ASIC ReRAM Compute Test Chip

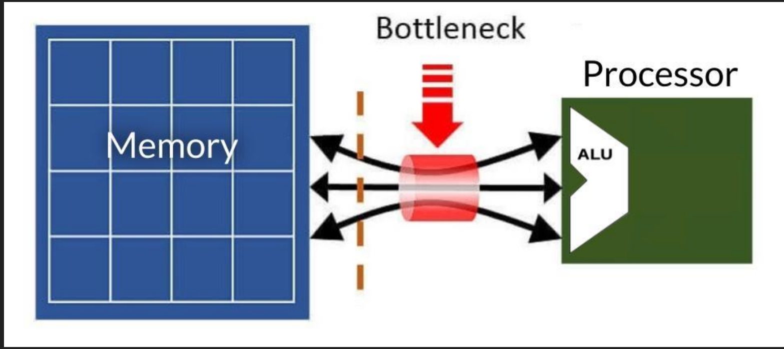
Team sdmay25-19

By: Noah Mack, Olivia Price, Travis Jakl, Sam Burns

Clients/Advisors: Dr. Henry Duwe and Dr. Cheng Wang

Problem Statement

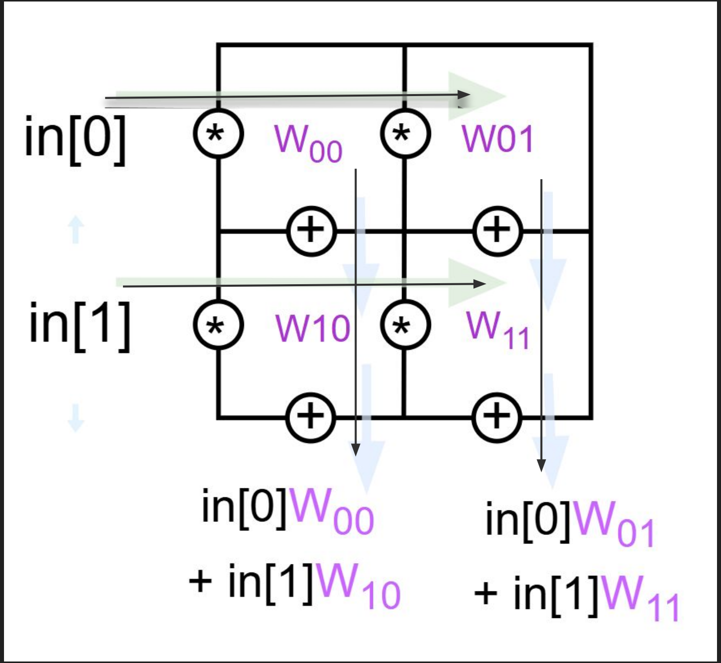
- AI is growing in demand, and right now it's an energy intensive process
- Matrix-vector multiplication (MVM) is crucial in machine learning
- Data transfer between memory and CPU creates bottlenecking
- **What can be done about this?**



Source [1]

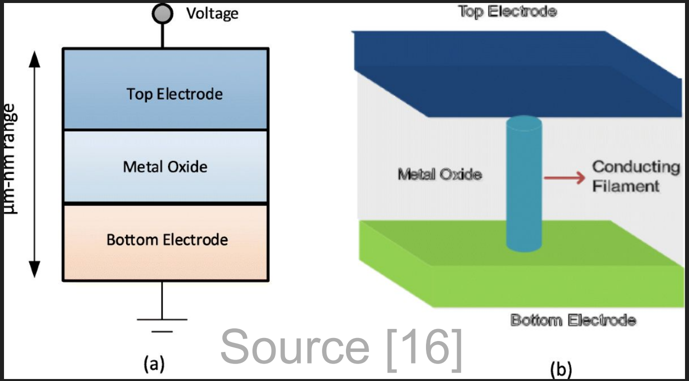
Compute-In-Memory

- MVM is performed within memory system
- Alleviates bottleneck between memory and CPU
- Works via multiply and accumulate process
- Many technologies for this system
  - Resistive Memory (ReRAM)

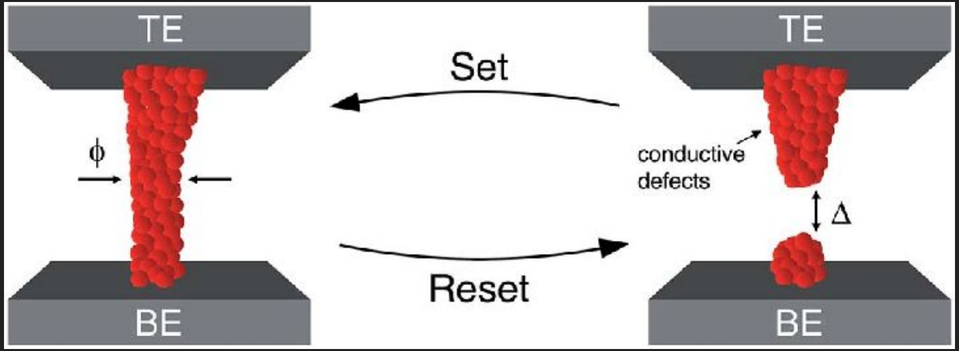


What is ReRAM?

- Non-volatile memory technology, that is presently used for storing memory
- Filament formation controls current flow
- High and low conductance states
- By abusing the ReRAM we can allow for analog computation



Source [16]

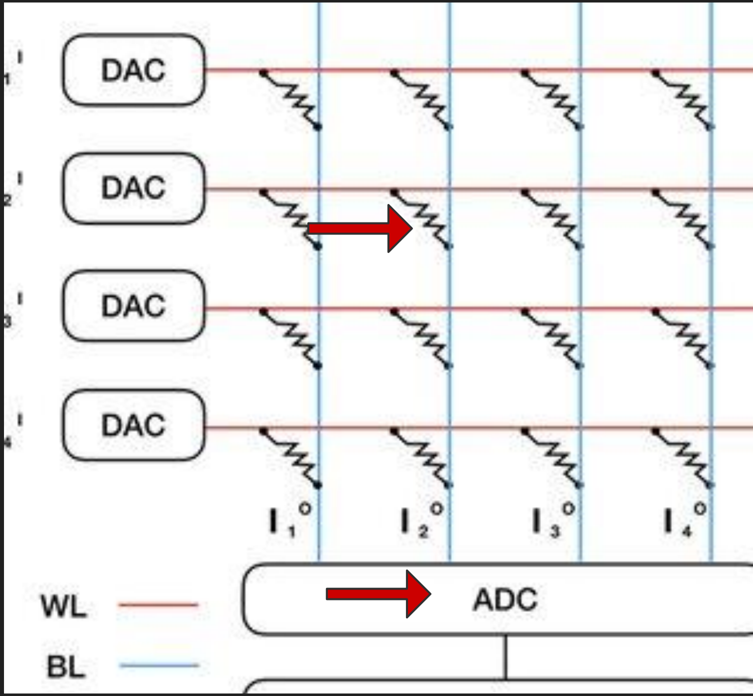


Source [4]

For more information about ReRAM see section 1.3 on the design document

Noise

- Since we are abusing ReRAM, there will be a lot of noise
  - Other components in the design that will cause noise in the designs include:
    - ADC (differential nonlinearity noise)
    - Bitline or word line
    - Transistor (minimal)
    - TIA
- Researchers need to better understand the characterization of noise



Source [17]

Our Solution

- Design a test chip with four unique ReRAM-based compute-in-memory architectures to allow us to understand how the noise will affect circuitry.
- Use Skywater 130nm process and Efabless tools for fabrication and tape-out

Source [2]

Source [3]

Users

Primary Users:

- ISU ECpE faculty and research teams (graduate and undergraduate).

Secondary Users:

- ChipForge: An Iowa State University club that started this semester
- Students in CPRE 4870/5870, EE 5260

Tertiary Users (Public):

- Public (which will further research in the field and help the field evolve)

For more information, see section 1.2 of the design document

Requirements

Requirements

- Four different ReRAM compute crossbar architectures
    - Different configurations of source, bit, word lines
    - Different peripheral circuitry
    - True crossbar vs. 1T1R grid
- Component circuits are individually characterizable and accessible through external analog I/O
- Uncertainty evaluation for implemented architectures
    - Crossbar noise
    - ADC noise
    - Design for worst case crossbar noise within one ADC step

For more information, see section 2.1 of the design document

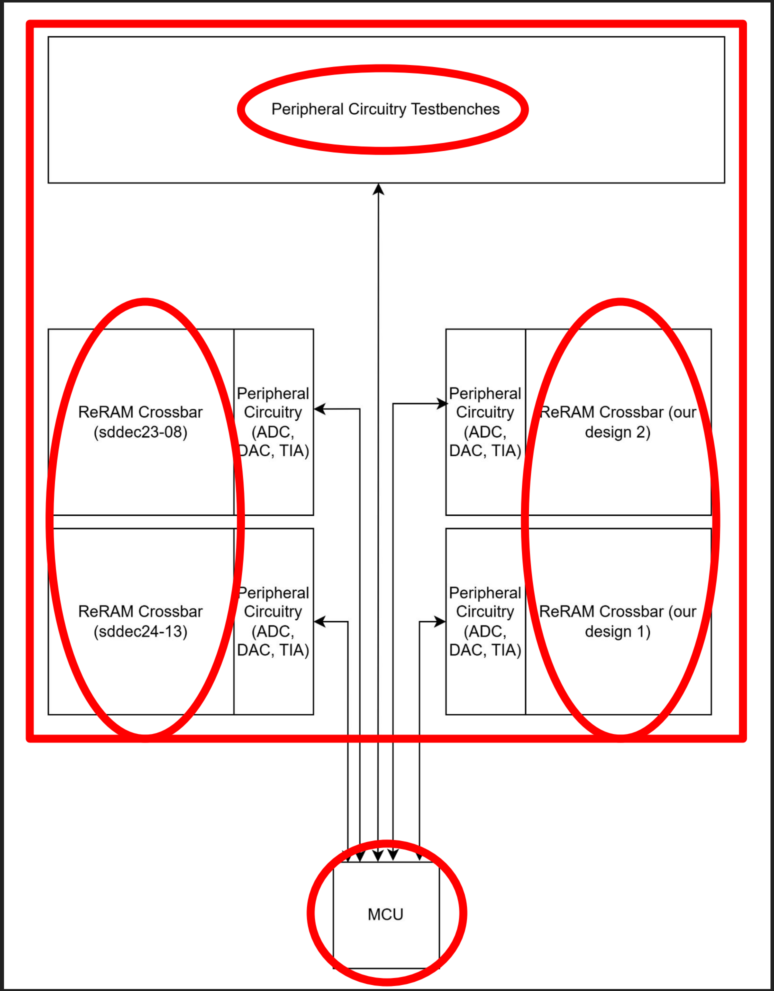
Requirements

- Bring-up Documentation and code for validation and debug
    - Characterizing the component circuitry via individual test benches.
    - FORMing the ReRAM cells
    - C Code for the MCU to interface with the ReRAM that enables testing and demonstrates that the ReRAM can compute an MVM within an epsilon tolerance.

For more information, see section 2.1 of the design document

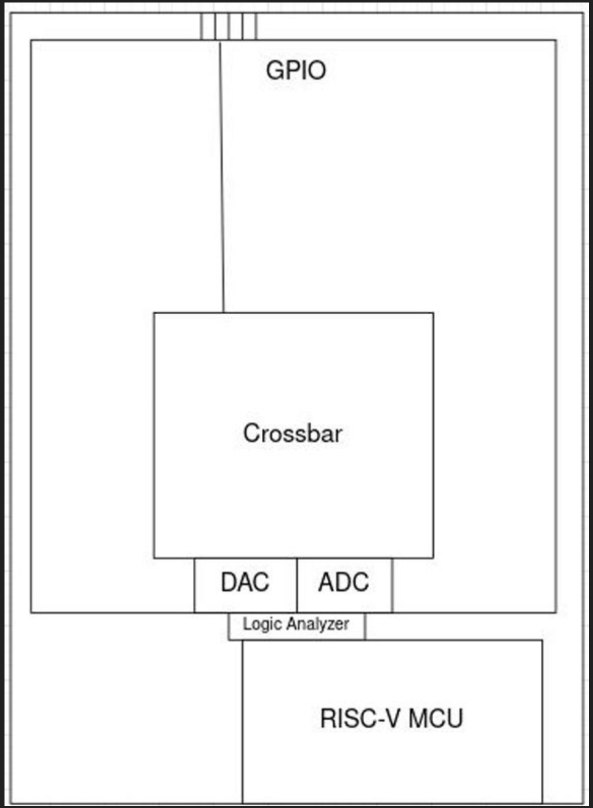
Designs

Top Level Design


Peripheral Circuitry Testbenches
ReRAM Crossbar (sddec23-08)
Peripheral Circuitry (ADC, DAC, TIA)
Peripheral Circuitry (ADC, DAC, TIA)
ReRAM Crossbar (our design 2)
ReRAM Crossbar (sddec24-13)
Peripheral Circuitry (ADC, DAC, TIA)
Peripheral Circuitry (ADC, DAC, TIA)
ReRAM Crossbar (our design 1)
MCU

Team sddec23-08

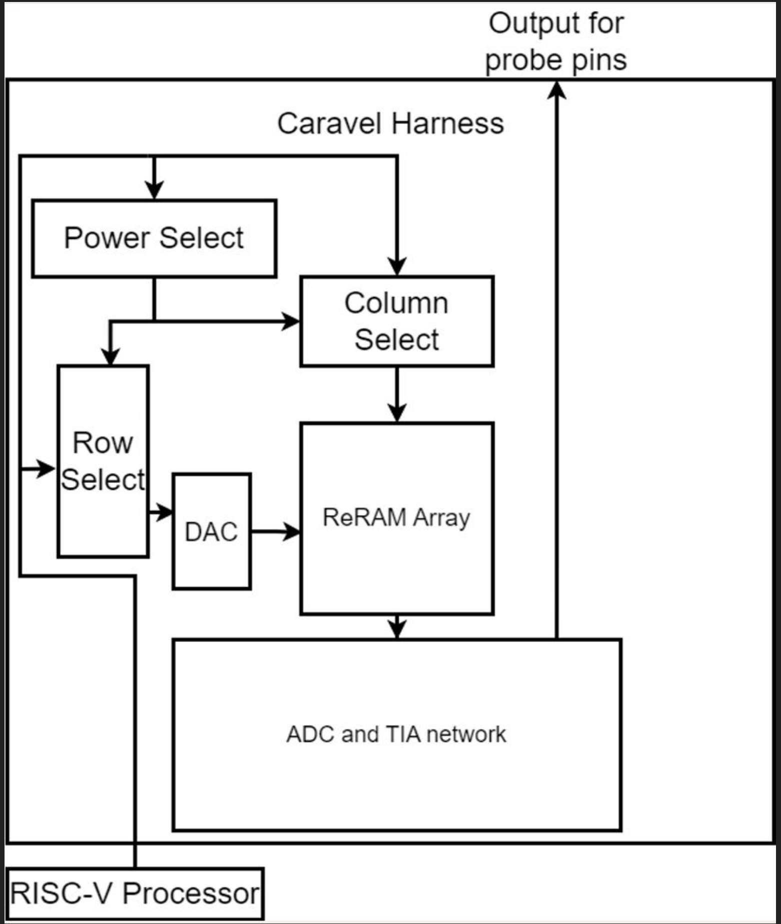
- Pioneered the tools used in this project
- Provided tool documentation
- ReRAM Architecture
  - One bit sample & hold ADC
  - DAC
  - TIA
  - Read from Source Line



Source [5]

Team sddec24-13

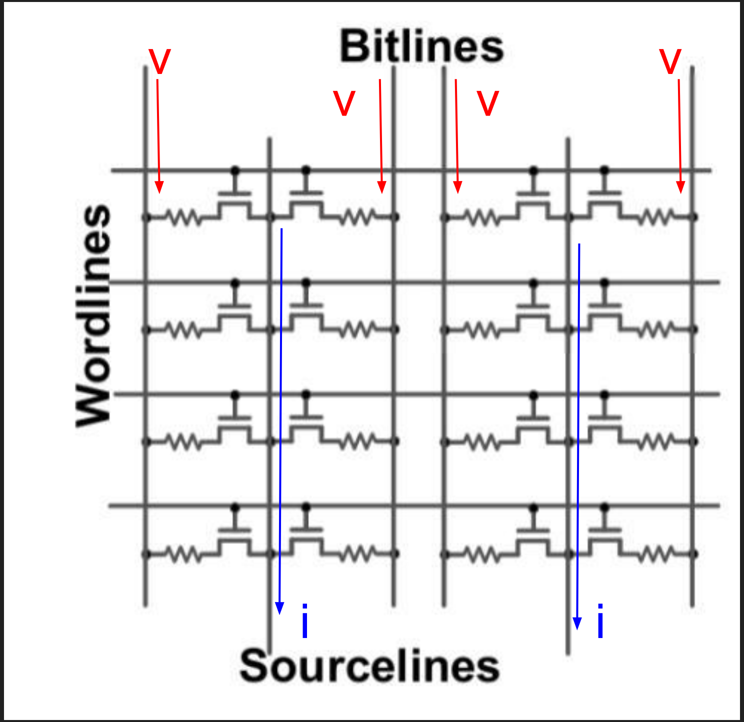
- Added to tool documentation
- ReRAM Architecture
  - Four bit flash ADC
  - DAC
  - TIA
  - Strong Arm Comparator
  - Read from Source Line



Source [6]

Our Team sdmay25-19

- New Architecture 1
  - Parallelize the source and bit lines
  - 4-bit ADC
  - Read from source line
  - Reduce noise on word lines

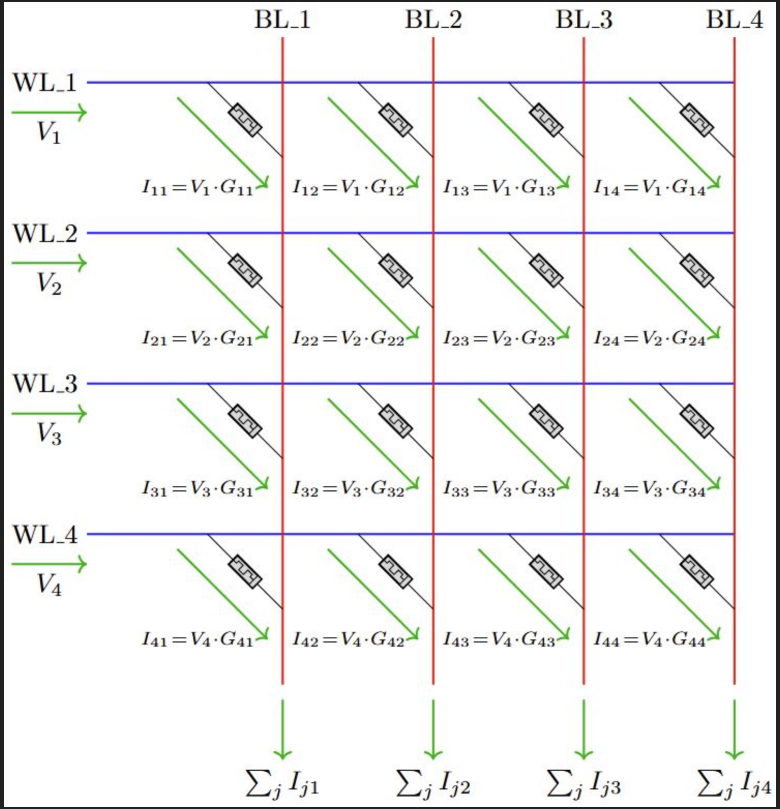


Source [7]

For more information, see section 6 on the design document

Our Team sdmay25-19

- New Architecture 2
  - True crossbar (no transistors)
  - 1-bit ADCs
  - Increases density
  - More susceptible to noise



BL_1
BL_2
BL_3
BL_4
WL_1
V_1
WL_2
V_2
WL_3
V_3
WL_4
V_4
$I_{11}=V_1 \cdot G_{11}$
$I_{12}=V_1 \cdot G_{12}$
$I_{13}=V_1 \cdot G_{13}$
$I_{14}=V_1 \cdot G_{14}$
$I_{21}=V_2 \cdot G_{21}$
$I_{22}=V_2 \cdot G_{22}$
$I_{23}=V_2 \cdot G_{23}$
$I_{24}=V_2 \cdot G_{24}$
$I_{31}=V_3 \cdot G_{31}$
$I_{32}=V_3 \cdot G_{32}$
$I_{33}=V_3 \cdot G_{33}$
$I_{34}=V_3 \cdot G_{34}$
$I_{41}=V_4 \cdot G_{41}$
$I_{42}=V_4 \cdot G_{42}$
$I_{43}=V_4 \cdot G_{43}$
$I_{44}=V_4 \cdot G_{44}$
$\sum_j I_{j1}$
$\sum_j I_{j2}$
$\sum_j I_{j3}$
$\sum_j I_{j4}$

For more information, see section 6 on the design document

Risks

- Taped-out chip does not perform as expected
  - Risk: 65%
- Past team’s design does not pass pre-check
  - Risk: 100%
- Integrated top-level wrapper design fails pre-check
  - Risk: 55%
- Flicker noise is more impactful on the Skywater process than expected
  - Risk: 15%

For more information, see section 3.5 on the design document

Risk Mitigation

- Dedicate time towards verifying and adapting past teams architectures
  - Pre-layout & post-layout simulation
  - Pre-check
- Perform thorough background research to inform design decisions
  - Create testbenches and detailed simulations
- Follow proper layout practices for a space efficient designs
- Increasing clock speed into the MHz region decreases flicker noise
- Reach tape-out readiness early
  - Talk with eFabless if necessary

For more information, see section 3.5 on the design document

Resources/Tools

XSCHEM

Source [8]

Magic

Source [9]

GTKWave

Source [10]

slack

Source [11]

- **Netgen**

For more information, see section 3.7 on the design document

Testing

Unit Testing Workflow

1. Create schematic testbench using XSchem
2. Simulate schematic using Ngspice
3. Create layout schematic using Magic
4. Extract SPICE netlist from layout
5. Simulate layout netlist using Ngspice
6. Perform LVS check using Netgen

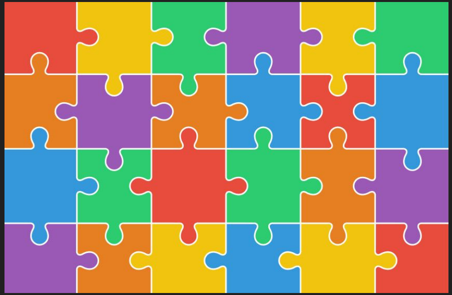
For more information, see section 5 on the design document

Other Testing Strategies

- Integration
- Regression Testing
- Acceptance Testing
    - Complete by **April 21st**

Source [12]

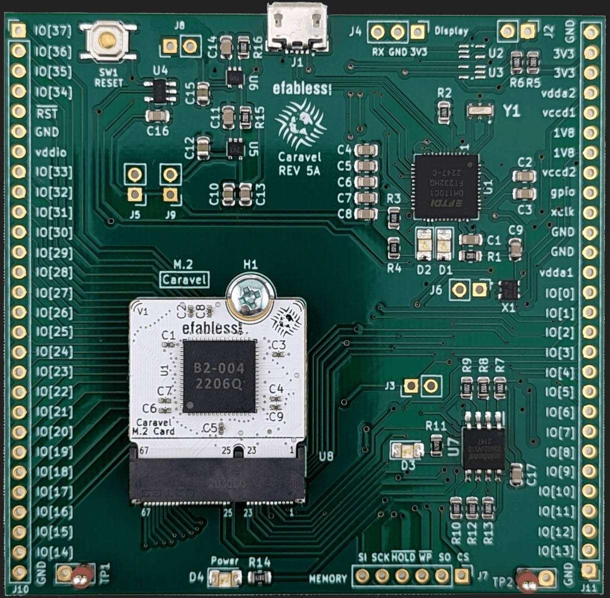
Source [13]


Source [14]

For more information, see section 5 on the design document

Bring-up Testing

- Provide testing strategies in bring-up documentation
- Properly form ReRAM cells
- Verify peripheral circuitry function
- Testing practices for each architecture


Source [15]

Results

- Testing work of team sddec23-08
  - Many components work
  - Some need updating
- Same process will occur with team sddec24-13 work

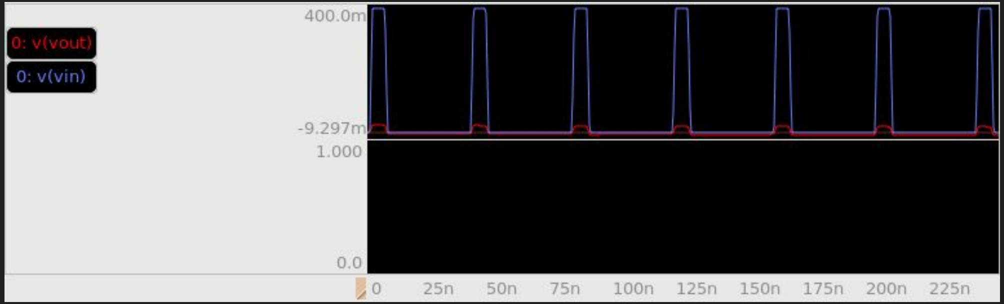


Figure 5.5: Resulting waveforms from buffer testbench from team sddec23-08

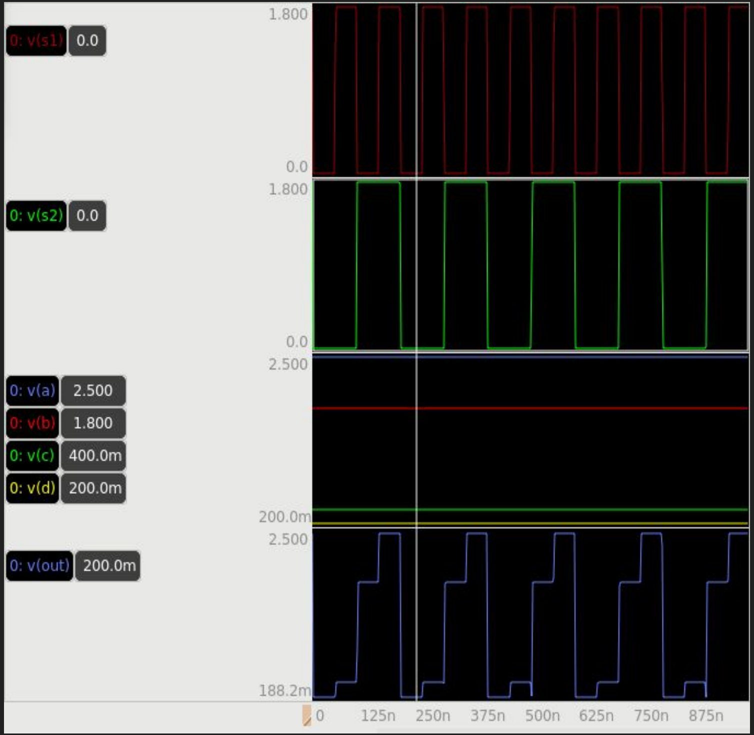
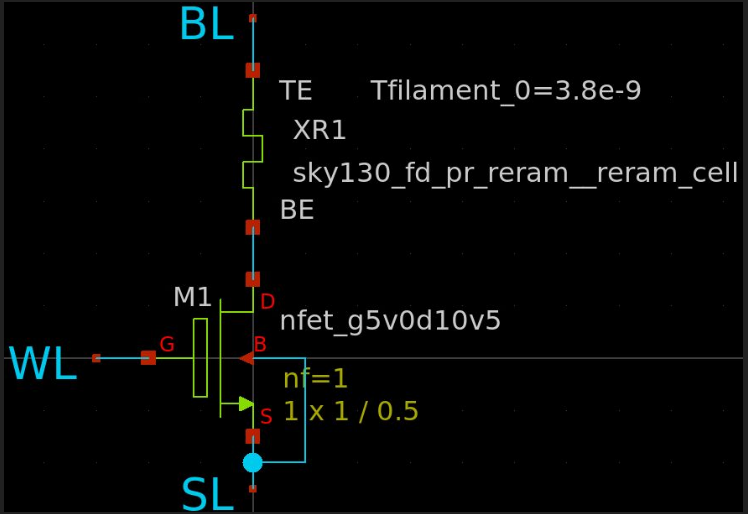


Figure 5.7: Resulting waveforms from 4-to-1 multiplexer testbench from team sddec23-08

For more information, see section 5.8 on the design document

Results

- We have also tested a 1T1R cell (pre-layout sim)
  - Perform Set and Reset Operations



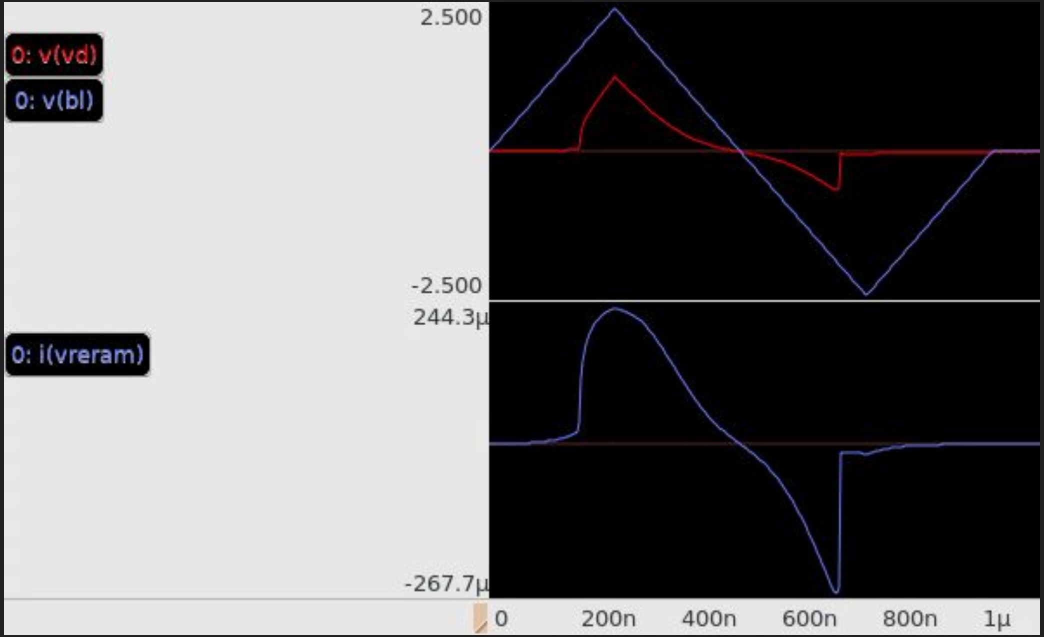
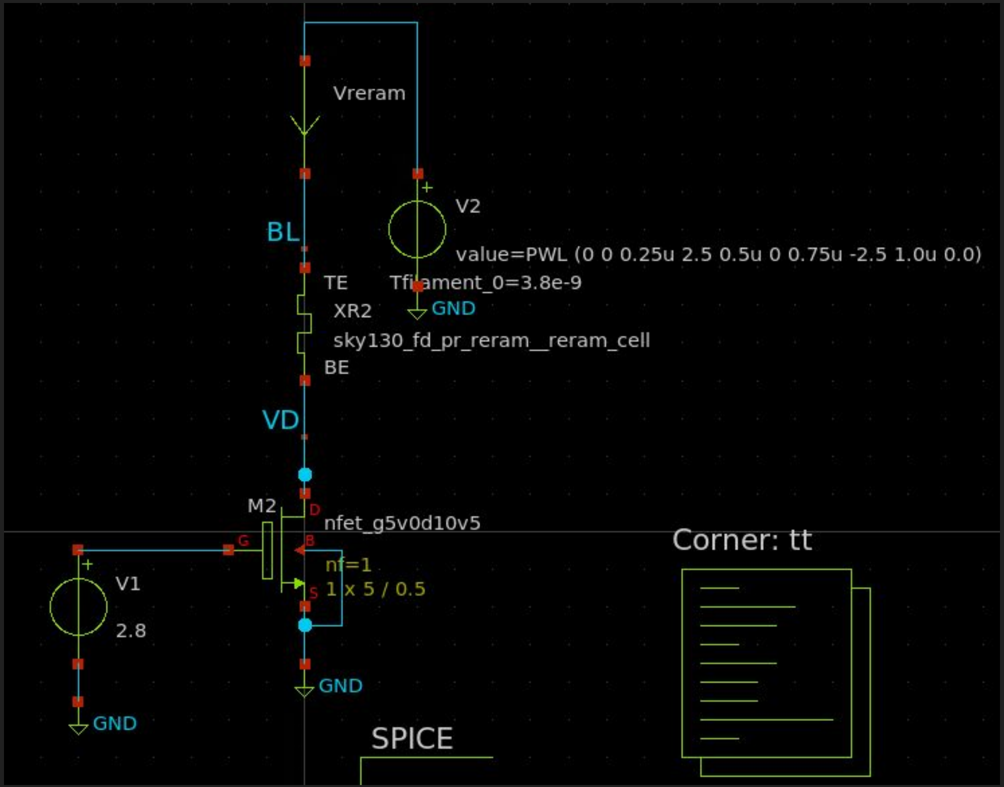
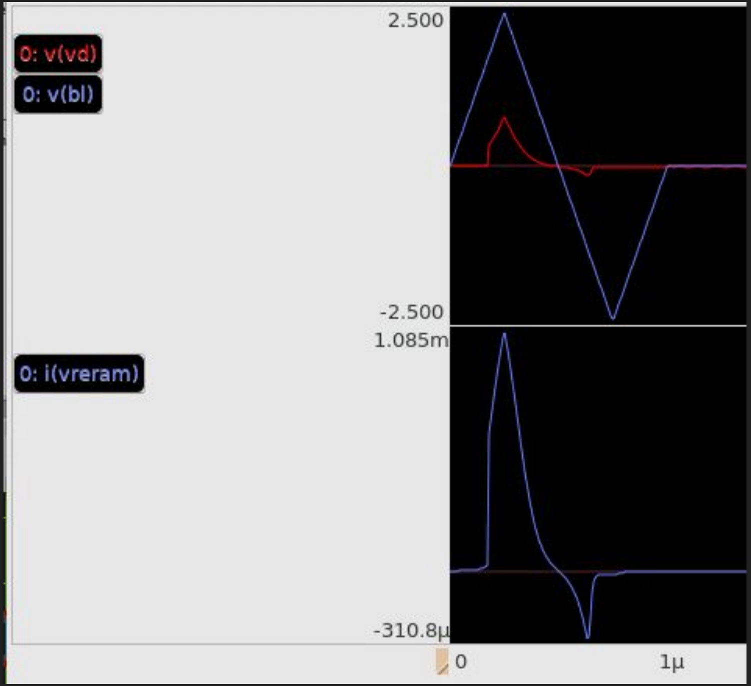


Figure 5.10: Waveforms from our 1T1R testbench

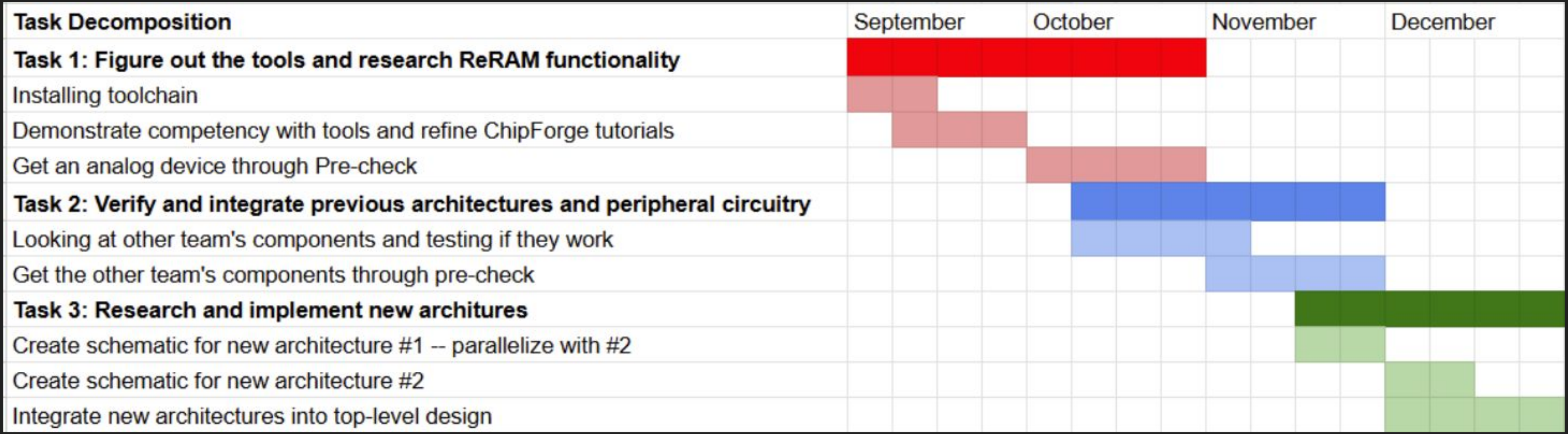
For more information, see section 5.8 on the design document

Results


Vreram
BL
V2
value=PWL (0 0 0.25u 2.5 0.5u 0 0.75u -2.5 1.0u 0.0)
TE
Tfilament_0=3.8e-9
GND
XR2
sky130_fd_pr_reram__reram_cell
BE
VD
M2
D
nfet_g5v0d10v5
G
B
nf=1
S
1 x 5 / 0.5
V1
2.8
GND
GND
Corner: tt
SPICE


2.500
0: v(vd)
0: v(bl)
-2.500
1.085m
0: i(vreram)
-310.8µ
0
1µ

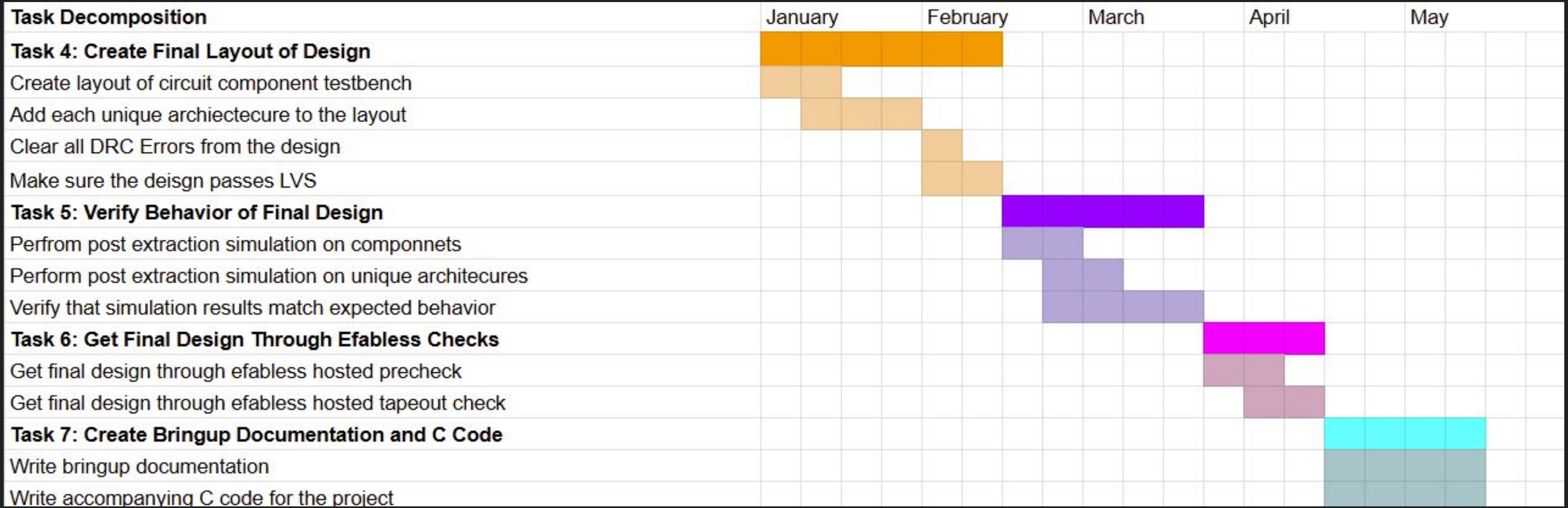
First Semester Timeline



Task Decomposition	September				October				November				December			
Task 1: Figure out the tools and research ReRAM functionality	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥								
Installing toolchain	🟥	🟥														
Demonstrate competency with tools and refine ChipForge tutorials		🟥	🟥	🟥												
Get an analog device through Pre-check					🟥	🟥	🟥	🟥								
Task 2: Verify and integrate previous architectures and peripheral circuitry						🟦	🟦	🟦	🟦	🟦	🟦	🟦				
Looking at other team's components and testing if they work						🟦	🟦	🟦	🟦							
Get the other team's components through pre-check									🟦	🟦	🟦	🟦				
Task 3: Research and implement new architures											🟩	🟩	🟩	🟩	🟩	🟩
Create schematic for new architecture #1 -- parallelize with #2											🟩	🟩				
Create schematic for new architecture #2													🟩	🟩		
Integrate new architectures into top-level design													🟩	🟩	🟩	🟩

For more information, see section 3 on the design document

Second Semester



Task Decomposition	January				February				March				April				May			
Task 4: Create Final Layout of Design	■	■	■	■	■	■														
Create layout of circuit component testbench	■	■																		
Add each unique archiectecure to the layout		■	■	■																
Clear all DRC Errors from the design					■															
Make sure the deisgn passes LVS					■	■														
Task 5: Verify Behavior of Final Design							■	■	■	■	■									
Perfrom post extraction simulation on componnets							■	■												
Perform post extraction simulation on unique architecures								■	■											
Verify that simulation results match expected behavior								■	■	■	■									
Task 6: Get Final Design Through Efabless Checks												■	■	■						
Get final design through efabless hosted precheck												■	■							
Get final design through efabless hosted tapeout check													■	■						
Task 7: Create Bringup Documentation and C Code															■	■	■	■		
Write bringup documentation															■	■	■	■		
Write accompanying C code for the project															■	■	■	■		

For more information, see section 3 on the design document

Conclusions

- Familiarize ourselves with analog toolflow
- Tested sddec23-08 components
  - Identified working & faulty components
  - Found that design does not pass pre-check
- Created own testbenches for 1T1R cell
  - Device operates in triode region
  - Demonstrates SET and RESET operations
- Planned two unique ReRAM architectures
  - Parallelize bit and source lines in 1T1R grid
  - True crossbar

Questions?

Image Sources

[1] https://www.linkedin.com/pulse/memory-bottleneck-ali-farahany/

[2] https://skywater-pdk.readthedocs.io/en/main/

[3] https://efabless.com/

[4] https://pubs.rsc.org/en/content/articlehtml/2019/fd/c8fd00106e

[5] https://sddec23-08.sd.ece.iastate.edu/final_presentation.pdf

[6] https://sddec24-13.sd.ece.iastate.edu/SDDEC24_13_DESIGN_DOCUMENT%20(2).pdf

[7] C. Xu et al., "Overcoming the challenges of crossbar resistive memory architectures," 2015 IEEE 21st International Symposium on High Performance Computer Architecture(HPCA), Burlingame, CA, USA, 2015, pp. 476-488, doi: 10.1109/HPCA.2015.7056056.

[8] https://marketplace.visualstudio.com/items?itemName=barakh.vscode-xschem-viewer

Image Sources

[9] http://opencircuitdesign.com/magic/

[10] https://commons.wikimedia.org/wiki/File:Gtkwave_256x256x32.png

[11] https://1000logos.net/slack-logo/

[12] https://www.dreamstime.com/photos-images/adding-puzzle-pieces.html

[13] https://www.vectorstock.com/royalty-free-vector/puzzle-pieces-together-vector-14398401

[14] https://www.vectorstock.com/royalty-free-vector/colorful-puzzle-vector-6821228

[15] https://info.efabless.com/chipignite-v2

Image Sources

[16] https://link.springer.com/article/10.1186/s11671-020-03299-9

[17]
https://www.researchgate.net/figure/Resistive-random-access-memory-ReRAM-crossbar-structure-with-bitlines-BLs-and_fig2_360786529