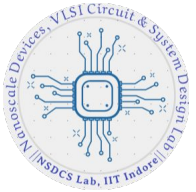
PROGRESS MADE SO FAR

February 1, 2022

Domains Explored

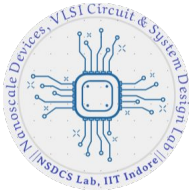
- In-memory Computing
- Memory Compilers

Key Takeaways: In-memory Computing



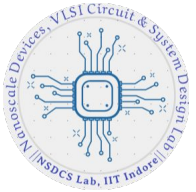
- **Need** - traditional Von Neumann architecture which separates the Memory Unit and the ALU suffers from long latency and high power consumption, especially in data-centric applications, due to the movement of data between the Memory and ALU. This is called the Von Neumann bottleneck or memory wall
- **TCAM** - A memory architecture that allows data search in one clock cycle by integrating the search circuitry within the memory, which performs bit-wise XOR/NOR operations between the search key and the stored data for matching and discharges the match-lines accordingly
- **ReRAM-based Memory** - A Resistive RAM device has a Metal-Metal Oxide-Metal structure, with a high resistance state (HRS) in the unbiased situation. When a voltage above the threshold is applied to it, ionization occurs and electricity is carried along by the oxygen ions formed, creating a low resistance state (LRS). They can have either unipolar or bipolar switching.

Key Takeaways: In-memory Computing



- Familiarity with 6T SRAM and its operations, and terms such as TOPS/W, TOPS/mm^2, bit-line boosting, write margin, static noise margin, sensing margin, sleep transistors, MTCMOS, pass gate, CMOS transmission gate, tie-cell, differential non-linearity, process corners
- Some concepts are still not clear, such as read-disturb, body-bias control, STTRAM, half-selected cells, Wallace tree adder, strongARM latch comparator, cascode, world-line under-drive, bit-line regeneration, column-interleaved SRAM, WB delay, decoupled ports, hierarchical bitline, negative cell virtual ground, etc.

Key Takeaways: Memory Compilers



- **Need** - to automate generation of larger memories of different types and configurations based on the parameters supplied by the user, such as word size, memory size, multiplexing, banking, architecture, technology node
- **Basic components of memory** - memory array, decoder, read/write circuit, timing circuit
- **Memory scaling approaches** - banking (increases columns or word size by adding another array with the same decoder inputs) and multiplexing (increases rows or memory size by adding another array with same read/write connections, selected using a MUX)
- **Plugin-based Design** - memory architecture and technology node are provided as plugins, so that the relevant files can be plugged into the program as required. This makes the compiler independent of the architecture and the technology node

References

In-memory Computing

- Y. Chen, L. Lu, B. Kim and T. T. -H. Kim, "**Reconfigurable 2T2R ReRAM Architecture for Versatile Data Storage and Computing In-Memory**," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 12, pp. 2636-2649, Dec. 2020, doi: 10.1109/TVLSI.2020.3028848.
- Y. Chen, L. Lu, B. Kim and T. T. -H. Kim, "**A Reconfigurable 4T2R ReRAM Computing In-Memory Macro for Efficient Edge Applications**," in IEEE Open Journal of Circuits and Systems, vol. 2, pp. 210-222, 2021, doi: 10.1109/OJCAS.2020.3042550.
- V. T. Nguyen, J. -S. Kim and J. -W. Lee, "**10T SRAM Computing-in-Memory Macros for Binary and Multibit MAC Operation of DNN Edge Processors**," in IEEE Access, vol. 9, pp. 71262-71276, 2021, doi: 10.1109/ACCESS.2021.3079425.

- L. Lu, T. Yoo, V. L. Le and T. T. Kim, "**A 0.506-pJ 16-kb 8T SRAM With Vertical Read Wordlines and Selective Dual Split Power Lines**," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 6, pp. 1345-1356, June 2020, doi: 10.1109/TVLSI.2019.2956232.
- K. Lee, J. Jeong, S. Cheon, W. Choi and J. Park, "**Bit Parallel 6T SRAM In-memory Computing with Reconfigurable Bit-Precision,**" 2020 57th ACM/IEEE Design Automation Conference (DAC), 2020, pp. 1-6, doi: 10.1109/DAC18072.2020.9218567.

Memory Compilers

- R. Goldman, K. Bartleson, T. Wood, V. Melikyan and E. Babayan, "**Synopsys' Educational Generic Memory Compiler**," 10th European Workshop on Microelectronics Education (EWME), 2014, pp. 89-92, doi: 10.1109/EWME.2014.6877402.

PROGRESS MADE SO FAR

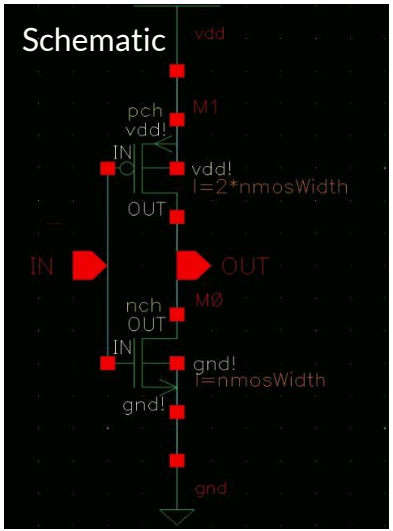
February 8, 2022

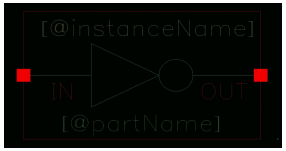
Domains Explored

- In-memory Computing
- Memory Compilers

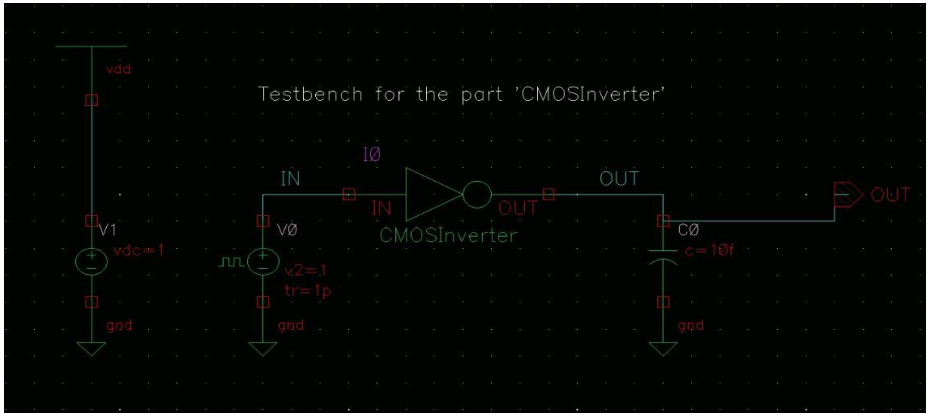
Progress Made

- **Learned Cadence -** Learned the basic concepts such as the design of schematic, symbol, layout, their validation, accounting for parasitics, parametrization, and simulation types
- **Tried a Cadence example** - Tried designing a CMOS Inverter on Cadence including schematic, symbol, and a test bench, containing a few parameters.

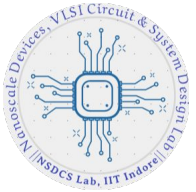




Symbol



Progress Made



- **Question** - What is a ‘via’ in layout design

- Timeline
  - **1st week of Feb** - Learning Cadence and Starting Implementing Components
  - **2nd week of Feb -** Decoding OpenRAM’s Source Code
  - **End of Feb -** Compiling all the components, code, and understanding how to join the pieces together
  - **March -** Solving issues faced before, and working on our memory compiler
  - **April -** Writing a paper

- **Next Step** - Decoding OpenRAM’s Code

PROGRESS MADE SO FAR

February 15, 2022

Progress Made

- **Learned Physical Design Theory -** Attended a workshop detailing 'Physical Design of ASICs,' which gave a holistic overview of the design process of ASICs, with a thorough discussion of physical design and layout.

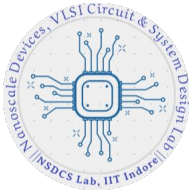
- **Resource Compilation** - Searched and compiled resources on openRAM which contained details on the implementation and file structure of the code. Also retrieved the first version of openRAM for better understanding the code.

- **openRAM Publication** - Went through the paper by the developers of openRAM, which gave a systematic overview of the implementation. The paper also mentioned a memory compiler named FabMem, which uses Cadence.

Progress Made

- **Understood some of the code -** Top-down approach, decoded the globals.py and the options.py files successfully, which are used to provide the user interface, store options, and set paths for appropriate directories along with a number of small checks and modifications

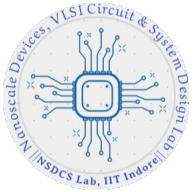
Key Takeaways: openRAM Implementation



- **Structure** - The front-end consists of the compiler (which generates SPICE model and GDSII files) and the characterizer (generates SPICE simulation results), while the back-end consists of programs and scripts to generate annotated timing and power models.

- **File Hierarchy** - The high-level classes such as openRAM.py, sram.py, bank.py, and globals.py coordinate and control the execution of the program and other classes. The block classes such as sense_amplifier.py and sense_amplifier_array.py have information about the base cell and tiling of the cell, respectively. Finally, low-level classes pertaining to parametrized transistors and logic gates are also present.

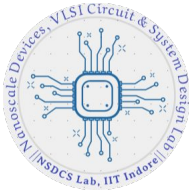
Key Takeaways: openRAM Implementation



- **Technology Independence** - A technology directory which contains the technology specific parameters such as technology information, design rules, and standard cell designs is employed to make openRAM portable across technologies

- **Standard cells used** - The following cells are pre-designed in each technology:
  - 6T cell
  - sense amplifier
  - master-slave flip-flop
  - tri-state gate
  - write driver

Progress Made



- **Question** - What is bisection search?

- Timeline
  - **1st week of Feb** - Learning Cadence and Starting Implementing Components
  - **2nd week of Feb -** Decoding OpenRAM's Source Code
  - **End of Feb -** Compiling all the components, code, and understanding how to join the pieces together
  - **March -** Solving issues faced before, and working on our memory compiler
  - **April -** Writing a paper

- **Current Step** - Decoding OpenRAM's Code

References

openRAM

- M. R. Guthaus, J. E. Stine, S. Ataei, Brian Chen, Bin Wu and M. Sarwar, "**OpenRAM: An open-source memory compiler,**" *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016*, pp. 1-6, doi: 10.1145/2966986.2980098.