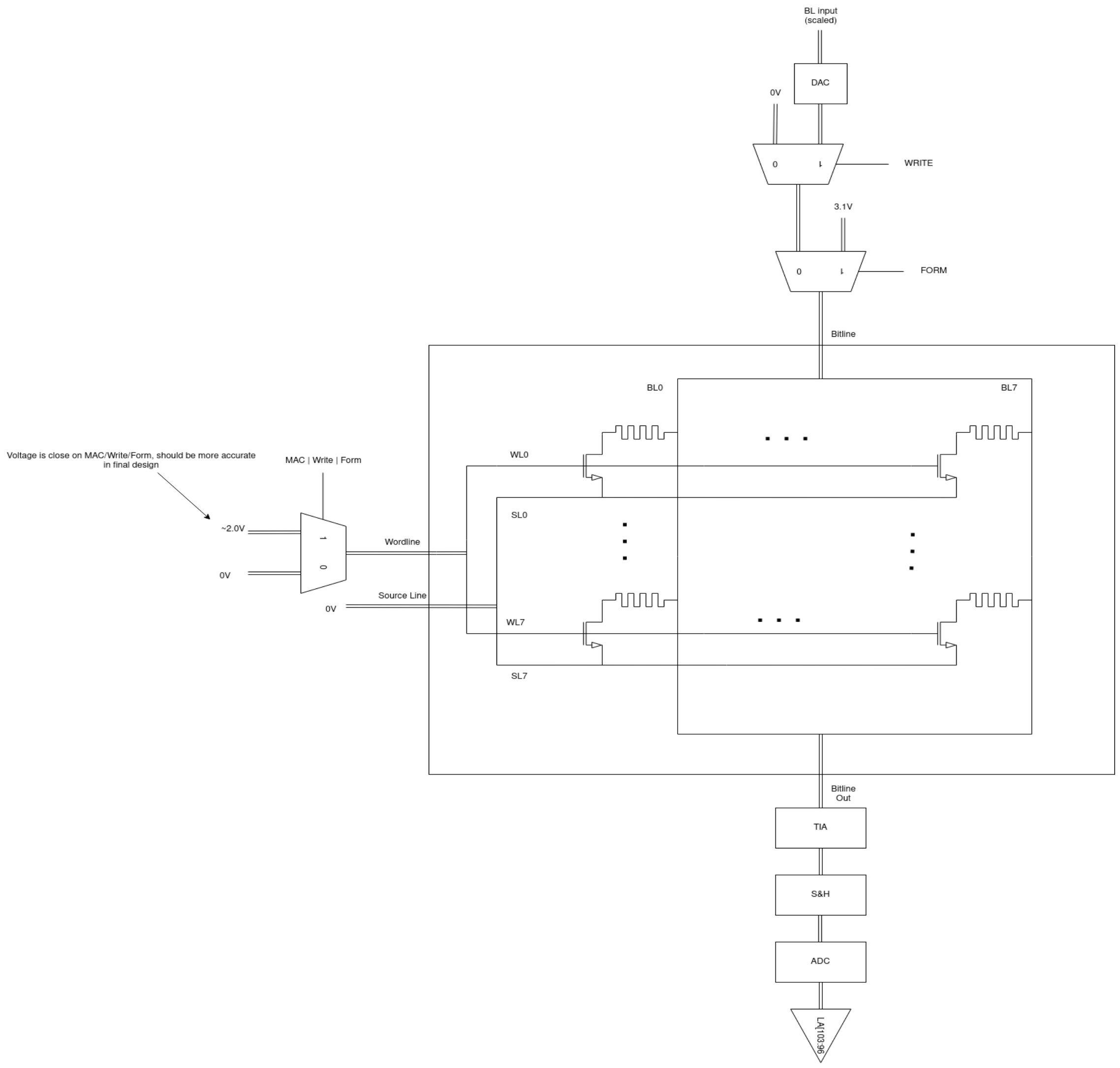
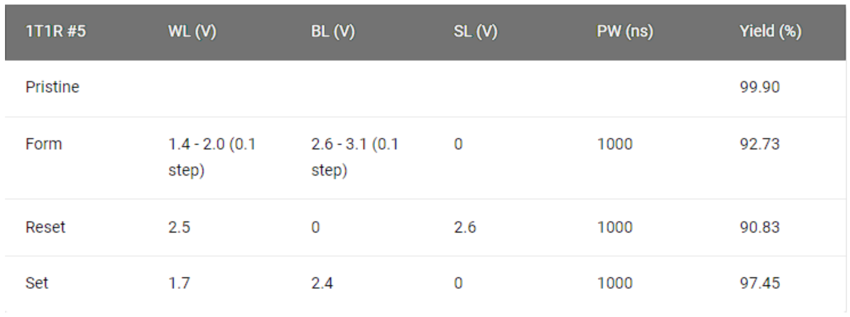


Figure 5: User Area Top Level Diagram

3.4 How the Design Changed During 492

3.4.1 Design Visual and Description

Throughout 492 our design has changed quite a bit based on our changing understanding of the limitations of the Caravel Harness. We also have simplified the design so that it would be easier to implement during the layout process. While we have changed our top-level design, the overall idea of how we perform calculations remains the same. We will still take in voltages from the logic analyzer and shift them to the needed voltages that will be input to the crossbar depending on what operation is occurring. The table below comes from the SkyWater documentation for the ReRAM module we are using and shows the recommended voltages for the different write operations that are needed for ReRAM.



1T1R #5	WL (V)	BL (V)	SL (V)	PW (ns)	Yield (%)
Pristine					99.90
Form	1.4 - 2.0 (0.1 step)	2.6 - 3.1 (0.1 step)	0	1000	92.73
Reset	2.5	0	2.6	1000	90.83
Set	1.7	2.4	0	1000	97.45

Figure 6: Table for Recommended Voltages for ReRAM Operations [4]

So, using the information in this table we will use the level shifters (voltage dividers) to supply the correct voltages for the VDDs of DAC. These DACs will then input the correct voltages into the crossbar. From there once we have written the crossbar, we will then do computations on it, where we will input a voltage which will be multiplied by the conductance which will then accumulate current on a column. These accumulated currents will then be converted back to a digital voltage through a transimpedance amplifier, inverting amplifier, and 1-bit ADC. Our updated top-level design can be seen in the figure below.

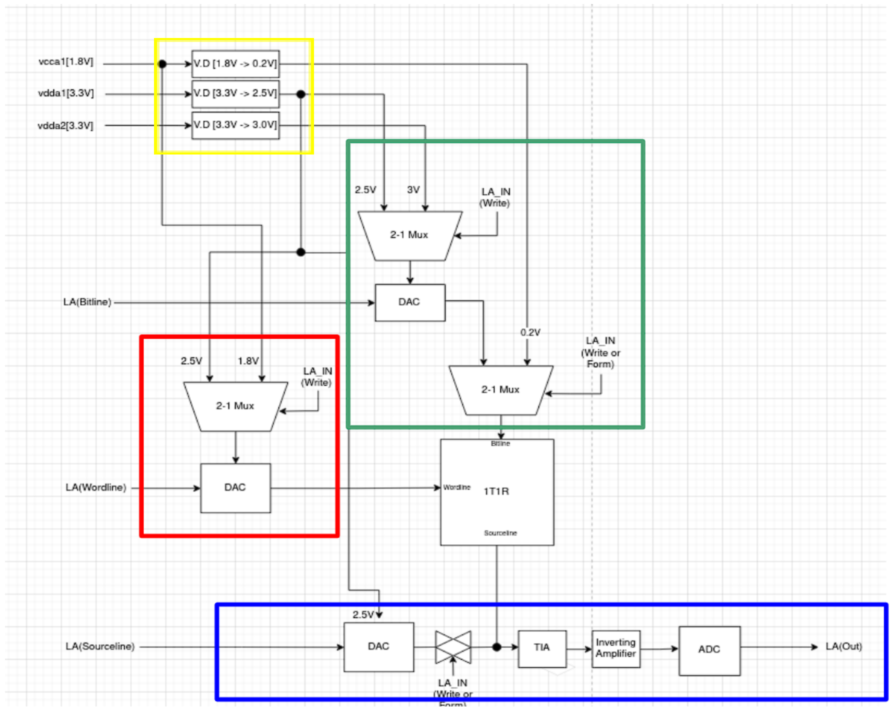


Figure 7: Updated Top-Level Design

The circuits that are enclosed in the yellow box are our level shifters. These will take the different power voltages supplied by the Caravel Harness and shift them down to the needed voltages for the crossbar. The circuitry in the green box is the bit-line. In the red box is the circuitry for the worldline. Finally, the blue box encompasses the select line circuitry. We also got rid of the sample and hold circuit as we realized that it is not needed for the 1-bit ADC we are using. It should also be noted that we changed the 1T1R cell slightly for this updated design. The updated 1T1R cell is shown in the figure below.

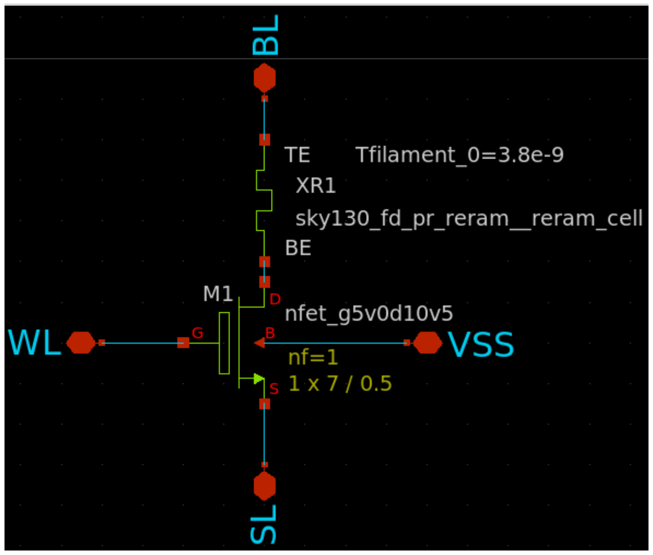


Figure 8: Updated 1T1R cell

In this updated design, we have switched where the bitline and selectline are. This is because we changed which side of the 1T1R cell we will output current from. After doing some more reading during this semester we realized that it was pretty interchangeable which side of the cell you outputted current from, so we decided to switch it to this orientation.

With our updated design we were also able to do all of the needed operations (form, write, read, MAC) which was not clear with the previous design. The figures below show how the voltage travels through our design to do the different operations.

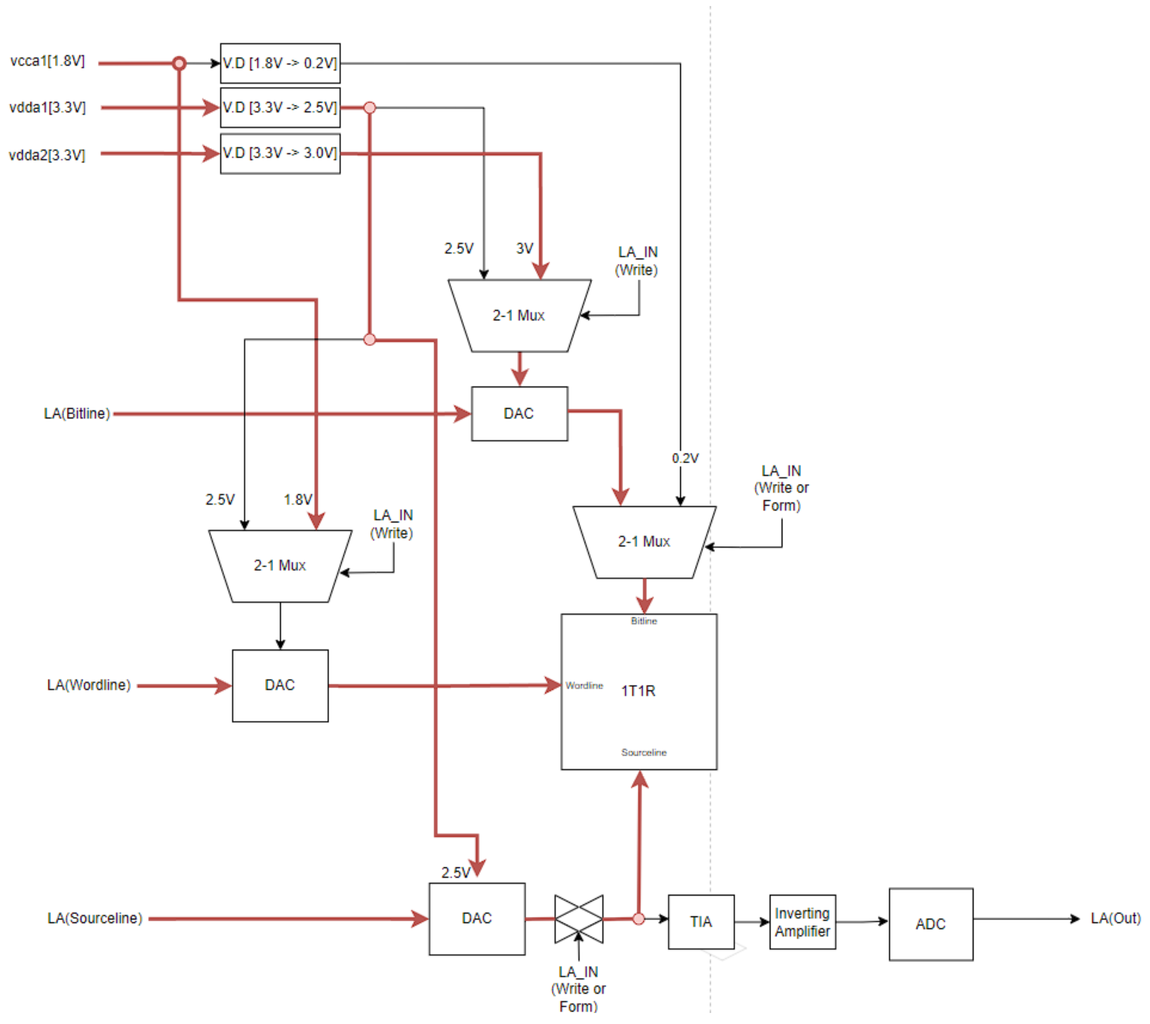


Figure 9: Form Operation

The figure above shows the form operation. This operation will need to be done so that we can do all other operations on the crossbar. The bitline is being supplied 3 V, the worldline is being supplied 1.8 V and the select line is inputting 0 V (LA(Souceline) is supplying a 0 V input, so even though the VDD of the DAC is 2.5 V, only 0 V go through).

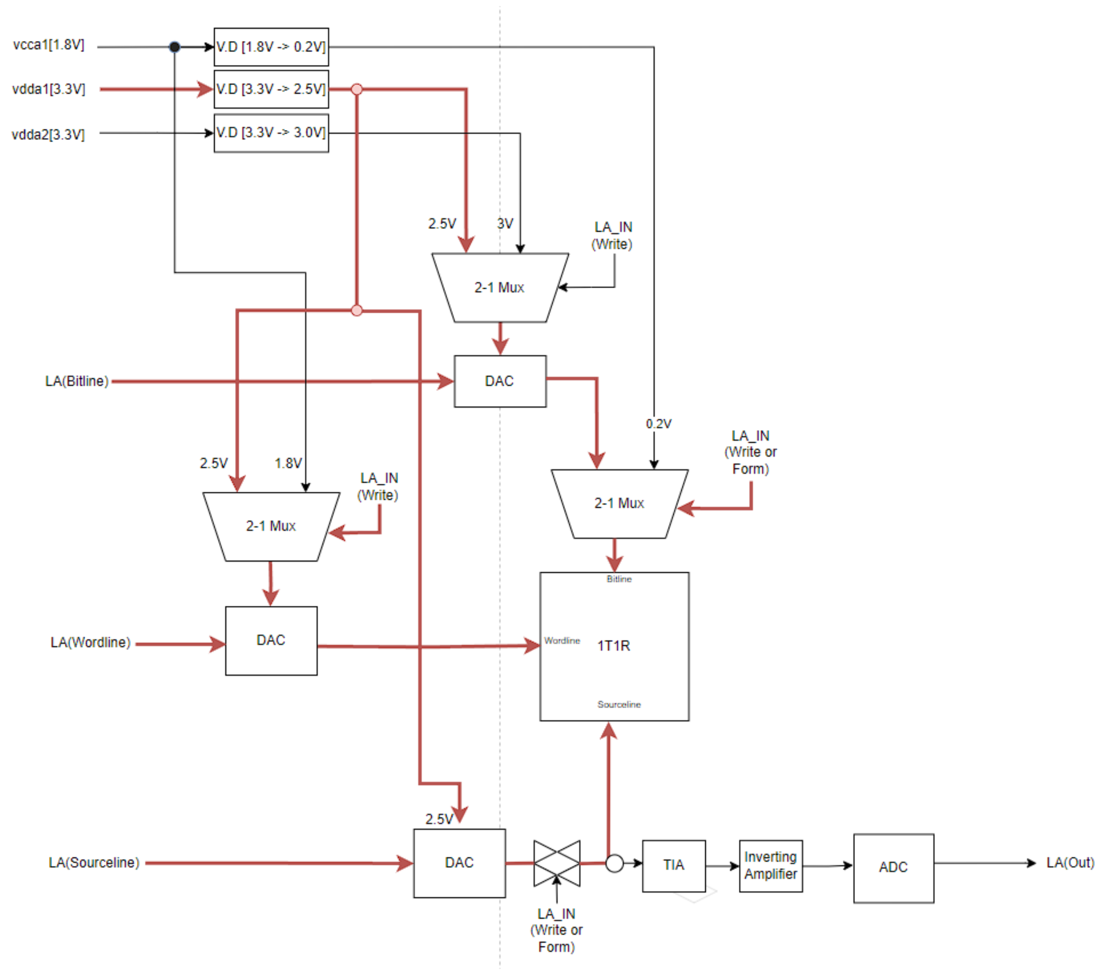


Figure 10: Write Operation

The figure above shows the write operation. In this operation, the ReRAM storage cell is either “set” (LRS) or “reset” (HRS). In this operation the worldline is getting 2.5 V is that row is being written to. If it is not being written to it will be supplied 0 V. The bitline is getting 0 V or 2.5 V depending on if it is “resetting” or “setting” respectively. The selectline is getting 2.5 V or 0 V depending on if it is “resetting” or “setting” respectively.

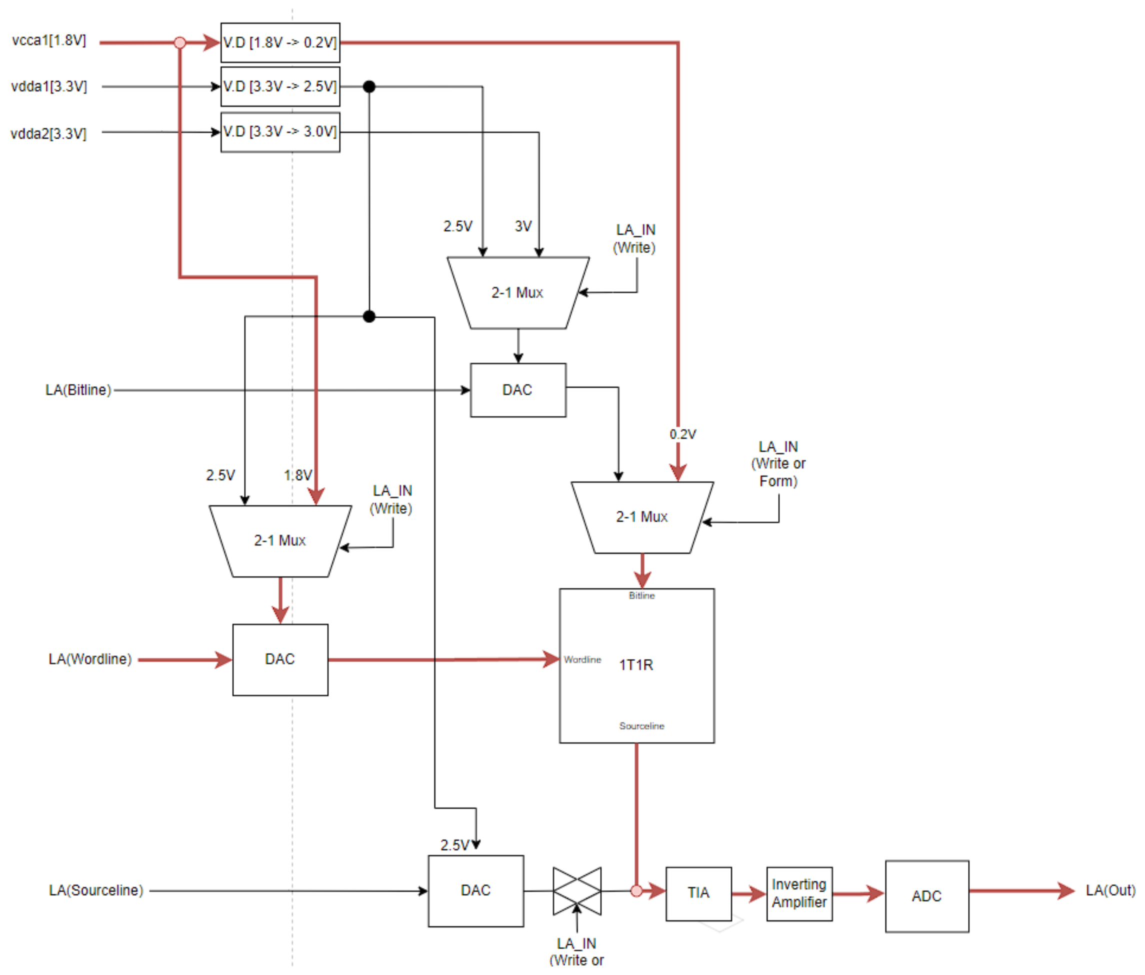


Figure 11: Read/MAC Operation

The figure above shows the read/MAC operation. Specifically for the MAC operation 0.2 V is supplied to the bitline, 1.8 V is supplied to the worldline and the summed current goes through the transimpedance amplifier, inverting amplifier, and ADC to the logic analyzer. During the read operation the worldline is toggled to 0 V or 1.8 V depending on if that row of cells is being read.

3.4.2 Functionality

Once our design is fabricated, a user will have our ASIC chip printed out on a board with numerous controls that are dictated by the Caravel Harness. This board is provided in the fabrication process through the eFabless program. The user will be able to select digital inputs that they want to input into the chip to be computed by the ReRAM crossbar. They may also want to write weights to the ReRAM crossbar, which they will also be able to do with the circuitry around our chip. Once they input the values, they will then be able to read the digital output of these values on one of the output pins of the board. This design will meet all functional requirements of the ReRAM crossbar, as values will be able to be input and weights will be able to be written to the cells.

3.4.3 Areas of Concern and Development

The primary concern of our project is fully instating all of the analog devices that will be used in this project and getting them past precheck. For each analog device, we will have to create a schematic, simulate that schematic, create a layout of that device, and pass DRC and LVS tests on that layout. Once we have that done for each component, we have to create a top schematic and run through everything again. Once that is done, we have to connect everything up in the analog wrapper that is provided by eFabless and try to pass precheck. As can be seen, by this description, there are a lot of steps, and that means a lot of places where things can go wrong or get bottlenecked for some time in our design. We will attempt to address this concern by being proactive and working ahead of schedule. If we run into any trouble, we will use a Slack channel that is used for open-source chip design for help.

3.5 Technology Considerations

Simulators:

- Ngspice
  - Better integrates into eFabless framework
  - Easier to setup
  - Cannot simulate VerilogA well, but has support that can make it easier
- Xyce
  - Harder to setup (need to configure it properly)
  - Is able to simulate VerilogA
  - Requires other software (Trilinos)
  - None of our components other than ReRAM have VerilogA netlists

For our simulator we chose Ngspice because most of our components have Spice netlists, so it is much easier and faster to simulate with it. The only hard part comes with simulating the VerilogA netlist of the ReRAM device. Luckily for us during the time of this project, Ngspice received an update that allowed it to do simulations on VerilogA netlists.

User area communication:

- Logic analyzer
  - Easy to use
  - Easy to implement for hardware
  - Has 128 bits
- GPIO
  - Can communicate with user area using configuration
  - Not very many available communication bits
- Wishbone
  - More flexible, follows a standard
  - Difficult to implement
    - Hard to do a handshake with analog circuitry

We decided to use the logic analyzer because it allows for us to place more components in the user area for our design. We also wanted to reserve the GPIOs for external characterization of our analog devices along with a smaller ReRAM crossbar.

3.6 Design Analysis

Our design is a great prototype for how periphery circuits interact with a ReRAM crossbar. Our design is theoretically able to supply all the needed voltages to the crossbar to perform all of the needed operations. We should also be able to perform both read and MAC operations with this design. This design should be able to scale up to as large as a crossbar as is required, so long as the layout area can fit it. If this design is fabricated we hope that it will show that not only does our design work, but that ReRAM computation can be achieved through fabrication with the SkyWater 130 nm process.

That is not to say that our design is perfect. There could be many improvements made to it. From our work, a future team can make a more complex design, such as having a more complex ADC to remove quantization effects or by including decoders/encoders to do computations at a much faster rate. They may also be able to improve upon the footprint of our design, finding ways to make this process more robust or easier to layout on silicon. Overall, our design is a great first step towards figuring out ReRAM computation through the open-source analog design flow through eFabless using the SkyWater 130 nm process.

3.7 Design Plan

Our design plan is to design and test all analog components individually. Once we have verified that the individual components meet our functional requirements (this includes testing of the extracted post-layout results), we will move to testing the functionality of the individual components when put together. The goal of this is to ensure that the functionality of one device will not interfere with the functionality of another device. If we notice that there are undesired results when we simulate multiple components together, we will tweak the components until we do get the desired results. Once we have verified that all components work together, we plan on creating a top-level layout and hooking up the Caravel harness. Once our circuit is hooked up to the Caravel harness, we will run multiple test cases to ensure the functionality of our circuit still remains intact when hooked up the harness.

Going through this design plan should ensure that we meet all requirements of our project. We will be able to show the functionality of our circuit when not hooked up to the Caravel harness and when it is hooked up to the harness. With our layout, we will ensure that all DRC and LVS tests are passed so that our design will be able to pass precheck. With all of these parts of our design plan, we should be able to meet the goals of this project and successfully tape-out or design.

4 Implementation

4.1 Implementation of Devices

In order to implement our design we must go through the open-source analog design flow that is required by eFabless. This took quite a while to figure out as there is little to no documentation on it. However, the basic tool flow is shown in the figure below.

Tools	Function
Xschem	Schematic Editor
Ngspice	Spice Circuit Simulator
Magic	Layout Editor & DRC
Netgen	LVS

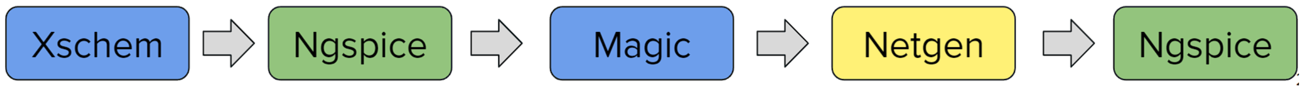


Figure 12: Tool Flow

In the analog design flow, we start off in Xschem which is a schematic editor. In this tool, this is where we create our schematics at a transistor level. These transistor models are provided to us through the SkyWater 130 nm process. Once we have created our schematic and sized the devices in it we move on to the next tool which is Ngspice. Ngspice allows us to simulate the netlists of our schematics. This tool allows us to run different simulation tests on our designs. We will then plot the waveforms and see if the results match our expectations.

Once we have verified that our schematic behaves the way we expect it to, we then move over to Magic which is the layout editor. This will allow us to layout our design and how it would actually be fabricated on a chip. In this tool, we create our layouts and run a DRC (design rule check) to ensure that our design can actually be fabricated without failing any fabrication rules (such as two nets being too close together). From Magic, we then use Netgen to compare the netlists generated by our schematic level design and our layout level design in an LVS (layout vs. schematic) check.

If this passes, we then extract an R+C version of the netlist generated from the layout. This extracted netlist attempts to incorporate the different resistances and capacitances that will exist in a design once fabricated. We then pop this back into Ngspice and run a post-layout simulation on it to see if the parasitics that were created hamper the behavior of the design. If after the post-layout simulation, the design still behaves as expected, then that design is finally implemented through the open-source analog design flow. A theoretical design may have to go through this entire process again if it does not behave correctly when implemented with other designs.

If this sounds like a lot of steps, that is because it is. It should be noted that for every design we did we had to go through this process at least once. The designs that we pushed through this design flow include:

- Inverter
- Buffer
- Transmission Gate
- 2-1 MUX
- 4-1 MUX (not used in final design)
- 1-Stage Operational Amplifier
- 2-Stage Operational Amplifier (not used in final design)
- 1-bit ADC
- 3-bit ADC (not used in final design)
- Transimpedance Amplifier(s)
- Inverting Amplifier
- Level Shifters
- ReRAM Device
- 1T1R cell

Some of these designs we did not end up using for varying reasons, such as realizing that we didn't end up needing them or that they were not fully implemented in time once system level design began.

Once we had these devices created through this design flow, we had to put them together so that we could hook them up to the Caravel Harness in layout. A rough concept sketch of how we would do this is shown in the figure below.

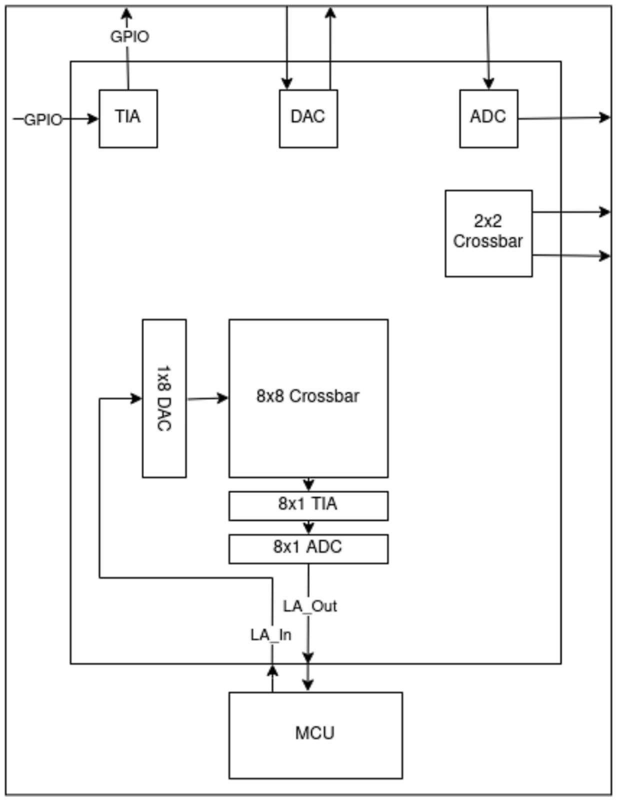


Figure 13: Concept Sketch of Desing Hooked up To Caravel Harness

We would hook up our compute crossbar to the logic analyzer pins on the Caravel Harness and we would hook up individual components/a smaller crossbar to the GPIOs for external characterization. We also did this as a risk mitigation method in case our design didn't work once fabricated, we would at least have external circuits we could test to ensure that actual designs can be fabricated using the SkyWater 130 nm process. We also included a smaller 2x2 so we could do our own individual characterization of it.

To create this layout, we created an 8-line version of the worldline, bitline, select line, and the 8x8 ReRAM crossbar independently. We then hooked up all of these components together before finally hooking all of them up to the Caravel harness. We also created a second version of the compute crossbar for a different read/MAC voltage to be placed in the layout as well. We did this so that we could further risk mitigate our possible fabricated design. The only difference between the two was the 8-line selectline output that had a different-sized transimpedance amplifier to account for the different read/MAC voltage. Once crossbar was for a 0.2 V read/MAC and the other was for a 0.4 V read/MAC.

After we had hooked both of these up we hooked up all of the individual components on the side and connected them up to the GPIOs. Once this was all finalized we had successfully created the layout of our final design. The schematic level hook-up of our final design is shown in the figure below (you may have to zoom in on it as it is a massive picture).

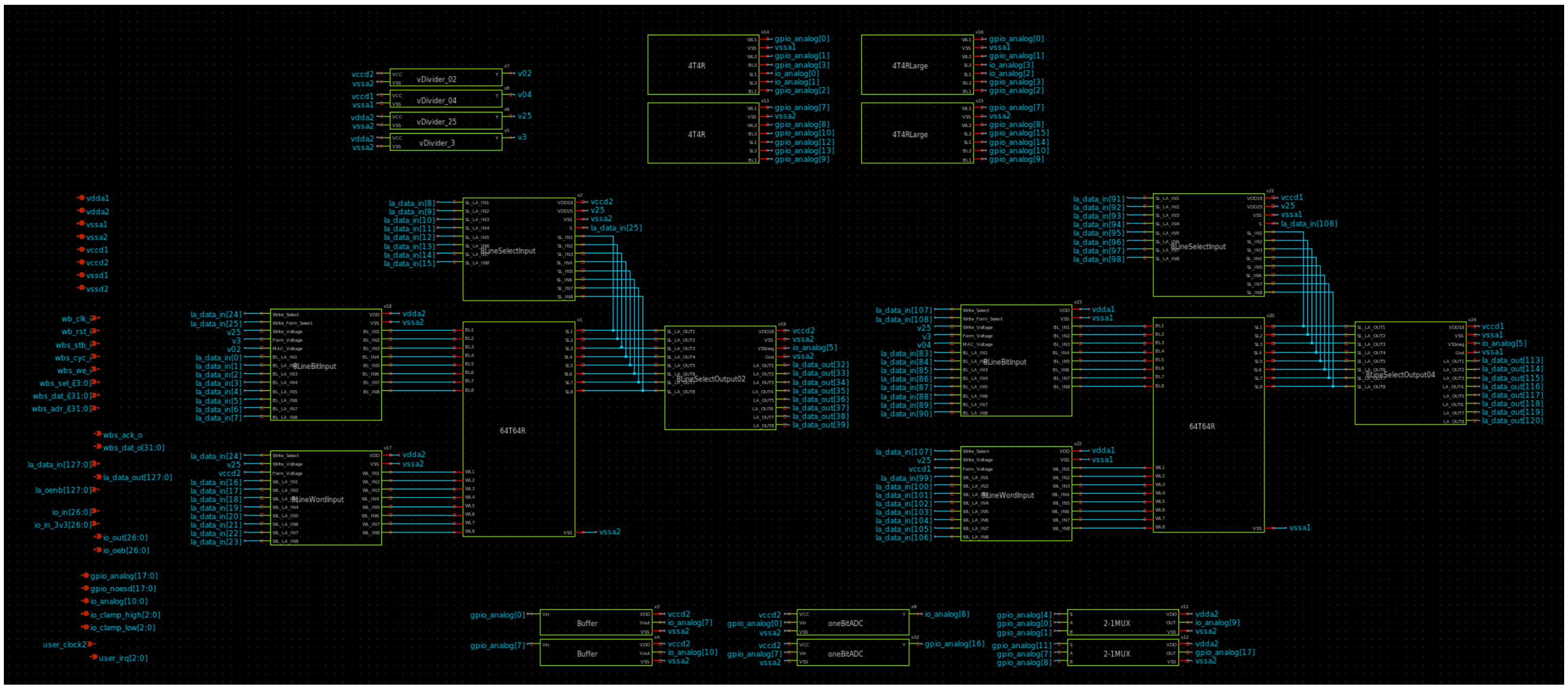


Figure 14: Schematic Level View of Hook-up to Caravel Harness

The actual layout of our design with all of the components can be found in the figure on the following page.

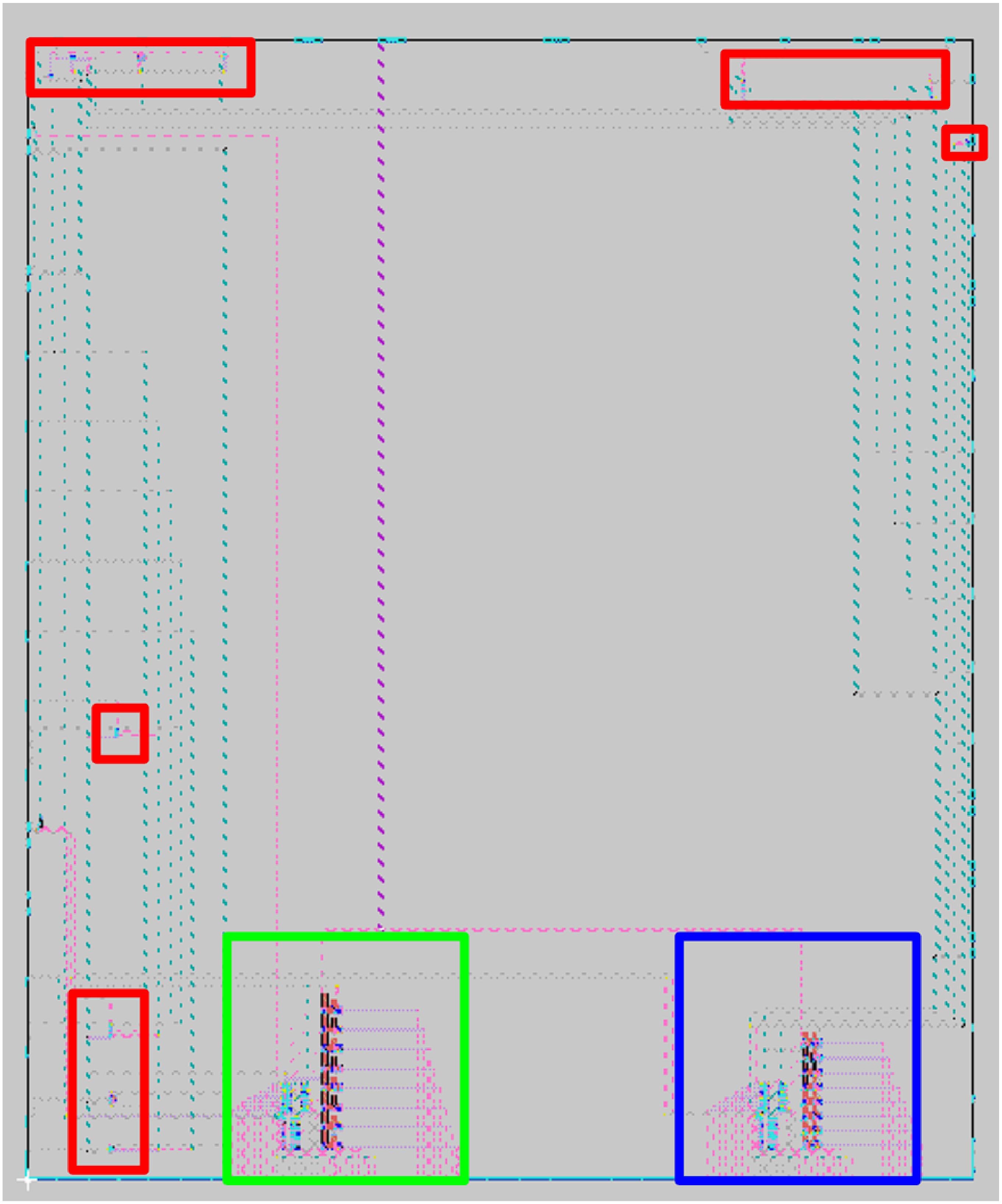


Figure 15: Layout Level View of Hook-up to Caravel Harness

The layout area is massive, so it is hard to grab an all-encompassing picture that highlights all of the devices and connections. The stuff in the green box is the 0.2 V compute crossbar, in the blue box is the 0.4 compute crossbar, and the red boxes show where the external circuitry is that is connected to the GPIOs.

4.2 Implementation Issues

Unfortunately, with our final design, we were not able to pass an LVS check. There are two reasons for this. The first reason is that we connected the substrate of our PMOS devices to multiple different pins. This is an issue because this substrate is a singular layer meaning that it can only support one voltage level. When we have multiple different pins connected to it, the program assumes that they will be connected to different voltages, so it does not match the schematic netlist. The way to fix this is to isolate the substrate of the PMOS devices so that each device can get its respective voltage/ground. However, we were not able to figure this out in time, but it does look like it is possible with the SkyWater 130 nm process.

The other reason that it does not pass LVS is because of the ReRAM device itself. As this whole process is essentially in "alpha," things are still being worked on and developed. One of those things is the ReRAM device. In its current state, it is unable to pass an LVS check due to netlist errors that arise from Xschem and Magic. This is due to the developer of Xschem having to write special model statements in the netlist of the ReRAM cell to get it to function properly with Ngspice. However, these model statements do not get extracted to Magic, and so LVS fails as it can not properly compare the two netlists.

We also attempted to take a 1T1R cell through precheck to see if that is possible, but as of the time of writing this, it is not. This is because of the updated model of the ReRAM not being able to pass LVS, and thus it fails precheck with an error. We have been in contact with the developers of these tools and the people at eFabless themselves and they are currently working on a fix.

5 Testing

5.1 Unit Testing

Our design will consist of many units. The main units will be the analog circuitry, such as the DACs, ReRAM cells (1T1R), transimpedance amplifiers, and ADCs. These units will be thoroughly tested using the open-source tools we have available to us. For each device, we will have to make a schematic of it using XSchem. From that schematic, we will then create a testbench in Xschem to observe its behavior. These testbenches will be simulated using Ngspice or Xcye. Once we have verified if that device meets our design requirements and expectations, we will then have a layout created for it using Magic. We will then obtain the post-layout parasitics using Magic and simulate that parasitic netlist using Ngspice. An example of what this would look like is shown in the figure below for the inverter design.

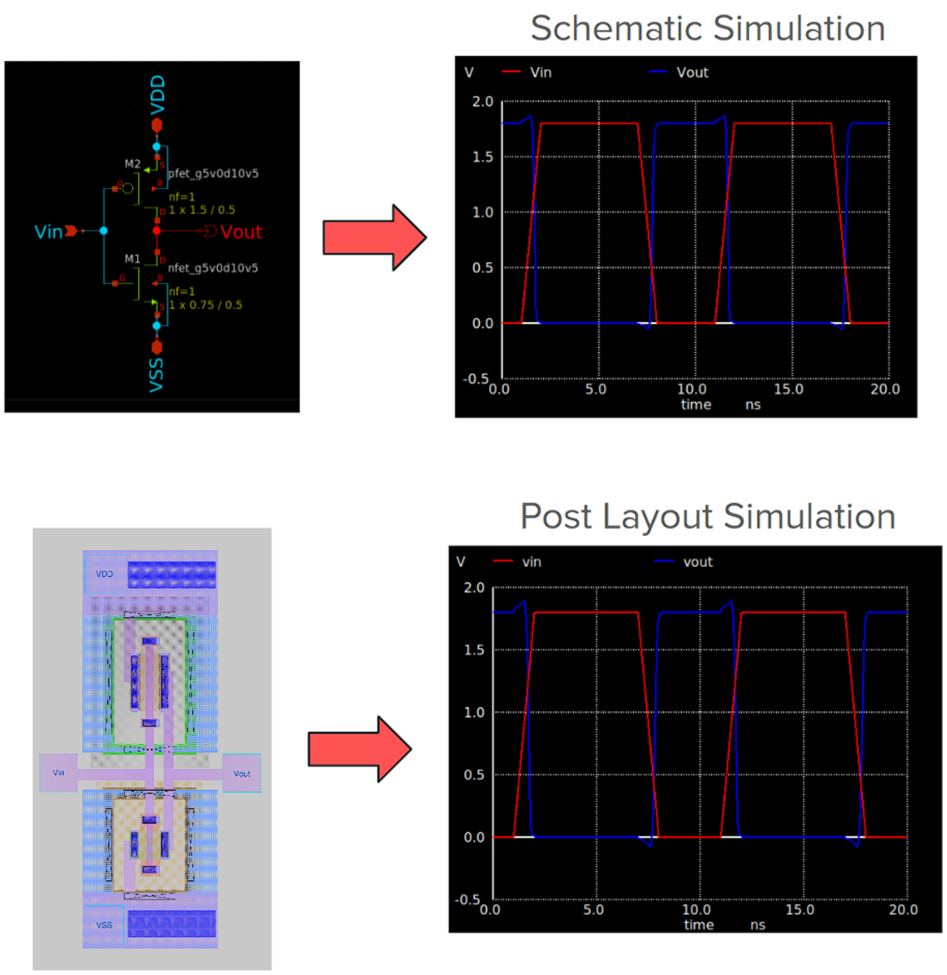


Figure 16: Example Unit Testing on Inverter

5.2 Interface Testing

The interface between each of our components will be wires. The number and name of the wires will vary based on what is needed from them. In our project, we have a user area which needs to be communicated with via a RISCV CPU. This communication will be done over a configurable 128bit logic analyzer. Whether each bit is an input or output is configurable.

The composition of multiple units is being tested the same way as individual components, which is creating testbenches in XScehm, simulating through Ngspice, creating a layout using Magic, verifying LVS, and extracting parasitics to rerun simulations with.

5.3 Integration Testing

The main integration path in our design comes from the connection of all of our analog components to the outside Caravel harness. This path is fairly linear when we are computing data. Digital data will be entered into our circuit from the Carvel harness, where our chip will do computations in the analog domain before returning a digital value back to the harness. Before hooking up our circuit to the Caravel harness, we will ensure that the analog circuitry is behaving as expected. This will be done by creating a top-level schematic of all the components connected together in Xschem. We will then create a testbench that simulates the behavior of this circuit. This testbench will simulate the conditions that the circuit will be in when connected to the Caravel harness. We will run these simulations using Ngspice. Once this is done, we will use Magic to connect together all of the components and extract post-layout parasitics. We will then simulate these parasitics, and if the results are still acceptable, we will move on to connect our circuit to the outside harness. We will run further testing on this to ensure that everything still behaves as expected.

5.4 System Testing

This part is awkward; unlike with fully digital designs, we are unable to fully verify our product. We must test in two separate parts. We currently have a fully functional digital implementation of our project, which has the same behavior as the ReRAM crossbar, so it demonstrates the same functionality in digital simulation. The fully digital simulation allows us to determine if our port mappings are going to work, along with if our drivers will work.

For the second part, we need to determine if our analog ReRAM crossbar component's behavior matches our digital implementation. This must be done through analog simulations since it uses analog components. Our strategy here consists of writing many top-level tests to ensure all of the operations are working.

5.5 Regression Testing

The testing will be very iterative to ensure that everything works properly together. In our design, we need to make sure that each unit works not only independently but they all work together. For this regression testing, we will first build and simulate a schematic of each individual unit to ensure it works as intended using Xschem. Next, we will hook up two units together that will be connected in our final design. For example, we will hook up the 1-bit DAC with the 1T1R cell. We will then create a testbench and run simulations using Ngspice to view how they behave together. If we observe that they are not behaving as expected, we will have to tweak the individual components and run simulations until they do behave as expected. Once we have verified that the two components work together, we would then connect one more component and run through the same process again. We would repeat this process until all components have been connected and simulated with acceptable results. Running through the testing this way, we can ensure that all components will work together, but it will also allow us to have an easier time troubleshooting if something deviates from the expected behavior, as we will be able to identify the component that is causing the problem.

5.6 Acceptance Testing

Our design will be tested for non-functional acceptability through the precheck process the eFabless uses to verify that our design meets certain requirements. Some of these prechecks include DRC and LVS checks. This precheck essentially runs multiple DRC checks to ensure that our layout does not violate any rules and is able to be fabricated on silicon. For the functionality, we will have to run simulations and test cases on our design to verify that it behaves as expected. These simulations will be done using Verilog to simulate how our circuit will operate within the Caravel harness. From these simulations, we will obtain waveforms that will help verify the functionality of our chip.

5.7 Issues With ReRAM Simulation

As was mentioned earlier, the current ReRAM model is currently still being developed and tested in this open-source design flow. As a result of this, simulations are currently not able to be done with anything other than the model by itself - and even then, it is not reliable. So when we attempted to do simulations on a 1T1R cell, the simulation failed due to convergence issues. We have informed the eFabless team, and they seem to be working on making it better for the future. In the meantime, we still had to try to do simulations. Luckily for us, the SkyWater documentation has a section on ReRAM characteristics and it shows a graph of the ReRAMs conductance during its different states. This figure can be seen in the figure below.

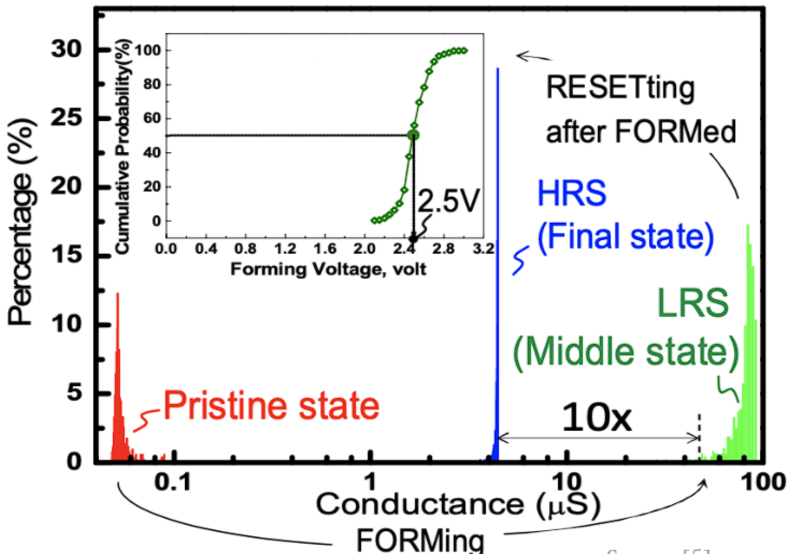


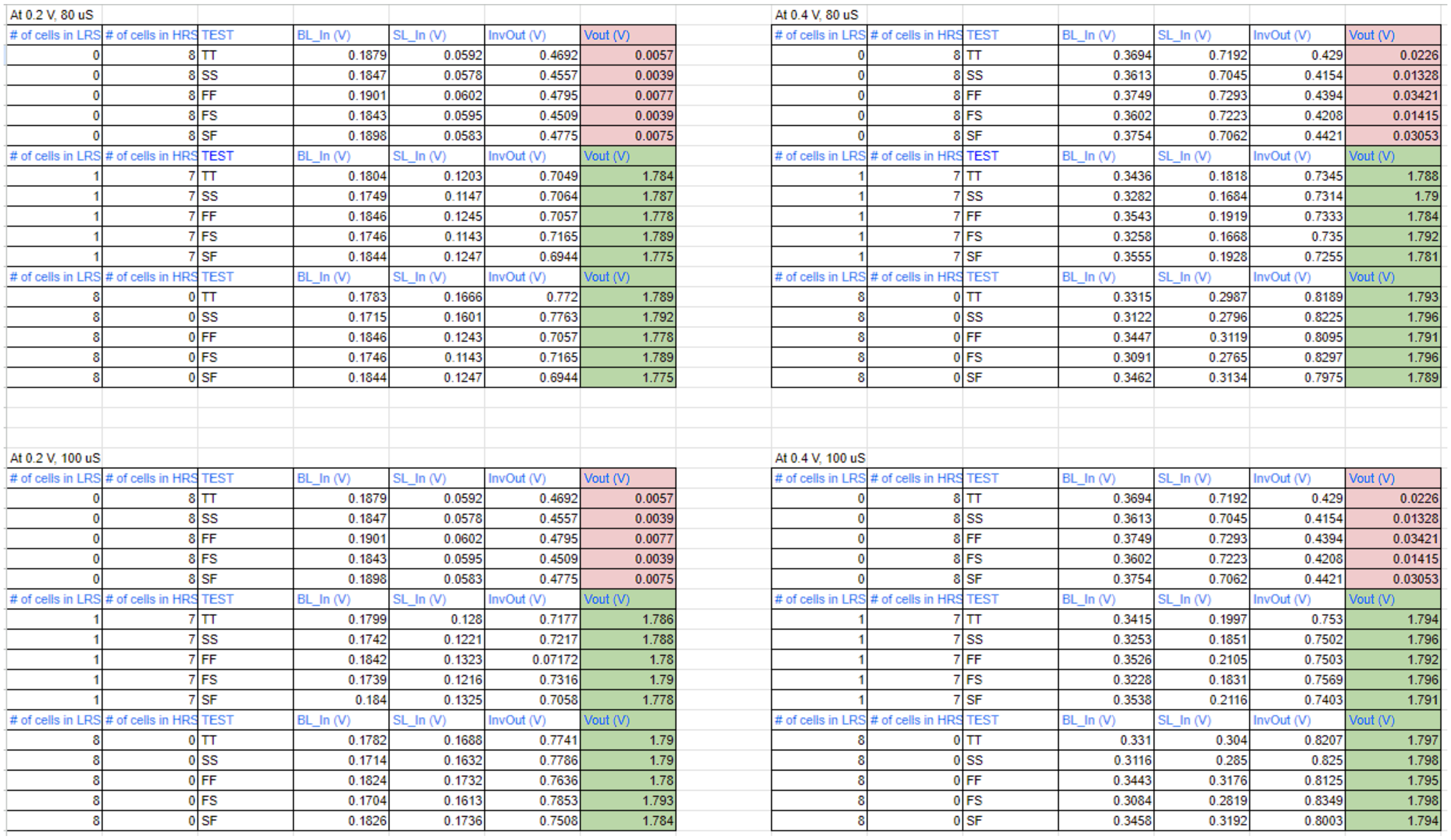
Figure 17: ReRAM Conductance Table [4]

Using this table we were able to simulate the different states of the ReRAM with a regular resistor by converting the conductance to a resistance and sizing a transistor to match that.

5.8 Results

We were able to test and verify that all of our individual analog circuitry worked as intended. We were also able to push all of them successfully through the analog design flow (with the exception of ReRAM stuff not passing LVS). Once we had done that, we hooked up our design in a top-level schematic and ran simulations on that. For form and write operations we were able to verify that the needed input voltages could get to where they needed to get to. There may be some issues with the loading effects of the resistances impacting the voltage that arrives. It did not seem to be a huge issue for the lower voltages, but as the voltage increased, the more the difference in expected vs. actual voltage differed.

For read/MAC operations, we had to do some estimation of the ReRAM as was explained in section 5.7. As a result of this, we also did some error testing by doing “minimum conductance” and “maximum conductance” tests. Since the LRS varies quite a bit in conductance, we grabbed a minimum conductance of 80 uS and a maximum conductance of 100 uS. Along with this error testing, we also did corner testing to see how/if that affected the results of our computations. Finally, we did all of these tests for both the 0.2 V and 0.4 V versions of the compute crossbar. The figure below shows some tables of the results of these tests.



At 0.2 V, 80 uS

# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
0	8	TT	0.1879	0.0592	0.4692	0.0057
0	8	SS	0.1847	0.0578	0.4557	0.0039
0	8	FF	0.1901	0.0602	0.4795	0.0077
0	8	FS	0.1843	0.0595	0.4509	0.0039
0	8	SF	0.1898	0.0583	0.4775	0.0075
# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
1	7	TT	0.1804	0.1203	0.7049	1.784
1	7	SS	0.1749	0.1147	0.7064	1.787
1	7	FF	0.1846	0.1245	0.7057	1.778
1	7	FS	0.1746	0.1143	0.7165	1.789
1	7	SF	0.1844	0.1247	0.6944	1.775
# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
8	0	TT	0.1783	0.1666	0.772	1.789
8	0	SS	0.1715	0.1601	0.7763	1.792
8	0	FF	0.1846	0.1243	0.7057	1.778
8	0	FS	0.1746	0.1143	0.7165	1.789
8	0	SF	0.1844	0.1247	0.6944	1.775

At 0.4 V, 80 uS

# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
0	8	TT	0.3694	0.7192	0.429	0.0226
0	8	SS	0.3613	0.7045	0.4154	0.01328
0	8	FF	0.3749	0.7293	0.4394	0.03421
0	8	FS	0.3602	0.7223	0.4208	0.01415
0	8	SF	0.3754	0.7062	0.4421	0.03053
# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
1	7	TT	0.3436	0.1818	0.7345	1.788
1	7	SS	0.3282	0.1684	0.7314	1.79
1	7	FF	0.3543	0.1919	0.7333	1.784
1	7	FS	0.3258	0.1668	0.735	1.792
1	7	SF	0.3555	0.1928	0.7255	1.781
# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
8	0	TT	0.3315	0.2987	0.8189	1.793
8	0	SS	0.3122	0.2796	0.8225	1.796
8	0	FF	0.3447	0.3119	0.8095	1.791
8	0	FS	0.3091	0.2765	0.8297	1.796
8	0	SF	0.3462	0.3134	0.7975	1.789

At 0.2 V, 100 uS

# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
0	8	TT	0.1879	0.0592	0.4692	0.0057
0	8	SS	0.1847	0.0578	0.4557	0.0039
0	8	FF	0.1901	0.0602	0.4795	0.0077
0	8	FS	0.1843	0.0595	0.4509	0.0039
0	8	SF	0.1898	0.0583	0.4775	0.0075
# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
1	7	TT	0.1799	0.128	0.7177	1.786
1	7	SS	0.1742	0.1221	0.7217	1.788
1	7	FF	0.1842	0.1323	0.07172	1.78
1	7	FS	0.1739	0.1216	0.7316	1.79
1	7	SF	0.184	0.1325	0.7058	1.778
# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
8	0	TT	0.1782	0.1688	0.7741	1.79
8	0	SS	0.1714	0.1632	0.7786	1.79
8	0	FF	0.1824	0.1732	0.7636	1.78
8	0	FS	0.1704	0.1613	0.7853	1.793
8	0	SF	0.1826	0.1736	0.7508	1.784

At 0.4 V, 100 uS

# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
0	8	TT	0.3694	0.7192	0.429	0.0226
0	8	SS	0.3613	0.7045	0.4154	0.01328
0	8	FF	0.3749	0.7293	0.4394	0.03421
0	8	FS	0.3602	0.7223	0.4208	0.01415
0	8	SF	0.3754	0.7062	0.4421	0.03053
# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
1	7	TT	0.3415	0.1997	0.753	1.794
1	7	SS	0.3253	0.1851	0.7502	1.796
1	7	FF	0.3526	0.2105	0.7503	1.792
1	7	FS	0.3228	0.1831	0.7569	1.796
1	7	SF	0.3538	0.2116	0.7403	1.791
# of cells in LRS	# of cells in HRS	TEST	BL_In (V)	SL_In (V)	InvOut (V)	Vout (V)
8	0	TT	0.331	0.304	0.8207	1.797
8	0	SS	0.3116	0.285	0.825	1.798
8	0	FF	0.3443	0.3176	0.8125	1.795
8	0	FS	0.3084	0.2819	0.8349	1.798
8	0	SF	0.3458	0.3192	0.8003	1.794

Figure 18: Top-Level Simulation Results

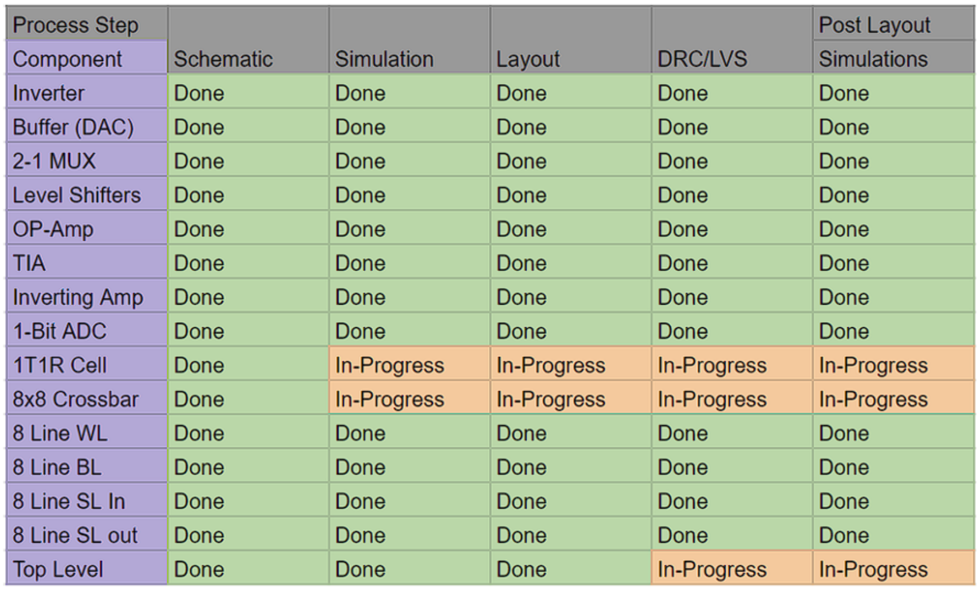
From the top-level simulations, we were able to verify that our crossbar could perform form, write, read, and MAC operations on both the 0.2 V and 0.4 V compute crossbar designs. We verified this with extensive corner and error testing.

6 Project Outcomes

We have accomplished a lot with this project and have many different deliverables. The sections below will quickly highlight the outcome of each of them.

6.1 Final 8x8 Compute Crossbar

We were successfully able to push all individual components through the open-source analog design flow and verify their post-layout functionality. We were able to successfully do all operations on the crossbar that we were tasked with doing - form, write, read, and MAC. We were able to create a final layout of our design as well as hook up the two crossbars to the logic analyzer and external analog circuitry out to GPIOs for characterization. Unfortunately we were unable to pass precheck due to having outlying issues with the substrate connections on our PMOS devices, but also the ReRAM model itself still being in development. A status matrix of the work accomplished for our components can be seen in the figure below.



Process Step					Post Layout
Component	Schematic	Simulation	Layout	DRC/LVS	Simulations
Inverter	Done	Done	Done	Done	Done
Buffer (DAC)	Done	Done	Done	Done	Done
2-1 MUX	Done	Done	Done	Done	Done
Level Shifters	Done	Done	Done	Done	Done
OP-Amp	Done	Done	Done	Done	Done
TIA	Done	Done	Done	Done	Done
Inverting Amp	Done	Done	Done	Done	Done
1-Bit ADC	Done	Done	Done	Done	Done
1T1R Cell	Done	In-Progress	In-Progress	In-Progress	In-Progress
8x8 Crossbar	Done	In-Progress	In-Progress	In-Progress	In-Progress
8 Line WL	Done	Done	Done	Done	Done
8 Line BL	Done	Done	Done	Done	Done
8 Line SL In	Done	Done	Done	Done	Done
8 Line SL out	Done	Done	Done	Done	Done
Top Level	Done	Done	Done	In-Progress	In-Progress

Figure 19: Status Matrix of Components

6.2 GitHub Repository

All of our components and their files are stored in a GitHub repository. These files include the schematics and layouts of each component. We also have the final top-level schematic and layout included in the repository as well. We also included all of the digital work that was done throughout the course of this project, such as the digital model, drivers, and C code. The link to the repository can be found here: https://github.com/Jthater1/ReRAM_Crossbar_Project

6.3 Post-Fabrication Bring-up Plan

A post-fabrication bring-up plan was created that detailed all of the steps that would be completed if our chip was fabricated once the outstanding issues were resolved. It includes how to test all of the external circuitry as well as how to test the compute crossbars. This bring-up plan can be found on our senior design website, but it is also included in appendix 8.1.4.

6.4 Tutorial Documentation on Analog Design Flow + Other Documentation

While working on this project, we documented how we went through the open-source analog design flow. This work was compiled and edited into a tutorial document for the open-source analog design flow. This documentation is very detailed but thorough on all steps of the design process. The end result was a document that was 80 pages long. The documentation included sections on:

- Intro
- Environment setup
- Tool Installation
- Tool usage with example "hello world project" with inverter
- Integration of tools with SkyWater 130 nm Process
- Interfacing analog design with Caravel Harness

Along with this additional documentation was created for how to run precheck on an analog design along with documentation on the digital behavioral model that was created for this project. This documentation can be found on our project website.

7 Closing Material

7.1 Discussion

We were able to design a ReRAM crossbar that was able to perform form, write, read/MAC operations at a schematic level. We were also able to create a top-level layout that hooked two different compute crossbars to the Caravel Harness. In this Caravel Harness, we were able to hook up external analog circuitry that would be used for characterization if fabricated. Unfortunately, we were not able to have our design pass precheck and be ready for fabrication partially due to issues in our layout, but also outstanding issues with the ReRAM model currently used in this design flow.

We were able to create detailed tutorial documentation that outlined the entire open-source analog design flow. This documentation ended up being very long, but it will be very beneficial to people who utilize it in the future as it highlights all of the needed steps for analog design through eFabless. We struggled for a very long time - multiple months and hours and hours of troubleshooting - to figure out the design flow. So, by having it fully documented we hope to alleviate all of that possible frustration for future people.

The work we accomplished, plus all of the documentation we created should allow for a future senior design team to hit the ground running to fix our design and even make improvements to it.

7.2 Conclusion

While we may not have been able to have our design ready for fabrication we still accomplished a lot of very good work. We were able to figure out the open-source analog design flow and thoroughly document it for future use here at ISU. We also were able to show that our design worked at a schematic simulation level and even created a layout of how all of our designs would be hooked up to the Caravel Harness. The work we accomplished, plus all of the documentation we created it should allow for a future senior design team to hit the ground running to fix our design and even make improvements to it.

7.3 References

[1] A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea (South), 2016, pp. 14-26, doi: 10.1109/ISCA.2016.12. https://ieeexplore.ieee.org/document/7551379

[2] T. Chou, W. Tang, J. Botimer, and Z. Zhang, Electrical Engineering and computer science, https://web.eecs.umich.edu/~zhengya/papers/chou_micro19.pdf.

[3] T. Boucard, "Spotlight on Resistive Ram (reram) – an interview with Weebit Nano," Yole Group, https://www.yolegroup.com/player-interviews/spotlight-on-resistive-ram-reram -an-interview-with-weebit-nano/.

[4] "Sky130_fd_pr_reram - sky130 reram (Skywater provided)¶," sky130_fd_pr_reram - SKY130 ReRAM (SkyWater Provided) - SkyWater SKY130 PDK 0.0.0-22-g72df095 documentation. [Online]. Available: https://sky130-fd-pr-reram.readthedocs.io/en/latest/technology_specifications

html#forming

8 Appendices

8.1 Appendix 1 - Operation Manual

For the operation of our project there are two parts. The first part is getting the pre-fabrication environment/files set up. The second part is the post-fabrication bring-up plan once our design has passed precheck and is sent back to us.

First, we will detail out the pre-fabrication stuff. Most of this information is pulled directly from the tutorial analog design flow documentation was created. To see the full version of this document check out our senior design website. Here, we will just include the setup of the environment and tools that are required to start working on the pre-fabrication side of our project.

8.1.1 Pre-Fabrication Environment Setup

This will provide a general guide on how to set up a virtual environment that will be able to house all the tools that are needed for the open-source analog design flow.

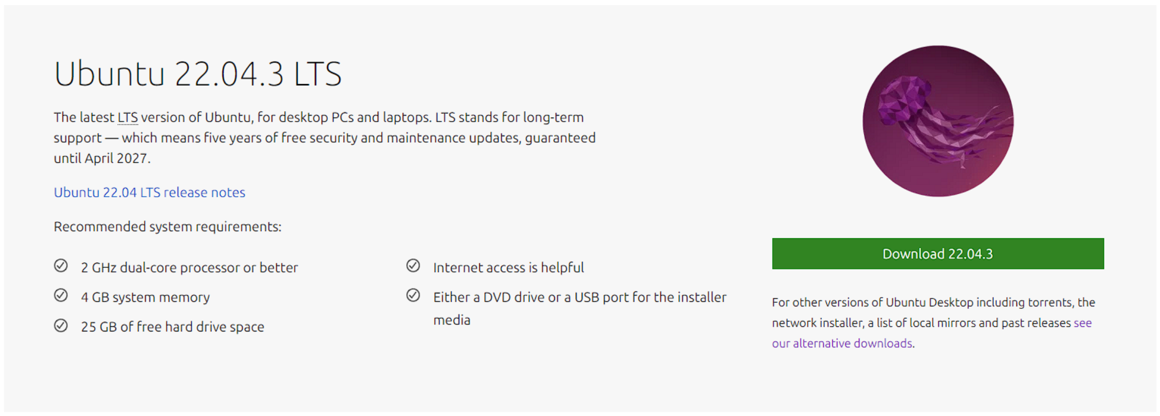
To start things things off, in order to do the analog design flow, it is *heavily recommended* to use a Linux system or MAC system. It may be possible to use a Windows operating system, but certain software may be out of date or not available at all.

The rest of this section will cover how to download and install a Ubuntu VM with the specific hardware requirements needed for this analog design flow. If you do not plan to use an Ubuntu VM for your environment, then you can safely skip this section. However, be sure that your environment will be able to run all the tools that are needed.

It is recommended to use Virtual Box for your VM. Below is the link to download Virtual Box: https://www.virtualbox.org/wiki/Downloads

Once you have Virtual Box downloaded, it is time to set up the Ubuntu VM. First, head to the webpage linked below. Ubuntu Desktop Virtual Machine Set Up:
https://ubuntu.com/download/desktop

Once there, download an Ubuntu image. It shouldn't matter which one you choose; just make sure it is past Ubuntu 20 (which all of them on the download page should be anyway). For this example, Ubuntu 22.04.3 was used, which can be seen below.



Ubuntu 22.04.3 LTS

The latest LTS version of Ubuntu, for desktop PCs and laptops. LTS stands for long-term support — which means five years of free security and maintenance updates, guaranteed until April 2027.

Ubuntu 22.04 LTS release notes

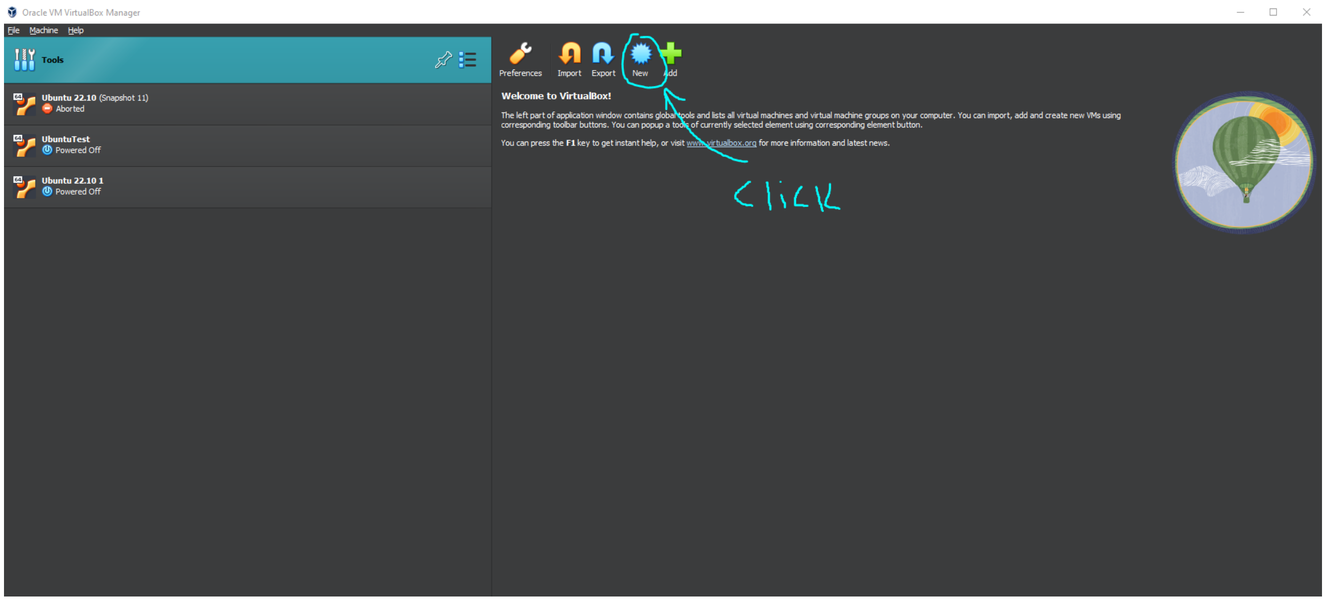
Recommended system requirements:

- 2 GHz dual-core processor or better
- 4 GB system memory
- 25 GB of free hard drive space
- Internet access is helpful
- Either a DVD drive or a USB port for the installer media

For other versions of Ubuntu Desktop including torrents, the network installer, a list of local mirrors and past releases see our alternative downloads.

Figure 20: Ubuntu Image

This ISO is a few GB, so it may take a bit to install. Once it is installed, you can head over to Virtual Box. Once in Virtual Box, you should see a screen similar to the one seen below. To begin setting up the virtual machine, click on "New" as seen in the figure below.



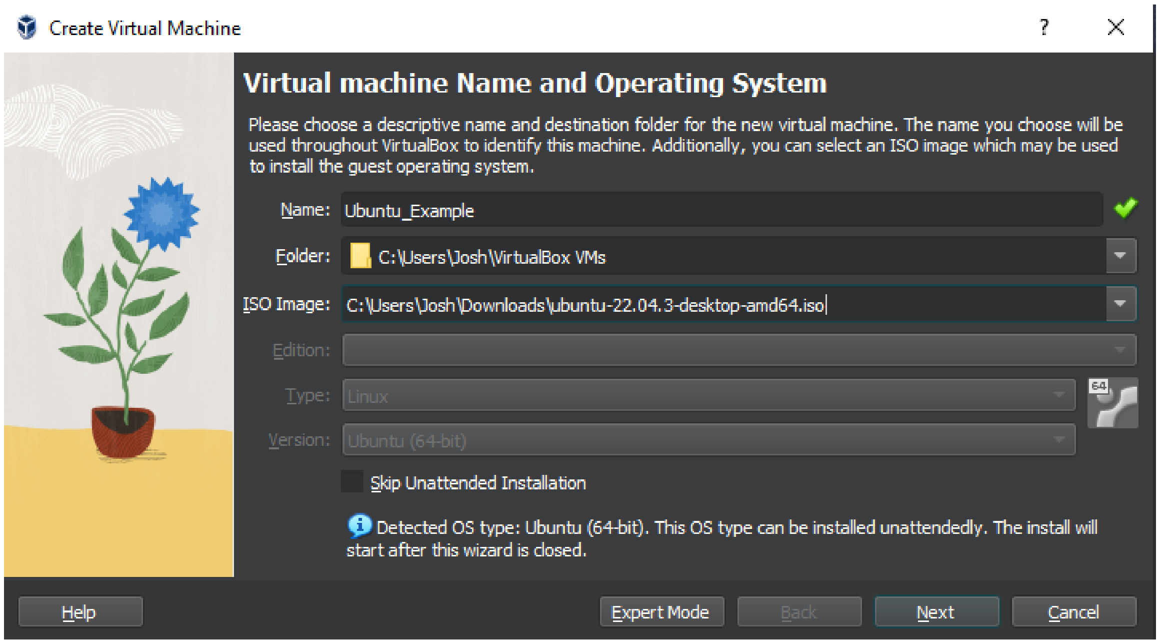
Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the F1 key to get instant help, or visit www.virtualbox.org for more information and latest news.

Figure 21: Virtual Box Setup

Once that is done, simply give a name to your virtual machine and select the ISO image you just downloaded (the ISO should be in your downloads directory). If you did all that correctly, it should look something like this.



Virtual machine Name and Operating System

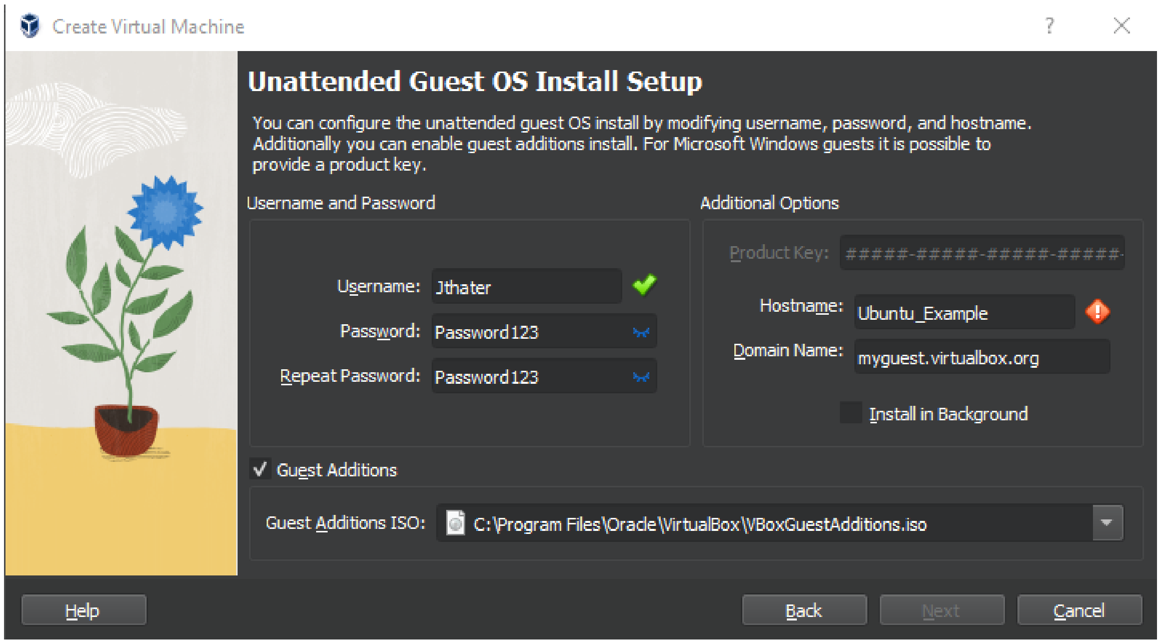
Please choose a descriptive name and destination folder for the new virtual machine. The name you choose will be used throughout VirtualBox to identify this machine. Additionally, you can select an ISO image which may be used to install the guest operating system.

Name: Ubuntu_Example
Folder: C:\Users\Josh\VirtualBox VMs
ISO Image: C:\Users\Josh\Downloads\ubuntu-22.04.3-desktop-amd64.iso
Edition:
Type: Linux
Version: Ubuntu (64-bit)
Skip Unattended Installation

Detected OS type: Ubuntu (64-bit). This OS type can be installed unattendedly. The install will start after this wizard is closed.

Figure 22: VM Name

Once this is done, click next. On the following screen, you can change the username and password of your machine. It is recommended that you change these settings so that you can get sudo access on your machine. Don't worry about changing the hostname or domain name. Finally, on this screen, make sure to select the box for "Guest Additions"; this adds a bunch of quality-of-life features, such as dynamic screen resolution. Once this is all done, it should look something like this.


Create Virtual Machine

Unattended Guest OS Install Setup

You can configure the unattended guest OS install by modifying username, password, and hostname. Additionally you can enable guest additions install. For Microsoft Windows guests it is possible to provide a product key.

Username and Password

Username: Jthater
Password: Password123
Repeat Password: Password123

Additional Options

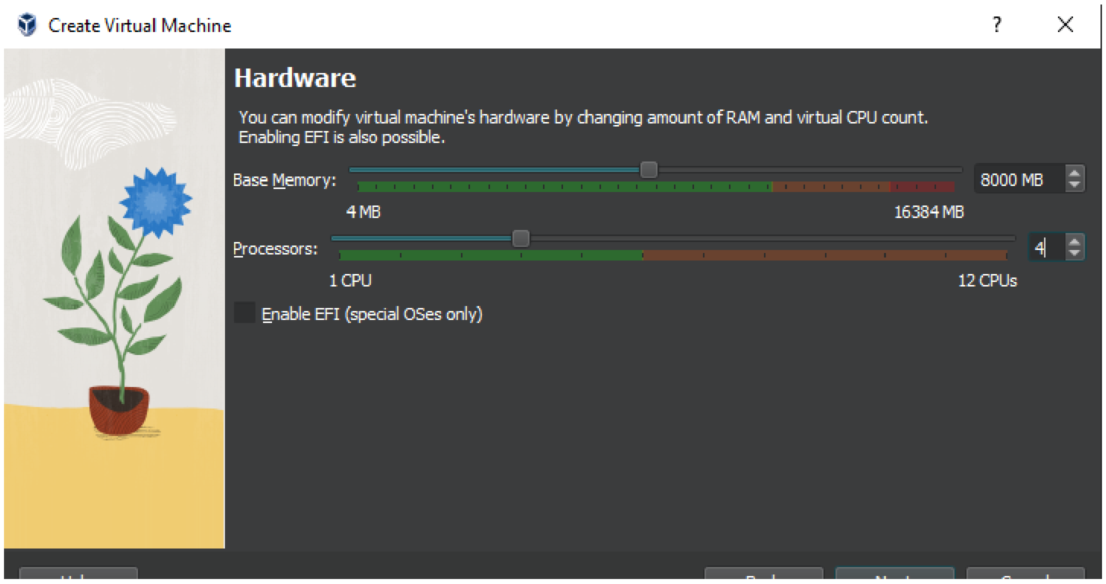
Product Key: #####-#####-#####-#####-
Hostname: Ubuntu_Example
Domain Name: myguest.virtualbox.org
Install in Background

✓ Guest Additions

Guest Additions ISO: C:\Program Files\Oracle\VirtualBox\VBoxGuestAdditions.iso

Figure 23: VM User Setup

Once you click next, you will be greeted with resource allocation. This is gonna be up to your system, but it is recommended to allocate around 8GB of RAM and 4 CPUs. However, it is more important to stay within the green so that your host machine will still be stable. This is what my settings look like.


Create Virtual Machine

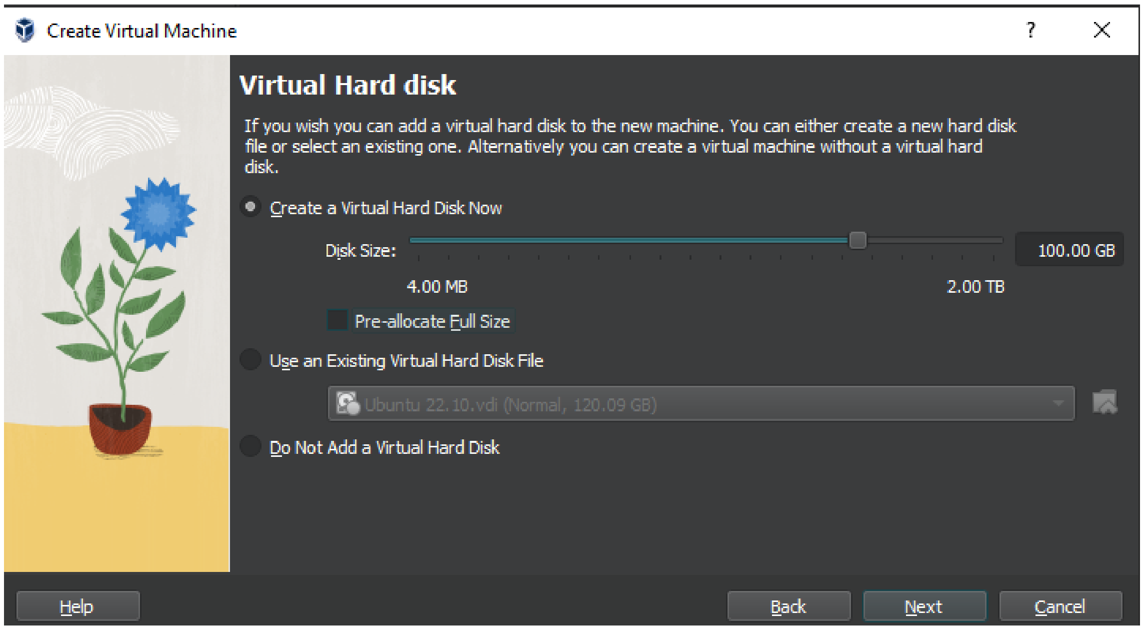
Hardware

You can modify virtual machine's hardware by changing amount of RAM and virtual CPU count. Enabling EFI is also possible.

Base Memory: 8000 MB
4 MB — 16384 MB
Processors: 4
1 CPU — 12 CPUs
Enable EFI (special OSes only)

Figure 24: VM Hardware Allocation

Once you click next, you will be greeted with setting up the virtual hard disk. For the environment that will be set up, **a minimum of 75 GB is recommended**, but depending on what you end up doing, you may want up to 120 GB allocated. For this example, 100 GB of space was allocated. You may also opt to pre-allocate the full size, but for this example it is turned off.



Virtual Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select an existing one. Alternatively you can create a virtual machine without a virtual hard disk.

● Create a Virtual Hard Disk Now

Disk Size: 100.00 GB

4.00 MB — 2.00 TB

☐ Pre-allocate Full Size

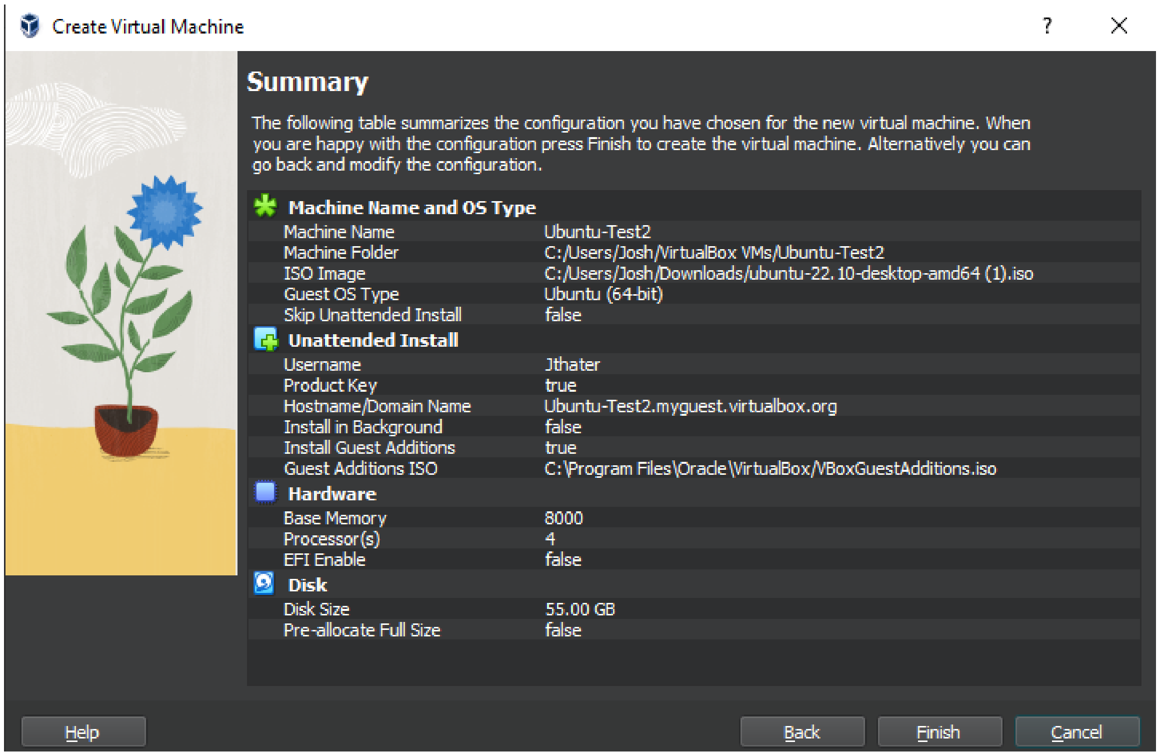
Use an Existing Virtual Hard Disk File

Ubuntu 22.10.vdi (Normal, 120.09 GB)

Do Not Add a Virtual Hard Disk

Figure 25: VM Hard Disk Allocation

Once all of that is done, you will see a summary page. Ensure that everything is set up correctly, and click "Finish."



Summary

The following table summarizes the configuration you have chosen for the new virtual machine. When you are happy with the configuration press Finish to create the virtual machine. Alternatively you can go back and modify the configuration.

Machine Name and OS Type	
Machine Name	Ubuntu-Test2
Machine Folder	C:/Users/Josh/VirtualBox VMs/Ubuntu-Test2
ISO Image	C:/Users/Josh/Downloads/ubuntu-22.10-desktop-amd64 (1).iso
Guest OS Type	Ubuntu (64-bit)
Skip Unattended Install	false
Unattended Install	
Username	Jthater
Product Key	true
Hostname/Domain Name	Ubuntu-Test2.myguest.virtualbox.org
Install in Background	false
Install Guest Additions	true
Guest Additions ISO	C:\Program Files\Oracle\VirtualBox/VBoxGuestAdditions.iso
Hardware	
Base Memory	8000
Processor(s)	4
EFI Enable	false
Disk	
Disk Size	55.00 GB
Pre-allocate Full Size	false

Figure 26: VM Setup Summary

Once you hit "Finish", Virtual Box will start running the machine and will begin downloading the ISO. It will take a few minutes to get everything fully set up. Once it is done downloading, it should kick you to a login screen. It should look something like this:

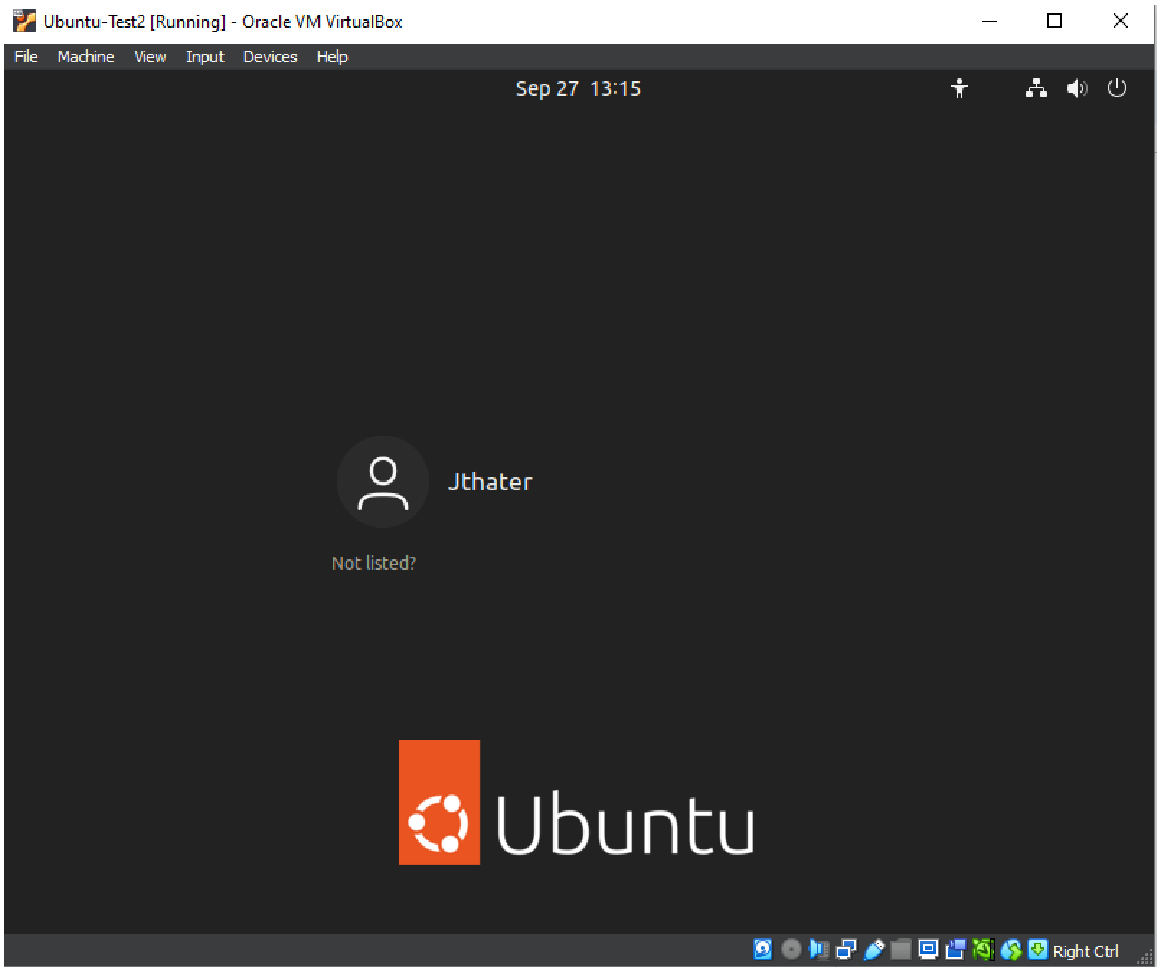

Ubuntu-Test2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Sep 27 13:15
Jthater
Not listed?
Ubuntu
Right Ctrl

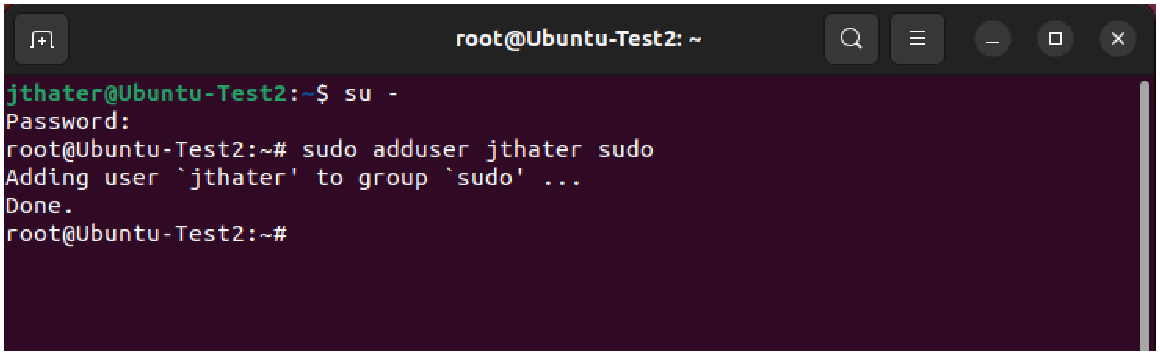
Figure 27: VM Setup Login

Go ahead and log in with your password. The last step is to ensure that the guest additions are downloaded correctly. Go full screen, and if the virtual machine also goes full screen, that means the guest additions have downloaded correctly! (NOTE: It seems that the virtual machine will not go full screen on 1440p monitors, only 1080p and smaller resolutions). With all of this done, you have successfully set up the virtual machine environment!

Also note, for some reason, sometimes you will not be added to the "sudoers" file. There are many ways to add yourself to this file, but one way to do so is shown below.

- Open up the terminal and type: ***su -***
- Enter your password
- Type: ***sudo adduser [your username] sudo***

If you did this correctly, you should see a message that you have been added to group sudo. Your terminal should look something like the figure below.



```
jthater@Ubuntu-Test2:~$ su -
Password:
root@Ubuntu-Test2:~# sudo adduser jthater sudo
Adding user `jthater' to group `sudo' ...
Done.
root@Ubuntu-Test2:~#
```

Figure 28: VM Sudo Setup

You can test if this worked by exiting out of the terminal, opening it back up, and typing something like "sudo ls." If you don't get an error, then you have successfully added yourself to the sudo group and can run commands as root.

NOTE: you may have to reboot the system to see the change go into effect (power off the machine).

8.1.2 Pre-Fabrication Tool Setup

Now that you have set up your virtual machine environment, it is time to begin downloading the tools needed for the analog design flow. It should be noted that since this whole process is still essentially in "alpha," things are changing quite a bit, so by the time you read this, some of this information may be outdated.

This part of the guide will just give a brief overview of the tools and how to install them; it will not go in-depth about their use and how to use them to go through the analog process flow. This information can be found in the next section of this document.

One last reminder that the rest of this guide is written assuming you are using an Ubuntu system. If you are not running an Ubuntu system, then you will have to figure out how to run these commands on your own system.

Before starting to download things, it is probably a good idea to update/upgrade your *"apt-get"* command as it may not be the most recent after a new download of a virtual machine. To do this, run the commands:

- ***sudo apt-get update***
- ***sudo apt-get upgrade***

If you run these commands and they give you errors, look up the error numbers you are getting, but it most likely means that you are running an outdated version of Ubuntu and need to upgrade your machine.

Once these commands are run, you should have no problem using "*sudo apt-get install [xxx]"* to download packages and dependencies for these tools.

Required Packages

In this section, some needed/useful packages that you will want to download for your use in this environment. These packages are not dependencies for the software you will be downloading, but they are helpful to have as you work. The first package you will need is the Git package. This will allow you to clone repositories, which is vital for installing some of the software. To install Git, simply type this command in the terminal:

- **sudo apt-get install git**

The next package you want to install is some type of text editor that will allow you to read and edit files within Linux. There are many text editors to choose from, but Vim is recommended. To install Vim, simply type the following command:

- **sudo apt-get install vim**

Magic VLSI

The first tool that you want to download and configure will be a tool called Magic VLSI. This tool is essential for creating a lot of libraries and will have you download many of the dependencies that will be needed for other tools. Magic will be a tool that you can use for your layouts and to perform device extractions for LVS and post-layout parasitics. It can also generate GDS and LEF files.

Documentation of Magic can be found here:
http://opencircuitdesign.com/magic/index.html

This is the page that has all the information you would want to know about Magic, so look around.

In order to install Magic, navigate to the install page. Here, you will find instructions on the dependencies, configurations, and setup of this tool. For the sake of simplicity, the dependencies and the commands to get them for an Ubuntu system. Some of these dependencies are optional, depending on how you want to configure Magic. However, it is recommended getting all of them because they do not take up much space, and some of the other software will need these dependencies installed anyway. These dependencies and commands to get them are as follows:

- **sudo apt-get install m4**
- **sudo apt-get install tcsh**
- **sudo apt-get install csh**
- **sudo apt-get install libx11-dev**
- **sudo apt-get install tcl-dev tk-dev**
- **sudo apt-get install libcairo2-dev**

Optional:

- **sudo apt-get install mesa-common-dev libglu1-mesa-dev**
- **sudo apt-get install libncurses-dev**

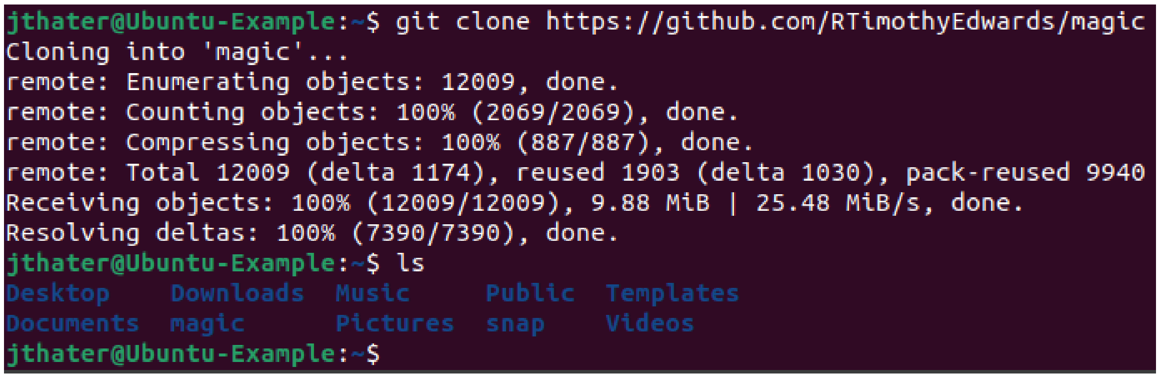
<u>NOTE: You can download all of these dependencies at once by typing them all in a single line. Such as:</u>

- **Sudo apt-get install m4 tcsh csh libx11-dev tcl-dev tk-dev libcairo2-dev**

Once you have all of the dependencies downloaded successfully, you should be able to compile and install Magic successfully. It is recommended to install it from the GitHub repository as it contains the most recent version. To do this, simply clone the git repository using the command:

- **git clone https://github.com/RTimothyEdwards/magic**
- **ls**

This will clone the GitHub repository into a directory on your virtual machine. This directory should show up in your current directory. You should be able to run the *"ls"* command to see it. The figure below illustrates this cloning and where the repository got cloned.



```
jthater@Ubuntu-Example:~$ git clone https://github.com/RTimothyEdwards/magic
Cloning into 'magic'...
remote: Enumerating objects: 12009, done.
remote: Counting objects: 100% (2069/2069), done.
remote: Compressing objects: 100% (887/887), done.
remote: Total 12009 (delta 1174), reused 1903 (delta 1030), pack-reused 9940
Receiving objects: 100% (12009/12009), 9.88 MiB | 25.48 MiB/s, done.
Resolving deltas: 100% (7390/7390), done.
jthater@Ubuntu-Example:~$ ls
Desktop    Downloads  Music     Public  Templates
Documents  magic      Pictures  snap    Videos
jthater@Ubuntu-Example:~$
```

Figure 29: Magic Repossitory Cloned

Once you have confirmed that you have cloned the Magic repository successfully, simply move into the directory using *"cd". I.e.:*

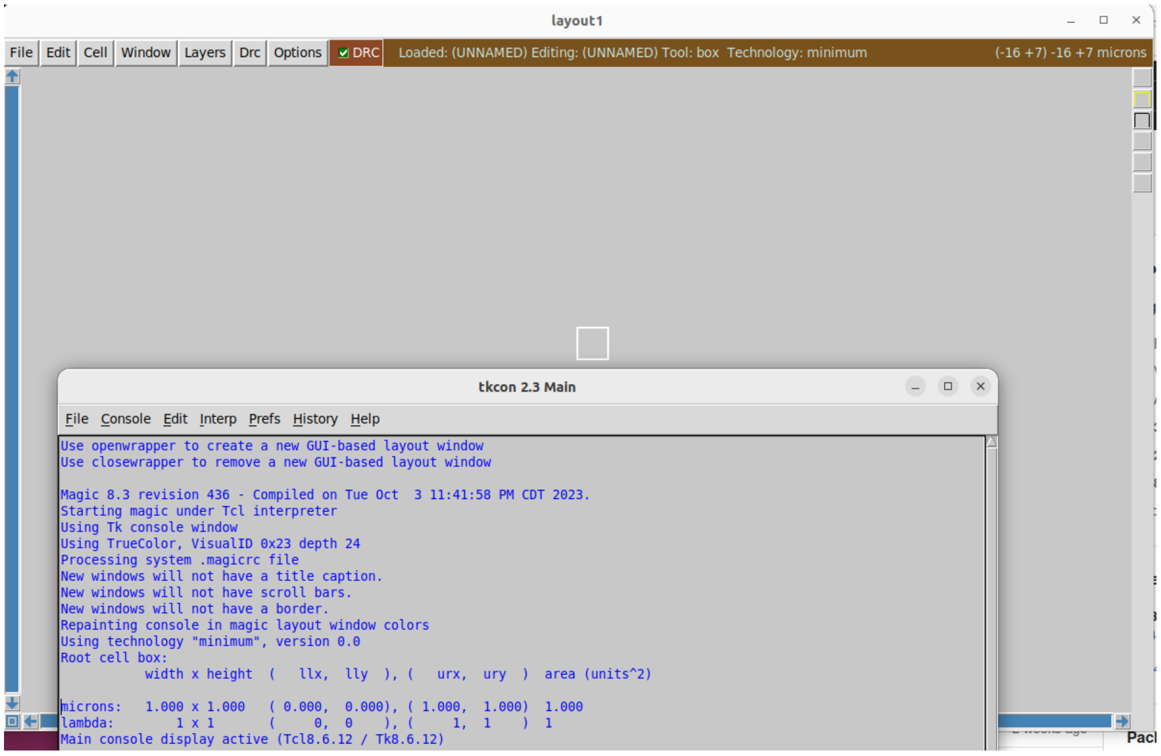
- **cd magic**

Doing this will move you into the Magic directory, and from here, you can manually configure and install Magic. To configure Magic, simply type the following commands:

- **./configure --enable-cairo-offscreen**
- **make**
- **sudo make install**

With this specific configuration it should help to alleviate any glitches that may occur with the software when using the OpenGL graphics interface if you choose to use it (read more about this in the Magic documentation).
It should be noted that the default path for the installation places all of the files in the directory /usr/local/bin/magic.

Once this is done, you can verify that you have installed Magic successfully by simply typing "*magic*" into the console. You should see the Magic tool come up, and you should be able to navigate around it. The figure below shows what you should be seeing.



```
Loaded: (UNNAMED) Editing: (UNNAMED) Tool: box  Technology: minimum          (-16 +7) -16 +7 microns
```

```
Use openwrapper to create a new GUI-based layout window
Use closewrapper to remove a new GUI-based layout window

Magic 8.3 revision 436 - Compiled on Tue Oct  3 11:41:58 PM CDT 2023.
Starting magic under Tcl interpreter
Using Tk console window
Using TrueColor, VisualID 0x23 depth 24
Processing system .magicrc file
New windows will not have a title caption.
New windows will not have scroll bars.
New windows will not have a border.
Repainting console in magic layout window colors
Using technology "minimum", version 0.0
Root cell box:
           width x height  (   llx,  lly  ), (   urx,  ury  )  area (units^2)

microns:   1.000 x 1.000   ( 0.000,  0.000), ( 1.000,  1.000)  1.000
lambda:        1 x 1       (     0,  0    ), (     1,  1    )  1
Main console display active (Tcl8.6.12 / Tk8.6.12)
```

Figure 30: Magic Startup Screen

If you are able to open Magic with no issue, then you have successfully installed the software and are ready to move on to the next step.

Xschem

The next tool that you will be downloading is Xschem. Xschem is a schematic editor that allows you to create schematics, symbols, test benches and will create SPICE netlists for testing. It is recommended over other open-source schematic editors, such as Xcircuit, because it feels and operates closer to commercial-grade tools (like Cadence).

Documentation for Xschem can be found here:
https://xschem.sourceforge.io/stefan/index.html

Just like with Magic, this page will contain all information about how to install and configure Xschem. The documentation on how to use this software is very in-depth, so it is highly recommended to go through and read the documentation once you get everything set up.

This software also requires some dependencies to be installed, but luckily, most of them have already been installed when Magic was installed, so only a few more need to be added. To get the rest of the dependencies, simply type:

- **sudo apt-get install flex**
- **sudo apt-get install bison**
- **sudo apt-get install libxpm-dev**

With all of these dependencies installed, Xschem can now be configured and installed. If you are following this guide section by section, then make sure you are back in your home directory before running this next step. This can simply be done with:

- **cd**
  - This will move you back to your home directory and should be done each time you begin installing new software

Now, simply clone this repository using the command:

- **git clone https://github.com/StefanSchippers/xschem**

Just like with Magic, if you did everything properly, you should be able to *"ls"* and see the Xschem directory. To install this software, simply type the commands from your home directory:

- **cd xschem**
- **./configure**
- **make**
- **sudo make install**

The default installation path of this software should be /usr/local/bin/xschem. NOTE: You can change this default directory when you run the "./configure" command if you so choose.

To ensure that everything is installed correctly, simply type the command *"xschem"* into the terminal, and you should be greeted with a similar-looking screen, as seen in the figure below.

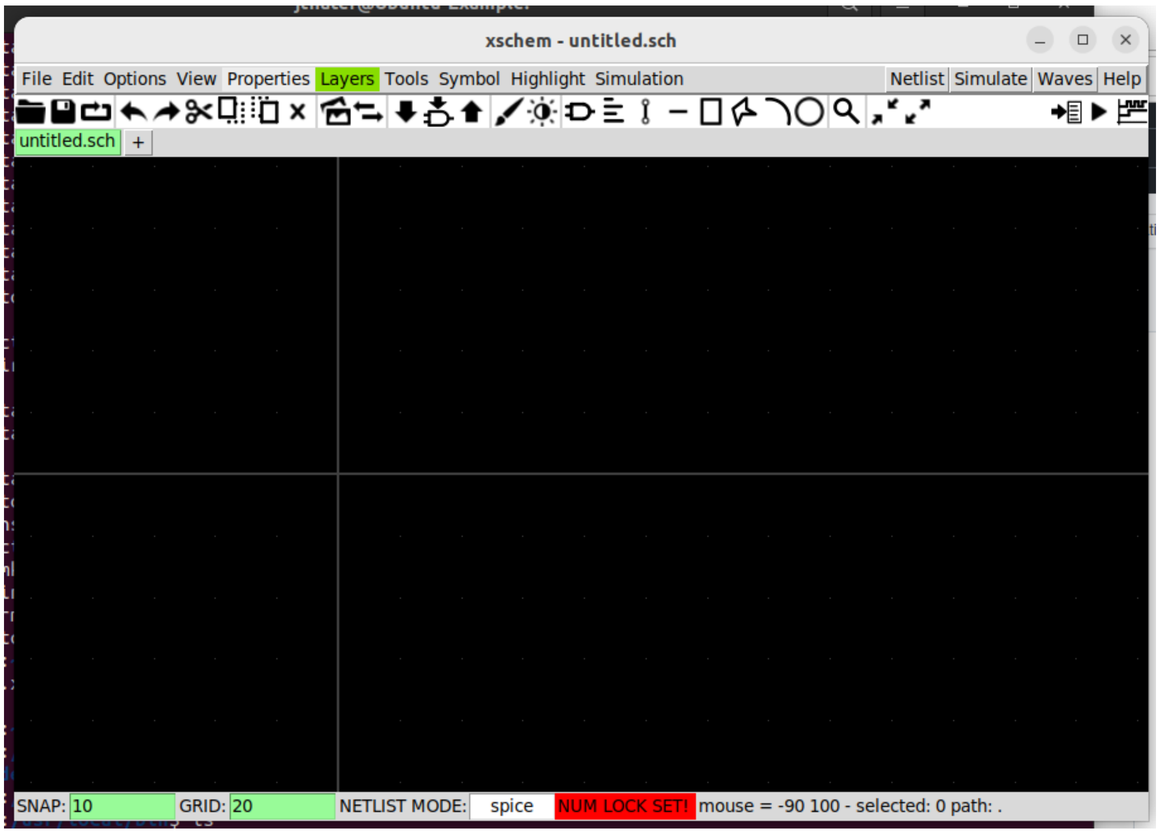

xschem - untitled.sch
File Edit Options View Properties Layers Tools Symbol Highlight Simulation
Netlist Simulate Waves Help
untitled.sch +
SNAP: 10 GRID: 20 NETLIST MODE: spice NUM LOCK SET! mouse = -90 100 - selected: 0 path: .

Figure 31: Xschem Startup Screen

If you see this screen, then you have successfully installed Xschem and are ready to move on to the next step.

Ngspice

The next tool that you will install is Ngspice. Ngspice is an analog/mixed-signal simulation tool. This will allow you to read in SPICE netlists (and, with more recent versions, more complex netlists) and run different types of circuit analysis on them, such as DC sweeps, transient simulations, AC tests, and many, many more. It also allows you to simulate digital design through Xspice, which is installed as part of this software.

The Ngspice documentation can be found here:
https://ngspice.sourceforge.io/

To get a fully featured Ngspice, there are a few dependencies that you must download. They include:

- **sudo apt-get install libxaw7-dev**
- **sudo apt-get install lib64readline8 libreadline-dev**
    - It should be noted that this dependency can be very finicky, so you may have to change the 64 to a 32, or there might be a new version, so you might need to change the 8 to a 9. Don't worry too much if you can't get this to work, as it is highly unlikely that you will use readlines for Ngspice for anything that you will work on.

You can download the Ngspice software from a Git repository like the other software, but this is more complicated than downloading it from a tar file in this case. So, it is recommended to download this software from the tar file found here:
https://sourceforge.net/projects/ngspice/files/ng-spice-rework/41/

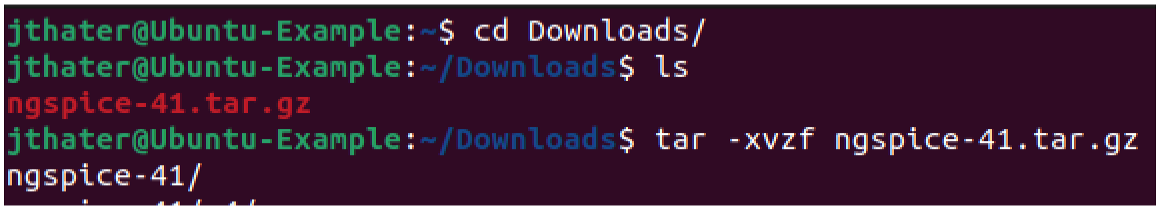
Download that tar file and navigate to the downloads file from the terminal. This can be achieved by being in your home directory and typing the command

- **cd Downloads**

Once in this directory, type *"ls"*, and you should see a file named "ngspice-41.tar.gz" or something similar. Once you have confirmed the file is there, type the following command to unzip the file.

- **tar -xvzf ngspice-41.tar.gz**

If you've done everything correctly, it should look something like the figure below.



```
jthater@Ubuntu-Example:~$ cd Downloads/
jthater@Ubuntu-Example:~/Downloads$ ls
ngspice-41.tar.gz
jthater@Ubuntu-Example:~/Downloads$ tar -xvzf ngspice-41.tar.gz
ngspice-41/
```

Figure 32: Unzipped Ngspice File

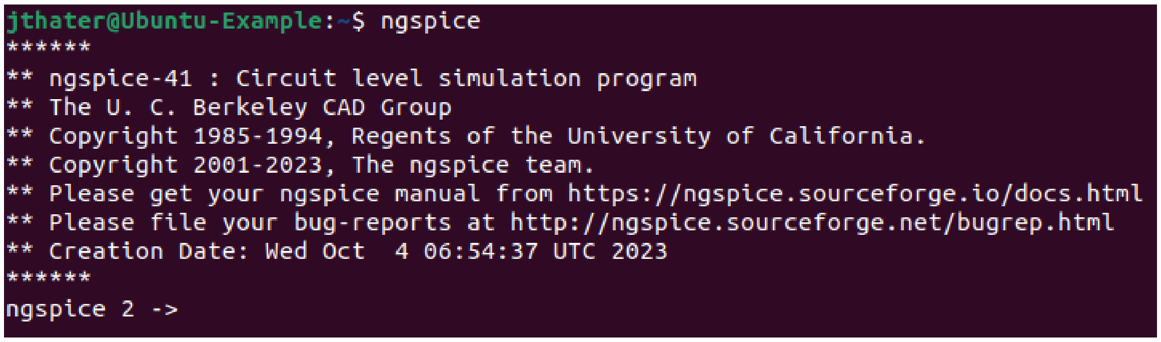
Unzipping the file will result in a bunch of lines being output to the terminal. Once everything is done, it is recommended to move the file into your home directory. This can be done with the command:

- **mv ngspice-41 ..**

If done correctly, you can navigate back to your home directory, and you should be able to use *"ls"* to see it. Once you have confirmed that the file is there, you can run the following commands:

- **cd ngspice-41**
- **mkdir release**
- **cd release**
- **../configure --with-x --enable-xspice --disable-debug --enable-cider --with-readlines=yes --enable-predictor --enable-osdi --enable-openmp**
    - Must configure with osdi in order to support ReRAM simulations using Ngspice
- **make 2>&1 | tee make.log**
- **sudo make install**

Running these commands will take longer than any of the setups for the other software, so be prepared to wait a few minutes for this installation. To ensure that the download has been successful, simply type *"ngspice"* into the terminal. If you do so, you should see something similar to the figure seen below.



```
jthater@Ubuntu-Example:~$ ngspice
******
** ngspice-41 : Circuit level simulation program
** The U. C. Berkeley CAD Group
** Copyright 1985-1994, Regents of the University of California.
** Copyright 2001-2023, The ngspice team.
** Please get your ngspice manual from https://ngspice.sourceforge.io/docs.html
** Please file your bug-reports at http://ngspice.sourceforge.net/bugrep.html
** Creation Date: Wed Oct  4 06:54:37 UTC 2023
******
ngspice 2 ->
```

Figure 33: Ngspice Startup Screen

If you see something similar to this screen, then you have successfully installed Ngspice and are ready to move on to the next software.

GAW (optional)

GAW is a way to view the waveforms that are generated from Xschem. There are many options to view waveforms, including through Xschem directly, through Ngspice, or other waveform viewers. It is recommended to look up waveform viewers yourself and find something that fits your needs. GAW is mentioned since that is what was used for us.

To download GAW, go here and download the latest version:
https://download.tuxfamily.org/gaw/download/

To download GAW, you will need this dependency:

- **sudo apt-get install libgtk-3-dev**

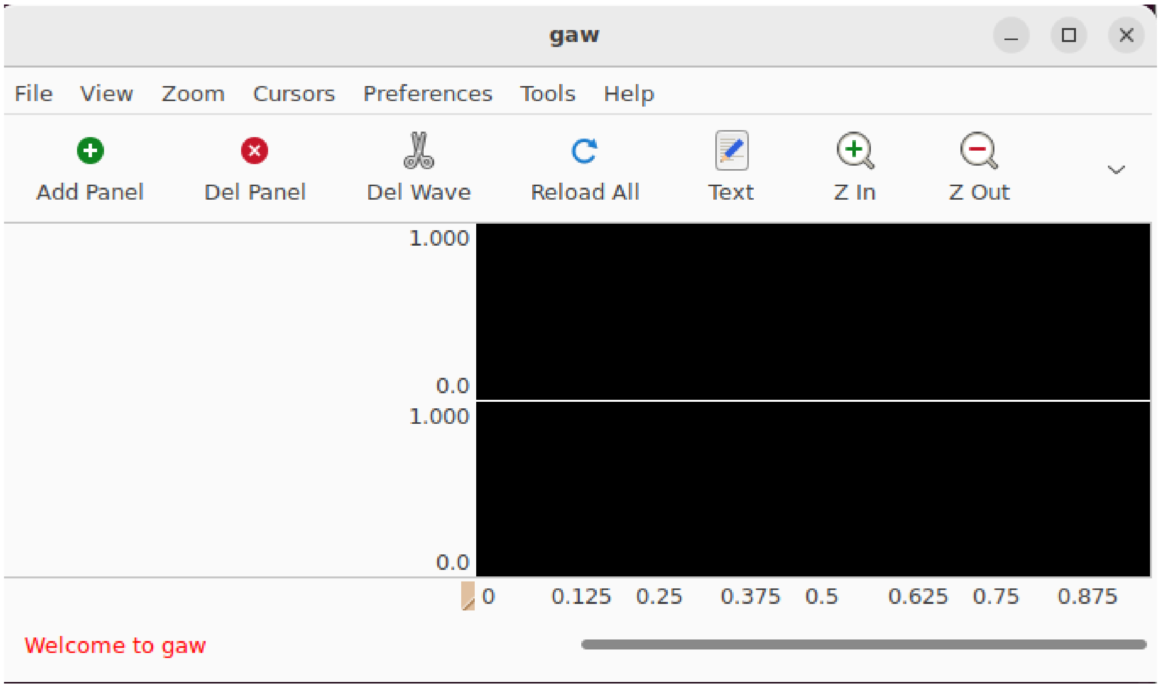
Optional:

- **sudo apt-get install xterm**
  - This is an emulator that will allow a Ngspice terminal to open when doing simulations in Xschem. This will allow you to use Ngpsice to view waveforms.

Then, from your home directory, enter the following commands:

- **cd Downloads**
- **tar -xvzf gaw3-20220315.tar.gz**
- **mv gaw3-20220315 ..**
- **cd**
- **cd gaw3-20220315**
- **./configure**
- **make**
- **sudo make install**

To confirm that GAW has been installed correctly, type gaw into the terminal, and you should see something similar to the figure below pop up.



Welcome to gaw

Figure 34: GAW Startup Screen

Netgen

The next tool that must be installed is Netgen. Netgen is a tool that will be able to perform LVS on two netlists, whether it be SPICE or Verilog.

Documentation for Netgen can be found here:
http://opencircuitdesign.com/netgen/index.html

This documentation page is much like Magic (as it is made by the same person). There are guides on how to install this software, and there are some tutorials on how to use the software. It’s a good idea to look around in the “Tutorials” and “References” pages for very detailed information about all of the different things Netgen can do.

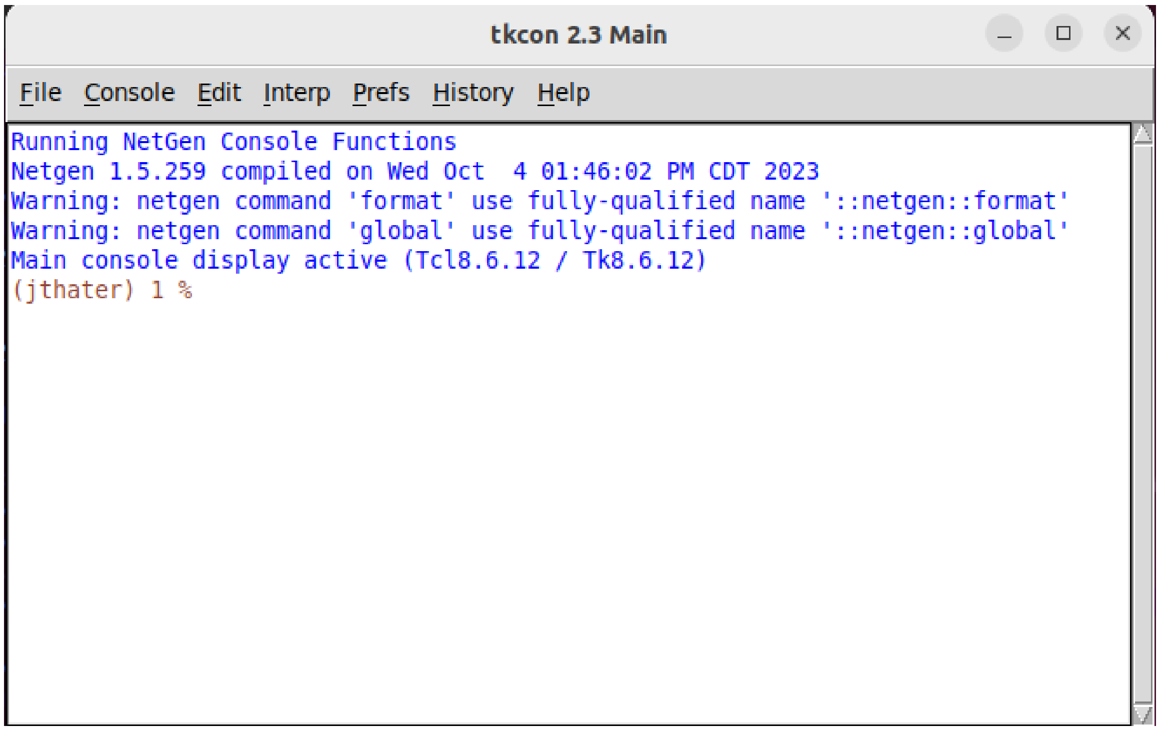
The installation of Netgen is very simple. Make sure you are in your home directory before performing this next command. Simply copy the Netgen GitHub repository using the command:

- **git clone https://github.com/RTimothyEdwards/netgen**

Once you have successfully cloned the repository, enter into the Netgen directory and run the following commands.

- **./configure**
- **make**
- **sudo make install**

If you have successfully installed Netgen, then simply type *“netgen”* into the terminal, and you should see something similar to what is shown below.



```
Running NetGen Console Functions
Netgen 1.5.259 compiled on Wed Oct  4 01:46:02 PM CDT 2023
Warning: netgen command 'format' use fully-qualified name '::netgen::format'
Warning: netgen command 'global' use fully-qualified name '::netgen::global'
Main console display active (Tcl8.6.12 / Tk8.6.12)
(jthater) 1 %
```

Figure 35: Netgen Startup Screen

Once you have verified that you have installed Netgen, you are ready to move on. If you are planning on doing simple analog designs, then you can most likely stop installing software here and move on to the Open PDKs Install section. The next two software are optional.

Open PDKs Install

The next install is the PDK that you will be designing devices with. As mentioned earlier, the installation will be done through Open PDKs as it has simplified the process of configuring the PDK for open-source tools and files. All you have to do is clone the repository, configure your installation for whichever PDK you want to download - SkyWater or GF180 MCU - and install the files, cells, and ruleset for that PDK. For this installation, it is assumed that you are installing the SkyWater 130 nm PDK.

Documentation for Open PDKs can be found here:
http://opencircuitdesign.com/open_pdks/

Clicking through the different tabs will walk you through configurations and installation steps for your respective PDK. This page, however, does not contain much information about the specific PDK you are downloading (like the different cells, rules, etc.).

To find this information, navigate to the documentation for the SkyWater 130 nm process, which can be found here:
https://skywater-pdk.readthedocs.io/en/main/

Once you have read through a little bit of the documentation, you can download and install Open PDKs. This will operate much in the same way as all of the other installations from the source code in GitHub. However, this install has many different configurations depending on which libraries from the PDK you want installed or not. Looking through both the SkyWater documentation as well as the install page on Open Circuit design should show you how to specifically configure these installs for you. It should be noted that you can always add libraries later if you find out you need something from them.

For most general users, the standard build of the SkyWater 130 nm PDK should be fine. To do this, simply type this into the terminal. NOTES: For this install, you need to have Magic and all of the dependencies installed. This is also a very large install, so make sure you have plenty of disk space available. As this is a large download, it will take quite some time, so be patient.

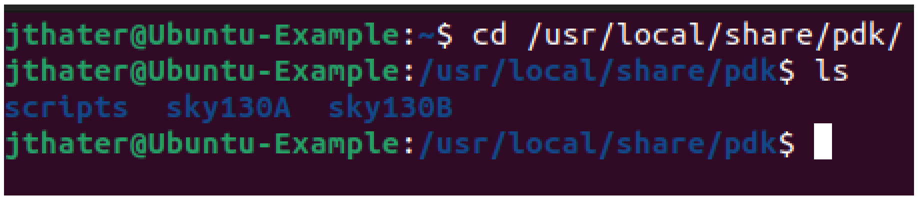
- **git clone https://github.com/RTimothyEdwards/open_pdks**
- **cd open_pdks**
- **./configure --enable-sky130-pdk --enable-sram-sky130**
- **make**
- **sudo make install**
- **make distclean**

While installing this massive script, you may see some warning and errors that pop up throughout the installation process - this is normal. The only thing you need to worry about is if, after going through any of these steps, you encounter “Error 2”. If you see this, then something went wrong, and it will direct you to a logfile where you can see what went wrong.

Once you have gone through this installation process, you can check if everything is installed correctly by using the commands shown below to navigate to the pdk installation to view the files:

- **cd /usr/local/share/pdk**
- **ls**

If you do this, you should see something similar to the figure below.



```
jthater@Ubuntu-Example:~$ cd /usr/local/share/pdk/
jthater@Ubuntu-Example:/usr/local/share/pdk$ ls
scripts  sky130A  sky130B
jthater@Ubuntu-Example:/usr/local/share/pdk$
```

Figure 36: Open PDKs File Directory

If you see this, then it is likely that everything was installed correctly. You may notice that there are two Sky130 directories. One is “A” and the other is “B”. You don’t need to know all about the minute differences between these two. The only difference that matters is that the “sky130A” is the standard process, and the “sky130B” is a modified process that supports the manufacturing of ReRAM. **So, if you plan to use ReRAM in any part of your design, you must use the sky130B process for all of your designs. Otherwise, if you have no plans to use ReRAM, then you can go ahead and use the standard sky130A process.**

It is a good idea to move into these files and look around to see all that was downloaded. Each directory will contain a libs.ref and a libs.tech directory. The libs.ref directory houses all of the libraries that were downloaded and contains the files for the cells contained in these libraries (read the SkyWater 130 nm PDK documentation for more information about these libraries). The libs.tech directory will house important files for analog design and configuration of the tools for the SkyWater process. This will be covered in the next section of this guide.

8.1.3 GitHub Repository Cloning

Now that the environment has been setup and the tools installed, our GitHub repository can now be cloned. This repository can be found at the link here:
https://github.com/Jthater1/ReRAM_Crossbar_Project

Simply clone this repository into your home directory using the command:

- **git clone** https://github.com/Jthater1/ReRAM_Crossbar_Project

In this repository you will find all the pertinent files we created during our time working on the project. The most important ones will be in the *xschem* and *magic* folders. The *xschem* folder houses all of our schematic designs as well as testbenches for some of them. The *magic* folder houses all of the layouts of our components as well as the final layout hooked up to the Caravel Harness. All of the other folders hold relevant files either to the digital parts of the project or stuff related to eFabless fabrication. With this repository cloned you have all of the files related to our project and are able to view, edit, modify and improve upon our designs.

8.1.4 Post-Fabrication Bring-up Plan

Setup

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

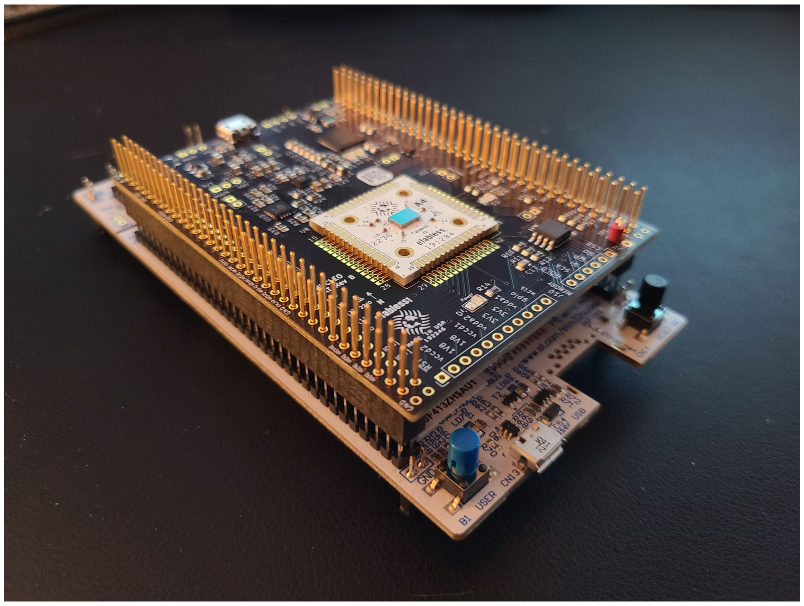


Figure 37: Caravel Nucleo Board [1]

Configuration

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 bend, 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 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 on copied on to Nucleo. It is also used to retrieve the gpio_config_def.py file after the diagnostic completes.

Installation

This will guide the user through installing necessary tools to run diagnostic software through a Micropython image on the Nucleo board.

1. Install the required tools including **mpremote**, **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.

```
git clone https://github.com/efabless/caravel_board.git
cd caravel_board/firmware/mpw2-5/nucleo
make setup
```

- **mpremote** is used for connecting the Micropython
- **mpy-cross** is a cross compiler for Micropython the 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.

- For macOS:

```
brew install stlink
```

- On Ubuntu download and install a release deb from https://github.com/stlink-org/stlink/releases

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.

- NOTE: For some linux platforms, the PYBFLASH volume is not automatically installed.

Test Process:

I. Hardware Tests

To ensure the caravel Nucleo board is working correctly a few basic checks should be run on the received board. Compare tests with Schematic of the Caravel Nucleo Board shown Below.

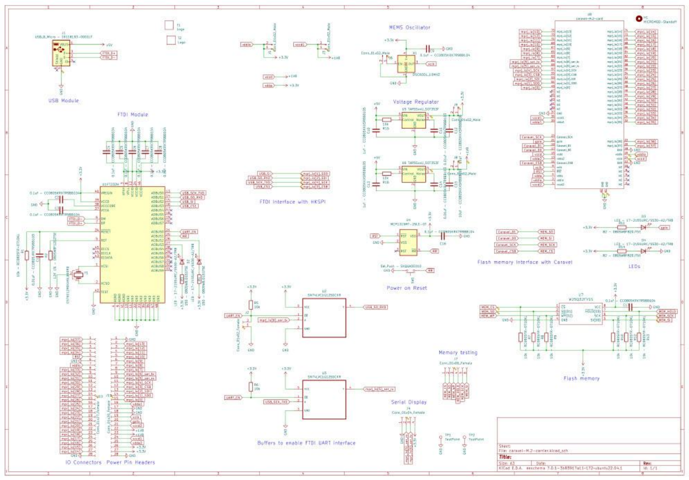


Figure 38: Caravel Nucleo Board [2]

- Use multimeter to verify Expected connectivity compared to schematic by measuring resistances for shorts and Open circuits
- Send and measure test signals through pins to confirm signal integrity.
- Measure voltages on power rails to verify power delivery of pins
    - vccd1 should supply 1.8v

- vccdio should supply 3.3v

II. Software Tests

This section provides information necessary to setup and run diagnostic software to ensure that the Caravan harness is working properly.

To run the diagnostic, enter the following commands:

```
cd caravel_board/firmware/mpw2-5/nucleo
make run_analog PART=<part id>
```

The test will now run with the green light on the Nucleo flashing 5 times. When the test concludes, the green and red lights 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

Type Ctrl-C to exit the test diagnostic once it completes. You can run

```
make get_config
```

To get a configuration file. The file will indicate if the IO was successfully configured.

III. ReRAM Firmware Test

This is a test plan specific to our ReRAM Crossbar implementation for the Efabless program to validate our crossbar's functionality. The user will need to create test cases using the provided ReRAM crossbar driver APIs. Here is the source code for the APIs that provide a convenient interface with the ReRAM crossbar:

```
#define WAIT_ITERATIONS 10

void write(uint8_t value, uint8_t line){
  uint32_t op;
  // la[ 7: 0]
  uint32_t bitline     = value;
  // la[15: 8]
  uint32_t selectline = (0xFF ^ (value)) << 8;
  // la[23:16]
  uint32_t wordline    = (1 << line) << 16;
  uint32_t write_control = (0b11) << 24;
  op = bitline | selectline | wordline | write_control;
  reg_la0_data = op;
  reg_la2_data = op << 19;
  reg_la3_data = op >> 13;
  for(int i = 0; i < WAIT_ITERATIONS; i++){
      __asm("nop");
  }
  reg_la0_data = 0;
}

uint8_t read(uint8_t line){
  uint32_t op;
  uint32_t wordline = (1 << line) << 16;
  op = wordline;
  reg_la0_data = op;
  reg_la2_data = op << 19;
  reg_la3_data = op >> 13;

  for(int i = 0; i < WAIT_ITERATIONS; i++){
      __asm("nop");
  }
  uint8_t result = (uint8_t) reg_la1_data;
  reg_la0_data = 0;
  return result;
}

uint8_t mac(uint8_t value){
  uint32_t op;
  uint32_t selectline = 0;
  uint32_t bitline    = 0;
  uint32_t wordline   = (value) << 16;
  op = selectline | bitline | wordline;
  reg_la0_data = op;
  reg_la2_data = op << 19;
  reg_la3_data = op >> 13;

  for(int i = 0; i < WAIT_ITERATIONS; i++){
      __asm("nop");
  }
  uint8_t result = (uint8_t) reg_la1_data;
  reg_la0_data = 0;
  return result;
}

void form(){
  uint32_t op;
  uint32_t selectline = 0;
  uint32_t bitline    = 0xFF;
  uint32_t wordline   = 0xFF << 8;
  uint32_t form_control = (0b10) << 24;
  op = selectline | bitline | wordline | form_control;
  reg_la0_data = op;
  reg_la2_data = op << 19;
  reg_la3_data = op >> 13;

  uint8_t result = (uint8_t) reg_la1_data;
  return result;
}
```

It exposes 4 functions: write, read, mac, and form. Write will write a value to a line in the crossbar, read will read a line, mac will do a multiply and accumulate operation given a vector value, and form will form all of the ReRAM.

Here is an example of a main function that could use these functions to write an identity matrix, read a row, and then do a multiply and accumulate:

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

void main(){
  // configure logic analyzer
  reg_la0_oenb = reg_la0_iena = 0xFFFFFFFF;
  reg_la1_oenb = reg_la0_iena = 0x00000000;
  reg_la2_oenb = reg_la0_iena = 0xFFFFFFFF;
  reg_la3_oenb = reg_la3_iena = 0xFFFFFF00;

  // Write identity matrix
  write(0x01, 0);
  write(0x02, 1);
  write(0x04, 2);
  write(0x08, 3);
  write(0x10, 4);
  write(0x20, 5);
  write(0x40, 6);
  write(0x80, 7);

  int r = read(3);

  int m = mac(0xFF);

}
```

Expected outputs can be generated using the digital behavioral model. This can be run by following the Digital Model Guide and modifying

`verilog/dv/crossbar_la_test/crossbar_la_test.c`

To match your new test case. You can also use the binary generated by running the test on the digital model to be the firmware on the Nucleo board.

IV. Component Testing Through GPIO:

These tests will allow you to verify the functionality of the components in the user project. Pinout for components in the user project is included at the end of this section. GPIO and IO pins require external user inputs and readings

8x8 Crossbars

This section will cover the testing of the 0.2V and 0.4V crossbars and their analog control circuits. All tests will be run on both crossbars.

Note: Pin io_analog[5] needs to by supplied -1.8v

The first test to be performed is testing all of the individual 1T1R cells on the crossbars to verify their functionality by writing and checking the High Resistive state and then the Low Resistive State.

1. Perform a write function of 00000000
2. Then Read each cell, the expected output for all cells is 0
3. Perform a write function of 11111111
4. Then Read each cell, the expected output for all cells is 1

Next test the MAC operation, in this test a MAC will be performed

1. Write 00000000
2. Perform MAC 00000000, the expected output is 00000000
3. Perform MAC 11111111, the expected output is 00000000
4. Write 00000001
5. Perform MAC 00000000, the expected output is 00000001
6. Perform MAC 11111111, the expected output is 00000001
7. Repeat 7 times bitshifting the write input left 1 before each Subsequent Test, the expected output will be the same as the write input

The final test to perform on the crossbar is disruption testing. This is to test the effect that performing repeated Read and MAC operations can have on the filament.

1. Write 00000000
2. Perform MAC of 1111111 repeatedly until output is no longer 00000000 or after many attempts with no change
3. Perform Read on all cells to confirm which cell was disrupted
4. Write 11111111
5. Perform MAC of 1111111 repeatedly until output is no longer 11111111 or after many attempts with no change
6. Perform Read on all cells to confirm which cell was disrupted

4x4 Crossbar

There are four 4x4 Crossbars, 2 with normal sized filaments and 2 with large filaments. To test operations on the ReRAM Cells below are the required voltages

Form	
Bit Line	2.6v - 3.1v
Word Line	1.4v - 2.0v
Select Line	0v

Write	
Bit Line	2.5v / 0v
Word Line	1.8v - 2.5v
Select Line	0v / 2.5v

Read	
Bit Line	0.2v - 0.4v
Word Line	1.8v
Select Line	Output

MAC	
Bit Line	0.2v - 0.4v
Word Line	1.8v
Select Line	Output

DAC

There are two one bit DACs included, the expected output is either 0 or 1.8V

Buffer	
Vin	0v / 1.8v
Vout	Output
VDD	1.8v
VSS	0

ADC

There are two one bit ADCs included, the expected output is either 0 or 1.8V, the threshold voltage is ~.59v (R1 = 1943 Ohms, R2 =943 Ohms)

ADC	
Vin	0v - 1.8v

Y	Output
VCC	1.8v
VSS	0

2-1 MUX

The MUXs are analog MUXs so they should output the same input voltages, but there is a loss of current

Buffer	
A	0v / 1.8v
B	0v / 1.8v
S	0v / 1.8v
Vout	S=1 -> A, S=0 -> B
VDD	1.8v
VSS	0

Pinout

Pin	Connection
la_data_in [0]	BL 0.2v 1
la_data_in [1]	BL 0.2v 2
la_data_in [2]	BL 0.2v 3
la_data_in [3]	BL 0.2v 4
la_data_in [4]	BL 0.2v 5
la_data_in [5]	BL 0.2v 6
la_data_in [6]	BL 0.2v 7
la_data_in [7]	BL 0.2v 8
la_data_in [8]	SL in 0.2v 1
la_data_in [9]	SL in 0.2v 2
la_data_in [10]	SL in 0.2v 3
la_data_in [11]	SL in 0.2v 4
la_data_in [12]	SL in 0.2v 5
la_data_in [13]	SL in 0.2v 6
la_data_in [14]	SL in 0.2v 7
la_data_in [15]	SL in 0.2v 8
la_data_in [16]	WL in 0.2v 1
la_data_in [17]	WL in 0.2v 2
la_data_in [18]	WL in 0.2v 3
la_data_in [19]	WL in 0.2v 4
la_data_in [20]	WL in 0.2v 5
la_data_in [21]	WL in 0.2v 6

la_data_in [22]	WL in 0.2v 7
la_data_in [23]	WL in 0.2v 8
la_data_in [24]	Write Select 0.2v
la_data_in [25]	Write Form Select 0.2v
la_data_in [83]	BL 0.4v 1
la_data_in [84]	BL 0.4v 2
la_data_in [85]	BL 0.4v 3
la_data_in [86]	BL 0.4v 4
la_data_in [87]	BL 0.4v 5
la_data_in [88]	BL 0.4v 6
la_data_in [89]	BL 0.4v 7
la_data_in [90]	BL 0.4v 8
la_data_in [91]	SL in 0.4v 1
la_data_in [92]	SL in 0.4v 2
la_data_in [93]	SL in 0.4v 3
la_data_in [94]	SL in 0.4v 4
la_data_in [95]	SL in 0.4v 5
la_data_in [96]	SL in 0.4v 6
la_data_in [97]	SL in 0.4v 7
la_data_in [98]	SL in 0.4v 8
la_data_in [99]	WL 0.4v 1
la_data_in [100]	WL 0.4v 2
la_data_in [101]	WL 0.4v 3
la_data_in [102]	WL 0.4v 4
la_data_in [103]	WL 0.4v 5

la_data_in [104]	WL 0.4v 6
la_data_in [105]	WL 0.4v 7
la_data_in [106]	WL 0.4v 8
la_data_in [107]	Write Select 0.4v
la_data_in [108]	Write Form Select 0.4v
la_data_out [32]	SL out 0.2v 1
la_data_out [33]	SL out 0.2v 2
la_data_out [34]	SL out 0.2v 3
la_data_out [35]	SL out 0.2v 4
la_data_out [36]	SL out 0.2v 5
la_data_out [37]	SL out 0.2v 6
la_data_out [38]	SL out 0.2v 7
la_data_out [39]	SL out 0.2v 8
la_data_out [113]	SL out 0.4v 1
la_data_out [114]	SL out 0.4v 2
la_data_out [115]	SL out 0.4v 3
la_data_out [116]	SL out 0.4v 4
la_data_out [117]	SL out 0.4v 5
la_data_out [118]	SL out 0.4v 6
la_data_out [119]	SL out 0.4v 7
la_data_out [120]	SL out 0.4v 8
gpio-analog[0]	4T4R 1 WL 1, 4T4R Large 1 WL 1, Buffer 1 Vin, ADC 1 Vin, MUX 1 A
gpio-analog[1]	4T4R 1 WL 2, 4T4R Large 1 WL 2 , MUX 1 B
gpio-analog[2]	4T4R 1 BL 1, 4T4R Large 1 BL 1

gpio-analog[3]	4T4R 1 BL 2 1, 4T4R Large 1 BL 2
gpio-analog[4]	MUX 1 S
gpio-analog[7]	4T4R 2 WL 1, 4T4R Large 2 WL 1, Buffer 2 Vin, ADC 2 Vin, MUX 2 A
gpio-analog[8]	4T4R 2 WL 2, 4T4R Large 2 WL 2 , MUX 2 B
gpio-analog[9]	4T4R 2 BL 1, 4T4R Large 2 BL 1
gpio-analog[10]	4T4R 2 BL 2, 4T4R Large 2 BL 2
gpio-analog[11]	MUX 2 S
gpio-analog[12]	4T4R 2 SL 1
gpio-analog[13]	4T4R 2 SL 2
gpio-analog[14]	4T4R Large 2 SL 1
gpio-analog[15]	4T4R Large 2 SL 2
gpio-analog[16]	ADC 1 Y
gpio-analog[17]	MUX 1 out
io_analog[0]	4T4R 1 SL 1
io_analog[1]	4T4R 1 SL 2
io_analog[2]	4T4R Large 1 SL 1
io_analog[3]	4T4R Large 1 SL 2
io_analog[5]	VSS Negatvive SL
io_analog[7]	Buffer 1 Vout
io_analog[8]	ADC 2 Y
io_analog[9]	MUX 2 out
io_analog[10]	Buffer 2 Vout

References

[1][2] Efabless. (n.d.). Efabless/caravel_board. GitHub. Retrieved November 29, 2023, from https://github.com/efabless/caravel_board

8.2 Team Contract

Team Members:

1) Joshua Thater___________________ 2) Aiden Petersen__________________

3) Matthew Ottersen_______________ 4) Regassa Dukele__________________

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
    a. Monday 11-1 in TLA
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    a. Discord group chat
    b. Microsoft Teams
3. Decision-making policy (e.g., consensus, majority vote):
    a. Consensus
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    a. As a team, take notes as needed
        i. Notes are stored in shared Google Doc

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    a. Expected to be on time for all meetings unless the team is notified beforehand.
    b. During the meeting, each member should contribute ideas, promote questions, and make progress on the project.
    c. It is expected that during meetings, the focus is solely on the project and no outside classwork or other distractions.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
    a. It is expected that each member will contribute to team assignments equally.
    b. All timelines and deadlines for deliverables should be on time unless otherwise communicated.
3. Expected level of communication with other team members:
    a. Check group chats at least once a day for any updates.
    b. Weekly updates on what each team member is working on for that week.
4. Expected level of commitment to team decisions and tasks:
    a. Each team member should voice their opinion when it comes to decisions on the project.
    b. If a team member says they will do a task, they are expected to fulfill that task.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
    a. Joshua Thater - team organization & documentation
    b. Aiden Petersen - Software documentation. Primary client communicator.
    c. Matt Ottersen - Individual Component Design, Simulations
    d. Regassa Dukele - Individual Component Design, Simulations
2. Strategies for supporting and guiding the work of all team members:
    a. Make sure to always be in communication with the team and document what progress is being done.
    b. If a member runs into trouble with something, they should share the issue in one of the group chats, so another team member may assist.
3. Strategies for recognizing the contributions of all team members:
    a. Having a record of completed tasks and research.
    b. Making sure that each member feels their contributions are helpful towards completing the project.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
    a. Joshua Thater: Effective communication, Verilog, experience in analog and digital circuit design, some experience with mixed-signal circuit design, Linux terminal, PCB design, solid troubleshooting skills.
    b. Aiden Petersen: Very experienced in Linux and troubleshooting it, verilog, C, digital design, general software engineering experience, computer architecture.
    c. Regassa Dukele: I have some experience in schematic and layout design, including creating schematics, developing board layouts, and troubleshooting design issues.
    d. Matt Ottersen: Experience in Analog circuit design, testing and layout. Some experience with digital, and mixed signal design. Also some experience with Verilog
2. Strategies for encouraging and support contributions and ideas from all team members:
    a. Making sure that each person has shared their opinion.
    b. Help members express their ideas in a positive environment.
    c. Hold brainstorming sessions where everyone can contribute ideas and express their feeling
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
    a. By informing the team that they feel their contributions are being obstructed.
    b. If issues still persist, contact the advisor.
    c. Identify any potential issues or concerns around collaboration in the team and tell them to fix the problem.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

   a. Obtain all of the background information needed and begin working on the eFabless MPW submission
   b. Fully design ReRAM ASIC to be fabricated, but not necessarily have it implemented.
2. Strategies for planning and assigning individual and teamwork:
   a. Assignments will be based on the skills each individual has in a specific area.
   b. Assignments and plans will meaningfully progress the project.
   c. Individuals will be assigned equal amounts of work.
3. Strategies for keeping on task:
   a. Weekly client meetings where we will discuss progress and define deliverables by the next week.
   b. Weekly in-person group meetings to discuss and assign tasks, along with asking for help if needed.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
   a. We will discuss these infractions with the member in a weekly meeting, try to figure out why they are happening and how to resolve them.
2. What will your team do if the infractions continue?
   a. We will discuss these infractions with our advisor and see if we can resolve the issues. If we cannot, we may need to remove them from the project.

****
a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*
b) *I understand that I am obligated to abide by these terms and conditions.*
c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*
1) Joshua Thater_______________________________________ DATE 02/19/2023_________
2) Aiden Petersen______________________________________ DATE 02/19/2023_________
3) Regassa Dukele _____________________________________ DATE 02/19/2023_________
4) Matt Ottersen_______________________________________ DATE 02/19/2023_________