# EECS 427 W15 Final Report
# 4KB ROM-Embedded SRAM

## Group 3

Ruohan Luo
luoruoh@umich.edu

Yuchen Fan
fanyuch@umich.edu

Yuxiang Mu
muyx@umich.edu

Yixin Ma
yixinma@umich.edu

Rui Zhou
ruizhou@umich.edu

## ABSTRACT

**This report summarizes our EECS 427 design project: a 4-kb ROM-embedded SRAM and its interaction with our 16-bit 2-stage pipelined RISC microprocessor. The ROM data is embedded into the SRAM cell, and the switching between the ROM mode and SRAM mode is controlled by one extra word line named RCON. We proposed some novel schematics for our Rom-embedded SRAM cells and block/word decoders for the purpose of delay and area reduction. Besides, we added sleep transistors into the SRAM cells to reduce the leakage current up to one tenth. We also sized up our transistors a little bit in order to meet the high performance requirement of our baseline processor. Finally, we replaced the IMEM and DMEM in the baseline processor by our own memory and reached a clock period of 6.4 ns with a very compact layout.**

## KEYWORDS

VLSI design, ROM-embedded SRAM, high performance

## 1. INTRODUCTION

Along with the continuing scaling down of CMOS transistors' sizes, a large on-chip ROM is not a good idea for designers nowadays, especially for the devices with fixed functions. The reason is that on-chip read-only memory not only takes a large portion of the area of the whole layout, but also requires abundant interconnection which induces more delay. Therefore, for some functional devices, the ROM of the processor is replaced by a ROM-embedded SRAM to improve the performance and save some area.

In fact, ROM-embedded SRAM is not a new thing. It is widely used in the pre-programmed and area-sensitive devices, such as the processor of micro medical robots. It is also applied to machines with fixed functions, such as the digital signal processing processor, math function evaluation devices, and built-in self-test hardware. For our ROM-embedded SRAM, we implemented the test functions into the ROM as the instruction memory, and we used the SRAM mode of our Rom-embedded SRAM as the data memory of our baseline processor, which is similar to the built-in self-test hardware.

One of the strengths of ROM-embedded SRAM is area-saving. At the same level of performance, the ROM-embedded SRAM saves the area of some commonly used components like decoders, pre-charge cells and MUX. At the minimum size, 8 transistor cell design will save the area by 30% approximately. However, the goal of our whole project is to design a high speed microprocessor based on moderate power consumption and area cost. Hence in order to make our final project consistent with our baseline processor design philosophy, we modified the traditional 8T cell and made some tradeoff between the memory area and its access time to keep the high speed and robustness of our baseline processor. We also spent a long time on the layout design in order to reduce the total area and routing.

The following parts gives the detailed explanation of our complete circuit design and validation, and it is organized as follows. Section 2 introduces our baseline processor schematic design, compact layout, and relevant simulation results. Section 3 discusses the circuit logic of our ROM-embedded SRAM and its interaction with our baseline processor. Section 4 makes a conclusion of our design and gives some suggestions for the future improvements.

## 2. BASELINE MICROPROCESSOR

Basically, our microprocessor adopts RISC instruction and follows 2-stage pipeline implementation in 16bits. In the first stage, the program counter (PC) calculates correct address to fetch the instruction from IMEM and sends it to instruction reg. In the second stage, the controller decodes the instruction and executes the corresponding operation through the Register File (RF), ALU, or shifter. The result is returned to the RF/DMEM, if necessary, in the last half cycle in order to avoid hazards.

Here shows the complete process of operating a single instruction through our microprocessor:

I. The first cycle is instruction fetch stage. When starting with a rising edge, the PC calculate the new address from the old one base on the signal of Br and Jmp. If jump condition is triggered, the PC will adopt the target address from RF and output the current PC+1 ready for writing back in RF in case JAL happen. If branch condition is triggered, then PC will read the displacement and branch address to PC+disp+1. If both signals are negative, then PC just adds 1 to old one as new address. At the falling edge, the IMEM receive the new address from PC and fetch the instruction to the Instruction Register (IREG). Notice that the branch/jump instructions will need extra cycle to return the proper Jmp/Br signal; hence we insert single Noop instruction right after each Bcond/Jcond/JAL operation.

II. The second stage is execution stage. When encountering the rising edge, the IREG passes the instruction to the controller, where the instruction is decoded and translated to the control signals for muxs and execution devices, make sure the data stream flow through the right path.

*a) RF*

We implemented a 16-word 16-bit data storage register file with 16 columns of slave latch as well as 1 column of master latch on the right side. To fit in the requirement of the baseline, the RF should be able to read after the instruction is decoded and write when the data is written back in a single cycle. Thus we designed the master cell to be triggered at high level of clock, which guarantees the calculated data is stored in the master before falling edge, while the slave latch is triggered at low level of clock to transfer the data from master. The 16-bits WE signal is activated at the same time to control which register to write.

*b) ALU*

ALU is responsible for the main calculation part of our microprocessor. We used a 16-bit carry-select adder with “2-2-3-4-5” structure to minimize the delay. The ALU can operate the required basic math calculation as well as the Z, F, N flag operation when CMP/CMPI instruction is adopted to further support Br and Jmp control.

*c) Shifter*

The shifter is constructed by 4 different sub-shifters which can operate the left shift operation with 1bit, 2bits, 4bits and 8bits. Through different combination we can fulfill any bits of left shift operation. The combination is controlled by a mux before the shifter. To further improve the performance, we insert drivers between every two sub-shifter.

Our microprocessor can complete all of the required instructions. At demonstration we are required to perform lines of instruction to calculate Fibonacci Sequences. The instructions is mainly designed with MOV/MOVI, ADD/ADDI and Bcond operation. Then we implement instruction to test the rest of the operation. The NC Verilog shows our results are correct.

When we further substitute the DMEM and IMEM with our ROM-Embedded SRAM, we encounter some challenges with delay control. These problems are solved with the careful clock design, which will be further discussed in following pages. In final, our data path can fit both DMEM/IMEM as well as our ROM-embedded SRAM with the correct operations and calculations.

Table 1 shows the final data of our baseline performance.

**Table 1. Performance of our microprocessor**

| Components | Worst-case delay (ps) | Area (um$^2$) |
|---|---|---|
| RF | 200 | 84.4*96 |
| ALU | 1400 | 32*96 |
| Shifter | 1088 | 32.4*96 |
| Controller | 554 | 56*36 |
| PC | 1226 | 56*60 |

The floor plan is shown in Figure 1. Notice that this is the final version of our baseline with our ROM-Embedded SRAM instead of DMEM and IMEM. We distribute the bits in column so the path goes from right to left in horizontal.

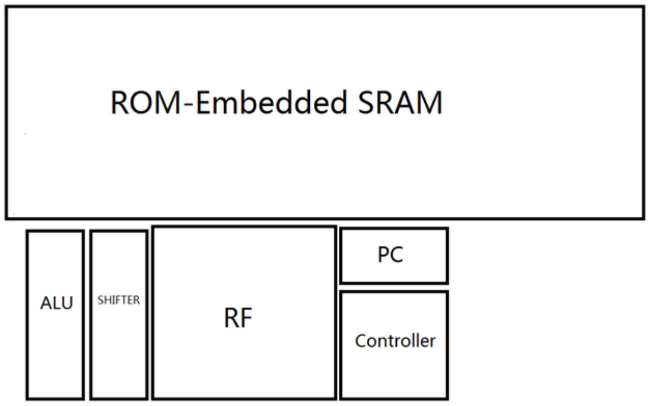

**Figure 1. Floor plan of our microprocessor**

## 3. ROM-EMBEDDED SRAM

### 3.1 Cell Design

Our ROM-embedded SRAM has two modes, decided by RCON. When RCON=1, the cell is in ROM mode. When RCON=0, the cell is in SRAM mode. To achieve this functionality, we use 8-T cell for ROM1 cell (which stores a “1” in ROM mode) and 10-T cell for ROM0 cell (which stores a “0” in ROM mode).

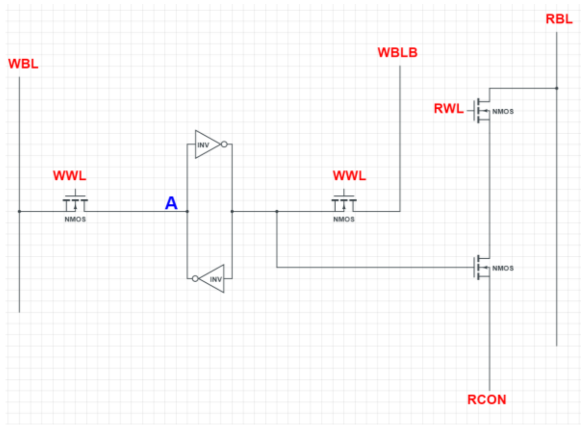

**Figure 2. ROM1 cell**

Figure 2 shows the design of ROM1 cell. Initially, RBL is always pre-charged to “1”. When RCON=1, the cell is in ROM mode. To read the value stored in ROM mode, WWL should be set to 0. Then a “1” will be read on RBL. When RCON=0, the cell is in SRAM mode. To read the value stored in SRAM mode, WWL should be set to 0 and RWL should be set to 1. Then the value stored at point A will be read on RBL. To write a VALUE to point A, the WBL and WBLB should be charged respectively to VALUE and VALUE_BAR. The WWL should be set to 1 and the RWL should be set to 0. Then a new value can be written into point A.

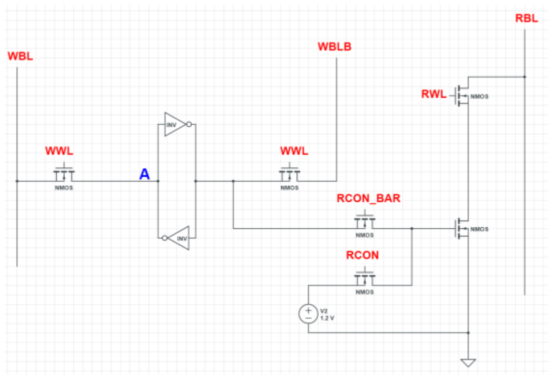

**Figure 3. ROM0 cell**

Figure 3 shows the design of ROM0 cell. Initially, RBL is always pre-charged to “1”. When RCON=1, RCON_BAR=0, the cell is in ROM mode.  To read the value stored in ROM mode, WWL should be set to 0 and RWL should be set to 1. Then a “0” will be read on RBL. When RCON=0, RCON_BAR=1, the cell is in SRAM mode. To read the value stored in SRAM mode, WWL should be set to 0 and RWL should be set to 1. Then the value stored at point A will be read on RBL. To write a VALUE to point A, the WBL and WBLB should be charged respectively to VALUE and VALUE_BAR. The WWL should be set to 1 and the RWL should be set to 0.  Then a new value can be written into point A.

## 3.2 Clock Design

The ROM-embedded SRAM is used in our 2-stage pipelined microprocessor. The ROM mode will be used as IMEM and the SRAM mode will be used as DMEM. In the first stage, a new instruction is fetched from the IMEM. In the second stage, the instruction is loaded into IREG. Then this instruction will be decoded and executed. The clock design is discussed below.

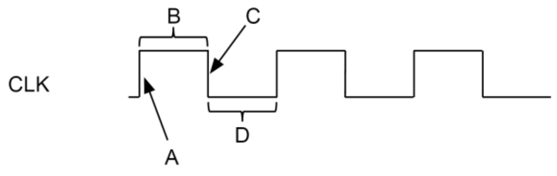

**Figure 4. Clock design**

For the first stage: When CLK=1 (time period B), the PC will be updated. When CLK=0 (time period D), the new PC is loaded into the IMEM and a new instruction will be fetched. For the second stage: At rising edge A, the instruction is loaded into IREG. When CLK=1 (time period B), the instruction will be decoded and executed. At falling edge C, the result value is written back to Register File.

To achieve this functionality, the ROM-embedded SRAM should be switched between ROM mode and SRAM mode. When CLK=1, all 256 words of ROM-Embedded SRAM are in the SRAM mode. When CLK=0, the word whose address is the current PC will be in ROM mode and the other 255 words will be in SRAM mode. Therefore, the equation for RCON is: RCON=RWL*CLK_BAR

We also design a CLK_Precharge to pre-charge the RBL. The pre-charge design using a PMOS is illustrated in Figure 5. When CLK_Precharge equals to 0, RBL is pre-charged to “1”. When CLK_Precharge is transited to 0, the RBL remains a floating “1”.

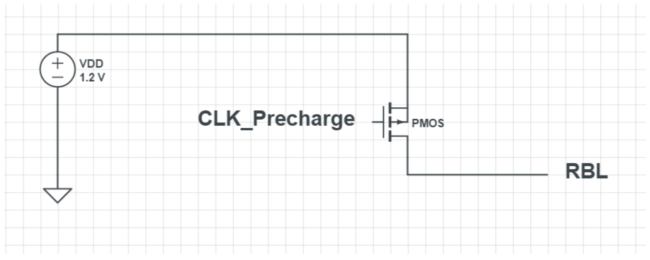

**Figure 5. Pre-charge schematic**

For every clock cycle, an instruction should be loaded from the ROM-embedded SRAM when CLK=0. For the baseline instruction LOAD, a value should be read from the ROM-embedded SRAM when CLK=1. Therefore, the RBL should be pre-charged twice a cycle. The design of CLK_Precharge is shown in Figure 6.

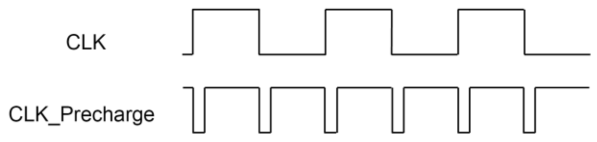

**Figure 6. CLK_Precharge design**

## 3.3 Hierarchical Blocks Design

Hierarchical architecture is applied in our design. There are three reasons for separating the original 256-word memory array into 16 hierarchical blocks with 16 words in each.

Firstly, in this way, the long bit lines in the original memory array which have the length of 256 cells are cut up, and thus the wire capacitances are greatly reduced which improves the speed of our circuit.

Secondly, since 256 is 2 to the power of 8, an 8-bit address need to be decoded to access a certain word. Usually, an 8-to-256 is not very efficient, and by dividing the 256 words into 16 blocks, the 8-bit address is split into 2 parts, 4 bits in each. The critical path now becomes passing two 4-to-16 decoders in parallel (one block decoder and one word decoder) plus one more logical gate. The decoding process is almost double efficient now.

Thirdly, hierarchical blocks match the design of sleep transistors. At any time, only one block will be accessed, and then only that block need to be active, and others could be in sleep mode.

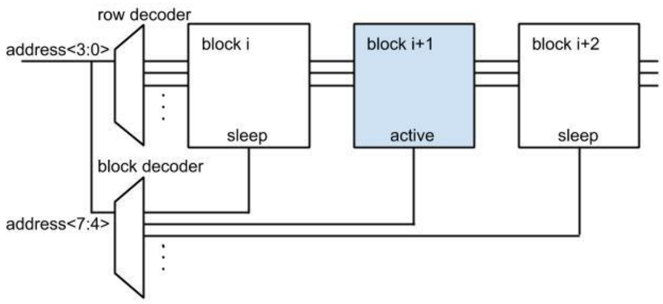

**Figure 7. Conceptual Diagram of Hierarchical Blocks**

## 3.4 Decoder Design

### *3.4.1 Block decoder & Word decoder*

For a 4KB memory, there are 256 words. We divide our Rom-embedded SRAM into 16 blocks with 16 words in each block. As a result, we build two 4:16 decoders to control the blocks and the words separately. The block decoder translates the first four bits of the eight-bit address into 16 blocks address, the word decoder translates the last four bits into 16 word line address at the same time. For example, when the first output of the block decoder and the first output of the word decoder becomes 0, the WWL will become 1 when WEN is 0, which means the first word is selected. The detailed logic is illustrated as following.

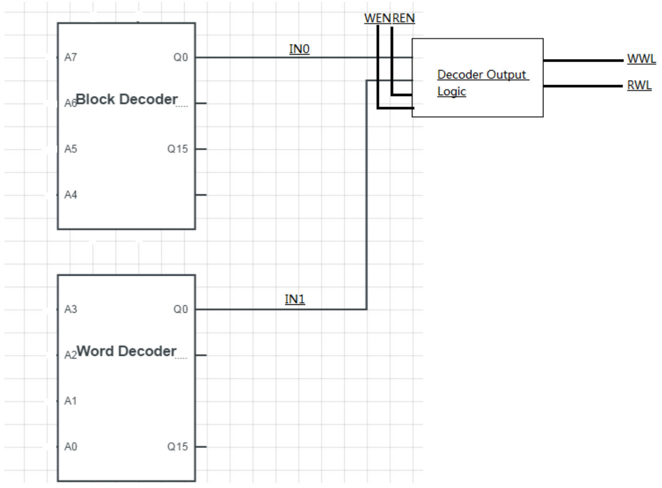

**Figure 8. Top level circuit of the decoder logic**

### *3.4.2 4:16 decoder Logic*

For the 4:16 decoders, we use NAND-OR logic to implement the function. For example, $Q_0 = (\overline{A_0} NAND\ \overline{A_1}) OR (\overline{A_2}\ NAND\ \overline{A_3})$

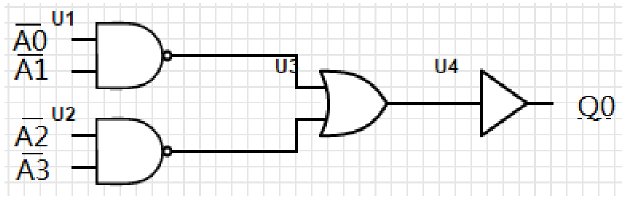

**Figure 9. Gate level circuit of the 4:16 decoder logic**

### *3.4.3 Decoder Output Logic*

There are 256 decoder output logic blocks which mapped 256 words. There are two 2:1 MUX and two 3-input NOR gates inside each block. When the CLK is 0, the Rom-embedded SRAM is in ROM model, RWL will be 1 when the word is selected (IN0 is 0 and IN1 is 0, shown is Fig1). WWL will always be 0. When the CLK is 1, the Rom-embedded SRAM is in SRAM model, now the value of RWL and WWL depend on the value of REN and WEN.

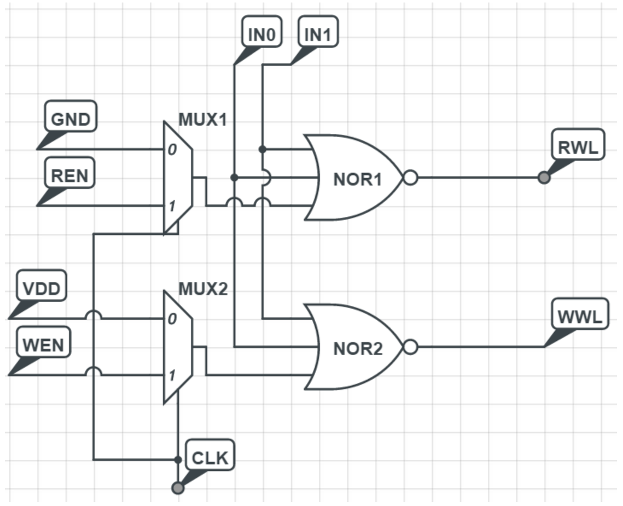

**Figure 10. Gate level circuit of the decoder output logic**

## 3.5 Sleep Transistors Design

The purpose of adding sleep transistors is obviously saving power. Beside sleep transistors, we also consider other power-saving approaches. Scaling down VDD is the most efficient way to save power, but scaling down the whole circuit will sacrifice the speed. In that case, to regain the performance, Vth could be scaled down at the same time, but the leakage current will increase significantly which makes VDD scaling meaningless. Sleep transistors let the circuit function normally when it is active and decrease the leakage current when the circuit is in the sleep mode. Sleep transistors theoretically don't save dynamic power, so they are especially useful if the active factor is pretty small. In our project, this effect is revealed particularly in the last several blocks which are not accessed often.

Since the memory we design is a SRAM, the VDD of the back-to-back inverters cannot be turned off completely, or the data stored in the cells might be lost. Then, only one sleep transistor won't work, and one more bias transistor is needed to maintain a lower virtual VDD (VDDV) in sleep mode as shown in Figure 11.

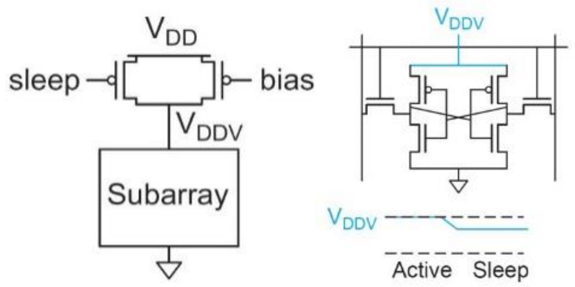

**Figure 11. Conceptual diagram of sleep and bias transistor [4]**

The gate of the sleep transistor of each block is directly connected to the block decoder as the output of the block decoder is "one-cold". Every time a certain block is accessed, the sleep transistor will be completely on for that half cycle. The sizing of the sleep transistor is 2.24um which is big enough to charge the capacitance in a single block. The bias transistor is connected to a constant analog value. To minimize the leakage current, the bias transistor is set to have the minimum size 280nm. The gate voltage of the bias determines its ability to offset the leakage, i.e. VDDV and the leakage current.

As the Hspice analog simulation results show, the block can maintain its functionality (keeping the data stored) for bias gate voltage less than 1.00V. For bias gate voltage less than 0.87V, VDDV exceeds 90% VDD which doesn't meet with our design expectations of power-saving. Figure 12 shows the inverse relationship between the bias gate voltage and VDDV (left axis), block leakage current (right axis).

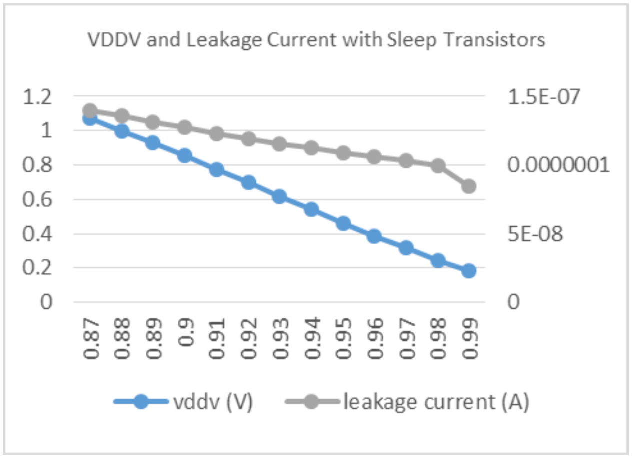

**Figure 12. VDDV and leakage current vs. bias gate voltage**

In order to wake up the sleeping block as fast as possible, VDDV is not used in pre-charge cells and output buffers. The block wake up time versus the bias voltage is shown in Figure 13. We define the wake up time as the time of reaching 90% VDD.

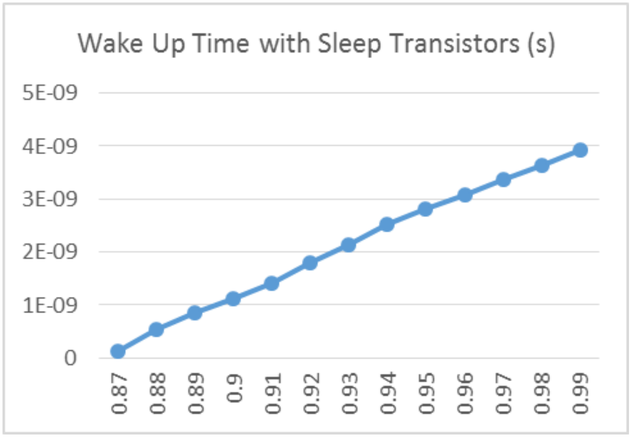

**Figure 13. Wake up time vs. bias gate voltage**

Dynamic current is also measured, as shown in Figure 14. As a reference, the operation current data for block without sleep transistors are respectively 3.76E-4 (SRAM Write), 6.41E-4 (SRAM Read) and 6.42E-4 (ROM Read). We found the dynamic current with sleep transistors is close to or slightly less than that without sleep transistors.

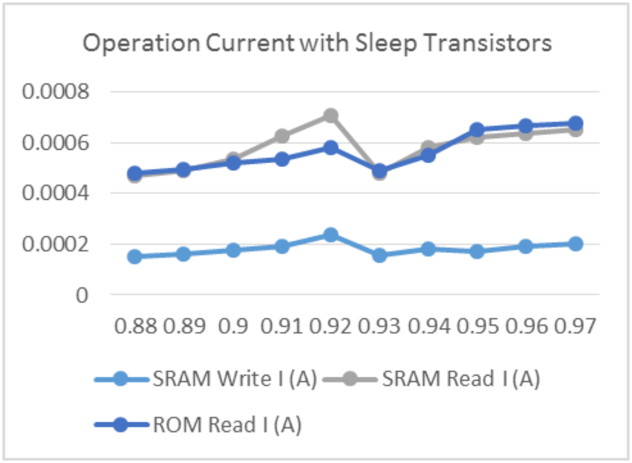

**Figure 14. Operation current vs. bias gate voltage**

Take all the factors above into consideration, the best bias gate voltage for both power saving and performance is 0.93V. However, in this case, the clock cycle of the baseline datapath need to add 2ns to execute the worst case. If we want to fit in the baseline clock cycle, we can set the bias gate voltage to 0.89V. For information, if only considering power saving, the best choice is 0.99V. The detailed data is shown in Table 2.

**Table 2. Block performance with bias gate voltage 0.93V**

| Bias Gate (V) | VDDV (V) | Leakage Current (A) | Wake Up Time (s) |
|---|---|---|---|
| 0.89 | 9.29E-1 | 1.31E-7 | 8.47E-10 |
| 0.93 | 6.19E-1 | 1.16E-7 | 2.15E-9 |
| 0.99 | 1.87E-1 | 8.47E-8 | 3.92E-9 |

## 3.6 Layout Design

### *3.6.1 Rom Cell Layout*

It is crucial to reduce the cell area in order to achieve an area-efficient design. The ROM0 cell layout which contains ten transistors shown in Figure 15 is compact.

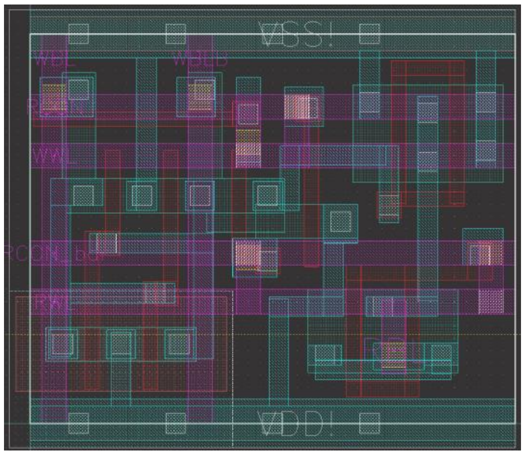

**Figure 15. Layout of ROM0 cell**

### *3.6.2 Rom-embedded SRAM Layout*

The layout of Rom-embedded SRAM shown in Figure 16 contains 16 blocks where Decoder Output Logic blocks are placed in front of them. In order to further reduce the area and minimize the wire length, the block decoder and the word decoder is divided into 16 segments which can be inserted into the gaps between blocks. The area of Rom-embedded SRAM layout is 398um*217.6um.

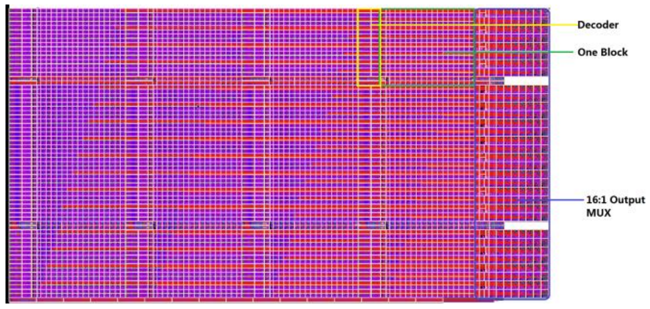

**Figure 16. Layout of Rom-embedded SRAM**

### *3.6.3 Top-Level Design Layout*

The floor plan of the microprocessor is shown in Figure 17. In order to minimize the delay of the critical path, we place shifter, ALU and RF close to each other from left to right. The layout after integration of the R-SRAM microprocessor is shown in Figure 18.

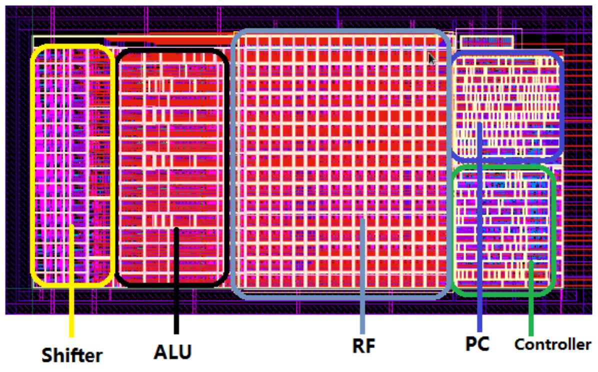

**Figure 17. Layout of the datapath**

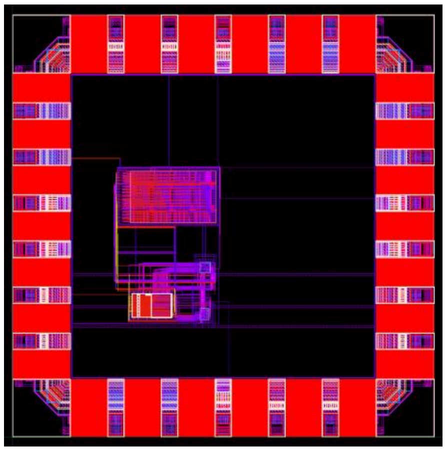

**Figure 18. Layout of the microprocessor after integration**

## 3.7 Performance

The ROM-embedded SRAM is properly sized up to achieve high performance and to avoid area penalty. The worst-case write access time of ROM-embedded SRAM is 0.68 ns and the worst-case read access time is 1 ns. Therefore, the average worst-case access time of ROM-embedded SRAM is 0.84 ns, which is much less than the worst-case access time of the provided IMEM (2.7 ns) and the provided DMEM (1.1 ns).

## 4. CONCLUSION

In this report we presents our high performance 16-bit 4kb ROM-embedded SRAM and its interaction with our 16-bit 2-stage pipelined RISC microprocessor. We proposed our 8-transistor cell and 10-transistor cell for different ROM data and modified the logic of decoder in order to achieve smaller area and faster memory access time. We also designed our layout carefully to reduce the total area and routing. By using our design, the average memory accessing time is 0.84 ns. Therefore, the clock cycle of our microprocessor can be reduced to 6.4 ns. Besides, we added sleep transistors into the SRAM cells, which reduces the leakage current by 10%.

## 5. ACKNOWLEDGEMENTS

First, we would like to thank Prof. Mazumder for his interesting and impressive lectures and advices on the interview, which laid us a solid foundation for our career.

Second, we would like to thank our GSI, Mahmood, for his generous help and suggestions on our CAD assignments and final project. Our project might not go on so smoothly without his fast email response.

## 6. REFERENCES

[1] Anand Ramalingam, Bin Zhang, Anirudh Devgan and David Z. Pan. *Sleep Transistor Sizing Using Timing Criticality and Temporal Currents.* Department of Electrical and Computer Engineering, the University of Texas, Austin, TX 78712. Austin Research Laboratory, IBM Research Division, Austin, TX 78758

[2] Benton H. Calhoun, Frank A. Honoré, and Anantha P. Chandrakasan. *A Leakage Reduction Methodology for Distributed MTCMOS.* IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 39, NO. 5, MAY 2004

[3] Dongsoo Lee, and Kaushik Roy. *Area Efficient ROM-Embedded SRAM Cache.* IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 21, NO. 9, SEPTEMBER 2013

[4] Neil Weste, and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective 4th.* Addison-Wesley Publishing Company, USA ©2010

[5] Vishal Khandelwal, and Ankur Srivastava. *Leakage Control through Fine-Grained Placement and Sizing of Sleep Transistors.* IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 26, NO. 7, JULY 2007