Group 7: Boolean Operations in 10T Differential Read-Decoupled SRAM

YuTing Shiau, Hsinling Lu, Hsiang-Yang Fan, Daniel Yu, Yong-Xin Huang

{ytshiau, hsinling, seanfan, dddaniel, yonhu}@umich.edu

Abstract:

This report presents the design and implementation of an 10T SRAM as a computational unit that provides READ, WRITE, AND, and NOR instructions. The use of SRAM can improve speed and reduce energy consumption in data transfers between processor and memory units. The 10T SRAM is chosen for better low voltage behavior and loose constraints in transistor sizing. The design includes optimization for read and write stability, area efficiency, and performance. The results show that the single cell of 10T SRAM has a wide and symmetric static noise margin. The 16x16 array of SRAM is connected with discharge and precharge circuits. This work involves implementing the bit line computation of AND or NOR and modifying the controller to support additional SRAM instructions.

Introduction:

Our final project implements an 10T SRAM and it is designed to be a computational unit that provides READ, WRITE, AND, and NOR instructions. The main advantage of using SRAM is speed and efficiency improvement. With a well-designed SRAM, the computation can be done inside SRAM instead of keeping moving data in and out of the memory. According to the course content, data movement takes more time than local computation. Furthermore, SRAM is tailored for custom features when a specific processor or application will be using it [1]. Thus, we modify our controller to support additional SRAM instructions and combine it with our baseline design.

We did not choose 6T SRAM because the cell has a coupled read-write path that is vulnerable to read-disturb failures. Activating multiple word-lines may also easily flip the cell states non-deterministically which is undesirable [2]. Moreover, 10T SRAM provides better low voltage behavior and loose constraints in sizing our transistors.

Design and Implementation:

System Overview of In-memory Computation

In our implementation provided in Fig. 1, we have two memory modules in the same hierarchical level. The DMEM module is used for regular load/store operations, while the SRAM module is customized to support in-memory computation. The SRAM module can support regular load/store operations as well, but such operations correspond to special operation codes to distinguish themselves from load/store to DMEM module (see Additional ISA support for details).

Fig. 2 shows the block diagram of our controller and SRAM. The controller of the SRAM needs to receive the instruction or data from the CPU controller in order to combine with the 16x16 SRAM circuits. When the controller receives the SRAM_AND or SRAM_NOR instruction, the controller will activate 2 addresses (two one hot addresses doing an OR). After receiving the address, the SRAM will then read the bitline out. Therefore, the controller will choose to activate 1 or 2 read word lines. We don’t need XOR to determine whether it is a 1 or 2 word line that has been activated.

Additional ISA Support

In this work, four additional instructions are added, and the specifications are shown in Table 1. <15:12> is the opcode to do SRAM calculation given by our controller, and <15:12> is the opcode to choose from 4 functions. SRAM_AND and SRAM_NOR are the two in-memory computation instructions which calculate the two addresses in <11:8> and <3:0>. SRAM_LOAD loads the value from SRAM_addr to Rdest; SRAM_STORE stores the value from Rsrc into SRAM_addr.

SRAM circuit design

In our single SRAM cell schematic, in order to ensure write stability, we make the pull-up PMOSs weakest. To reduce the leakage current, we need to size the length of N2/N4 slightly larger than the other transistors. As for the Boolean instructions, our SRAM supports 4 additional functions including SRAM_AND, SRAM_NOR, SRAM_READ and SRAM_WRITE. When the write word line triggers high, the write bit line will pass and save the value through the single cell of SRAM. We finished connecting the 16x16 array of SRAM with discharge and precharge circuits. Instead of connecting with the sense amplifier and level converter, we add buffers in the button of our bitlines. They can strongly pull high or low in our circuits.

Layout

In our design, we add buffers in the button of our bitlines. It can help strongly pull high or low in our circuits. In addition, multiple factors like read and write stability, area efficiency, and performance are considered carefully. With the increasing demand for smaller and more complex integrated circuits, reducing the area of each SRAM cell is a key goal. One approach to achieving this goal is to compress the height and width of the cell. The height/width of our single cell is

5.2/4.8 um. However, the tradeoff is that we can’t accommodate too many transistors in one layer, so we use more lines to do routing. The detailed design is shown in Fig. 3.

Results:

SNM Analysis

The butterfly curve of our 10T cell is in Fig. 4. The Stability Noise Margin (SNM) for upper and lower squares are 0.45V and 0.30V separately. In addition, we are interested in how the temperature will affect the curve, so we sweep the temperature from -65°C to 125°C as shown in Fig. 5. As the temperature goes up, the SNM becomes smaller. In Fig. 6, Monte Carlo simulation is also provided for mismatch and process variation.

NC-Verilog Simulation

We add NOP instruction between each instruction during NC-Verilog simulation to store operating results back. Therefore, there are several Xs in our graph. The data is available every cycle, however, we have a control signal sram_write_enable to control what time the data can be stored. As shown in Fig. 7, we first store and load to set the desirable value. Once the values are ready, we verify our functionality by doing NOR, AND instructions successively.

Analog Simulation

Fig. 8 and Fig. 9 show two calculations, AND and NOR, respectively in our analog simulation. In Fig. 8, we store data FFFF to address 0010 (Data <4> (in purple) and WWL<4>(in blue)), and data 0050 is stored at address 0040 (Data <6> (in brown) and WWL <6>(in pink)). Next we make RWL<4> and RWL<6> high to read data FFFF and 0050 from 0040 and 0010, and AND two data.

We get the correct result, 0050. In Fig. 9, we start to do NOR calculation. First, we store data FFFC to address 0010 and store data 0000 to address 0040. Then we read two data from 0010(data FFFC) and 0040 (data 0000) and do NOR calculation. The result is also correct (0030).

Layout

The layout of the whole design is shown in Fig. 10 and Fig. 11. The area of SRAM is 76.8μm ×92μm=7065.6μm^2. The area of the baseline processor is 487μm×535.4μm=260739.8μm^2, and the chip size (with IO pads) is 1424μm×1254μm= 1785696 μm^2.

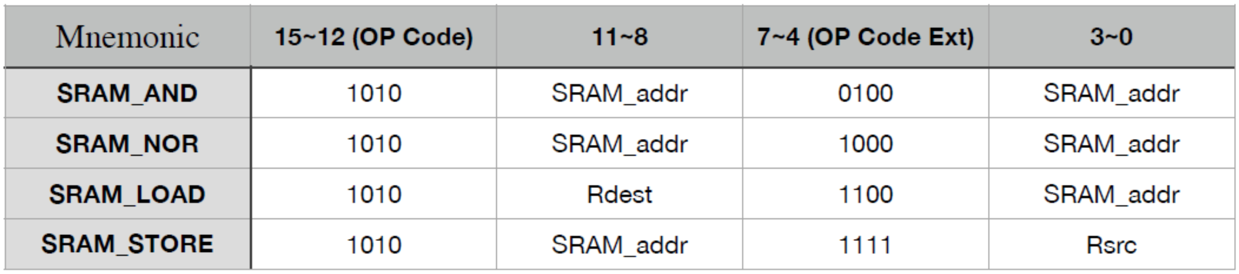
Conclusion:

Our SRAM helps eliminate parts of data movement by implementing in-SRAM Boolean operations. We propose a specification for the SRAM to be able to integrate with the baseline processor and simulate the design to verify the functionality. In the future, we hope to perform more Boolean functions in memory computations. Also, higher operating frequency, more robust and wider Static Noise Margin are expected.

References:

[1] IBM Applications Note: Understanding Static RAM Operation

[2] A. Agrawal, A. Jaiswal, C. Lee and K. Roy, "X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 12, pp. 4219-4232, Dec. 2018, doi: 10.1109/TCSI.2018.2848999.



Mnemonic	15~12 (OP Code)	11~8	7~4 (OP Code Ext)	3~0
SRAM_AND	1010	SRAM_addr	0100	SRAM_addr
SRAM_NOR	1010	SRAM_addr	1000	SRAM_addr
SRAM_LOAD	1010	Rdest	1100	SRAM_addr
SRAM_STORE	1010	SRAM_addr	1111	Rsrc

Table 1. Additional ISA Support

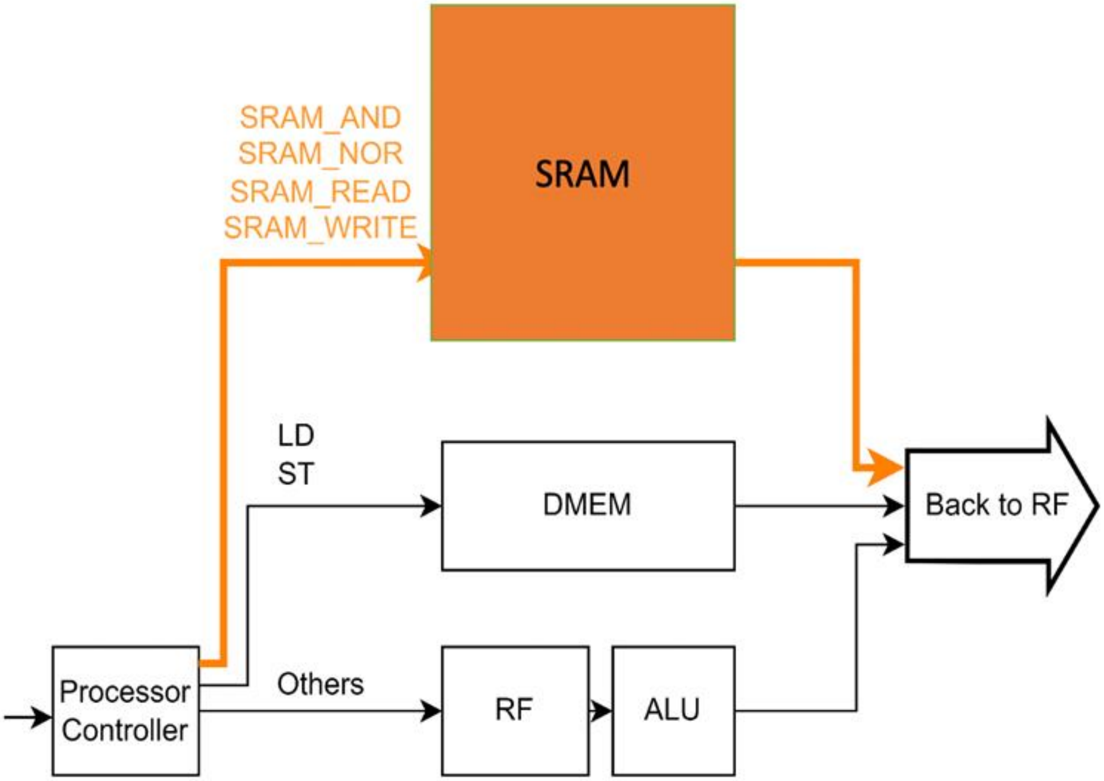


Fig. 1 Integration with baseline processor

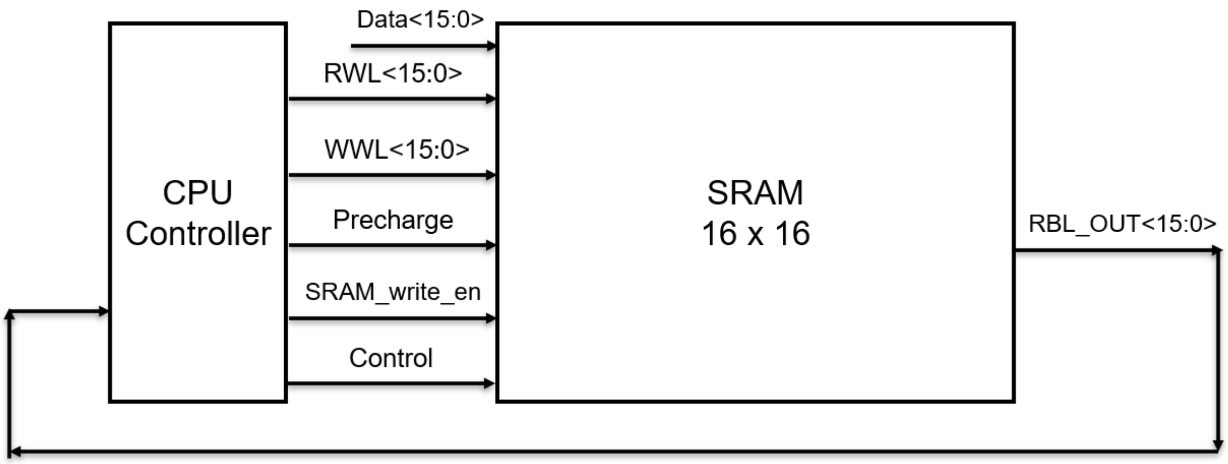


Fig. 2 CPU controller connected with SRAM block diagram

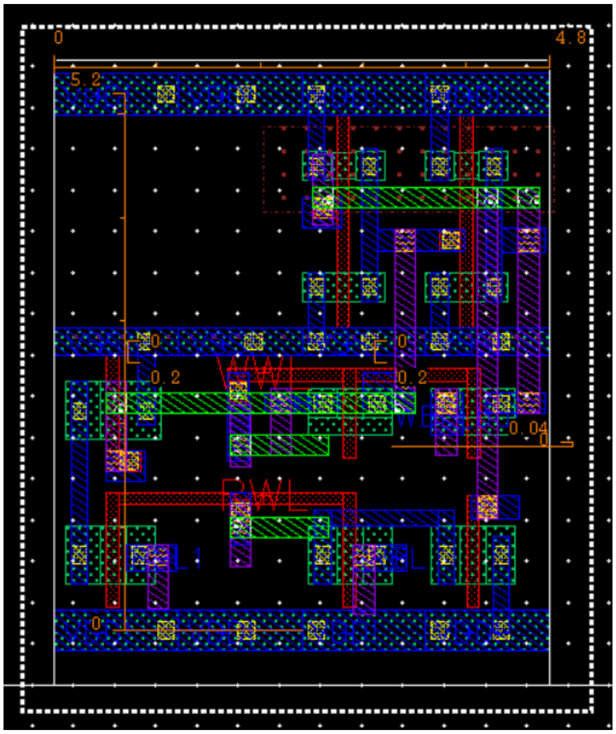


Fig. 3 Layout of 10 T single cell

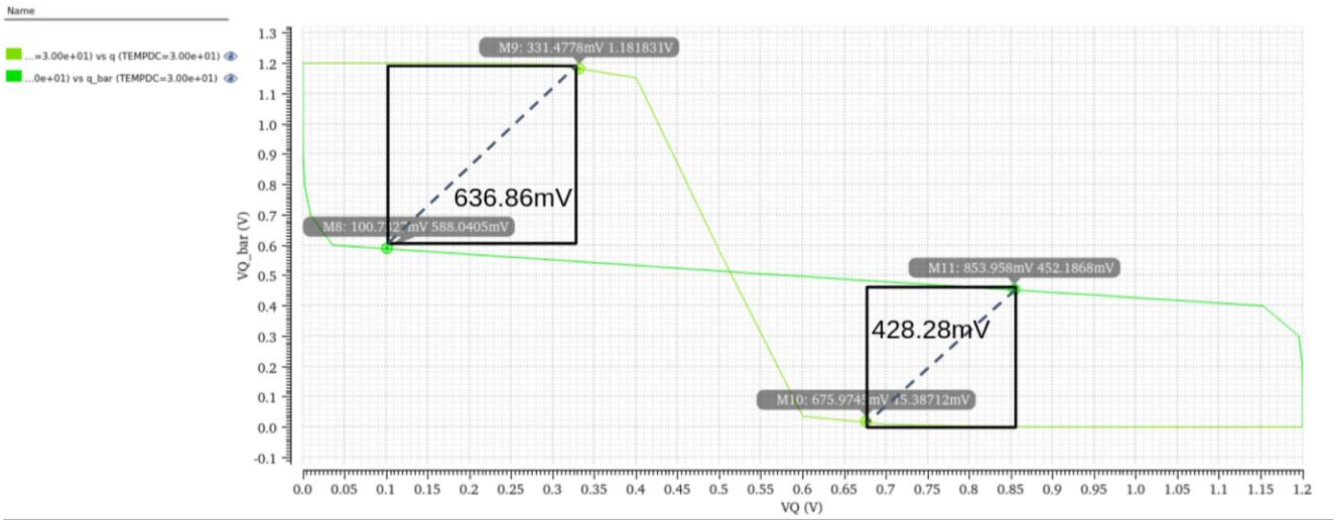


Fig. 4 Static Noise Margin analysis

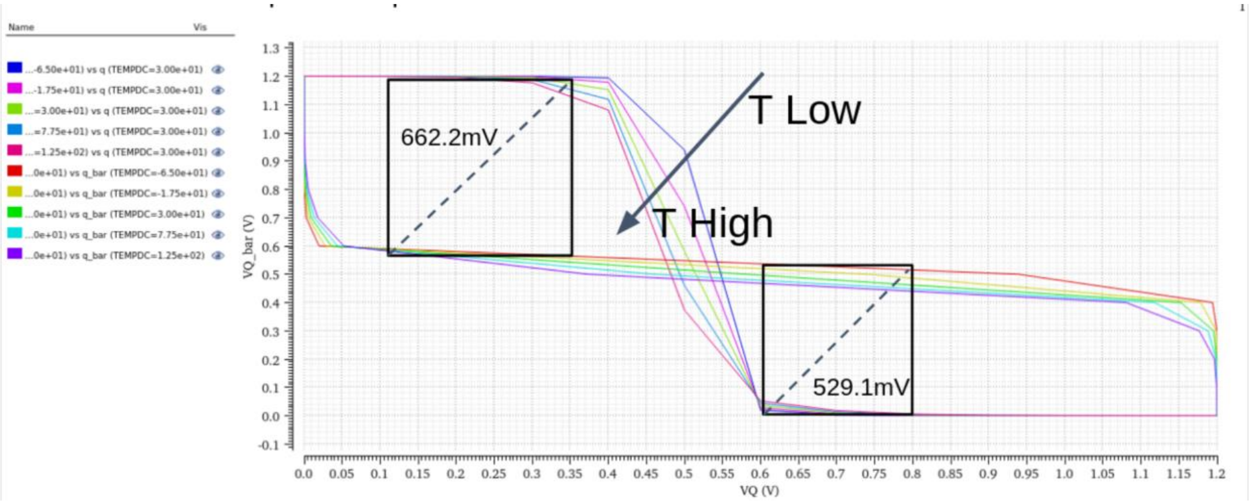


Fig. 5 Static Noise Margin analysis with temperature variation

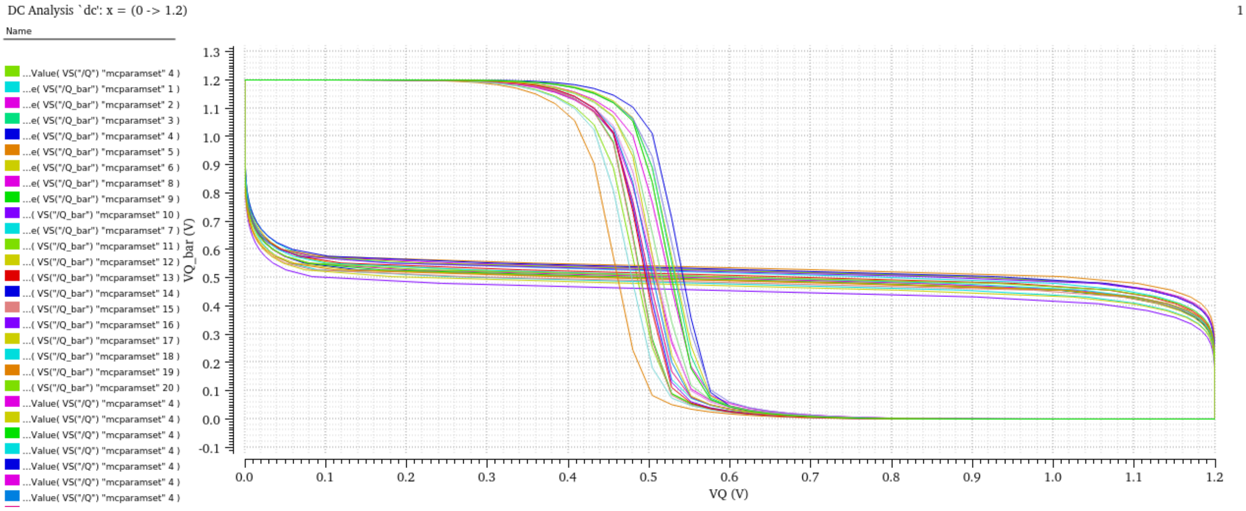
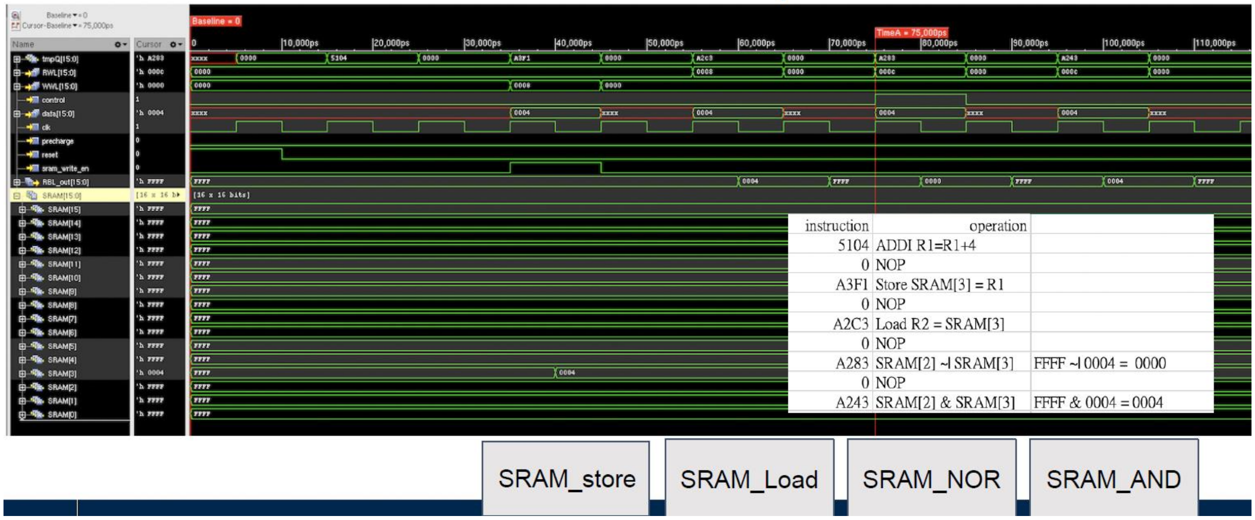
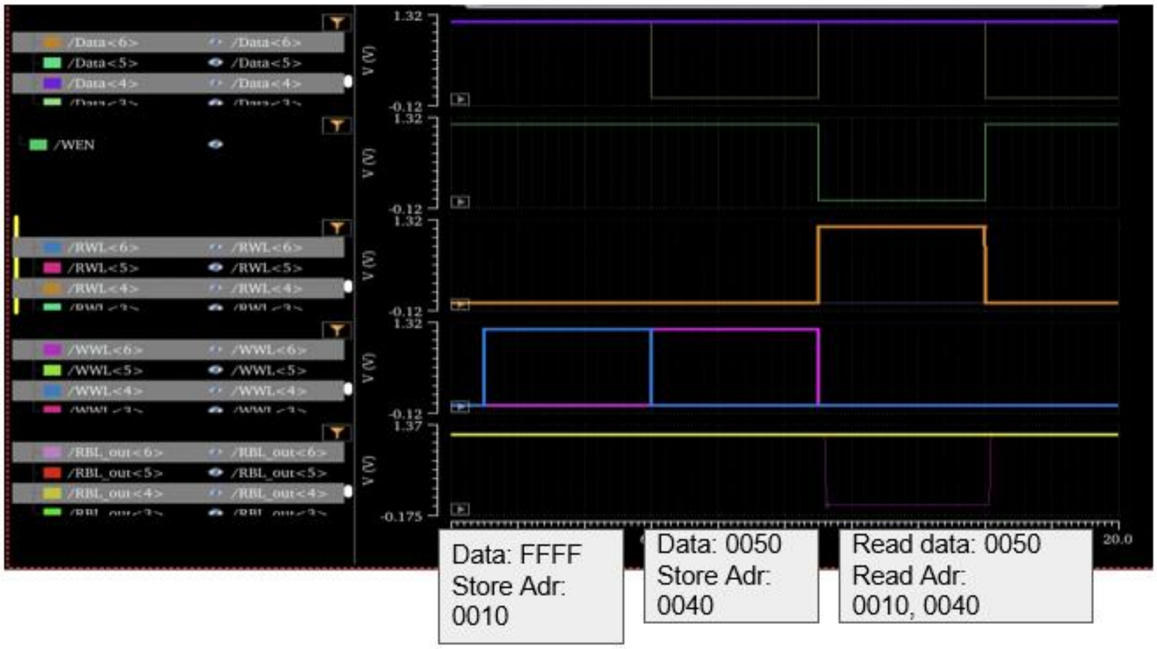


Fig. 6 SRAM Monte Carlo simulation



instruction	operation	
5104	ADDI R1=R1+4	
0	NOP	
A3F1	Store SRAM[3] = R1	
0	NOP	
A2C3	Load R2 = SRAM[3]	
0	NOP	
A283	SRAM[2] ~I SRAM[3]	FFFF ~I 0004 = 0000
0	NOP	
A243	SRAM[2] & SRAM[3]	FFFF & 0004 = 0004

Fig. 7 SRAM NC-Verilog simulation

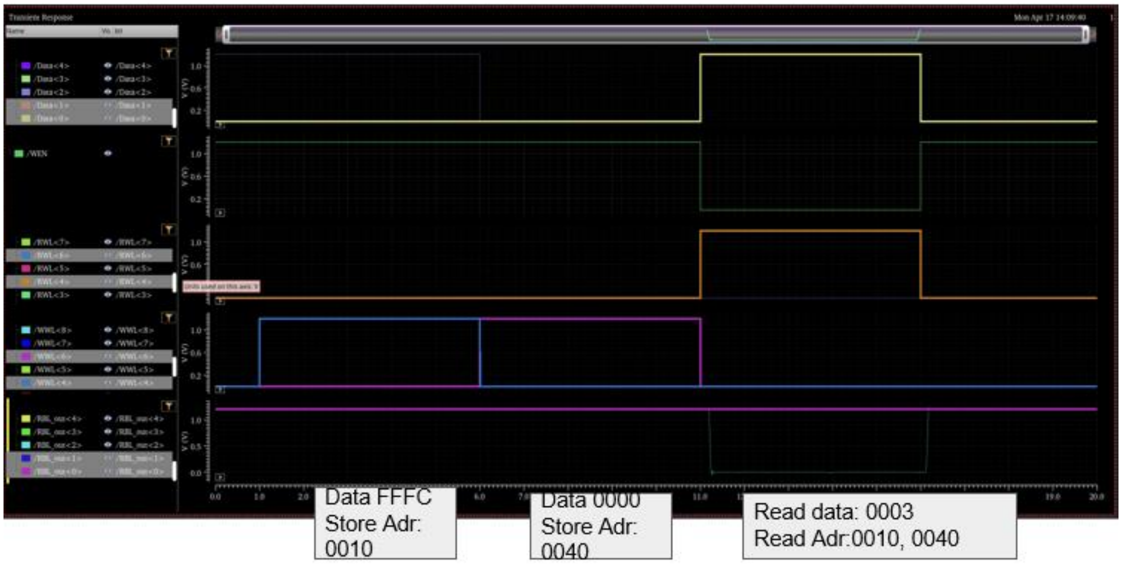


Data: FFFF
Store Adr:
0010

Data: 0050
Store Adr:
0040

Read data: 0050
Read Adr:
0010, 0040

Fig. 8 SRAM analog simulation - AND



Data FFFC
Store Adr:
0010

Data 0000
Store Adr:
0040

Read data: 0003
Read Adr:0010, 0040

Fig. 9 SRAM analog simulation - NOR

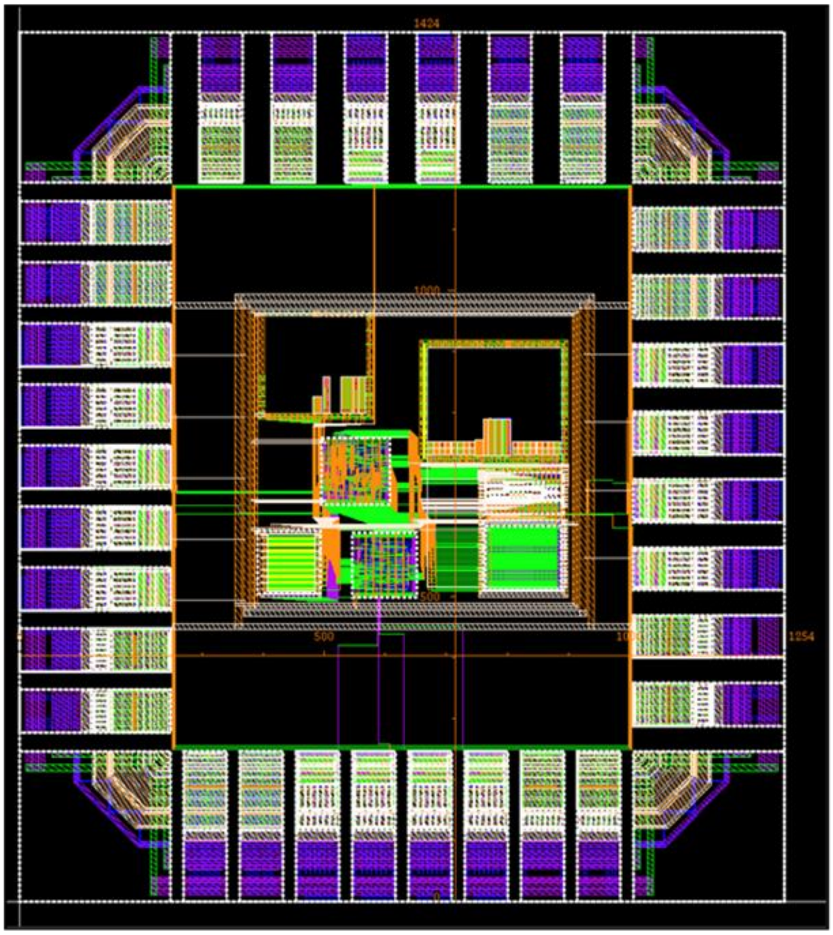


Fig. 10 Layout designs with Pads

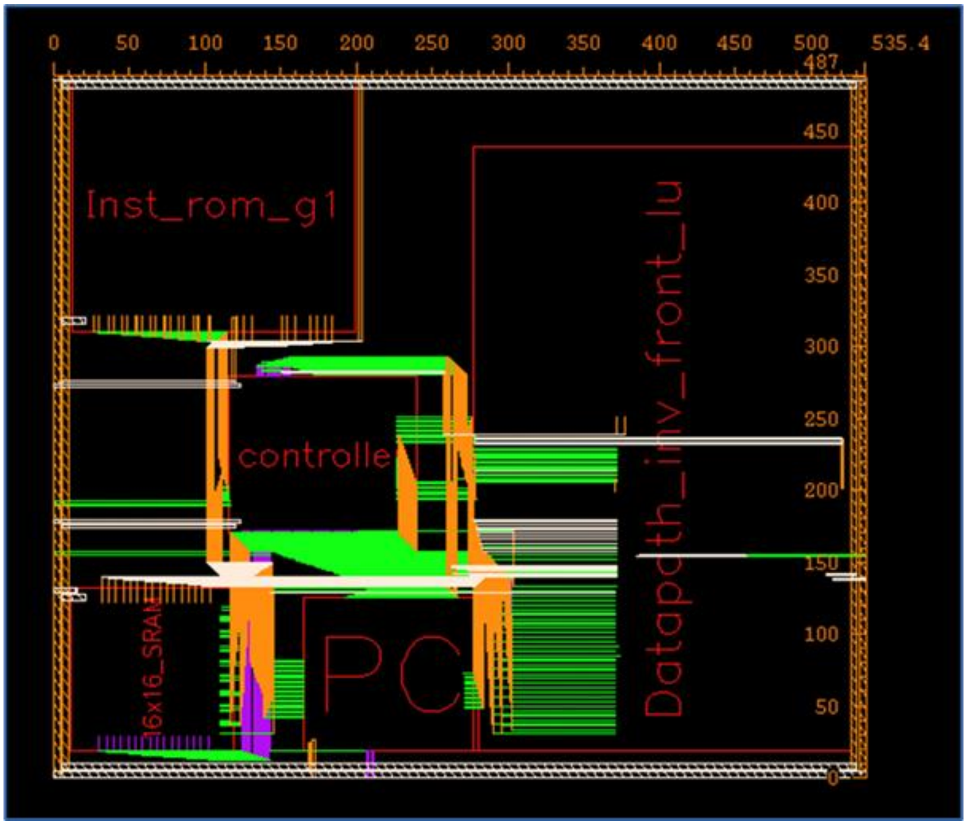


Fig. 11 Layout designs without Pads