Open-Source ReRAM-based FPGA

Po-hsun Wu, Paul Danciu, Paul George
Group3: EECS627
University of Michigan Ann Arbor, MI
{pohsunwu,ptdanciu,egpaul}@umich.edu

***Abstract*—A *Resistive Random Access Memory* (ReRAM) device can store information in vertically-integrated BEOL resistor. The application of ReRAM-based *Field Programmable Gate Array* (FPGA) structure is thus proposed in this project. FPGAs architecturally use heavy amounts of memory cells, we propose a ReRAM-based FPGA (built with open source tools) that will attempt to improve on existing approaches in literature (7-15% area reduction [1]) while improving or maintaining the performance and power.**

***Index Terms*—ReRAM, FPGA, Skywater, Open-Source, OpenFASOC OpenROAD, OpenFPGA, Full Custom Digital Design**

I. Introduction

Field Programmable Gate Arrays (FPGAs) are versatile devices that have enabled among other things, greater testing and validation of high performance Digital circuits, Reconfigurable computing, High-Frequency trading, and high bandwidth control systems / Networks, and other applications where latency reigns supreme but deployment numbers/ reconfigurability requirements make ASICs economically unsuitable. FPGAs are architecturally simple devices consisted by a matrix of *Configurable logic Blocks* (CLBs) and a programmable interconnect fabric and switch blocks. These programmable elements are typically implemented with CMOS latches. Technologies such as ReRAM (Resistive Random Access Memory) offer good potential in reducing device footprint as it is a *Back-End-of-Line* (BEOL) process.

New Open-Source EDA tools such as OpenFASOC [2], allow for the rich specification and customization of the layouts of circuits enabling us to build parameterized high-density tile-able FPGA macro-cells.

We draw heavily on prior work done by the open source electronic design community, most notably efforts from [3] [4], which also concurrently developing ReRAM eFPGA IP on Skywater-130, and seek and approach the same leveraging OpenFASOC to what extent possible.

A. Conventional FPGA Structure

FPGAs are types of integrated circuit built with configurable and programmable blocks to be customized for a user-specific application. The conventional FPGA structure consists of three types of modules: *Configurable logic blocks* (CLBs), programmable Interconnect, and *Input / Output* (I/O) blocks. The layout structure is formed by an array of tiles surrounded by I/O blocks, and each tile consists of CLBs and interconnect components. An individual CLB contains multiple *basic logic elements* (BLEs) interconnected by multiplexers. A single BLE consists a *Look-up table* (LUT) and flip-flops to hold the data for sequential logic purposes. LUT is used to configure any function with its variables equal or less to the input of LUT, hence a K-input LUT outputs one bit out of the 2^K possibilities. A conventional SRAM-based implementation requires 2^K SRAM cells and 2^K multiplexers to select the demanded information stored in the configured SRAM cells. The programmable routing interconnect, also known as routing fabric, laid the connection between CLBs, and contains programmable switches to control the configured routing. In the island-style architecture, a routing interconnect is divided into *Switch Matrix* (SM) and *Connection Blocks* (CBs) [3]. Switch Matrices are placed at the intersection of vertical and horizontal connections and each consist of a matrix of *switch blocks* (SBs) implemented in various topologies [3]. CBs provide the connection between the input and output channels of CLBs.

We will be using a memory bank structure as the base protocol to program our FPGA. A memory bank circuitry is organized as an array of configurable ReRAM memory elements. Each element can be accessed through a unique address to the Bit-line/Word-line decoders. We will be using 1 of these memory banks across the fabric.

B. ReRAM-based Memory Array

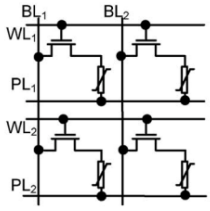

Fig. 1: Generic 1T1R Array unit

1T1R ReRAM array is the most well-known approach in creating large ReRAM storage devices, amenable for scaling from small 1-bit element to storage class arrays, and has been thoroughly explored in literature. 1T1R structures however have implicit driver circuitry required for each and bias point (Fig. 1) which makes it unsuitable to implement as an array for small highly repeating structures such as 16 element 4 input LUTs, we are exploring circuit solutions that can minimize this ancillary circuit repetition.

II. 1P1R-ReRAM cell

Exploring the sky-water-130 Re-Ram IP we characterized the given Verilog-A Model and obtained the following bias dependent state characteristics.

Operation	Transition	V Bias (Top, Bottom)
Form	Pure $\rightarrow LRS$	+ 2.4V
Reset	LRS $\rightarrow HRS$	- 1.2V
Set	HRS $\rightarrow LRS$	+ 2.2V
Readout	No Change	$\leq \pm 0.8V$

TABLE I: ReRAM Operation

Table.I highlights requirements inferred from characterizing Foundry provided Verilog-A model in order to be able to reliable perform operations on the state of ReRAM Storage element.This Verilog-A Model and foundry documentation did not contain any model parameter variability data, and independent work characterising the technology as well reported significant variation in device to device LRS(low resistance state) and HRS(high resistance state) resistance values [5].We therefore settled on a Maximum readout voltage of $\leq \pm 0.8V$.

We developed the 1P1R-ReRAM Cell to meet the device operating conditions and our design goals. to safely handle the higher voltages involved in the manipulation of the ReRAM device state we created high voltage PFETs following the PDK High Voltage Design rules [6] sized to a valid bin (of pfet device sky130_fd_pr_pfet_g5v0d10v5) .

A. Circuit

Fig. 2 outlines our 1P1R ReRAM Cell & Table II outlines important circuit operating points (simulated Fig: 3) , compared to a traditional 1T1R Array this requires only 2 global nets to program an individual cell(explored in further detail in section driver). The Re-Ram State is readout as the voltage drop across the device under the application of the weak readout voltage. In the high resistance state the V_{OUT} is $V_{READOUT}$ and in the LRS state it is $V_{READOUT} - V_{DIODE}$ a biased inverter (larger PMOS) amplifies this small voltage difference such that subsequent logic devices can resolve this. but herein lies the drawback of this readout technique in this inverter being partially turned on and hence incurring a large static leakage current(Fig4).

Operation	VWL	VBL	Current Path
Form	0V	$\geq 2.6V$	HRS , FB Nwell Diode
Reset	$\geq 2V$	0V	PMOS R_{DS} ON, LRS
Set	0V	$\geq 2.4V$	HRS , Nwell Diode
Readout	$\leq 1V$	0V	LRS/HRS + Weak FB Diode

TABLE II: 1P1R Circuit Operation

Layouts of a 4X ReRAM device (tileable) unit cell was made, with devices sharing a word-line also sharing a HV Nwell (Fig. 5).

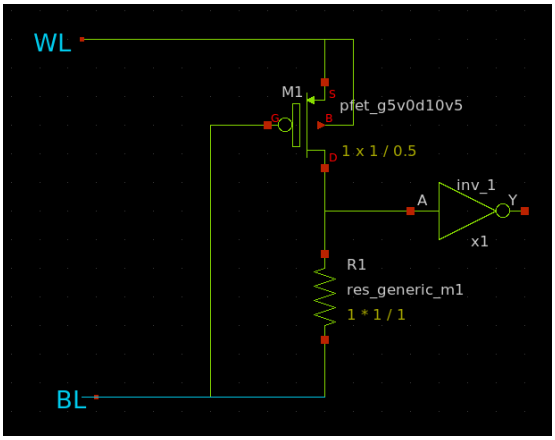

Fig. 2: 1P1R ReRAM Cell : {WL,BL}

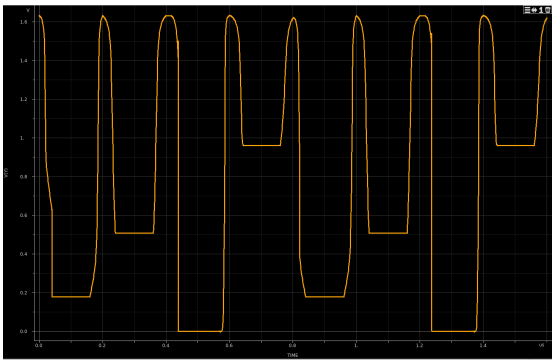

Fig. 3: 1P1R : Form-Read-Reset-Read-Set-Read-Reset-Read

B. Write Driver

A Driver was developed to program the 1P1R Cells when in an Array. It employs high voltage tri-state inverter cells from the sky130_fd_sc_hvl library. using multiple levels of pairs of positive and negative enable inverters achieves safe multiplexing between the various voltages the array bit and word-lines have to be driven to. When bits in a word-line are being written to, all other word-lines are undriven (high impedance). Since Cells are "Programmed" only when bit-lines and word-lines are strobed to opposite poles, only one

Cell	Area (4Cells)
PDK High Density D Latch	10.88 * 5.22 μm^2
ReRAM Cell	10.88 * 3.24 μm^2

TABLE III: Cell Density

EN	Mode	Data	BL_{SEL}	WL_{SEL}	BL	WL
1	PROG	0	0	V_{SET}	{Z,0}	Z
1	PROG	1	V_{RST}	0	{Z,V_{RST}}	Z
1	READ	NA	NA	NA	0	V_{RD}
0	NA	NA	NA	NA	0	0

TABLE IV: Write Driver Operation

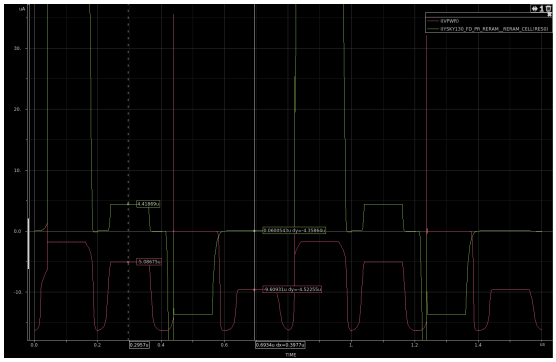

Fig. 4: 1P1R Readout Current (Pulse 2,4) : {Green Source:Buffer (1.8V) , Red : Source WL (1V)}

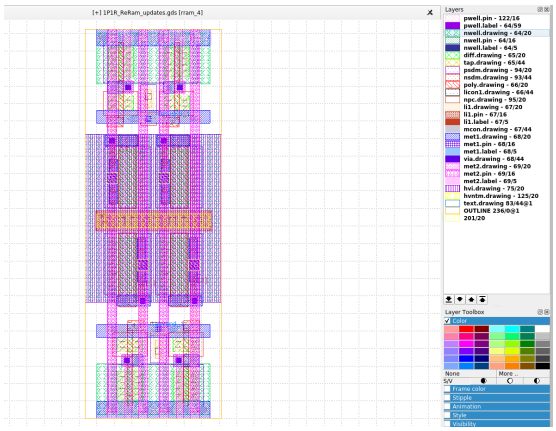

Fig. 5: 1P1R x4 Cell

state either ”0” or ”1” may be written to selected bits that share a word-line simultaneously requiring two bit-line / word-line strobes for any whole bit-line write. writing to cells one at a time does not see any difference, requiring non participating bit-lines and word-lines be undriven (Table.IV).

We were unable to at the time of writing this report verify the full functionality of the large 4X 1P1R cell in spice.

III. Implementation of Architectures

RTL-to-GDS programs are necessary in making an integrated circuit. Both the ReRAM-based and the baseline architecture are implemented by OpenFASOC, which uses open-source RTL-to-GDS flow tools, OpenLane and *OpenROAD flow scripts* (ORFS) to generate a synthesizable analog circuit.

A. OpenFPGA K4N4 structure

The top-level architecture is based on the K4N4 FPGA architecture generated by OpenFPGA. K4N4 architecture is a homogeneous FPGA architecture, and weaved by multiple tiles each consisting of one CLB and interconnected through CBs and SMs. Each CLB is made up of 4 BLEs. Each BLE contains a 4-input LUT, a 2-to-1 multiplexer, and a Flip-flop. Each CLB is surrounded by 4 CBs, and horizontal and vertical-oriented CBs are connected by Wilton-Style SM.

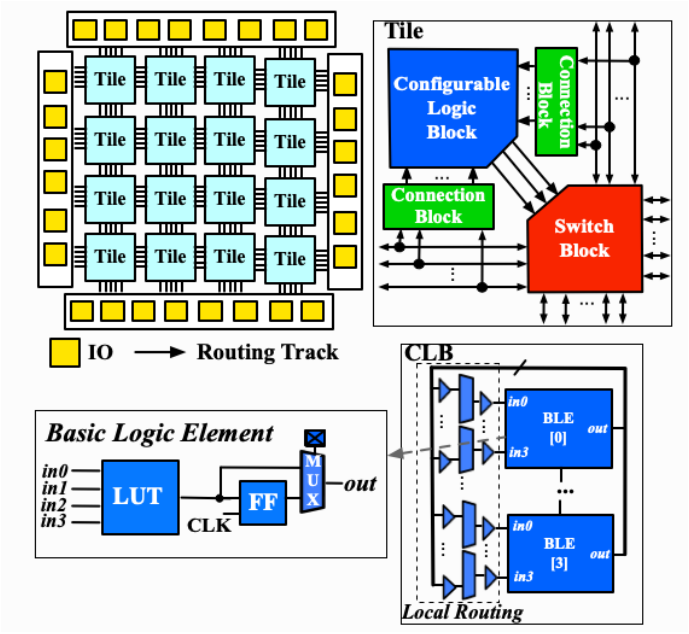

Fig. 6: K4N4 FPGA architecture [3].

B. Baseline Architecture

In the verilog format, the architecture can be imported in OpenFASOC directly, albeit require some manual adjustment in cases of incompatible format between the tools. As all submodules heavily relied on SRAM, which are not separately built by memory compiler, the open-source synthesis tool, yosys, default all SRAMs as D-type latches causing the massive overhead on baseline area estimation.

Since the layout generation is focused across the principle of maximizing density, thus we have also implemented a fully-custom manual-placed layout of BLE, shown in Fig.7 to maximize the density and eliminate some issues on RePlace, the open-source global placement tool. Contradictory to what we speculated, the implementation of manual-placed BLE does not achieve better area density in the generation of the single-tile baseline than automatically-placed layout due to the higher routing congestion causing by high density of bit-lines and word-lines access the single BLE cell. Hence, the final baseline layout for comparison is selected to be the automatically-placed version to give out more insightful detail.

A couple unsolved challenges are affecting the functionality of the design. LVS has failed even with the lower hierarchy of the implementation, it might be due to the incompatible source verilog codes, consisting of both behavioral and structural verilog format.

C. ReRAM-based Architecture

ReRAM-based architecture replace the use of default individual, unconnected memory cells with the customized ReRAM memory-row. The single memory-row has a single shared word-lines and 28 bit-lines and 28 data outputs, the dimension has height of precisely four rails, allowing us to directly integrate with the current power delivery network.

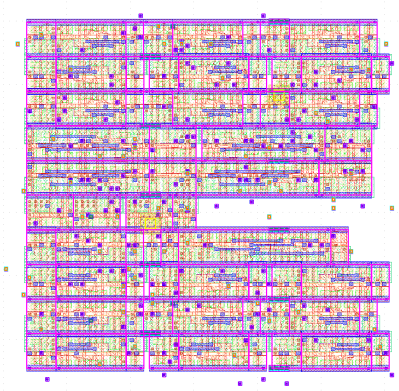

Fig. 7: Baseline Manual-placed BLE

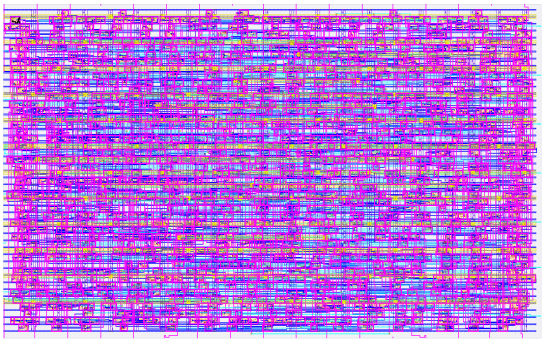

Fig. 8: Baseline Automatically-placed Single-tile FPGA

we have experimented with different placement strategy to maximize the density and reduce and routing congestion. The best area result is constructed with ReRAM memory-rows stacking in five columns each consisting of six rows. The even distribution of rows and columns should generate higher-density layout with less congestion. All memory-rows are 15.72μm horizontally and 4 rails (10.88μm) vertically apart from another.

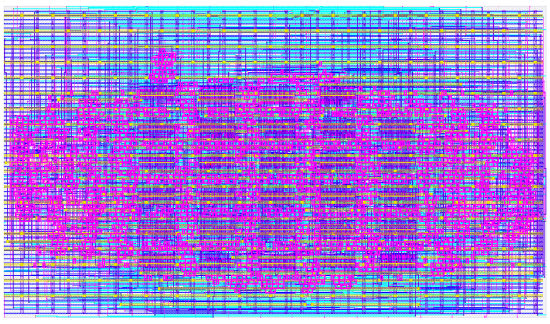

Fig. 9: Final ReRAM-based Single-tile FPGA

Fig. 9 shows the final ReRAM-based FPGA layout, the ReRAM memory-rows are placed in the middle and surrounded by standard cells. A substantial amount of area wasted by routing congestion can also be observed at the top and bottom of the layout.

IV. Performance

The Performance of the current 1-Tile baseline design is summarized in the table below.

	Baseline	ReRAM-based
Area	27300μm^2	70000μm^2
Density	46%	30%
Power	1.65$e^{-4}W$	2.13$e^{-4}W$

TABLE V: Performance Estimation of Baseline design

The area of the baseline FPGA architecture is the main indication of the project. The use of ReRAM memory rows does not yield higher density due to the high routing congestion and unplaceable halo and blockage areas. ReRAM cell does improve the memory cells density but does not improve the overall routing congestion. Different style of ReRAMs placement may yield better result, such as breaking the entire memory row into little chunk, but it also generate a larger total area of blockage and halo.

V. A Full Chip Floorplan

We will be implementing our chip floorplan using Efabless Caravel "harness" SoC. The Caravel chip is a ready-to-use test harness for creating designs with the Google/Skywater 130nm Open PDK [7]. Caravel consists of a harness frame, plus two wrappers; one for management area and one for user project area.

The Harness envelops both the management and user project wrappers, and main purpose is the handle the chip's I/O. It contains a clocking module, DLL, User ID, SPI, POR, and GPIO control.

The Management Area includes a RISK-V SOC, and a number of peripherals, like timers, UART, and GPIO. The management area will be used to configure User project I/O pads, Observe and control User project signals, and Control the User Project power supply. The Management Area also implements SRAM for the management of SoC.

The User Project area is around 2.92mm X 3.52 mm and has 38 I/O pads, and 4 Power pads. This area will be used to place our FPGA Fabric. Our goal is the use the RISK-V SOC from the Management Area to program the Fabric, and interface with it.

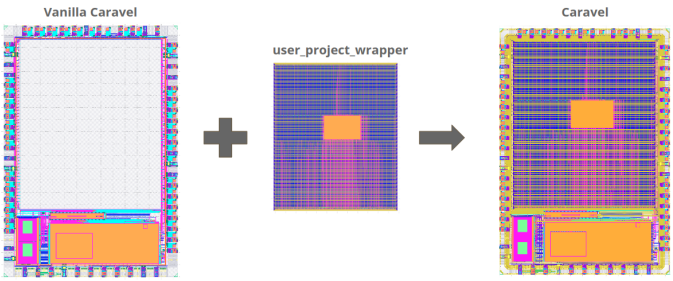

Fig. 10: Caravel floorplan integrated with example user project. [7]

For our final design we used the wishbone interface to allow the CPU to program and read from the FPGA. Wishbone is an open source computer hardware bus designed for connecting different cores to each other. To program our FPGA, the processor(Wishbone Master) sends a write request to our FPGA (Wishbone Slave) by writing to a set of addresses reserved for the user space (between 0x30000000 0x3000000C). The slave will receive the request and write the data to a set of local registers that map to the inputs to our FPGA fabric, and send back an acknowledgement. The processor can also read the values of these registers, as well as the FPGA's output by sending a read request.

We were at the time of writing this report unable to complete integrating our custom layout FPGA + wishbone slave + driver cells with the caravel framework with OpenLane.

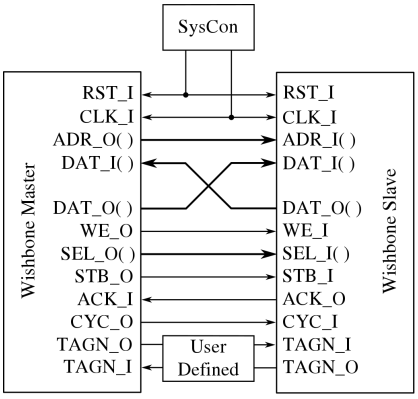

Fig. 11: Wishbone's Master and Slave Interfaces

VI. Verification and Test Plan

OpenFPGA Auto generates test benches to validate the correctness of their generated FPGAs. Their full testbench verifies in two stages. In the configure phase a bitstream is loaded to program the FPGA. Then in the operation phase the functionality of the programmed fabric is tested with random stimuli. We will use these testbenches to verify the functionality of our FPGA once the SRAM Cells in the design are replaced with our custom RRAM cells. So far we have successfully verified the correctness of our Baseline FPGA without RRAM cells.

To verify the functionality Wishbone interface, we can program the CPU and run RTL simulations using the Caravel interface. This is done by writing C code which is then compiled and programmed onto the CPU. A verilog test bench can then be used to monitor different signals to verify their functionality.

So far We are able to verify individually the SRAM functionality FPGA, and the Wishbone slave registers can be written to and read from. Due to time constraints however, we were unable to verify the combined functionality of our whole system. The main issue was that in order to program the FPGA 790 writes must be made through the wishbone interface to program each individual memory cell. There is not enough memory on the chip to hold all of those addresses and data. Further work would be needed to compress that data in a way that fit in program memory.

We planed to for signoff run a combined mixed signal simulation of the Digital (Caravel + FPGA controller) and the FPGA but were unable to setup the Mixed Signal Xyce VPI with iverilog, necessary to complete this step.

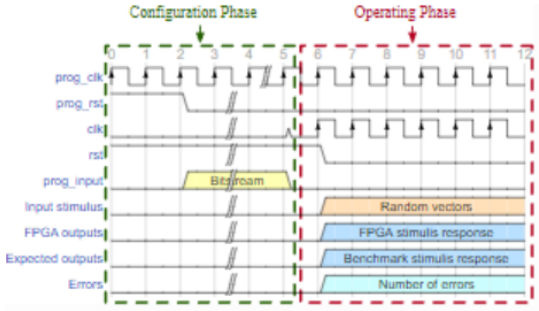

Fig. 12: An illustration of the waveforms of the full testbench

VII. Conclusions

The proposed Open-Source ReRAM-based FPGA structure can theoretically achieve better density than competing SRAM (and arguably ReRAM) solutions while retaining the performance of a standard conventional SRAM-based FPGA. Hand placement was explored and demonstrated the potential to optimize density even more. Our 1P1R ReRAM Cell while area efficient has poor Runtime Static Leakage, and because of poor/no variability information may not have useful yield. We in hindsight believe it to be possible to develop a ReRAM-latch with 2 more minimum sized transistors, that would eliminate the power draw problem the expense of area parity with a D-latch. Newer 1S-1R technology from Skywater EDA and WeeBit Nano hold great potential to eliminate much of the complexity of needing a dedicated HV-PFET and is promising to be the technology of choice to revisit creating high density efpga IP. The of the layout of these circuits was done in an area optimized manner with openFASOC in line with our broader global tile-ability constraints.

References

[1] X. Tang, E. Giacomin, N. Chetrit, and P.-E. Gaillardon, "Ultra-low-power rram-based fpga: A road towards reconfigurable edge computing," University of Utah Salt Lake City United States, Tech. Rep., 2019.

[2] T. Ajayi, S. Kamineni, Y. K. Cherivirala, M. Fayazi, K. Kwon, M. Saligane, S. Gupta, C.-H. Chen, D. Sylvester, D. Blaauw, R. Dreslinski, B. Calhoun, and D. D. Wentzloff, "An open-source framework for autonomous soc design with analog block generation," in *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, 2020, pp. 141–146.

[3] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P.-E. Gaillardon, "Openfpga: An opensource framework enabling rapid prototyping of customizable fpgas," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 367–374.

[4] N. C. Dao and D. Koch, "Memristor-based pass gate for fpga programmable routing switch," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[5] E. Hsieh, M. Giordano, B. Hodson, A. Levy, S. Osekowsky, R. Radway, Y. Shih, W. Wan, T. F. Wu, X. Zheng, M. Nelson, B. Le, H.-S. Wong, S. Mitra, and S. Wong, "High-density multiple bits-per-cell 1t4r rram array with gradual set/reset and its effectiveness for deep learning," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 35.6.1–35.6.4.
[6] "Skywater high voltage methodology," https://skywater-pdk.readthedocs.io/en/main/rules/hv.html, 2022.
[7] "Efabless-mpw5a," https://github.com/efabless/caravel_user_project_analog, 2022.