ABSTRACT

SHAH, TANMAY ANILKUMAR. FabMem: A Multiported RAM and CAM Compiler for Superscalar Design Space Exploration. (Under the direction of Eric Rotenberg).

An out-of-order superscalar processor has a complex microarchitecture that uses many multiported memory structures. Mutliported memories are special memories with many read and write ports. As the issue width of the processor increases, both the number of ports and sizes of the memory structures need to be increased. Due to their multiported nature, they consume a large fraction of the total area and power of the processor and limit its clock frequency. Consequently, effective design space exploration in the design of a superscalar processor requires accurate area, access time and energy consumption estimates of each of its memory structures.

To fulfill this goal, we have developed "FabMem", a tool which generates physical designs of arbitrary multiported Static Random Access Memory (SRAM) and Content Addressable Memory (CAM) structures using automated SKILL scripts. The generated memories are free of any DRC and LVS errors and detailed stability analysis confirms the generated memories are robust. FabMem generates accurate results for area, access time and energy consumption of multiported memories, essential data for superscalar processor design space exploration. To improve the access latency of large highly-ported SRAMs, pipelining of the SRAM and "constrained simultaneous access" are proposed.

FabMem: A Multiported RAM and CAM Compiler for
Superscalar Design Space Exploration

by
Tanmay Anilkumar Shah

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Computer Engineering

Raleigh, North Carolina

2010

APPROVED BY:

Dr. Eric Rotenberg Committee Chair	Dr. James Tuck
Dr. W. Rhett Davis	

DEDICATION

To my Family

BIOGRAPHY

Tanmay Shah was born on the 20^{th} of February, 1987. He received his Bachelors degree in Electronics and Communication from Nirma University of Science and Technology, Ahmedabad, India in 2008. He started his graduate studies in North Carolina State University, Raleigh, NC in Fall 2008. He is pursuing his research under the guidance of Dr. Eric Rotenberg. Upon graduation, he will be working as Physical Design Engineer at Intel Corporation.

ACKNOWLEDGEMENTS

First of all, I would like to thank my parents for making my life an enjoyable journey. I also would like to thank my brother and sister-in-law for their encouragement for my career.

It is my honor to pursue my research under Dr. Rotenberg. I would like to thank him for his support and encouragement to achieve goals throughout my Masters program.

I would like to thank my research group and friends, Rohan, Kushan, Jaimin, Rohit, Niket, Sandeep, Jayneel, Hiran, Saunder, Salil, Muawya, Elliot, Julian and Hashem for their guidance and valuable feedback on my thesis.

This work was supported by NSF grant No. CCF-0811707, Intel and IBM. Any opinions, findings and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

An out-of-order superscalar processor maintains a large pool of dynamic instructions, called the instruction window, from which it can extract instruction-level parallelism (ILP). Each cycle, a scheduler identifies independent instructions from this window that can execute in parallel.

In terms of physical implementation, a superscalar processor's instruction window corresponds to a number of key memory structures. The physical register file (PRF), rename map table (RMT), issue queue (IQ) payload and architectural map table (AMT) are implemented using Static Random Access Memory (SRAM), while the load queue (LQ) and store queue (SQ) address match logic and issue queue wake-up logic are implemented using Content Addressable Memory (CAM). Table 1.1 shows different memory structures used in the superscalar processor. Each of these memory structures is multiported so that they can be accessed by multiple instructions in parallel. For example, in the Register Read (RR) stage of the pipeline, each instruction might have two source operands, therefore, the number of read ports into the PRF is equal to twice the processor's issue width. In the same way, in the Write Back (WB) stage of the pipeline, all completing instructions might need to write to destination registers. Thus, the PRF requires as many write ports as the issue width of the processor. In this way, as the issue width of the processor increases from 1 to 8, the read port requirement for the PRF increases from 2 to 16 and the write port requirement increases from 1 to 8. The port requirements of all other RAM and CAM structures vary similarly with the issue width of the processor.

Table 1.1. Memory structures in a superscalar processor.

Pipeline Stage	Memory Structure
Fetch	Branch Target Buffer , Branch Predictor
Decode	Instruction Buffer
Rename	Rename Map Table, Free List
Issue	Issue Queue, Payload RAM
Register Read/Writeback	Physical Register File
Memory	Load Queue, Store Queue
Retire	Reorder Buffer, Architectural Map Table, Free List

In order to increase the performance (instructions per cycle) of the processor, its issue width is increased, by adding more parallel function units. To fully utilize the parallel function units of a wide-issue processor, we need a sufficiently large instruction window so that enough independent instructions can be identified each cycle. Thus, to extract performance, instruction window structures are needed that are both large and highly ported.

To design a superscalar processor, arbitrarily many candidate superscalar configurations should be considered. So, we require an automated tool to generate the synthesizable RTL of arbitrary superscalar designs. "FabScalar" [1] is a tool to generate the synthesizable RTL for arbitrary superscalar processors. We also require an automated tool to generate physical designs of multiported RAMs and CAMs of arbitrary sizes and number of ports. A synthesis tool, for example, Synopsys Design Compiler, can generate area, timing and energy results only for combinational and sequential logic, but not for the wide variety of multiported memory structures. To design a custom memory for each required configuration is a very time-consuming process. Thus, we require an automated tool that can generate accurate data for area, access time and energy consumption and also the actual memory layout, which can be used later for place-and-route purposes as well as to simulate the full memory in detail for points of interest. FabMem is a tool, which enables the automated design space exploration of multiported RAMs and CAMs and also generates their physical designs. FabMem is an insightful tool which can be used by microarchitects for the design space exploration of superscalar processors and is also useful for the physical design of arbitrary superscalar processors in conjunction with FabScalar [1].

To estimate area, access time and energy, it is possible to implement analytical models for SRAM and CAM. But multiported structures are very wire intensive structures. For example, an 8R8W SRAM bitcell has 32 bitlines and 16 wordlines in parallel. The same length wire can have different intrinsic and parasitic capacitances at different places depending on its neighbors' shapes and that will produce different driving times and also energies for different wires of the same length and metal layer. It is very hard to analytically model fine-grained parameters of the multiported memories. In addition to that, at any time, any number of ports may be reading from the same address. Thus, accurate analytical models require very detailed modeling of the loading of the bitcell for worst-case timing analysis.

For accurate analysis, the whole memory can be generated and its extracted netlist can be simulated in HSPICE, but HSPICE simulation of such a large netlist is very slow and requires a tremendous amount of storage space. Thus, we need to use a hybrid approach, which only generates and simulates the critical-path of the memory. The full memory is generated and simulated just for selected configurations of interest. This approach will save time as compared to always simulating the whole memory and will not suffer for accuracy.

1.2 Goal of this Work

The goal of this work is to automate the physical design and analysis process for multiported SRAM and CAM structures of arbitrary sizes and number of ports, which can be summarized as shown below.

- Design the automated data generation tool that produces accurate results for area, access time and energy consumption for the required multiported SRAM and CAM structures.
- Deliver the SRAM and CAM compiler, which automatically generates the full layouts for a wide variety of input configurations.
- Design the pipelined SRAM to improve the cycle time of large SRAMs.

1.3 Related Work

EMPIRE [3] proposes the “Empirical power/area/timing models for the register files”. They made the layout of various register files by varying the number of ports from 3 to 12, depth from 4 words to 64 words and width from 8 to 128 bits. The simulation results are obtained for these SRAMs and curve fitting is used to obtain analytical models for the Register File. Due to this approach, EMPIRE might not produce accurate results for point outside the above range. For example, if it is used to generate the read access time for 128 deep 32 wide 1R1W SRAM, then the read access time obtained is 1.123 ns less than the read access time of a 64 deep 32 wide 1R1W SRAM in 90nm process technology.

CACTI [4] is a tool, which uses detailed analytical models to estimate the area, timing and power for caches and other memories. CACTI 6.0 shows that the low-swing wire model used is within 12% error of SPICE-simulated values. The analytical models for distributed wordlines and bitlines are within 13% and 12% of the SPICE generated values. Due to error present at the fine-grained level, it might be possible that the overall delay is more inaccurate than the actual implementation. For example, the solutions generated by CACTI for the 90 nm SPARC L2 cache have 28-32% of average error in access time, area and power consumption. A flexible embedded SRAM compiler, proposed in [5], is able to generate actual layouts for single-ported and dual-ported SRAMs only, with maximum capacity of 1Mb.

Chapter 2

Multiported SRAM Design

The goal is to design each basic circuit within an overall SRAM, which can be later used for automated data generation tool and also for the automated layout generation tool. This includes designing multiported bitcells for each required configuration, as well as the row decoder, precharge circuit, column multiplexer, sense amplifier and write driver circuit. Each circuit is designed in such a way that:

1. It produces an overall efficient design.
2. It simplifies the placement and routing in the automated layout process.

2.1 Bitcell Design

The basic bitcell implemented is a 6T SRAM cell, which is a pair of cross-coupled inverters and two pass transistors connected at the output of each of the inverters as shown in Figure 2.1 below. The gates of both the pass transistors are connected through the word line, WL. The outputs of these pass transistors are bitlines, BTL and BTLB. These bitlines can be either read bitlines or write bitlines. The workings of basic read and write operations can be found in the literature [6].

A read operation can be done using a single bitline only. The single-ended bitline scheme has the following advantages and disadvantages:

Advantages:

1. It requires less area. For 4R4W bitcell (four read ports and four write ports bitcell), it requires 4 (cross-coupled inverters) +4 (four read ports) +8 (four write ports) = 16 transistors instead of 4+8+8 = 20 transistors. It will have 4 read bitlines and 8 write bitlines instead of a total 16 bitlines.
2. Due to having fewer bitlines, it will consume less energy in precharging the bitlines.

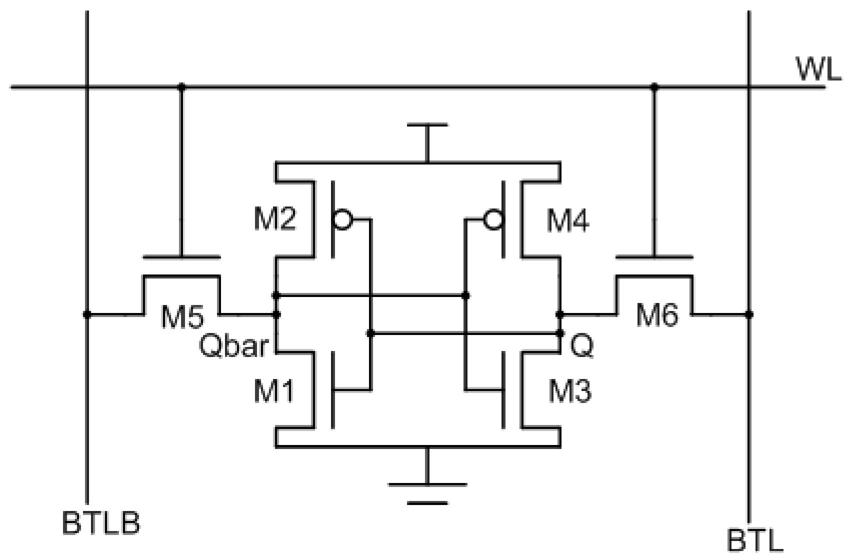


Figure 2.1. 6T SRAM bitcell.

Disadvantages:

1. The single-ended bitline architecture is not robust to the read upset problem. In the dual-bitline architecture, either bitline clamps Q (the storage node) to the V_{dd}, which makes inadvertent toggling of the cross-coupled inverter pair difficult [6].
2. The single-ended bitline architecture can not be used with a differential sense amplifier, and so will be slower as compared to the dual-bitline read scheme.

3. In a multiported bitcell, there will be more bitlines placed at the minimum allowable distance according to Design Rules. This will lead to significant coupling capacitances. When a logic value of "1" is read on a bitline, initially the bitline will be charged to V_{dd}, but it has the potential to discharge to V_{dd} - $V_{t,}$ because of the presence of the n-type pass transistor. If the adjacent bitline is discharging then this vulnerability can be triggered, affecting the bitline which is reading a "1".In dual-bitline bitcell, the same coupling might affect the bitline in similar manner, but it will not affect both the bitlines, BTL and BTLB. So, the single-ended bitline architecture is more sensitive to such coupling noise, as compared to dual- bitline architecture.

After evaluating both advantages and disadvantages, it was decided to implement the dual-bitline architecture.

For each extra port, another pair of pass transistors is added one on each side of the cross-coupled inverter. In Figure 2.2, a bitcell with two ports is shown. As the number of ports increases, the number of pairs of pass transistors will also increase. This will increase the wordline and bitline lengths significantly. Moreover, in a multiported bitcell, the length of the bitline is also larger and so the bitline capacitance (both intrinsic and parasitic) will be higher. So, even in the case of single read of a multiported bitcell, the read will be slower than before.

The current through that transistor is limited by the following equation given in the literature [6]:

$$\mathrm{I} = \frac{W}{L} K((V_{gs} - V_t)V_{min} - \frac{{V_{min}}^2}{2})$$

As shown in Figure 2.3, when all read ports are reading from the same address, all bitlines will be discharging through the single transistor highlighted at the bottom of the cross-coupled inverter. The current of each bitline is half of the current which would be flowing in the case of single port read of the bitcell. So, for a multiported bitcell, it will take quite a long time to discharge the bitline.

To reduce the read access time, it is necessary to facilitate high current and hence to discharge the bitline faster. For a faster read, the bitcell is implemented as shown in Figure 2.4 [17]. It is also possible to increase the width of the NMOS in the cross coupled inverter.

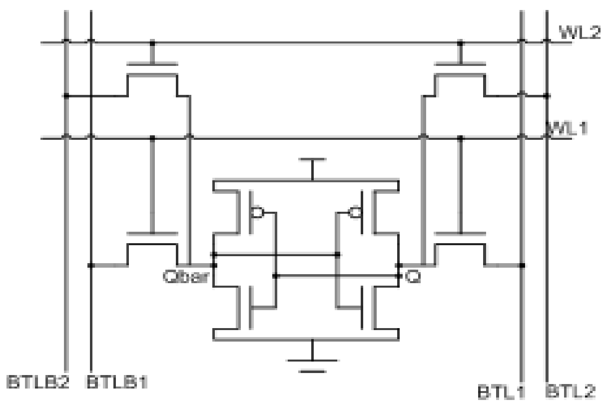


Figure 2.2. Two-ported SRAM bitcell.

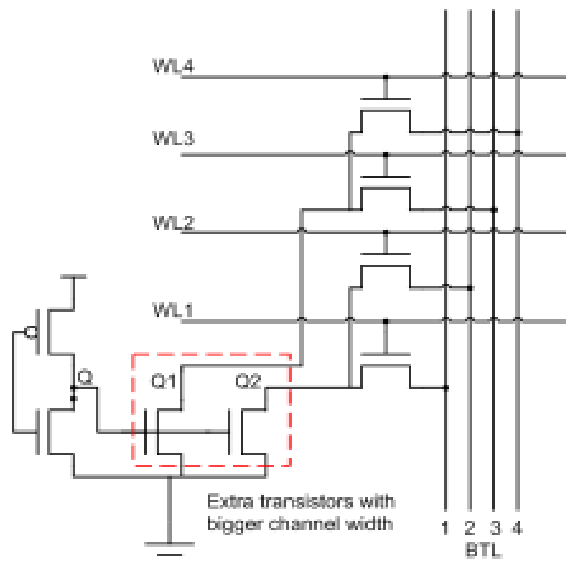


Figure 2.3. Faster read for multiported SRAM bitcell.

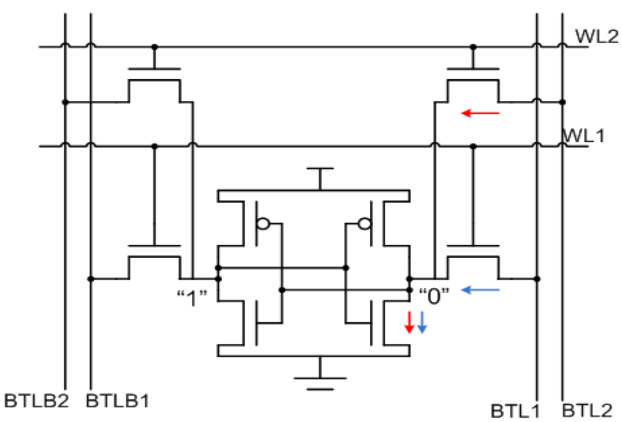


Figure 2.4. Simultaneous reading.

Figure 2.4 shows the example circuit for a four read ported bitcell. As shown, the bitlines corresponding to ports 3 and 4 will discharge through Q1, and bitlines 1 and 2 will discharge through Q2. In addition to that, the width of these extra transistors (Q1 and Q2) can be made larger to increase the current passing through them. This bitcell design has the following advantages and disadvantages.

Advantages:

1. The read access time is reduced due to two parallel paths to discharge the bitlines.
2. The read access time will be even less, if larger width transistors can be used.
3. The "read upset problem" will not occur, because now the bitlines are isolated from the cross-coupled inverter, therefore, during a read operation, the precharged bitlines can not raise the voltage of either Q or Qbar.

Disadvantages:

1. The extra transistors, Q1 and Q2, might increase the area of the bitcell.
2. The bitline will be reading the inverted value of the stored bit. If a "1" is stored in the bitcell, then, the bitline will be discharging and if "0" is stored in bitcell then bitline will not discharge. Though it is not a disadvantage, it puts the bitline in floating stage when the "0" is stored in bitcell. Because when "0" is stored in the bitcell it will turn off the Q1 and Q2 NMOS. During "read" phase precharging circuit is also off and this will put bitline in floating state.

The multiported bitcell is a very wire intensive circuit. The N-ported bitcell will have N wordlines in one direction and 2N bitlines in the other direction. So, those extra transistors can be placed under these wires. This way, it will not incur any extra area. The layout is managed in such a way that they will not increase area. It is difficult to hide the extra transistors under wires for a bitcell with too few ports, but such a bitcell is already faster due to its smaller size.

Using this approach, the following variety of bitcells is implemented:

1R1W, 2R2W, 3R3W, 4R4W, 5R5W, 6R6W, 7R7W, 8R8W

2R1W, 4R2W, 6R3W, 8R4W, 10R5W, 12R6W, 14R7W, 16R8W

The transistor sizes in the bitcell are decided according to the cell ratio (CR) and pull-up ratio (PR). As shown in Figure 2.1, if Qbar is "0" and Q is "1", then during the read operation, BTLB (which is precharged to V_{dd}) can raise the charge stored at Qbar. The rise in voltage at Qbar must not be greater than the switching threshold of the NMOS transistor, M_3, otherwise, it will create the "read upset problem" and might flip the bitcell's value. In the 45nm process node, during a read operation, M_1 will be in the "Triode Region" and M_5 will be in the "Saturation Region". The current flowing through both these NMOS transistors will be equal. The following equation is modified from the original equation presented in the literature [6].

$$K_n, M_5\left(\frac{(V_{dd} - V_{tn} - \Delta V)^2}{2}\right) = K_n, M_1((V_{dd} - V_{tn})\Delta V - \frac{\Delta V^2}{2})$$

The solution of the above equation is,

$$\Delta \mathrm{V} = (\mathrm{V_{dd}} - \mathrm{V_{tn}})\left(1 - \sqrt{\frac{\mathrm{CR}}{1+\mathrm{CR}}}\right)$$

$$\text{Where } CR = \frac{W_1/L_1}{W_5/L_5}$$

The voltage rise with respect to the CR for the 45 nm process node is plotted in Figure 2.5. As shown in this graph, for ΔV to be less than 0.4 V, CR should be greater than 0.3. By keeping CR equal to 1, the rise in voltage will be 0.214 V, which is verified with HSPICE simulation.

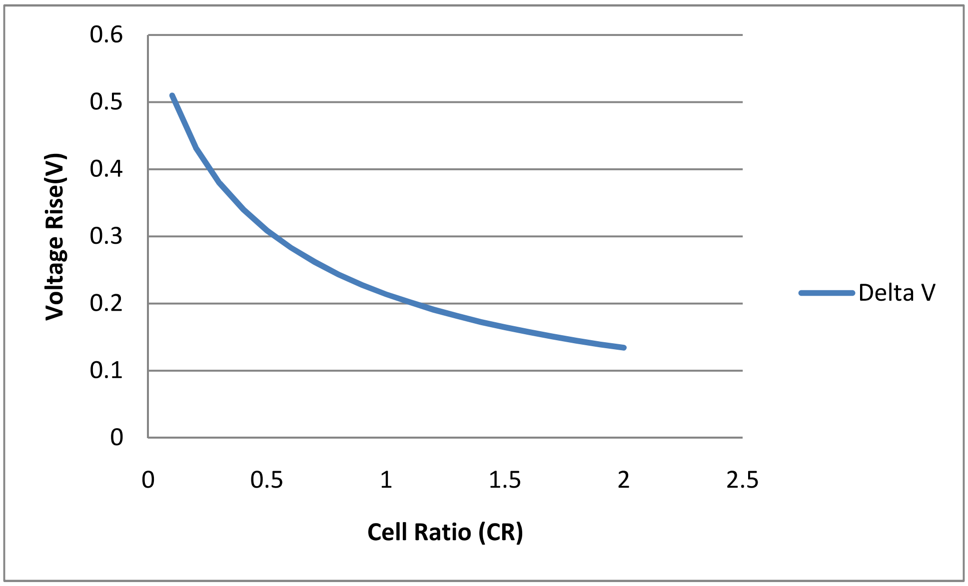


Figure 2.5. Change in voltage rise with cell ratio (CR).

For the PMOS transistor size, the pull-up ratio is taken into consideration. As shown in Figure 2-1, if Q is "1" and it is getting flipped to "0", then the voltage at Q must be lowered below the switching threshold of the M_1 NMOS transistor. During the write operation, BL will be at "0". For the 45nm process node, M_4 will be in the "Saturation Region" and M_6 will be in "Triode Region". Thus the current flowing from M_4 and M_6 will be equal. The following equation is modified from the original equation presented in the literature [6].

$$K_n, M_6\left(\frac{(V_{dd} - V_{tn})V_q - {V_q}^2}{2}\right) = K_p, M_4\left(V_{dd} - |V_{tp}|\right)^2/2$$

The solution to the above equation is,

$$V_q = V_{dd} - V_{tn} - \sqrt{(V_{dd} - V_{tn})^2 - \left|\frac{\mu_p}{\mu_n}\right| PR(V_{dd} - |V_{tp}|)^2},$$

$$\text{Where } PR = \frac{W_4/L_4}{W_6/L_6}$$

According to this solution, the cell voltage is plotted against PR as shown in Figure 2.6. For the cell voltage to be less than the threshold voltage of the NMOS transistor, PR should be less than 3.8. By selecting PR to be 2, the cell voltage will be 0.164 V.

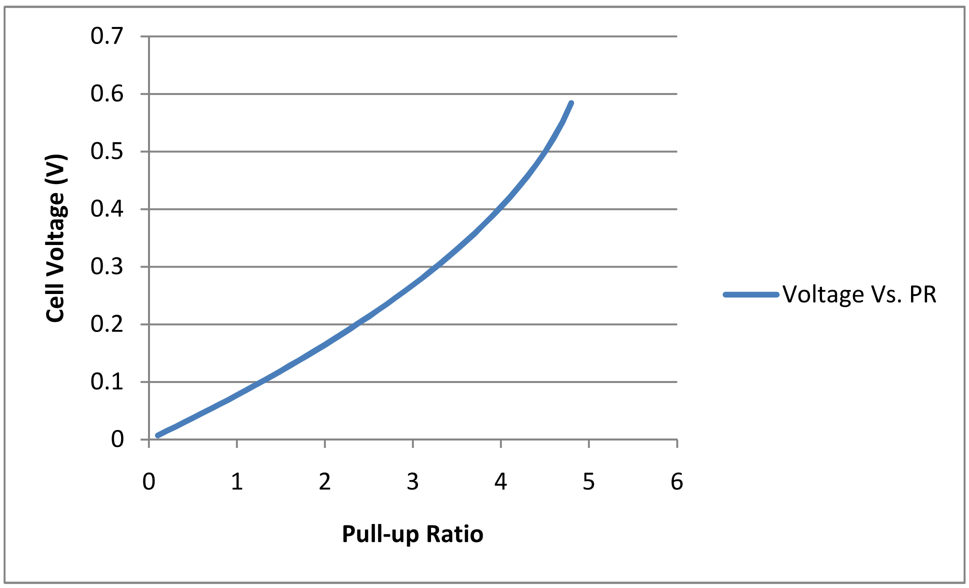


Figure 2.6. Cell voltage rise versus pull-up ratio (PR).

According to above CR=1 and PR=2, the NMOS width is selected to be 90nm and the PMOS width is selected to be 180nm. For a compact layout, wordlines are placed horizontally in metal layer-2 and bitlines are placed vertically in metal layer-3. The supply lines, V_{dd} and GND are placed on the top and bottom of the layout in the horizontal direction so that adjacent rows of bitcells can share one of the supply rails. All the internal routing including the connection to the gate of the pass transistor and the extra transistor is routed in metal layer-1. For a faster access time, the bitline is made smaller in length as compared to the wordline. Figure 2.7, Figure 2.8, Figure 2.9 and Figure 2.10 show the schematic and layout for the bitcell 4R4W and 8R4W, respectively.

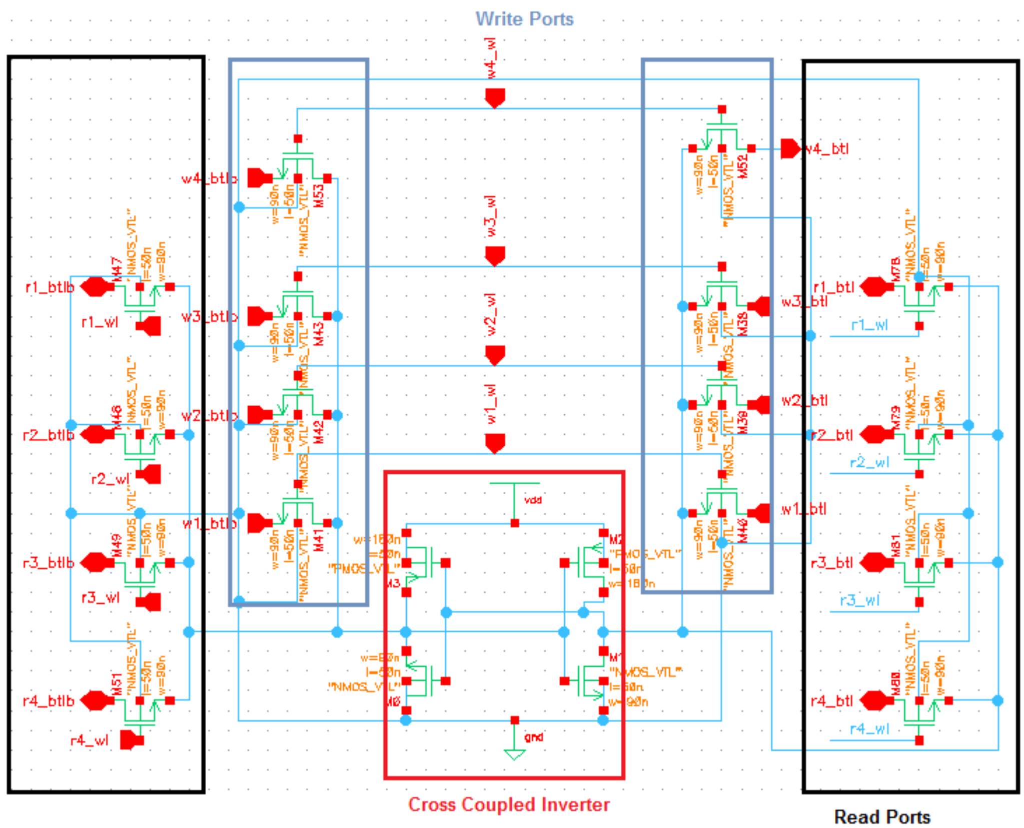


Figure 2.7. 4R4W bitcell schematic.

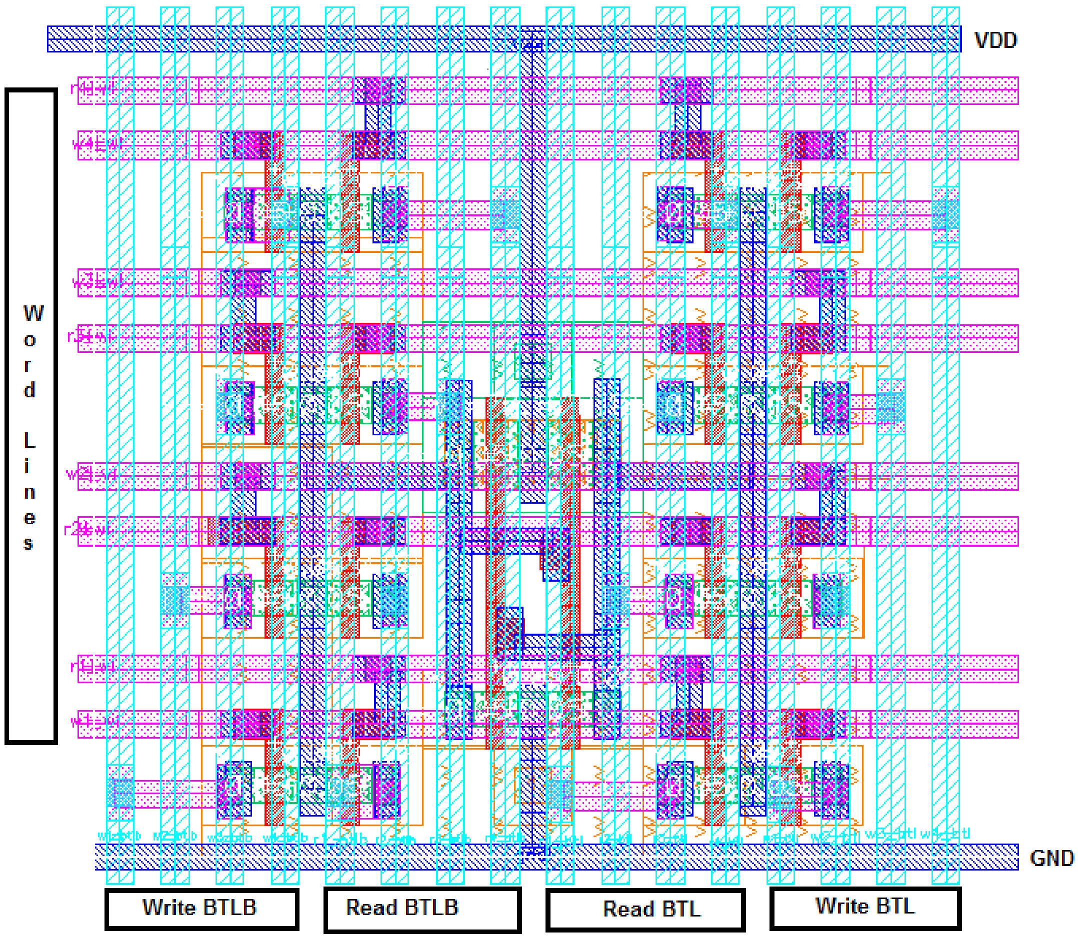


Figure 2.8. 4R4W bitcell layout.

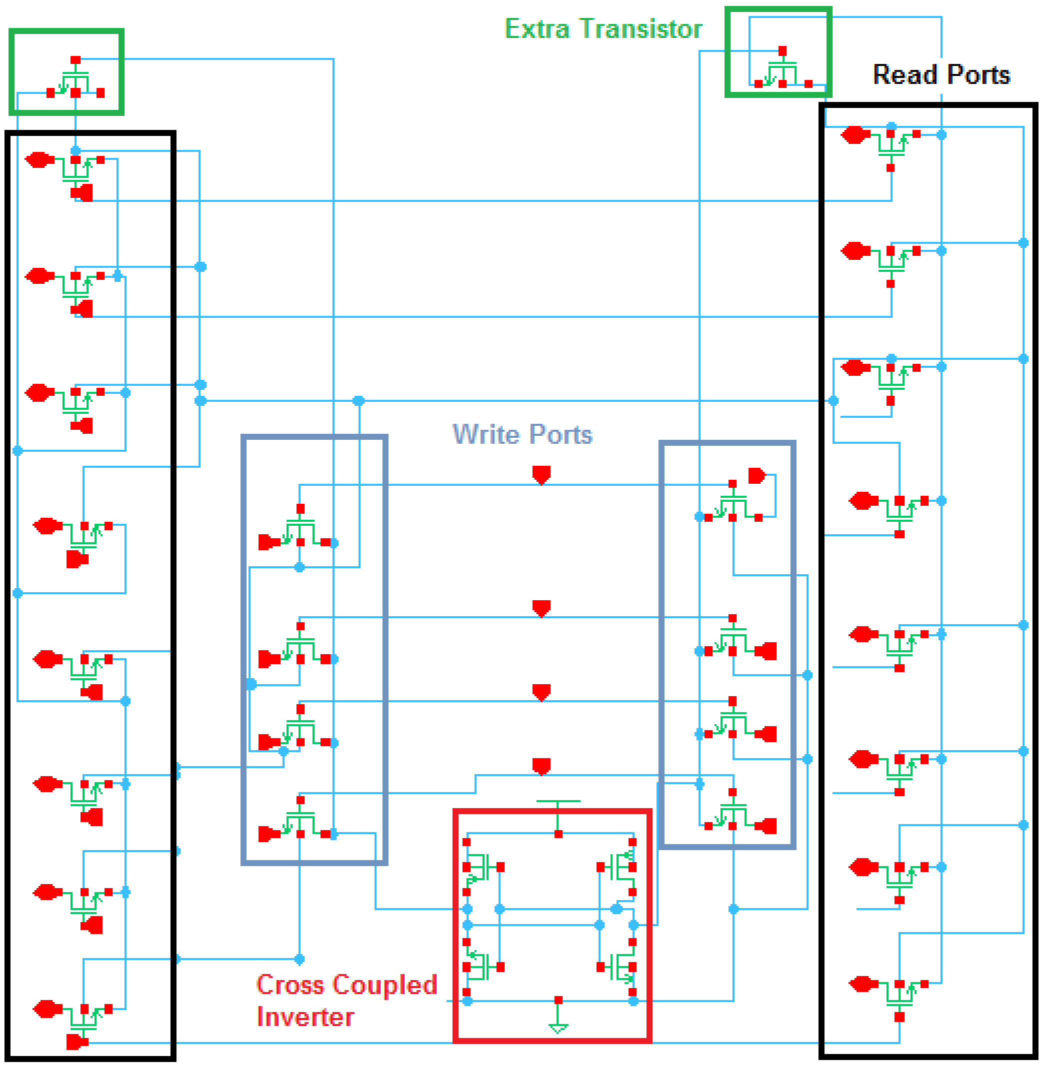


Figure 2.9. 8R4W bitcell schematic.

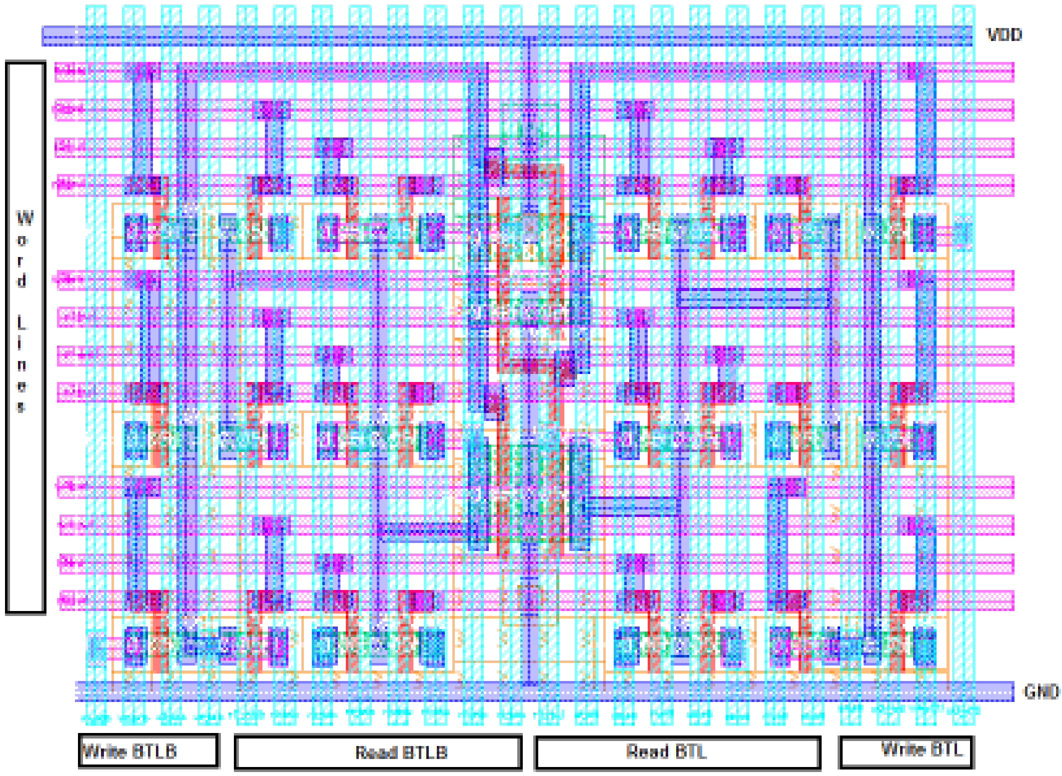


Figure 2.10. 8R4W bitcell layout.

Table 2.1 shows the area in μm^2 of all the bitcells implemented.

Table 2.1. Bitcell area.

Bitcell	Area (μm^2)	Bitcell	Area (μm^2)
1R1W	1.78	2R1W	1.69
2R2W	2.26	4R2W	2.67
3R3W	2.89	6R3W	5.06
4R4W	4.65	8R4W	8.17
5R5W	6.02	10R5W	11.99
6R6W	8.16	12R6W	16.51
7R7W	10.63	14R7W	21.73
8R8W	14.36	16R8W	27.65

2.2 Row Decoder Design

The decoder is used to trigger the wordline corresponding to the incoming address. There are two types of decoder: static and dynamic. A dynamic row decoder uses a clock for the precharge. Though a static row decoder does not require clock for decoding, in the case of an SRAM it must be clocked, because the wordline signal has to be LOW during the precharging of the bitline in the negative phase of the clock. Otherwise, a HIGH wordline during precharging might upset the stored value (unintended write). In addition to this, if the wordline is HIGH during precharging, then there can be high power consumption due to the direct path between V_{dd} and GND through the precharging bitline and the bitcell in the selected row.

In the static decoder, either the NMOS or PMOS transistors will be in series, depending on whether it is implemented using NAND or NOR logic. Due to this series connection, the static decoder is quite slower than the dynamic decoder. To increase the speed of the static decoder, it is necessary to size the NMOS and PMOS transistors for greater driving strength. In contrast to that, the dynamic decoder is faster due to better driving strength. The Dynamic decoder requires N+2 transistors for an N-bit decoder, thus, it has fewer transistors as compared to the static decoder. The dynamic power consumption of the dynamic decoder will be high because of the high probability of transitions as compared to the static decoder. After considering all the tradeoffs, it was decided to implement the dynamic decoder.

The dynamic decoder can be implemented using NAND or NOR logic. NOR logic will be faster because of the parallel NMOS transistors as compared to the series NMOS transistors in the NAND logic. Thus, for dynamic logic, the driving strength of NOR type decoder will be higher. For layout of the NOR type decoder, all the NMOS transistors must be connected in parallel, while for the NAND type decoder, all the NMOS transistors must be connected in series. For parallel connection between all the NMOS transistors corresponding to an address line, it is required to keep a minimum distance between the active regions of all transistors. In case of series connection of the NMOS transistors, their active regions can be made continuous and can use the multi-fingered transistor provided by FreePDK45 [2], thus the NAND type decoder consumes less area than the NOR type decoder. In addition to the area consideration, the NOR type decoder will consume more power as compared to the NAND type decoder. In case of NOR, all the precharged nodes will discharge during the evaluation phase except for the node corresponding to the selected wordline. While in the case of NAND, only the node corresponding to the selected wordline will be discharged, and all other will remain at the precharged value. This will significantly save power. Considering area, power and speed tradeoffs, it was decided to implement the row decoder using dynamic NAND type logic.

2.3 Precharge Design

The precharge circuit is used to precharge all the read bitlines before initiating the read operation. Bitlines are precharged to the V_{dd} using a single PMOS transistor per bitline as shown in Figure 2.11. For a memory with 32 or fewer entries, the PMOS width is kept 360nm, while for larger memories the PMOS width is set to 720nm.

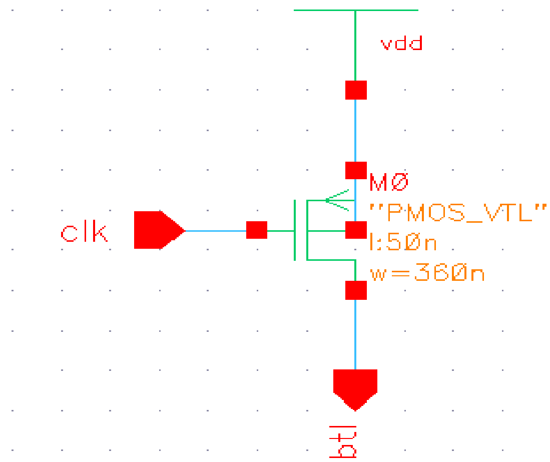


Figure 2.11. Precharge schematic.

2.4 Column Multiplexer Design

When the number of rows (words) in the SRAM is larger than the number of columns (word width), to reduce the access time, it is desirable to restructure the memory so that the number of rows is halved and the number of column is doubled. Due to the restructuring, we need a column multiplexer to select data from one of two bitlines. The column multiplexer is implemented using a transmission gate, as shown in Figure 2.12. If it is required to have one more level of restructuring, to further reduce the access time, another column multiplexer is used to select data from the outputs of two column multiplexers. In the current implementation, no multiplexing, 2-to-1 multiplexing and 4-to-1 multiplexing are supported.

For the write circuit, the column multiplexer is made part of the write data circuit, which is discussed in Section 2.6.

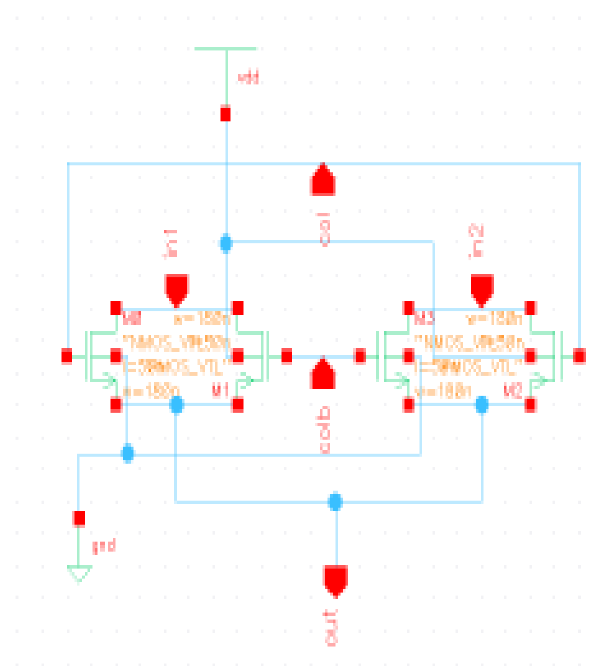


Figure 2.12. Column multiplexer schematic.

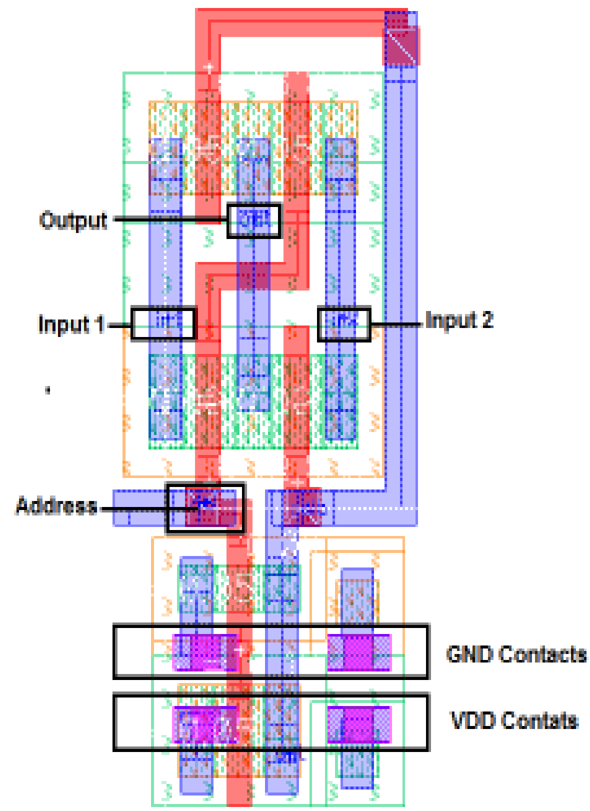


Figure 2.13. Column multiplexer layout.

As shown in the schematic, the NMOS and PMOS widths are set to 180nm for the transmission gate. For layout, an inverter is also attached to the transmission gate for inverting the address line locally for each transmission gate. The area of the column multiplexer is 1.023 μm^2. The layout is shown in Figure 2.13.

2.5 Sense Amplifier Design

The output data, either from the memory array or column multiplexer, if required, should be connected to a sense amplifier for faster read operation. A sense amplifier improves the speed of the read operation by rapidly discharging the bitline. The sense amplifier is implemented using the cross-coupled inverter latch discussed in the literature [6]. Its schematic and layout are shown in Figure 2.14 and Figure 2.15, respectively. But instead of the "sense enable" transistors in both the pull-up and pull-down paths, only NMOS transistor is used for the "sense enable" input. This will create a series connection with the NMOS transistors in cross-coupled inverter. For faster read, the inverter is sized up in the sense amplifier. The width of and PMOS transistor is 360nm and NMOS is also 360nm to maintain its driving strength due to its series connection with the 360nm "sense enable" NMOS transistor.

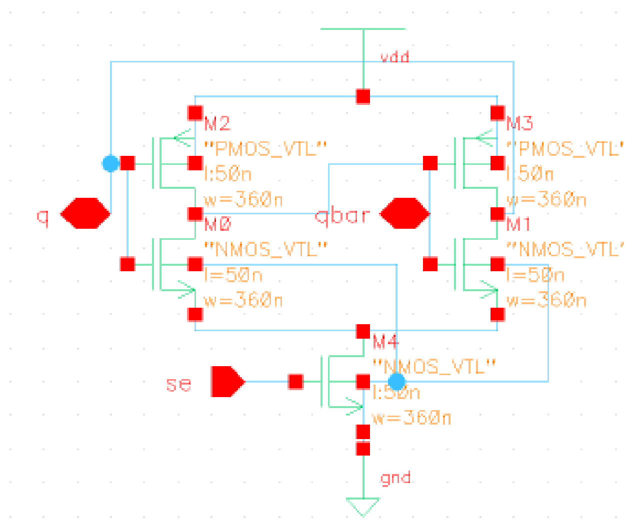


Figure 2.14. Sense amplifier schematic.

The total area of the sense amplifier is 1.0685 μm^2. The sense enable input NMOS transistor is 360nm, but it is implemented using two 180nm NMOS transistors in parallel, as it results in a more favorable geometry with respect to laying out the full SRAM.

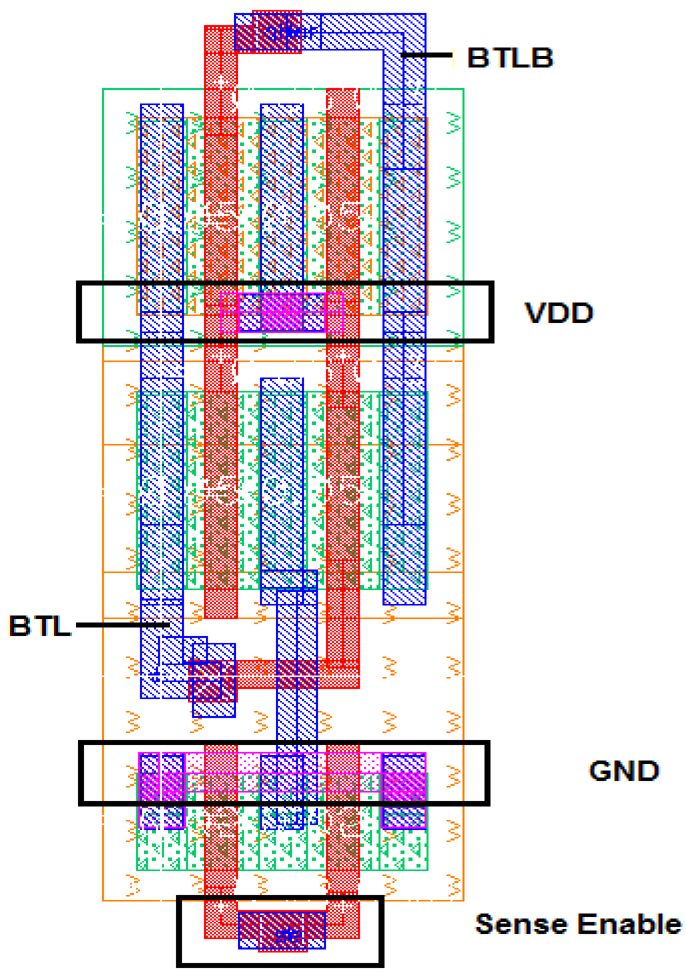


Figure 2.15. Sense amplifier layout

2.6 Write Driver Design

The write driver is used to drive the write BTL and BTLB to the desired logic level according to the write enable, clock and data input logic level. The desired logic levels of BTL and BTLB are given in Table 2.2.

Table 2.2. Write BTL and BTLB logic values.

Clock	Write Enable	Data In	BTL	BTLB
L	X	X	H	H
H	L	X	H	H
H	H	L	L	H
H	H	H	H	L

In the case of column multiplexing, there would be one or two more columns in the above table showing the write address MSBs. According to these bits, if the bitlines are not selected then they should remain HIGH, otherwise they should follow or complement the "Data In" signal in the case of BTL and BTLB, respectively.

The dynamic write circuit can be implemented using pass transistors connected in series for write enable, data in and also for address bits, in case of the column multiplexer. In this case the write bitlines must be precharged using the precharge circuit. For example, if Q of the bitcell is "1" and Qbar is "0" and we want to flip its value, then BTL will discharge to "0" and BTLB will remain at precharged value i.e., at V_{dd}, but it will be in high impedance state. When the wordline is triggered the access transistor turns ON and so the "0" at Qbar may start discharging BTLB in the initial phase of the write operation. The "1" at Q can not increase the voltage at the BTL because of the all ON NMOS transistors in the write circuit. Thus, BTLB will be lowered from the V_{dd} and BTL is at "0". This will result in the slower write operation. To avoid this problem, the robust circuit shown in the Figure 2.16 is used for write driver. As shown below, a dynamic NAND is used to create BTL and BTLB and these outputs from the NAND are given to the chain of sized inverters to drive the write bitlines. In this way, at any time the write bitlines will be driven by the inverters and will be in a low impedance state.

In the first inverter, the width of NMOS and PMOS transistors are 180nm and 360nm, respectively. In the second inverter the NMOS is 360nm wide and PMOS is 720nm wide, which are implemented as two NMOS transistors and two PMOS transistors in parallel to yield a favorable geometry for the overall SRAM layout. The area of the write driver for each case of column multiplexing is given in Table 2.3, below. For the write driver with 2-to-1 column multiplexing, there is only one “col” signal and so there are five transistors (2 for clock, 1 for write enable, 1 for data and 1 for one address line (“col”)). In the case of no column multiplexing, the write driver will have only four transistors.

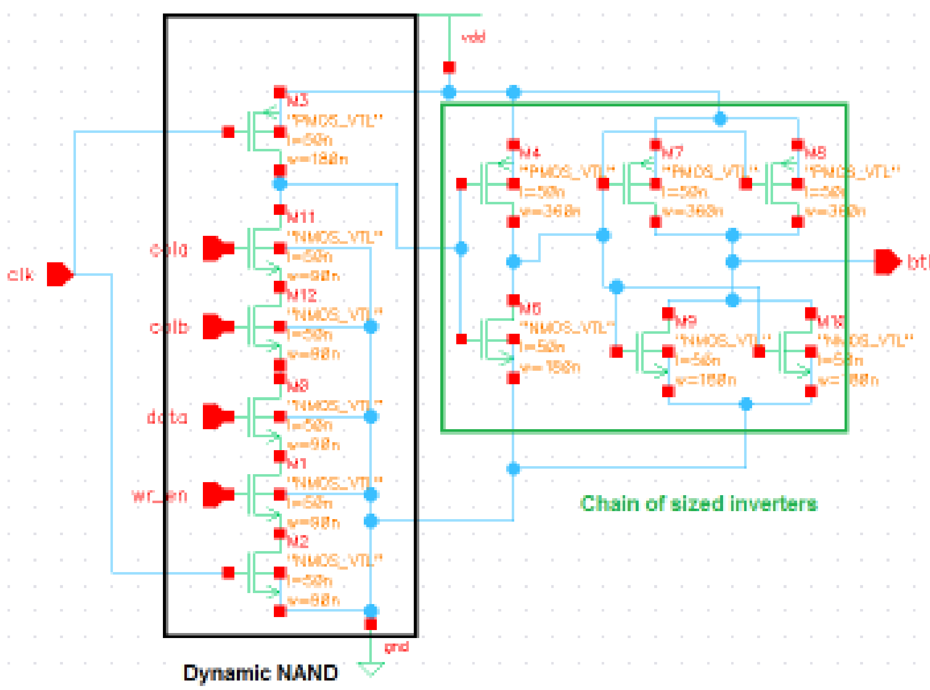


Figure 2. 16. Schematic of write driver for 4-to-1 column multiplexer.

Table 2.3. Write driver area.

Column Multiplexer	Area (μm^2)
NA	1.320
2:1	1.627
4:1	1.656

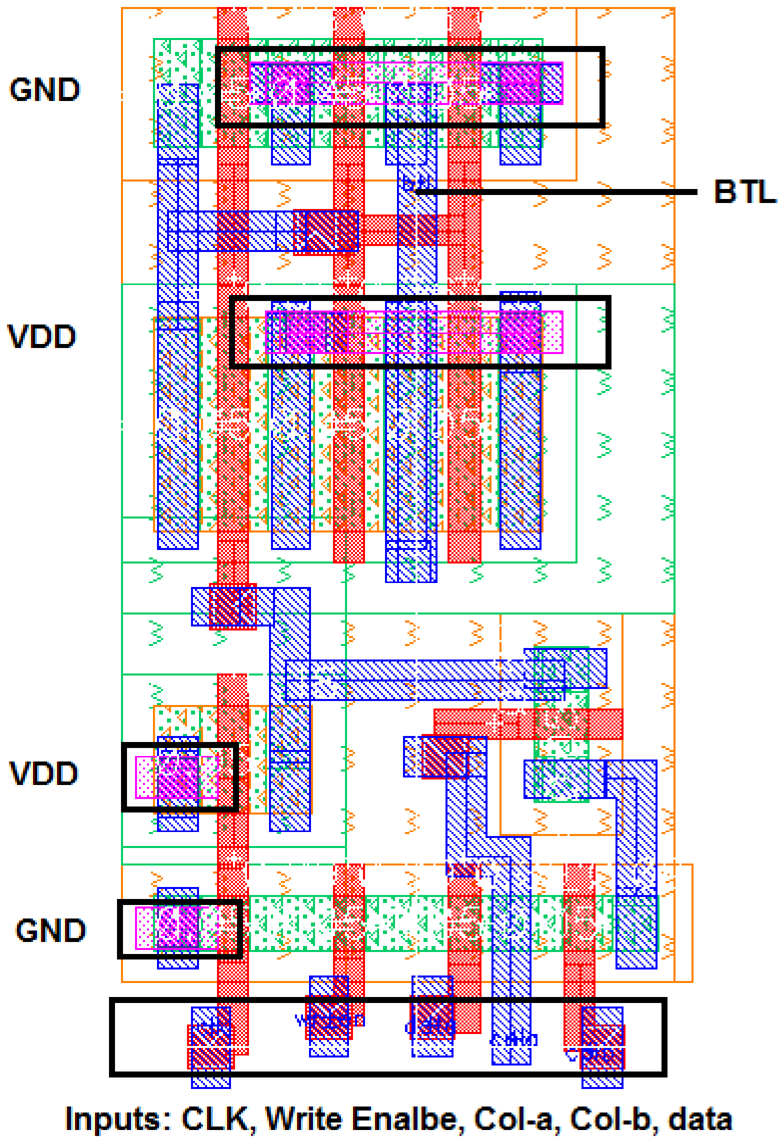


Figure 2. 17. Schematic of write driver for 4-to-1 column multiplexer.

Chapter 3

Multiported CAM Design

CAMs play a crucial role in an out-of-order superscalar processor. A CAM is a hardware algorithm to search for the address of a particular bit sequence in the whole memory, ideally in only one cycle. In a superscalar processor, a CAM is widely used in the "Wake-up" logic in the Issue Queue and in the "Disambiguation" logic in the Load-Store Unit. When an instruction is dispatched, it is allocated an entry in the Issue Queue, i.e., its source tags are written into the CAM port and all other required information (active list ID, load-store unit ID, branch mask, destination tag, PC, etc.) is stored in the payload RAM part. Now, when the instruction is "selected" to go forward to the Register Read (RR) stage of the pipeline, it needs to wake-up any dependent instructions. currently in the Issue Queue. For this wake-up procedure, it broadcasts its destination tag on a read port of the CAM. The CAM raises the match lines of the entries that have source tags that match the broadcasted destination tag.

3.1 Bitcell Design

The CAM bitcell can be designed as a NOR type or a NAND type, as described in the literature [7]. The NOR type CAM is widely used for its faster response time. The NOR type has better noise margins because of the full swing on the match line (ML). 10-T NOR type CAM bitcell is shown in Figure 3.1. In an overall CAM, all bitcells in a row form a word and are connected through a common ML. Data is broadcasted over the select line (SL) and complemented data on SLB. Whenever there is a mismatch (SL != Q), one of the pull-down paths will be ON and the ML will be discharged. If broadcasted data matches stored data in all of the bitcells in a row, the match line will not be discharged. These parallel pull-down paths among all the bitcells resemble the NOR type structure. As the number of write ports and read ports increases, same access transistor and pull-down path are added on both sides of the bitcell. The write port in the CAM bitcell is the same as that in the SRAM bitcell and so it will have the same write driver.

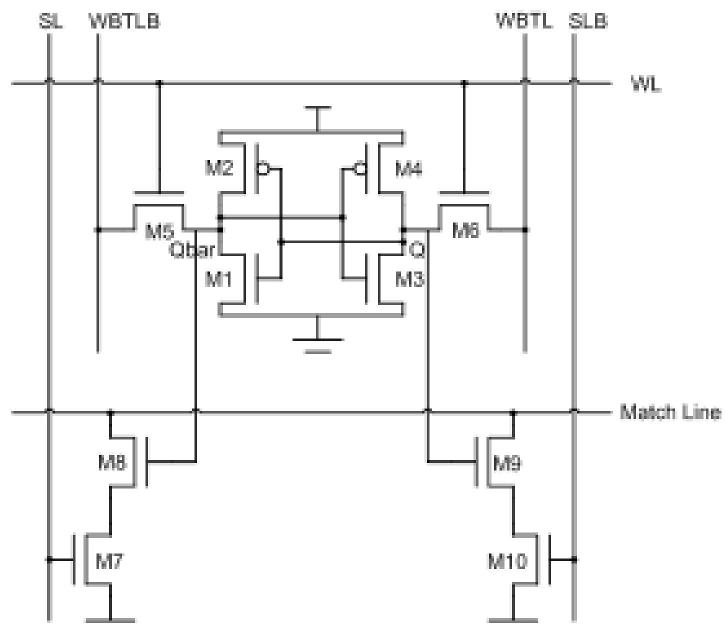


Figure 3.1. 10-T NOR type CAM bitcell.

To speed-up the discharging of the match line, the width of the pull-down NMOS transistor can be increased without impacting the area. Figure 3.2 shows the 4R4W CAM bitcell. Table 3.1 shows the areas of bitcells 1R1W, 2R2W, 3R3W, 4R4W, 5R5W, 6R6W, 7R7W and 8R8W.

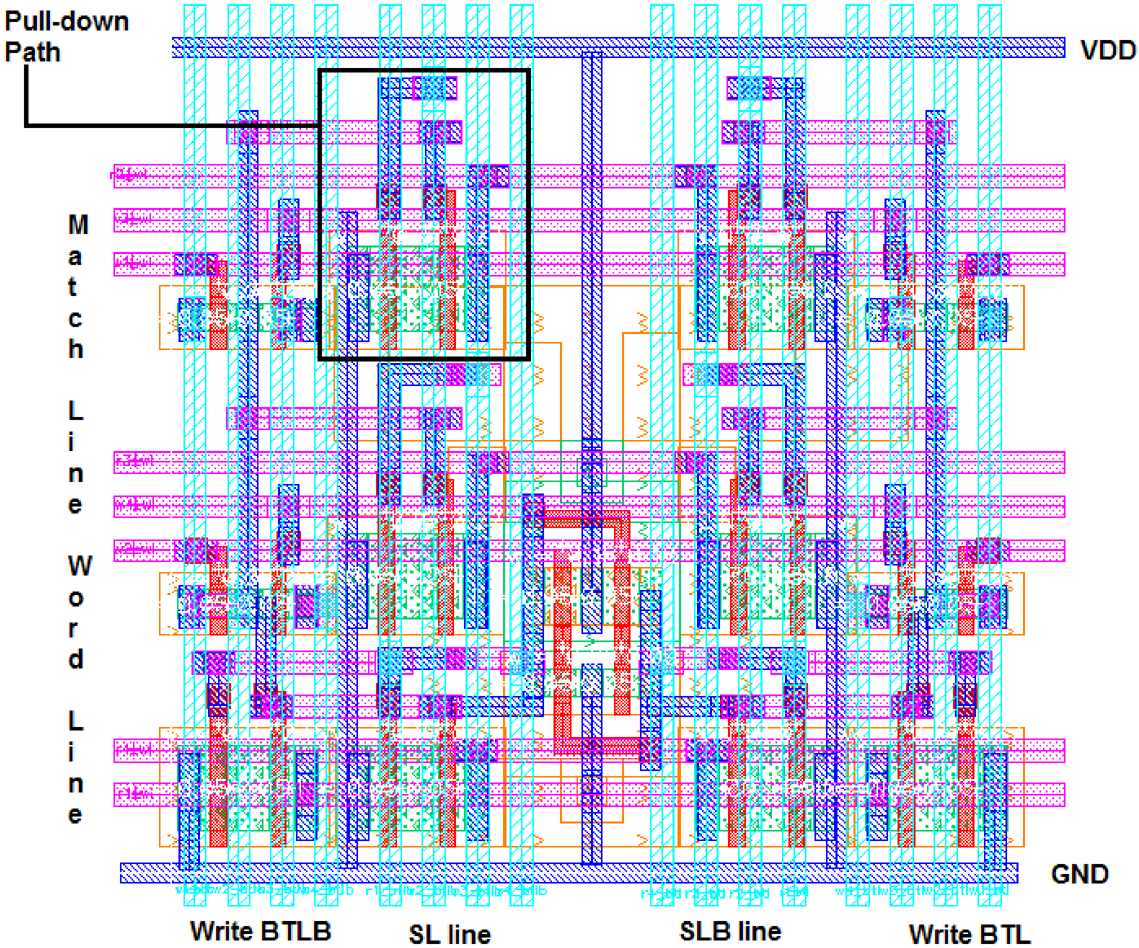


Figure 3.2. 10-T NOR type CAM bitcell.

3.2 Select Line Driver

The select line (SL) carries the broadcasted data, which is compared with the bit values stored in the CAM. The state of the select line depends on the Clock as well as the input data and it follows Table 3.2.

As shown in this table, the SL is simply the logical AND of the Clock and Data, while SLB is the logical AND of Clock and the complemented Data. The select line driver is implemented using dynamic NAND logic, the output of which is driven by a sufficiently sized inverter. In the NAND logic, NMOS transistors are in series and to improve driving strength, these NMOS transistors are sized up to 180nm. The area of the NAND in the select line driver is 0.469 μm^2.

Table 3.1. Multiported CAM bitcells and their areas in μm^2.

Bitcell	Area (μm^2)	Bitcell	Area (μm^2)
1R1W	2.554	5R5W	10.351
2R2W	3.133	6R6W	12.857
3R3W	5.798	7R7W	16.070
4R4W	7.140	8R8W	19.668

Table 3.2. Truth table showing the state of the select line.

Clock	Data	SL	SLB
L	L	L	L
L	H	L	L
H	H	H	L
H	L	L	H

3.3 Other Peripheral Cirucuits

The other peripheral circuits are the same as those used in the SRAM: These include the write driver and row decoder (for the CAM's write ports) as well as the match line precharge circuit (same as precharge circuit used for the read bitlines in SRAM).

Chapter 4

Data Generation and Automated Layout

4.1 Data Generation Tool Flow

The objective of the "Data Generation" tool is to implement an automated tool that, given a set of input parameters, produces accurate area, read access time, write time and energy consumption of the desired memory structure. As discussed in Chapter 1, analytical models suffer in accuracy. It is also not feasible to implement the whole memory structure and then simulate it at the transistor-level due to the large amount of time taken for simulation and also the tremendous memory required. Thus, hybrid approach is taken to just create the critical path and carried out simulation to measure the access time and energy consumption. While for points of interest, whole memory layout can be generated and simulated. The tool flow is little bit different for SRAM and CAM, because the worst case is different for both.

4.1.1 SRAM Tool Flow

4.1.1.1 Area

As shown in Figure 2.8 and Figure 2.10, the pass transistors (access transistors) present in the bitcell can be hidden under the bitline, but can not be hidden under the wordline due to the minimum allowable spacing by the technology. Thus, the width of the bitcell can be calculated by knowing the number of ports. This is true for higher number of ports. For too few ports, for example 2R1W, there are not enough bitlines to hide all the 6 access transistors. In this case, the width of the bitcell will be the sum of the width of the cross-coupled inverters and the number of access transistors placed in one row. So, the width of the bitcell is

$$width = Max\begin{pmatrix} (2 * (PITCH\ of\ Metal\ layer\ 3) * total\ number\ of\ ports), \\ (Number\ of\ access\ transistors * Transistor\ Width \\ + Width\ of\ the\ cross - coupled\ inverters) \end{pmatrix}$$

Thus, the width of the bitcell is the maximum of the above two factors. As the access transistors are not hidden under the word lines, they will contribute to the height of the bitcell. The total height of the bitcell can be calculated as shown below:

$$\begin{aligned} height = Total\ ports * (PITCH\ of\ Metal\ layer\ 2) \\ + (Height\ of\ the\ access\ transistors \\ * Number\ of\ access\ transistors\ in\ one\ column\ in\ bitcell) \end{aligned}$$

Using above equations, the height and width of the bitcell can be calculated. The total area consumed by the memory array will be the sum of areas of all the bitcells. The precharge height can be calculated by knowing the PMOS transistor width used for precharging. Although all the PMOS transistors will not fit into the width of the bitcell due to many bitlines (Figure 4.1), there can be two rows of PMOS transistors. If the column multiplexer is present, then its height should be added to the total height according to the degree of multiplexing. The sense amplifier height is also added to the total height. As with the precharge circuit, it is possible to have two rows of the column multiplexers and sense amplifiers. The number of these rows is calculated by dividing the bitcell width by the width of the multiplexer. This division shows that how many multiplexers can be placed in one row. The extra wire routing from the bitcell to the column multiplexer and to the sense amplifier is also taken into account. Approximately, there are as many as total read-port bitlines routed horizontally for interconnections to the multiplexer as well as to the sense amplifier. The write driver height is added to the total height. The sense enable, write enable signal and address bit signal separate for each port will be routed in horizontally to each circuit and that will increase the total height. The row decoder will increase the width of the total memory. The pull-down circuit (NMOS) of the row decoder is hidden under the address lines routed vertically (Figure 4.2). The width of this pull-down circuit will be 2 * PITCH of metal layer-2 * number of address lines used in the row decoder. The width of PMOS transistor, used in precharge circuit in row decoder, need to be added for all the ports. The width of the word line driver is also added for each port. Total area is calculated by calculating the total width and total height.

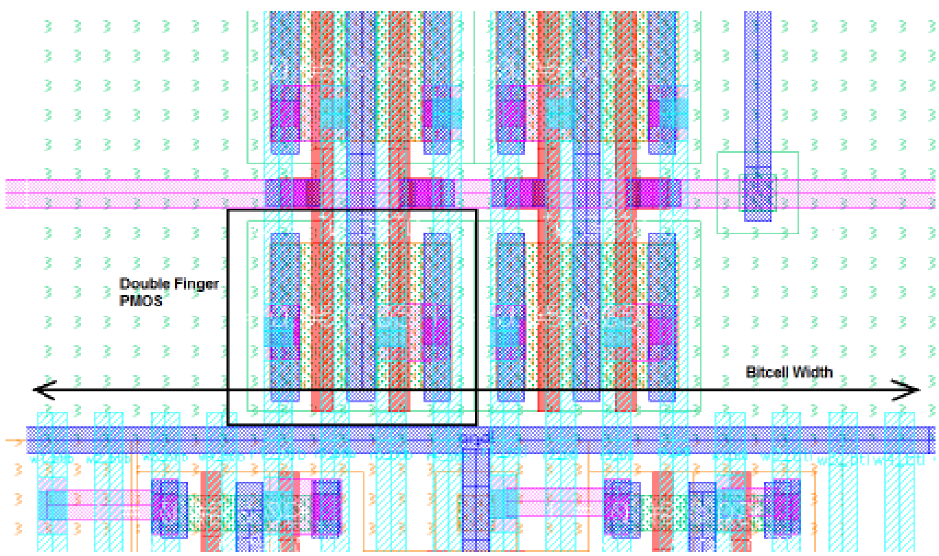


Figure 4.1. Placement of precharge circuits.

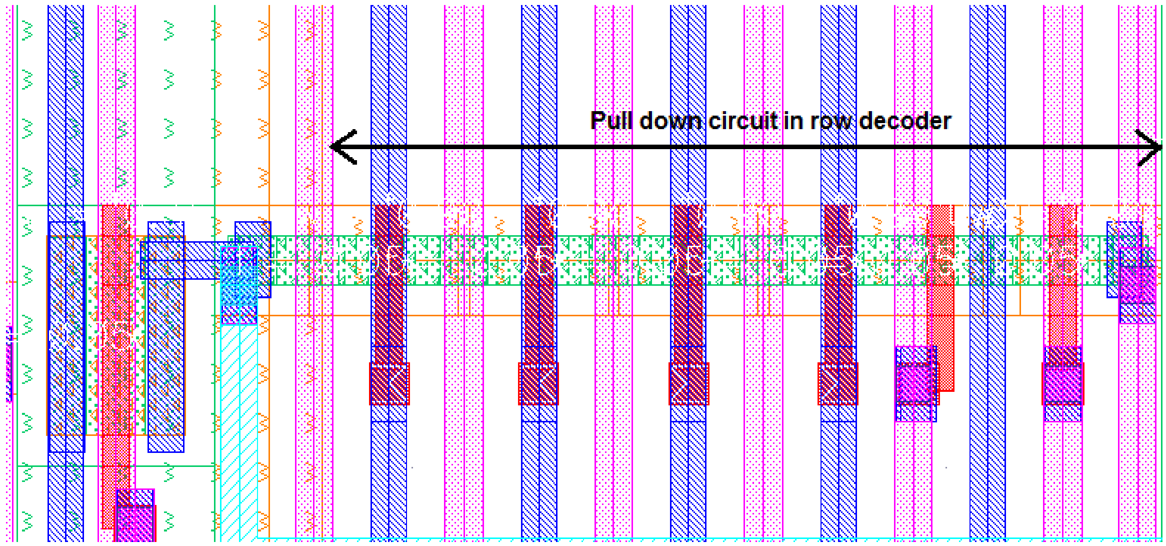


Figure 4.2 Pull-down circuit in row decoder hidden under address line wires.

4.1.1.2 Access Time and Energy Consumption

The access time and energy consumption are measured by simulating the critical path. For the critical path generation, first, the two rows and one column of SRAM bitcells are instantiated, as shown in Figure 4.3. The choice of the two rows of bitcells is discussed later in Section 4.2. If we take the largest SRAM produced by FabMem like 16R8W 512X32, then the length of the bitline will be 526 um. The resistance of such a wire is 1.8 kΩ, which is quite smaller than the resistance of an NMOS transistor. Thus, bitcells are C+CC (intrinsic and coupling capacitance) extracted netlist, but not R+C+CC. Due to the absence of resistance (R) in the extracted netlist, the location of the row and column will not affect the measurement of access time and energy. All the bitcells in the same row have a common wordline for each port but different bitline and all the bitcells in same column have common bitline for each port but different wordlines. After placing bitcells, a separate precharge circuit is instantiated for each read bitline. In the same way, a column multiplexer (if present), sense amplifier, write driver and output driver are placed with appropriate interconnections to each other and to the bitlines. For the input to the wordline, first the input address line driver is placed and its output is connected to the inputs of the two row decoders present for the two rows of memory. The output from the row decoder is given to the input of the wordline driver.

The basic bitcell netlist is an extracted netlist. So, it covers delays caused by the wordline and bitline, which are major components of the total access time delay. All other circuits are not extracted netlists and so they require modeling the intrinsic and parasitic capacitances of the remaining wires. The intrinsic and coupling capacitances are calculated using the following formula [8] for three parallel lines on the same layer over a ground plane. Here, S = spacing between two wires, W = width of the wire, T = thickness of the wire, H = height from the ground plane.

$$\frac{C_{int}}{\varepsilon_{ox}} = \frac{W}{H} + 2.24\left(\frac{W}{H}\right)^{0.0275}\left(1 - 0.85e^{-0.62\left(\frac{S}{H}\right)}\right) + 0.32\ln\left(\frac{T}{S}\right)\left(0.15\,\frac{S}{H}\,e^{-1.62\frac{T}{S}} - 0.12\,e^{-0.065\frac{S}{T}}\right)$$

$$\frac{C_{couple}}{\varepsilon_{ox}} = \frac{T}{S} + 1.31\left(\frac{T}{H}\right)^{0.073}\left(\frac{S}{H} + 1.38\right)^{-2.22} + 0.4\ln\left(1 + 5.46\frac{W}{S}\right)\left(\frac{S}{H} + 1.12\right)^{-0.81}$$

For measuring energy consumption, different voltage supplies are used for each circuit instantiation. The objective is to measure the read energy consumption and write energy consumption for a given memory structure. In the critical path only one column and two rows are placed. So, the bitlines corresponding to the row 1 and column 2 bitcell will be precharged very fast as compared to the row 1 and column 1 bitcell, and they will not represent the actual results. So, even for the same circuit, for example sense amplifier, a different voltage supply is used for the one port of the row 1 and column 1 bitcell, separate from the voltage supply for other ports and other bitcells. The supply voltage distribution is shown in Figure 4.4, below.

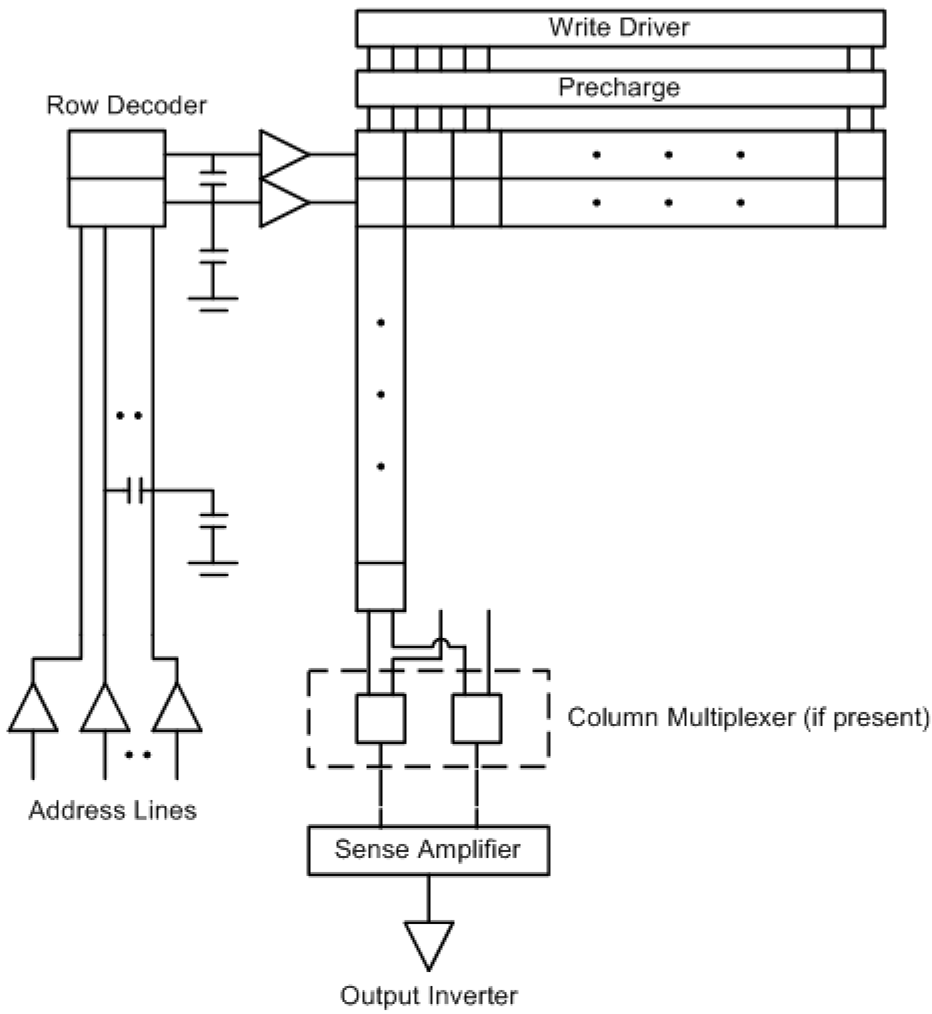


Figure 4.3. Critical path in SRAM.

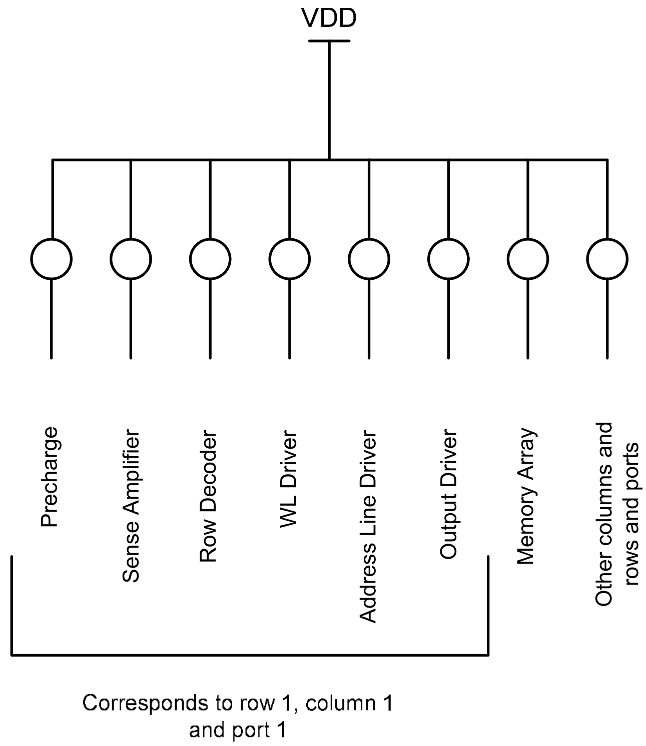


Figure 4.4. Supply voltage distributions.

Energy consumed in each of these supply voltages is measured to calculate the read energy and write energy consumption per port. This energy is multiplied by the width of the memory or depth of the memory, depending on the circuit, to calculate the total energy consumed in that circuit. For example, to measure the precharge energy, the energy obtained from the above supply is multiplied by the width of the memory (number of bitcells in each row).

4.1.2 CAM Tool Flow

4.1.2.1 Area

The CAM bitcell has more transistors than the number of bitlines. For example, the 4R4W CAM bitcell has 16 bitlines (8 write bitlines and 8 select bitlines), where it has 28 transistors (4 in cross-coupled inverters, 8 write access transistors, 16 for comparisons). Thus, the width of the bitcell can be calculated by knowing the number of transistors in each row. The width of the CAM bitcell is given by:

$$width = \left(\frac{number\ of\ row\ transistors}{2} + 1\right) * Width\ of\ double\ fingered\ transistor$$

The height of the CAM bitcell can be calculated from the total number of WL and ML and the number of NMOS transistors in one column in the bitcell layout.

$$\begin{aligned} height = {} & (read\ ports + write\ ports) * PITCH\ of\ metal\ layer\ 2 \\ & + number\ of\ transistors\ in\ one\ column * (height\ of\ the\ NMOS \\ & + PITCH\ of\ metal\ layer\ 2) \end{aligned}$$

The match line precharge circuit width is added to the total width. As in the SRAM case, the row decoder length and word line driver length is added to the total width. The select line driver height and write driver height are added to the total height. The increase in height and width due to the wire routing is also added to the total height and width.

4.1.2.2 Access Time and Energy Consumption

For the access time and energy consumption measurements the critical path shown in Figure 4.5 is simulated. The critical path consists of one row and one column of bitcells, connected by sharing a wordline and match line and bitline, respectively. For measuring of the energy consumption, the same voltage supply distribution is done same as in the SRAM case.

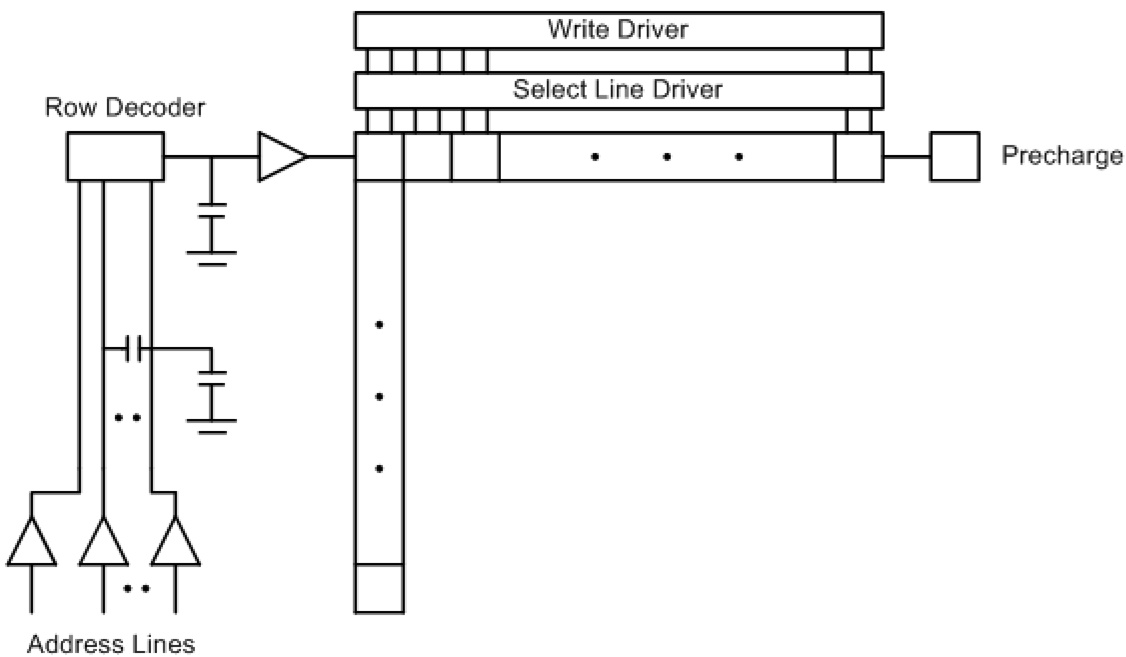


Figure 4.5. Critical path in CAM.

4.2 Test Vectors and Measurements

The test vectors to measure the access time and energy consumption should trigger the worst-case scenario. For the worst case, both the loading on the bitcell as well as the parasitic capacitance should be taken into account. The energy consumption will be more if the access time is more. So, during the energy consumption measurement it is assumed that the worst-case access time will give the maximum energy consumption. The test vectors and measurements for SRAM and CAM are described in Sections 4.2.1 and 4.2.2, respectively.

4.2.1 SRAM

The bitlines are placed on metal layer 3. In a multiported SRAM, there are many bitlines and so they are spaced by the minimum allowable distance in the 45 nm technology node to minimize the area of the bitcell. This creates large parasitic capacitances between adjacent bitlines. The bitlines are placed in the following order from left to right:

Write BTLB (1 to N) – Read BTLB (1 to N) – Read BTL (1 to N) – Write BTL (1 to N)

The coupling capacitance between two wires produces cross-talk noise [6] and introduces coupling delay, slowing down the driving of the victim wire. The simplified equation of capacitance of the victim wire is given by [9]. The β factor is positive if both the victim and aggressor are working in one direction and negative for the opposite case.

$$C = C_{int} + (1 - \beta)C_{couple}$$

Before initiating a "Read", all bitlines are precharged to V_{dd}. If the bitline is reading "0", then it will discharge towards GND and if it is reading "1", then it will stay at "1". Due to precharging to V_{dd}, two bitlines will never be working in opposite directions (one discharging and one charging), because one of them will be stable at V_{dd}. If both are discharging then β will be positive and so they will help each other and reduce the delay. So, in the worst-case, a bitline will be discharging and its adjacent bitlines will be stable at V_{dd}. Table 4.1 shows how the coupling capacitance varies in an 8R8W bitcell.

Table 4.1. Intrinsic and coupling capacitance in 8R8W bitcell.

R1_BTL intrinsic capacitance	0.036100 fF
Coupling Capacitance	
R1_BTL to R2_BTL	0.230800 fF
R1_BTL to R3_BTL	0.004000 fF
R1_BTL to R4_BTL	0.000693 fF

In a multiported SRAM, many ports can read the memory from different locations. If all the ports are reading from the same location, all the bitlines will be discharging through the same NMOS transistor. Such simultaneous reads will be the worst-case for the access time consideration. In summary, both coupling noise and simultaneous reads must be evaluated to determine the worst-case access time. Table 4.2 shows the access times for bitcells from 8R8W to 2R2W, varying the number of ports for simultaneous access. During the simulation, if the port is not simultaneously reading the bitcell then it is reading the bit value "1" in another location to maximize the coupling noise. The access time is measured on port 1. During this simulation, the write enable was kept LOW so that all the write bitlines remain at V_{dd}. This will maximize coupling delay with read bitlines.

Table 4.2. Access times of different bitcells until sense amplifier is triggered.

Simultaneous read on ports	1	2	3	4	5	6	7	8
8R8W	0.337	0.373	0.407	0.438	0.451	0.481	0.501	**0.502**
7R7W	0.3269	0.3588	0.3901	0.4201	0.4307	0.4448	**0.4608**	
6R6W	0.3193	0.3548	0.3808	0.3986	0.4232	**0.4369**		
5R5W	0.3178	0.3416	0.3614	0.3819	**0.3949**			
4R4W	0.3164	0.3407	0.3693	**0.3769**				
3R3W	0.3029	0.3581	**0.3869**					
2R2W	0.3073	**0.3516**						

From Table 4.2, it is clear that the worst case is simultaneous reads on all ports of the same bitcell. The second worst-case is simultaneous reads on Rp-1 ports, while the bitline adjacent to port 1 is reading “1”. On average, for all the bitcells, the second worst-case is near to the worst-case. The second worst-case covers both simultaneous reads, because Rp-1 ports are reading from the same location, and the worst-case coupling delay, because both of the bitlines adjacent to the port 1 are stable at V_{dd}. Thus it is clear that both the cases (Rp simultaneous reads and Rp-1 simultaneous reads) should be evaluated for finding the worst case. From the Table 4.1 it is clear that coupling capacitance is very significant for the adjacent bitline, while for the next-to-adjacent bitline it does quite insignificant and so do not have any effect on delay.

A “write” operation can not be done with more than one port writing to the same location. While the clock is LOW, all the write bitlines are kept HIGH. For a “write” to occur, either BTL or BTLB will transition from HIGH to LOW. For worst-case consideration, all ports except port 1 are disabled by keeping their write enables LOW so that adjacent bitlines will be at V_{dd}.

Based on the analysis above, testing is carried out using the following sequence of operations. A “RED” line indicates the bitline is at V_{dd}, while a “BLUE” line shows the bitline is discharging towards GND.

Cycle 1: Data = ‘1’ is written into Bitcell 0 in Row 0, as shown in Figure 4.6.

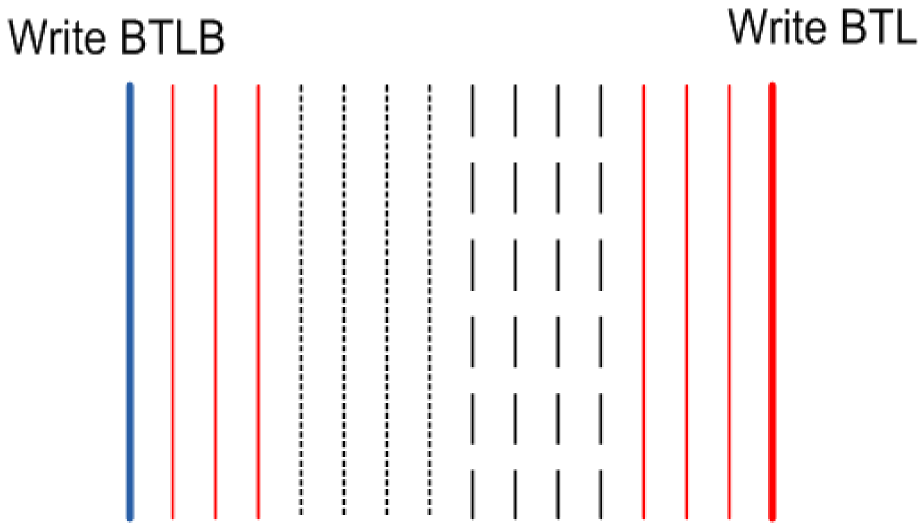


Figure 4.6. “Write” operation in Cycle 1.

Cycle 2: Data = ‘1’ is written into bitcell 0 in Row 1, as shown in Figure 4.7.

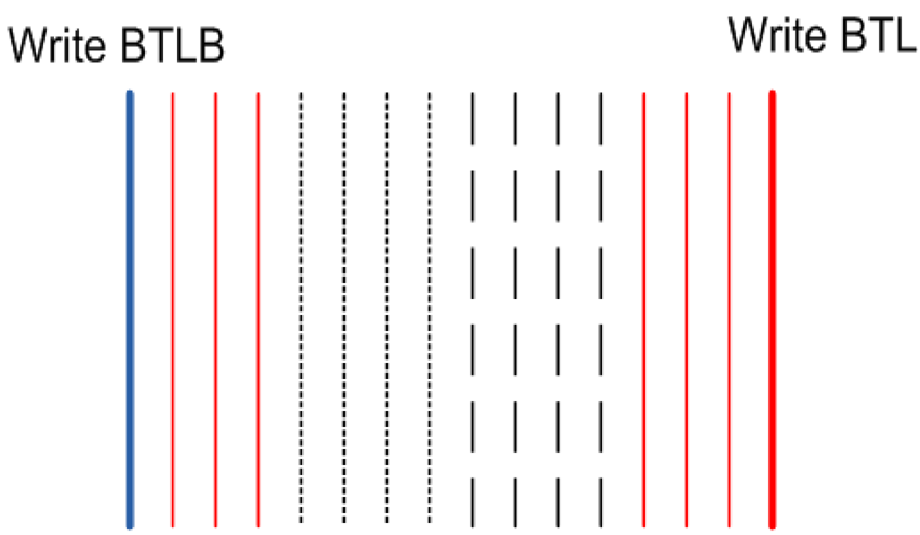


Figure 4.7. “Write” operation in Cycle 2.

Cycle 3: Data = ‘0’ is written into Bitcell 0 in Row 1, as shown in Figure 4.8.

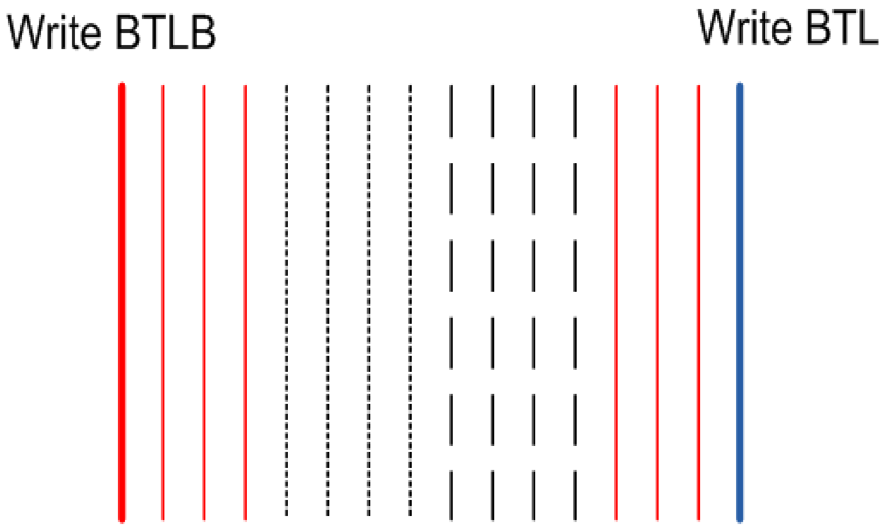


Figure 4.8. “Write” operation in Cycle 3.

For the “Read” operation:

Cycle 3: Simultaneous reads on all the ports on address 0 as shown in Figure 4.9.

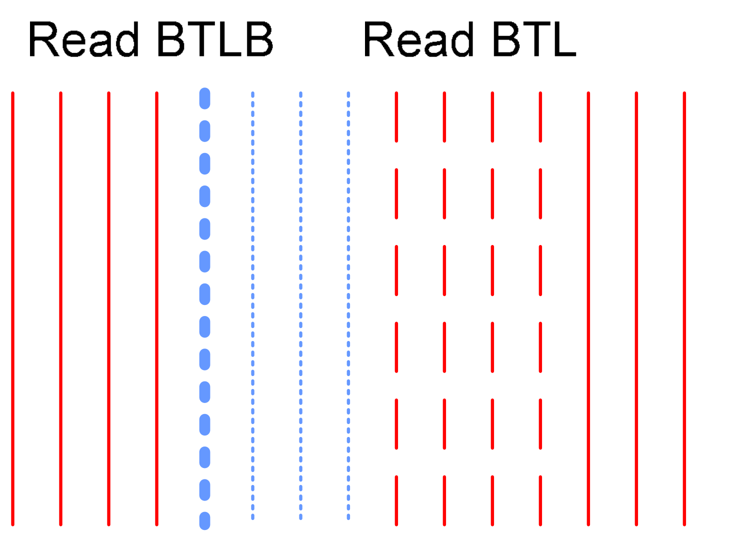


Figure 4.9. “Read” operation in Cycle 3.

Cycle 4: All the ports read address 0 (row 1) except for the port adjacent to the port, as shown in Figure 4.10.

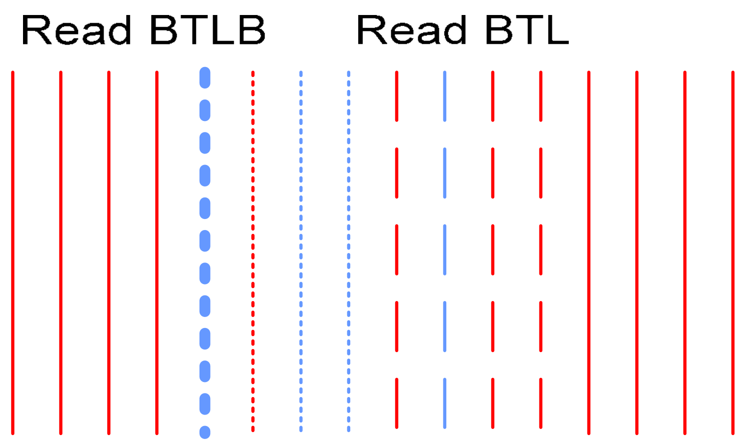


Figure 4.10. "Read" operation in Cycle 4.

During cycle 1, only the write operation on address 0 is active. During cycle 3, the opposite of the value in cycle 2 is written to the address 1 to ensure that the bitcell is actually getting written and its stored value is getting flipped. In some cases, it might be possible that from the start of the simulation, the bitcell is holding "1" and by writing "1" in cycle 2, the write latency can not be measured. By writing first "1" and then "0", it is ensured that the bitcell is getting written and correct write latency is measured. During cycles 1, 2 and 3, the dynamic write energy consumption can be measured in all the peripheral circuits as well as the bitcells. During cycle 4, the static energy consumption in the write driver, word line driver and row decoder can be measured.

During cycle 3, simultaneous reads are performed and the access time is measured. During cycle 4, read port-2 will be reading address 1 and so that bitline will be reading the opposite value as compared to other bitlines. The read access times will be the maximum of the read access time measured during cycle 3 and cycle 4. During cycles 1 and 2, the read ports are not reading and so static energy consumption in the row decoder, word line driver, sense amplifier, precharge, etc., can be measured. During the static energy consumption measurement, the gate leakage energy in peripheral circuits is not measured. The leakage energy for each bitcell is pre-computed and used to calculate the total leakage energy of the memory array. The bitcell leakage energy includes both the gate leakage and drain leakage.

4.2.2 CAM

In the CAM, the dominant contribution to total delay is the match line delay. If only a single bit is mismatched with the broadcasted data bits, then the match line will discharge through a single pull-down path present in that bitcell. Until this match line gets discharged to the desired voltage level, it is not known whether or not the entry matches. During discharge of the match line, if adjacent match lines are kept at V_{dd} then they will have additional coupling delay. For "write" operation, the following sequence of operations is performed:

Cycle 1: Write the bit pattern 101010… into address 0 as shown in Figure 4.11.

Cycle 2: Write the bit pattern 010101… into address 0 as shown in Figure 4.11.

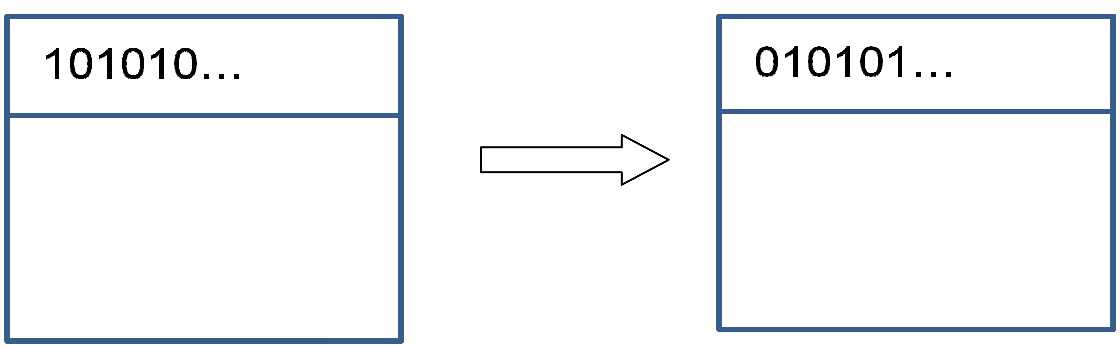


Figure 4.11 "Write" operations in cycle 1 and cycle 2.

Cycle 3: Write port is disabled.

Cycle 4: Write port is disabled.

For "read" operation, the following sequence of operations is performed:

Cycle 1: On all the read ports, data 111111… is broadcasted (don't-care case).

Cycle 2: On all the read ports data 111111… is broadcasted (don't-care case).

Cycle 3: Broadcast the bit pattern 010101… on all ports, as shown in Figure 4.12.

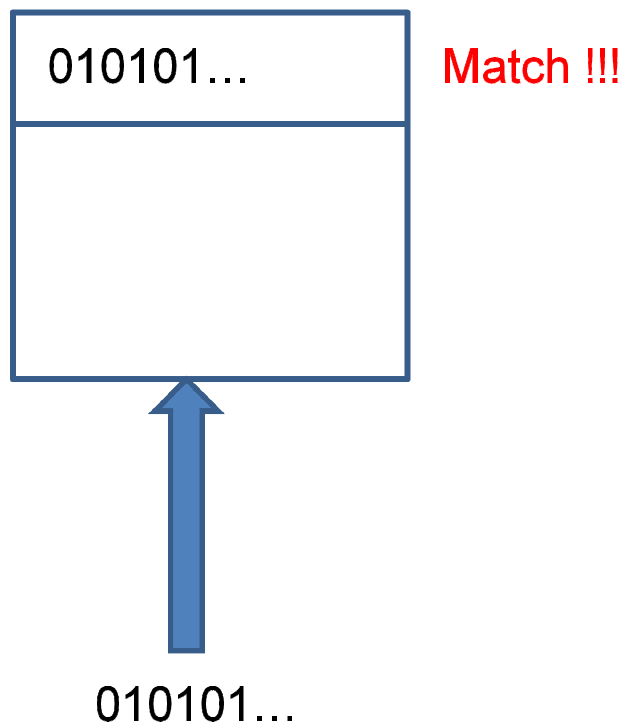


Figure 4.12 "read" operation in cycle 3.

Cycle 4: Broadcast the bit pattern 11010101 on port 1 and 010101 on all other ports as shown in Figure 4.13.

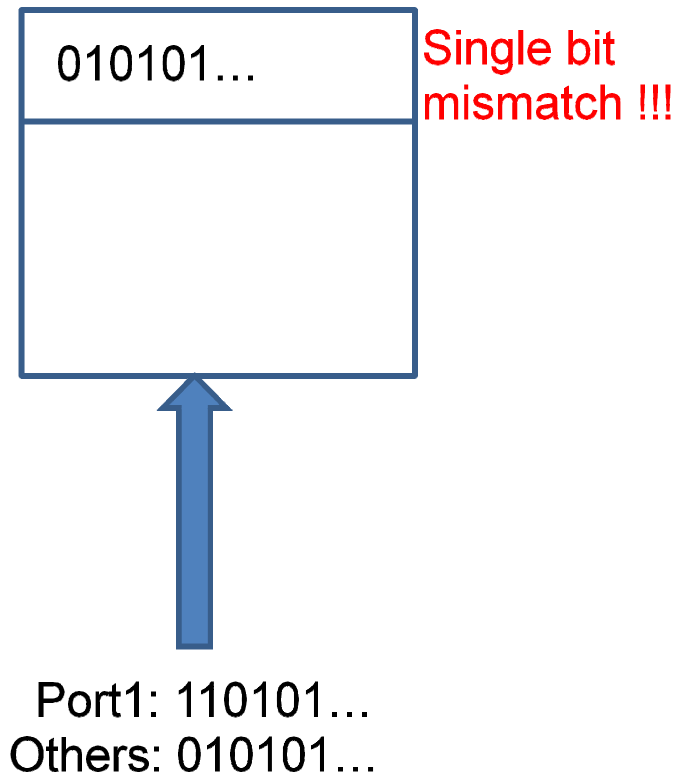


Figure 4.13. "read" operation in cycle 4.

From the above bit patterns, it is clear that in cycle 3, all the read ports will match with the stored data and their match line will not discharge. In cycle 4, the broadcasted data on port 1 will mismatch only in the first bitcell and the match line corresponding to that port will discharge through the pull-down path in that bitcell. All other matchlines will stay at V_{dd}, which will create coupling delay on the discharging match line. The static and dynamic energy consumed in the select line driver, matchline precharge, row decoder, write driver etc. are measured during their idle periods.

4.3 Automated Layout Tool Flow

The multiported SRAM and CAM compiler is a tool that generates a wide variety of compact memory structures. The tool uses manually implemented basic circuits for automated placement and route of each basic circuit using scripts.

4.3.1 SRAM

For the automated layout generation, the SKILL language supported by Cadence Virtuoso 6.1.2 is used. The placement and route of basic circuits follows the high level block diagram shown in Figure 4.14 for SRAM.

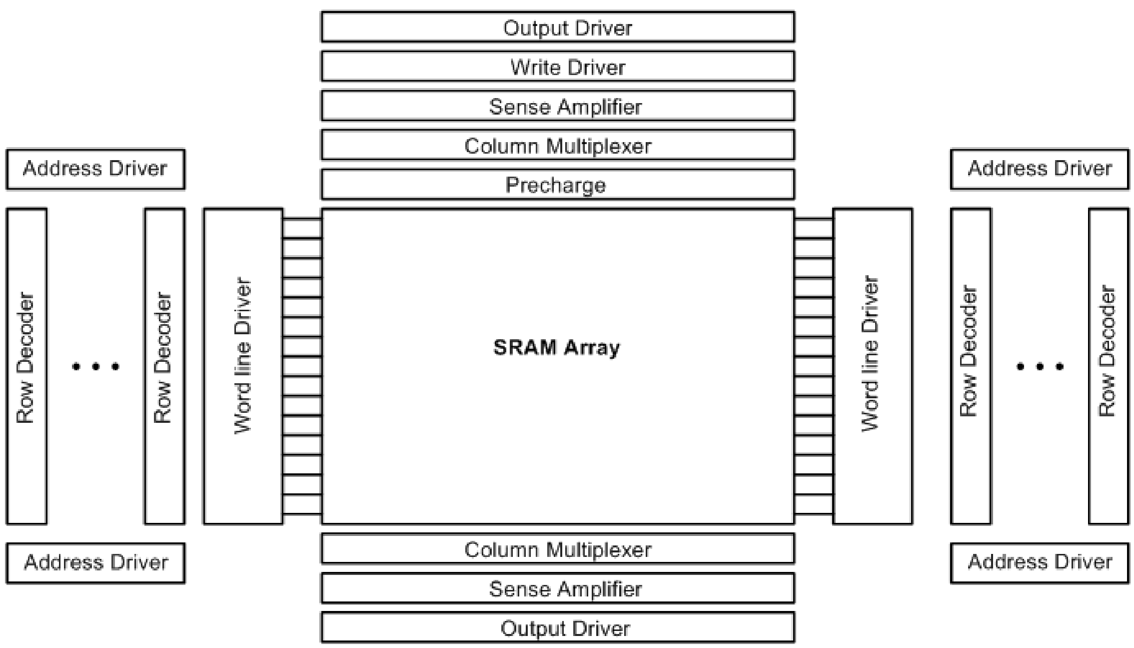


Figure 4.14. High level view of placement and route in SRAM.

The layout is initiated with the placement of bitcells to create the “Memory Array” as per the input of degree of column multiplexing. On every alternate row, the bitcells are flipped vertically so that two rows of bitcells can share either the GND or V_{dd} line. In the layout of bitcell, the bitline and wordline are kept long enough so that when they are placed, they got connected with adjacent bitcells. After the memory array, word line drivers and row decoders are placed. Half of the total numbers of address ports are distributed on each side of the memory array. For this distribution, first the layout of the bitcell is read using its “database object” and then it is queried to find the location of each word line. Then word lines are sorted in order and, for every alternate wordline, a row decoder is placed either side. As each port has different address lines, each row decoder requires separate space. For simplicity, the placement of the wordline driver is done in a similar fashion, resulting little more area as compared to an optimized placement. Such increase in area will be low for a small number of ports. In the current implementation, the selection of wordline driver is kept fixed (independent of the wordline length) and consists of three sized inverters in a chain. Half of the address line input ports are distributed on each of the upper and lower sides of the layout. Thus, address input ports are evenly distributed in each corner of the memory. This distribution is carried out to make the whole layout more compact and to contain the used area within a rectangular shape. Each row decoder has all address lines and inverted address lines as inputs. The address lines and inverted address lines are adjacent to each other and will have larger coupling capacitance. To decrease this coupling, they are alternatively placed in metal layer-1 and metal layer-2.

Precharge circuits are placed on the top of the memory array. The placement of all the circuits connected directly or indirectly to bitlines is carried out in such a way that bitlines are routed as little as possible and also bent as little as possible. The precharge circuits can not be placed linearly in one row, because if it is placed outside of the bitcell width then the bitline needs to be routed more. Such routing will increase further when we move to the next bitcell, and so on. Thus all the precharge required by a bitcell are accommodated in its width by placing them in two rows and these two rows share the clock in between them. Half of the read output ports are distributed on each of the top and bottom sides of the memory array. The column multiplexer is placed according to the degree of column multiplexing specified in the input. Write drivers are also placed in two rows to fit into the width of the bitcell. In the current implementation, IO pads are not implemented and so the input data to the write driver is not routed from outside of the layout, but it originates from the input to the write driver. Different examples of layouts are presented in the Section 4.4.

The separate netlist generator produces the netlist corresponding to the generated layout. This netlist can be used for LVS (Layout versus. Schematic) checking: confirming the layout matches the schematic. After generating a large number of memories, it is assured that the memories generated by the compiler successfully pass the DRC (Design Rule Check) and LVS checks. The parasitic extraction (PEX) of the error free layout can be carried out to generate extracted SPICE netlist of whole memory for simulation purpose. The integrated tool flow of the “Data Generation” and “Layout Generation” tools is shown in Figure 4.15. The tool flow also shows where human presence is required.

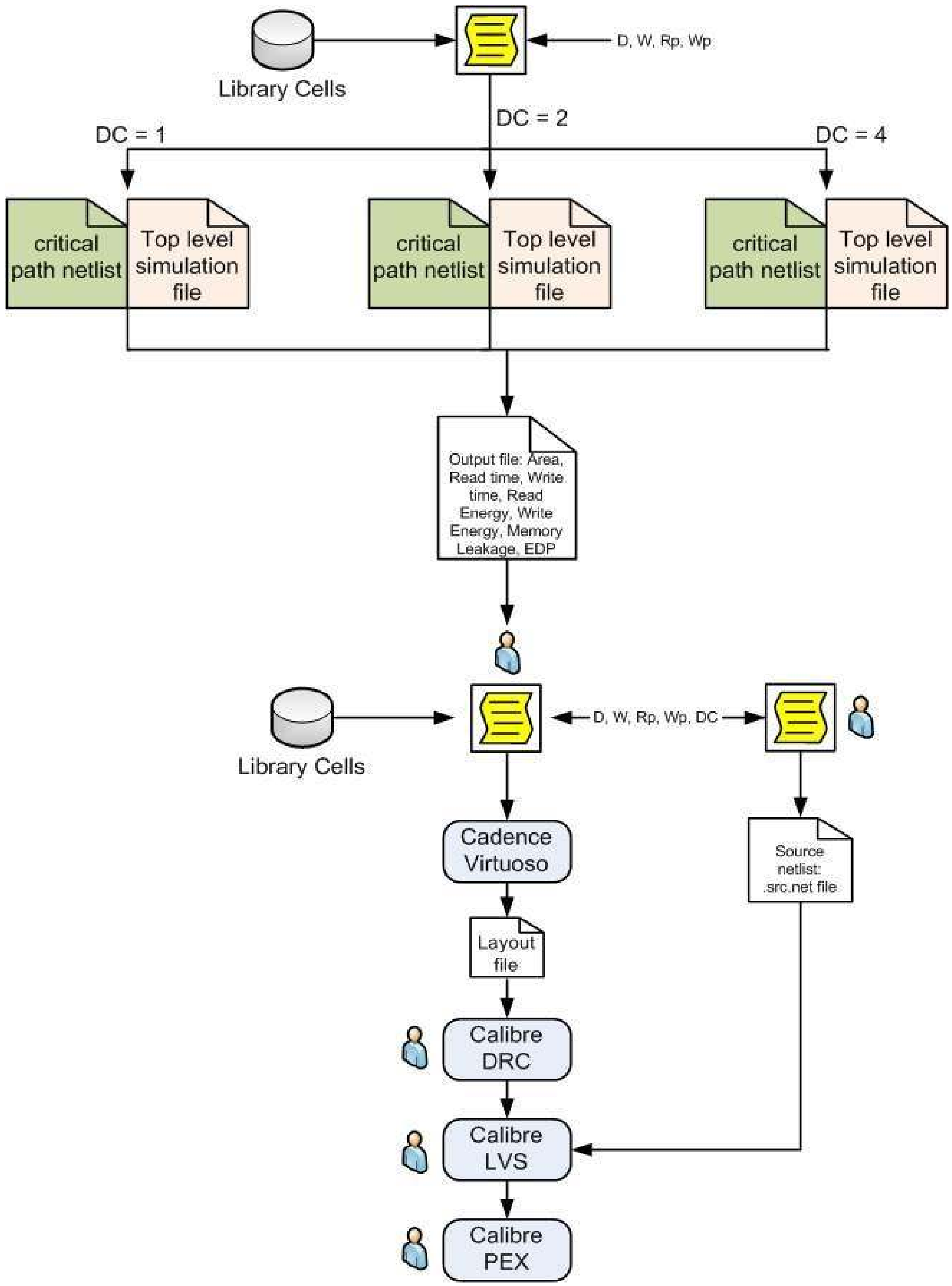


Figure 4.15. Integrated tool flow.

4.3.2 CAM

The CAM compiler is very similar to the SRAM compiler. The high level layout is shown in Figure 4.16.

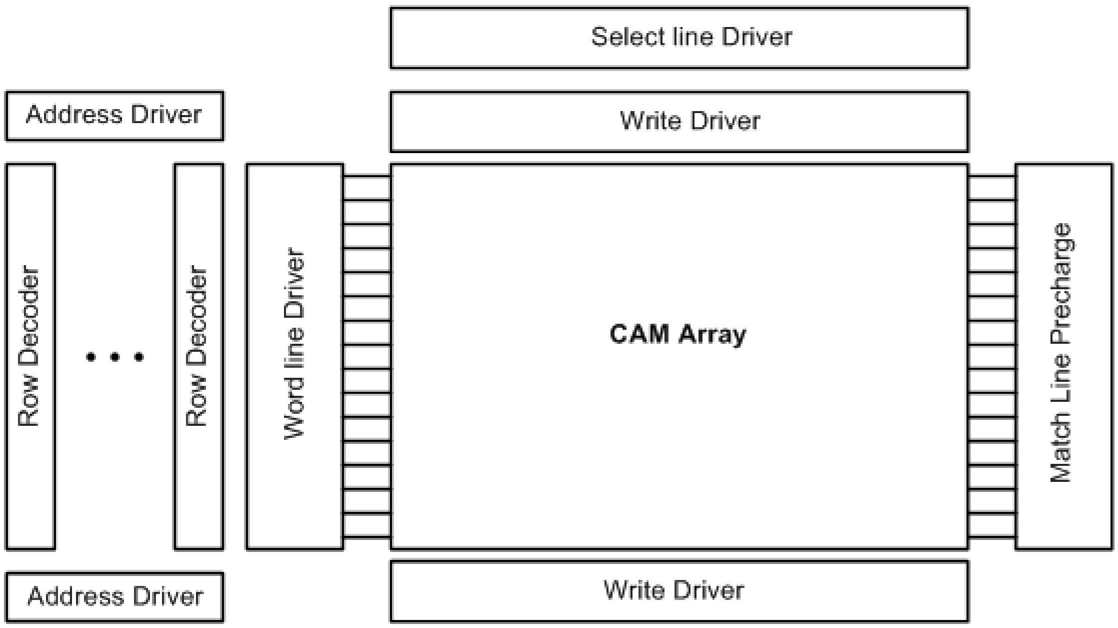


Figure 4.16. High level view of placement and route in CAM.

As shown in the figure, all row decoders corresponding to the write ports are placed on the left side of the memory array. The write drivers are divided and placed on both the upper and lower sides of the memory. To accommodate the bitcell width, the select line drivers are placed in multiple rows depending on the number of read ports and the bitcell width. The integrated tool flow for the CAM compiler will look exactly like Figure 4.15, except for the variability of degree of column multiplexing.

4.4 Sample SRAM and CAM layouts

Figures 4.17 to 4.20 shows the layouts of some sample SRAMs and CAMs generated by the tool.

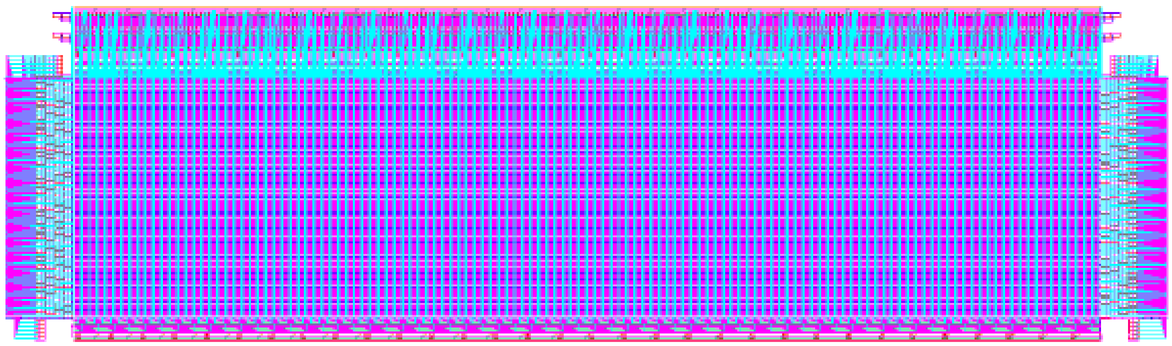


Figure 4.17. 128 X 32 4R4W with 4:1 column multiplexer SRAM layout.

Figure 4.18. 128 X 32 8R8W with 2:1 column multiplexer SRAM layout.

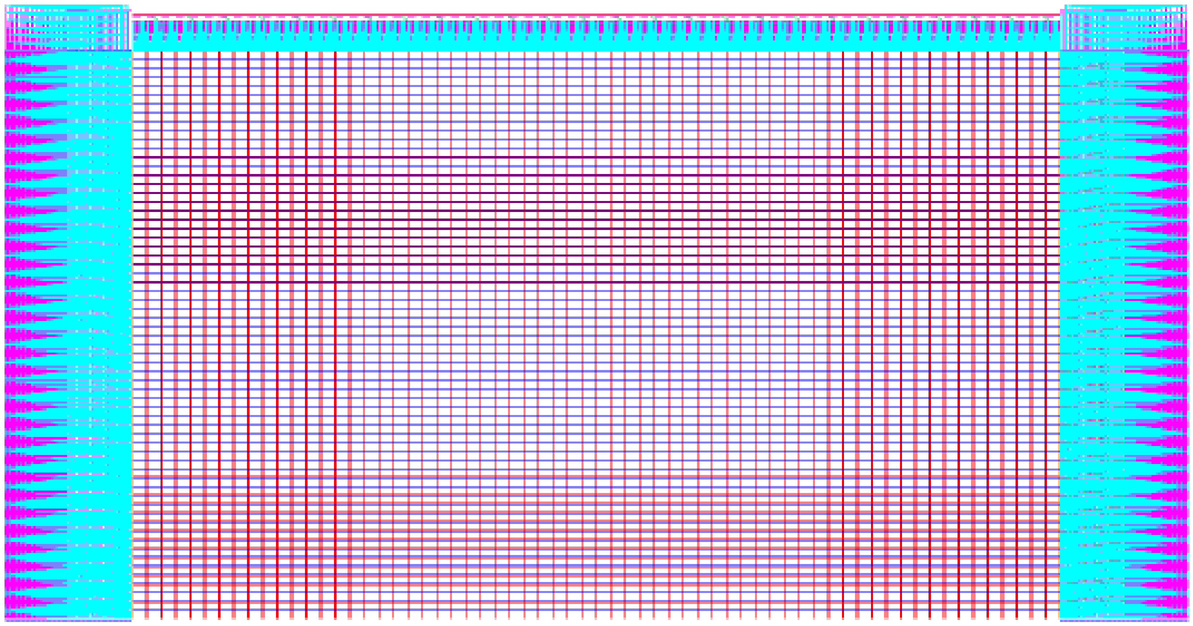


Figure 4.19. 64 X 64 16R8W SRAM layout (not all layers are visible).

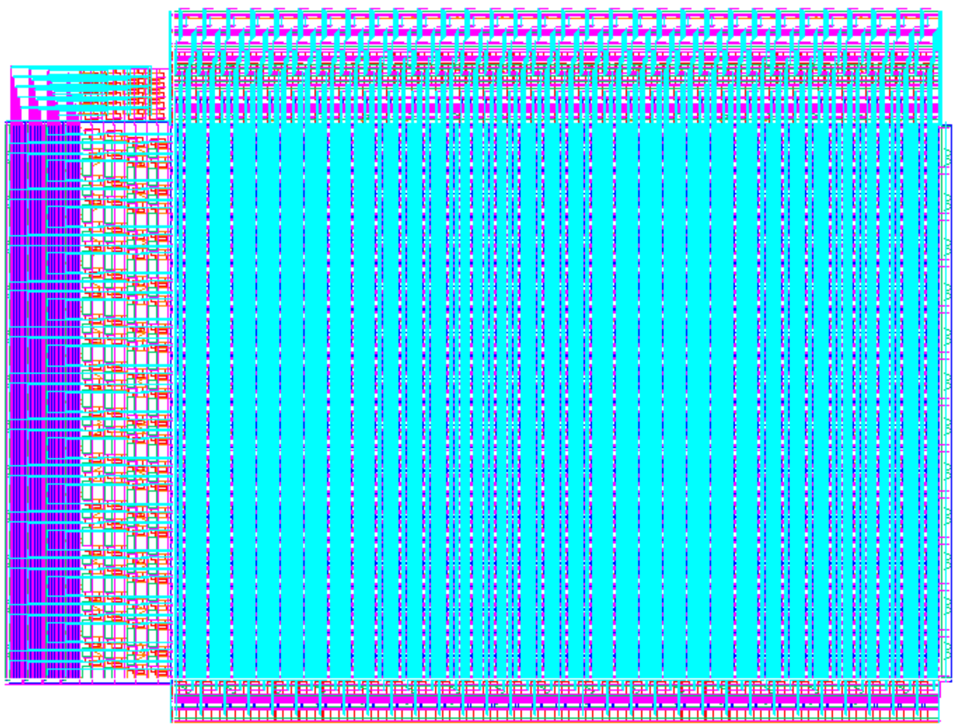


Figure 4.20. 24 X 32 4R4W CAM layout.

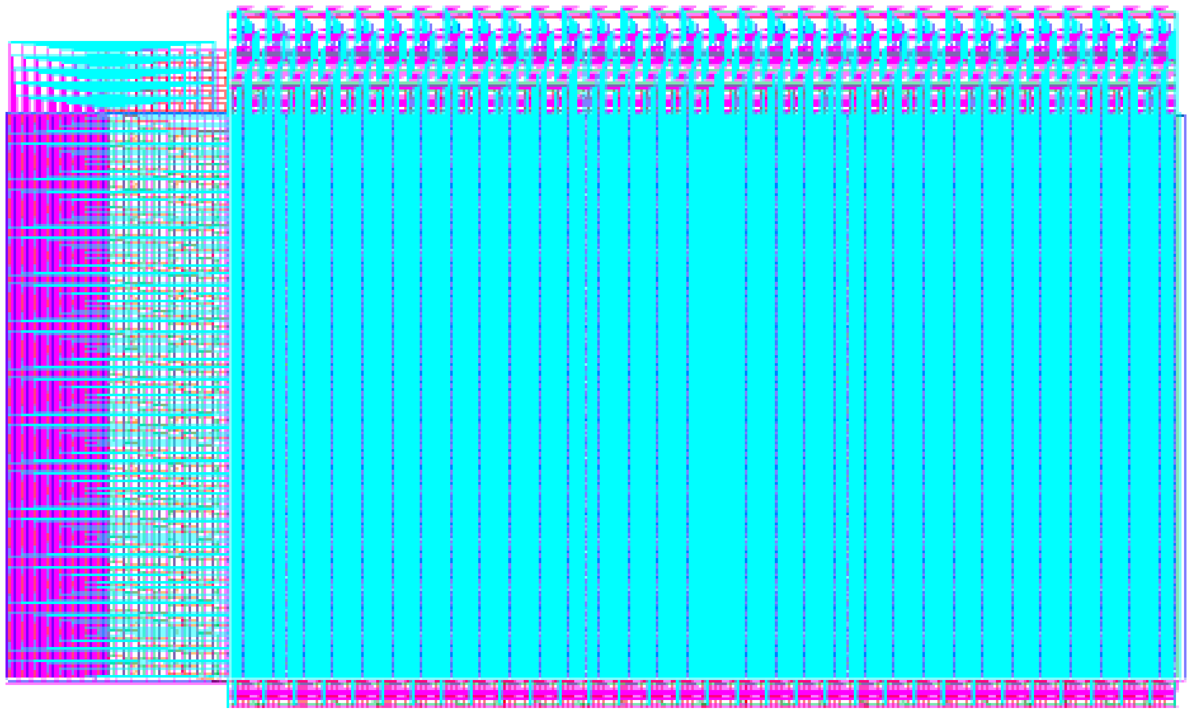


Figure 4.21. 24 X 32 8R8W CAM layout.

Chapter 5

Verification and Validation

The stability of the bitcell and noise margins are critical issues in sub-micron technology nodes. SRAM failures might occur due to alpha particle strikes, due to read noise or due to variations in process parameters. Thus, analysis of the tolerance of SRAM produced by memory compiler is required. It is also insightful to analyze the error in the estimates generated by the data generation tool, by comparing against simulations of the extracted full memory layout. It is also insightful to compare the tool with the cache modeling tool, "CACTI" [4,10].

5.1 Comparison of Results

5.1.1 Full Memory Layout

Area, access time and energy consumption estimates generated by the data generation tool are verified against actual results from the whole memory layout generated by the automated layout generation tool. Tables 5.1 and 5.2 show the accuracies of area, access time and power consumption for different configurations of SRAM and CAM, respectively. Due to both positive and negative errors, root-means-square (RMS) error is calculated. As shown in the Table 5.1 the RMS error is 5.97% in area, 13.62% in read access time, 13.52% in write time and 10.85% in power. For CAMs, RMS error is 9.78% in area, 15.28% in read access time, 13.05% in write time and 16.47% in power.

Table 5.1. Error analysis for SRAM: Comparison of estimates from data generation tool with simulation results from full memory layout.

Configuration	% Error (Area)	% Error (Read Time)	% Error (Write Time)	% Error (Power)
32X32 6R3W	-2.06	-10.48	-12.71	-7.95
32X32 8R4W	-3.74	-11.83	-11.34	2.05
32X32 10R5W	-5.05	-11.05	-10.52	1.04
32X32 12R6W	-4.98	-12.55	-18.73	15.88
32X32 14R7W	-5.99	-9.38	-12.54	21.78
32X32 4R4W	-5.90	-23.46	21.26	-6.92
32X32 5R5W	-5.14	-15.94	-8.02	1.73
32X32 8R8W	-10.94	-7.87	-6.13	9.82
RMS Error	**5.97**	**13.62**	**13.52**	**10.85**

Table 5.2. Error analysis for CAM: Comparison of estimates from data generation tool with simulation results from full memory layout.

Configuration	% Error (Area)	% Error (Read Time)	% Error (Write Time)	% Error (Power)
32X16 1R1W	-17.25	-28.22	-16.12	-15.80
32X16 2R2W	2.09	-7.04	-15.90	-25.64
32X16 3R3W	-8.80	-12.21	-10.72	-26.10
32X16 4R4W	-6.51	-4.24	-14.81	-17.42
32X16 5R5W	-7.73	-7.02	-11.88	-7.97
32X16 6R6W	-10.04	-18.86	-12.01	-4.04
32X16 7R7W	-7.34	-13.32	-12.28	-13.91
32X16 8R8W	-11.40	-16.49	-8.92	-2.55
RMS Error	**9.78**	**15.28**	**13.05**	**16.47**

5.1.2 CACTI

CACTI [10] is an analytical tool for modeling the area, access time, dynamic power and leakage power of caches and other memories. CACTI uses the ITRS roadmap to model device parameters. ITRS uses MASTAR [18] to estimate device characteristics like I_{on}, I_{off}, physical gate length, electron and hole mobility, threshold voltage, oxide thickness, etc. We scaled CACTI to match with FreePDK45 [2] technology parameters. The results from the data generation tool are also compared with the results from scaled CACTI 5.3. For generating results using scaled CACTI, a pure RAM interface with one bank structure and no single-ended read is assumed. The technology node is kept 45 nm (FreePdk45) and all transistors are of the type ITRS-LSTP [11]. The supply voltage (V_{dd}) is kept 1.1 V in both cases. The difference with respect to scaled CACTI 5.3 is shown in Table 5.3. The negative difference signifies that the result produced by FabMem is lower than the CACTI. For comparison the number of ports of 128X32 SRAM is varied from 3 to 24. The area, access time and read energy curves for CACTI and FabMem is shown in the Figure 5.1 and 5.2. As shown in figure, the difference in area is maximum while the difference between the read time is minimum.

Table 5.3. Comparison with CACTI.

Configuration	**% Difference (Area)**	**% Difference (Read Time)**	**% Difference (Energy)**
128X32 2R1W	-51.87	-23.64	138.76
128X32 4R2W	-74.61	-11.61	120.90
128X32 6R3W	-78.80	-13.82	91.46
128X32 8R4W	-80.89	-17.40	56.30
128X32 10R5W	-82.15	-20.67	34.85
128X32 12R6W	-82.98	-25.22	24.89
128X32 14R7W	-83.33	-23.77	21.87
128X32 16R8W	-83.64	-29.77	12.84
Average %	-77.28	-20.74	62.73

The CACTI targets the large cache memories and so it divides the given memory into banks, subbanks, mats and subarrays to produce hierarchical design, which is necessary for high performance cache memories. FabMem targets small size multiported memory compared to the cache memory and so it does not implement such a hierarchical design.

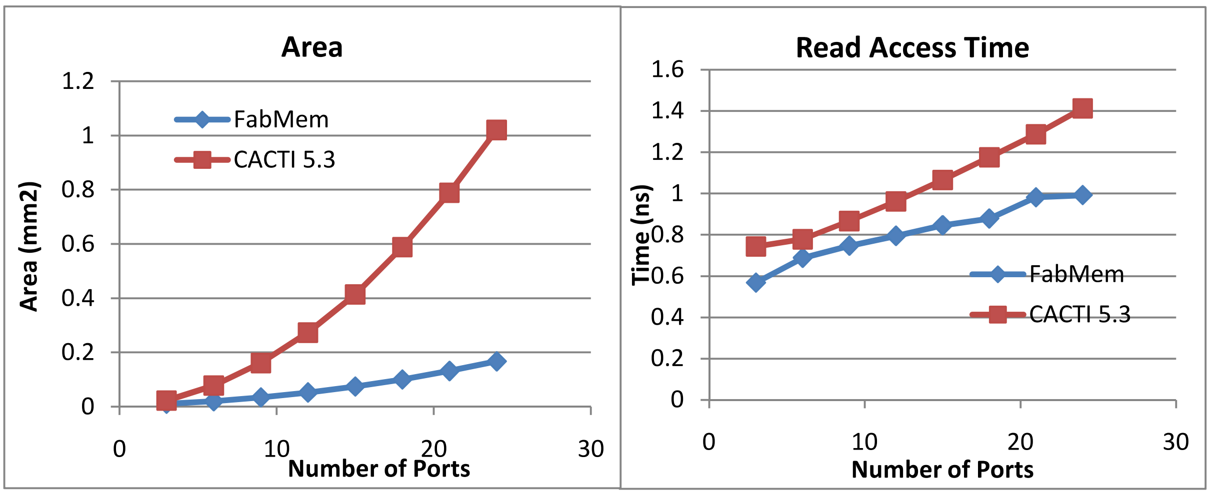


Figure 5.1 Area and Access Time Comparison of CACTI and FabMem

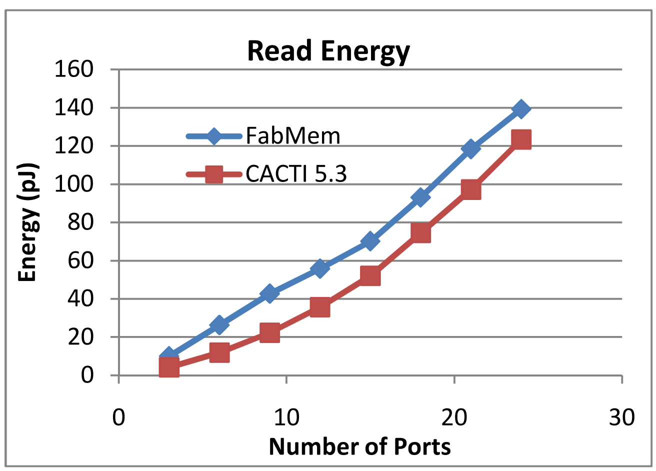


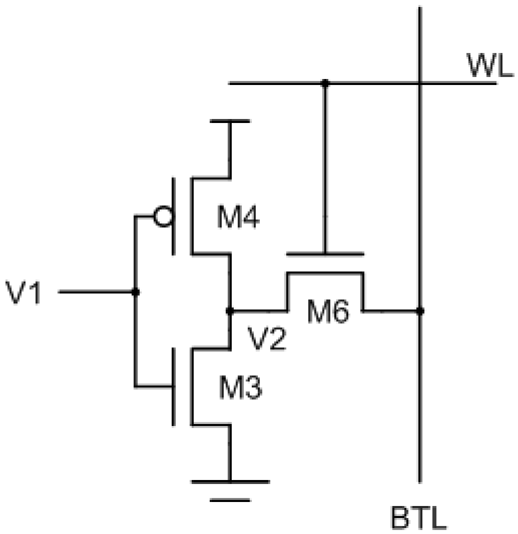
Figure 5.2 Read Energy Comparison of CACTI and FabMem

5.2 SRAM Noise Margin

Noise margin reflects the robustness of the design. It is resilience of the design in the event of a disturbance. If the disturbance is greater than the noise margin of the bitcell, then it will flip the stored value. To ensure that flipping does not occur, it is desired to have a sufficiently larger noise margin. In the case of SRAM, three types of noise margins need to be evaluated: hold noise margin, read noise margin and write noise margin.

5.2.1 Hold Noise Margin

When the bitcell is in IDLE condition, wordline WL = 0 and bitline BTL = 1, the noise margin is evaluated using the procedure mentioned in [12]. First, the feedback loop in the cross-coupled inverters is broken, as show in Figure 5.2. WL is kept LOW and BTL is kept HIGH. The voltage at V1 is swept from 0 to V_{dd} and the voltage at V2 is measured and the corresponding VTC (Voltage Transfer Curve) is plotted with Real(y) vs. Real(x). The same VTC is plotted again, but with Real(x) vs. Real(y) orientation. These two graphs are overlapped on each other to obtain the "Butterfly" curve shown in Figure 5.1. The Static Noise Margin (SNM) is determined by measuring the length of a side of the largest possible square in both openings of the butterfly curve. If the butterfly curve is asymmetric, the smaller square is considered for measurement. In the butterfly curve of Figure 5.5, the hold SNM is 376 mV.



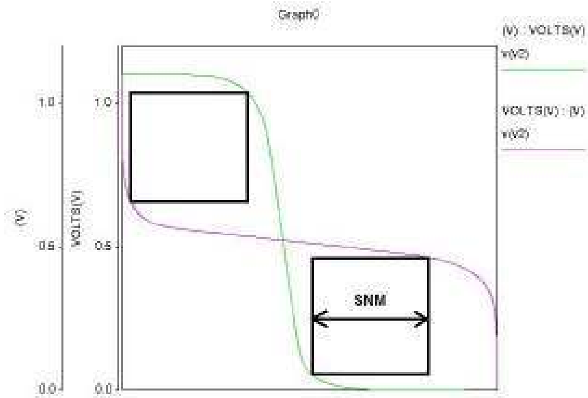


Figure 5.3. Butterfly curve.

Figure 5.4. Schematic for the VTC plot.

5.2.2 Read Noise Margin

Read noise margin is measured using the same procedure as for the hold noise margin, but the word line is held HIGH. All other conditions are the same as for the hold noise margin. As shown in Figure 5.6 and 5.7, the RSNM (Read Static Noise Margin) for a 4R4W bitcell is 153 mV, while the RSNM for a 5R5W bitcell is 378 mV. Due to decoupling between the stored charge and the bitline through an extra transistor, the RSNM for 5R5W (and even higher ports bitcell) is better than the RSNM of 4R4W.

5.2.3 Write Noise Margin

For the write noise margin measurement, two VTCs are plotted. For the first VTC, the bitline BTL is kept HIGH and for the second VTC, BTL is kept LOW [13]. The butterfly curve is then obtained to measure the static noise margin. As shown in Figure 5.8 the write SNM is 406 mV.

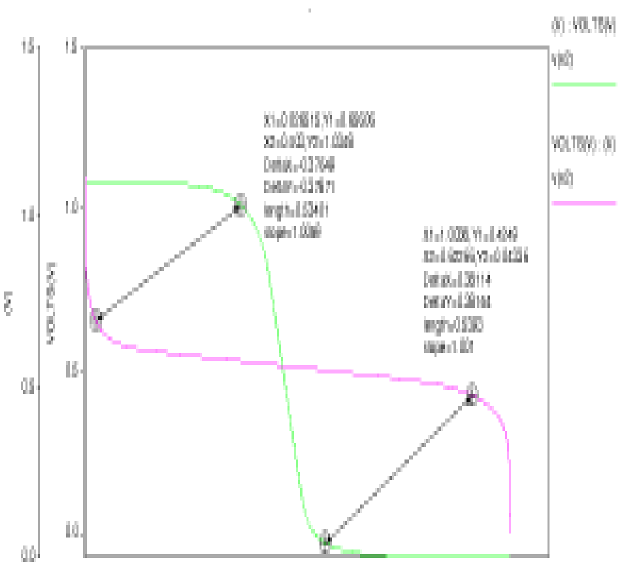


Figure 5.5. Hold Static Noise Margin.

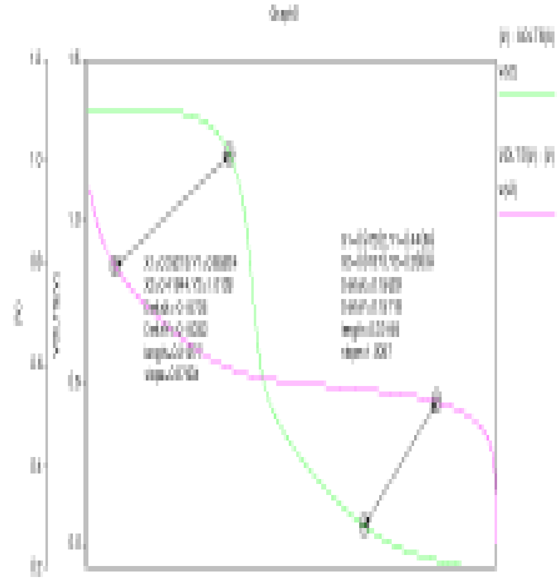


Figure 5.6. Read Static Noise Margin (4R4W SRAM).

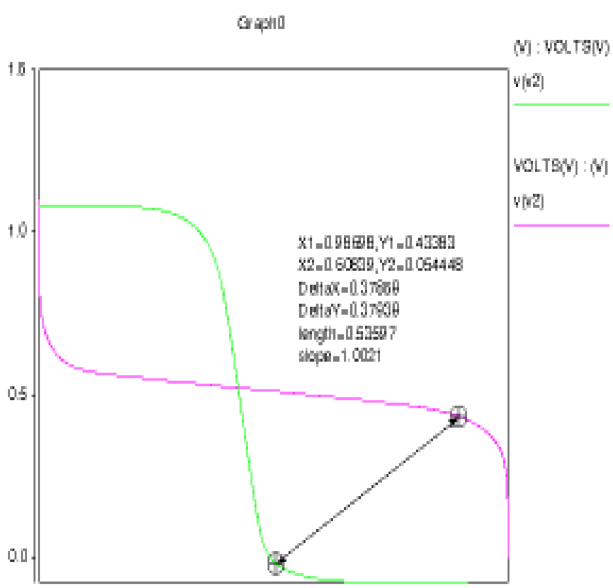


Figure 5.7. Read Static Noise Margin (5R5W SRAM).

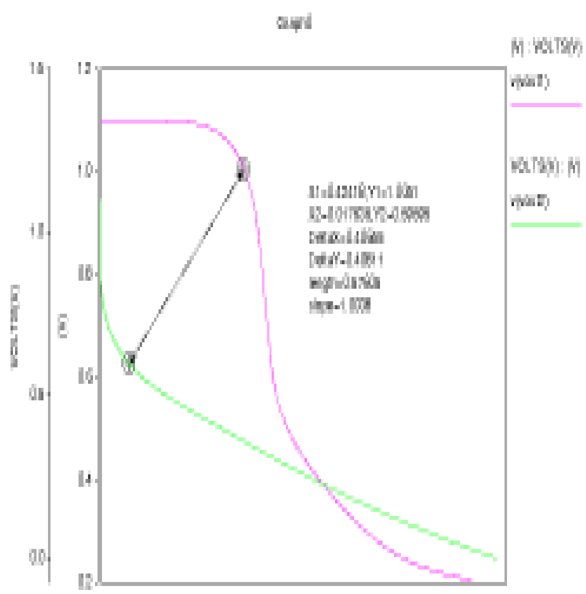


Figure 5.8. Write Static Noise Margin.

5.3 CAM Noise Margin

The write port into the CAM is same as that of the SRAM. So the write noise margin will be the same as that of the SRAM. The hold and read noise margins are measured the same way as we did for the SRAM. For the first VTC of read noise margin, the select line is kept HIGH and for the second VTC it is kept LOW. Figures 5.9 and 5.10 show the hold noise margin and read noise margin, respectively. For the CAM, the hold noise margin is 367 mV and the read noise margin is 379 mV. In the CAM, during the read operation, the only noise source is the gate leakage current. The gate leakage is very small in amplitude compared to the drain leakage and so the read noise margin is quite close to the hold noise margin.

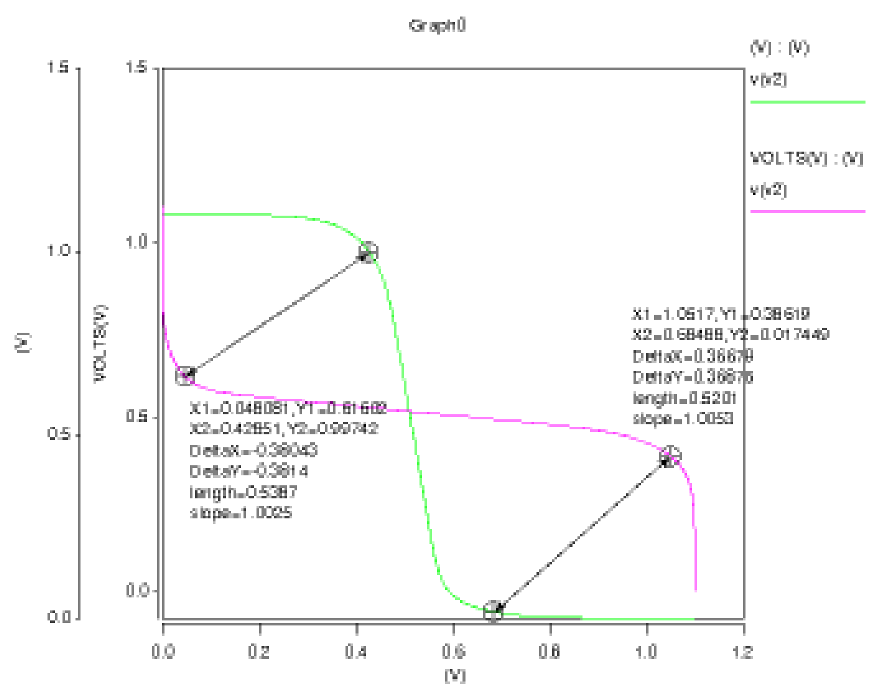


Figure 5.9. Hold noise margin in CAM.

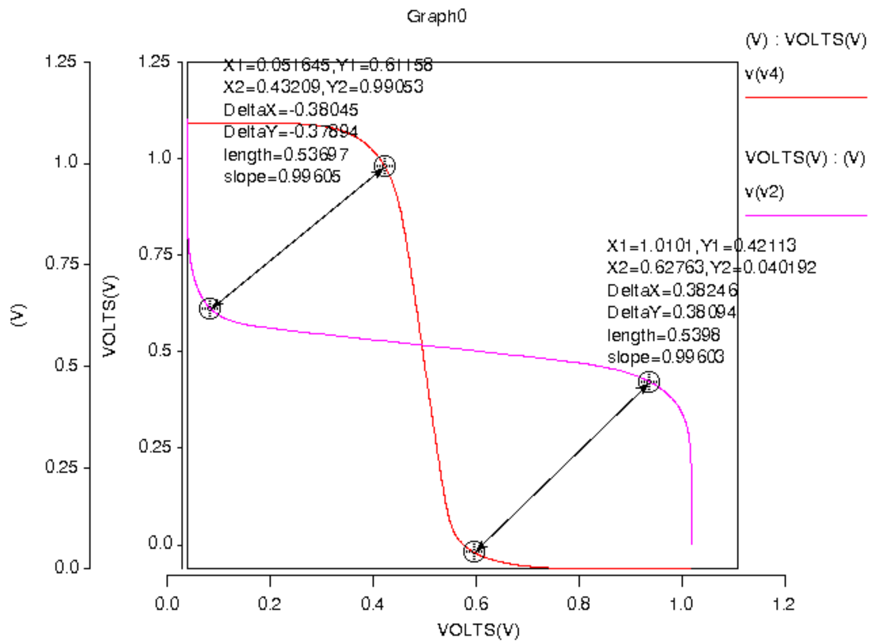


Figure 5.10. Read noise margin in CAM.

5.4 Bitcell Failure Analysis

Bitcell failure analysis is presented in [14] for both one-sided and two-sided disturbances. In a one sided disturbance, noise applied on either end of the cross-coupled inverters, while in a two-sided disturbance, noise is applied at both ends of the bitcell. SRAM failures due to read noise and alpha particle strikes [15] usually occur due to one-sided disturbances, but for the present analysis, the more conservative two-sided disturbance is considered.

Consider Figure 2.1 (basic 6T SRAM). Suppose a "1" is stored at Q and "0" is stored at Qbar. In [14], Q is represented as "R" (right side) and Qbar is represented as "L" (left side). For failure analysis, it is assumed that positive noise, $NOISE_L$, is applied at the "L" end and negative noise, $NOISE_R$, is applied at the "R" end. The condition for instability is shown below. The detailed explanation of this equation is given in [14].

$$(a \times V_{L(gain)}) + b > f(V_{L(gain)}) - NOISE_R$$

For any $NOISE_L$, as long as $NOISE_R \le f(V_{L(gain)}) - ((a \times V_{L(gain)}) + b)$, the bitcell will be stable. The coefficients a and b are calculated using values obtained from VTCs and equations presented in [14]. Using these values, different $NOISE_L$ and $NOISE_R$ are obtained and plotted in Figure 5.11. Figure 5.11 shows the required positive $NOISE_L$ and negative $NOISE_R$, to move the cell from a stable state to a metastable state. Beyond these values cell will fail and flip the stored value.

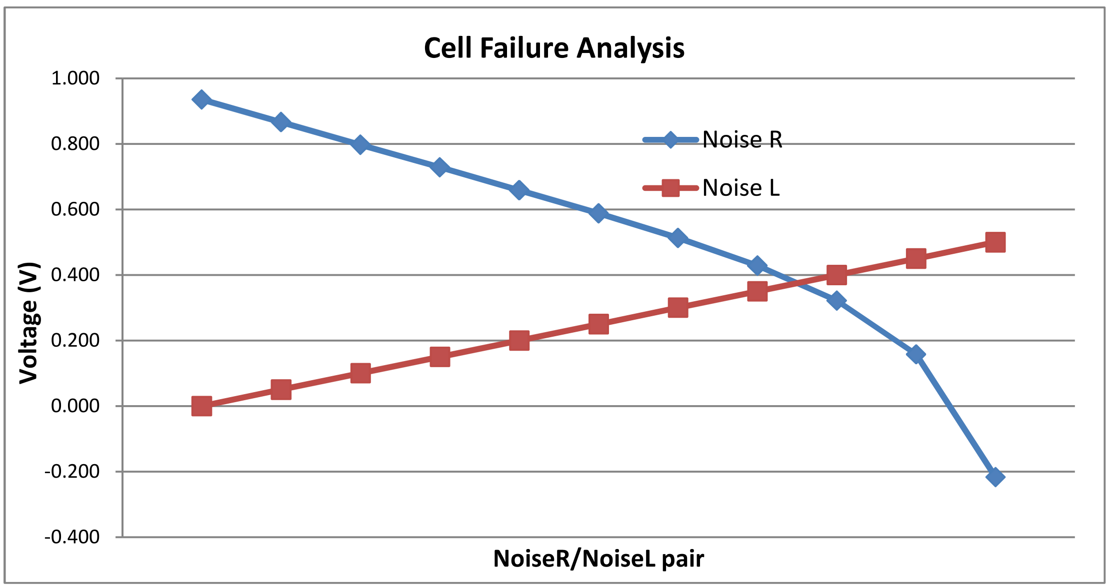


Figure 5.11. Bitcell failure analysis.

5.5 Variation in RSNM

Precharging the bitlines consumes a large fraction of the total energy consumption. To reduce the total energy consumption, the bitlines can be precharged to a different voltage than V_{dd}. During the read operation, the bitline voltage acts as a noise source to the bitcell voltage. So, as the precharge voltage is changed, the RSNM will be changed. Figure 5.12 shows that, as the precharge voltage is decreased from V_{dd} (1.1 V) to 0.5 V, the RSNM increases linearly, but after $V_{dd}/2$ (0.55 V), the RSNM decreases.

It is also possible to change the substrate voltage other than connecting it to the GND, in the case of NMOS, to reduce the leakage power. By dynamically changing the body bias in the access transistor, the threshold voltage can be changed dynamically and leakage current can be minimized, although changing the body bias might impact the operating speed of the transistor. Moreover, changing the body bias, it might impact the RSNM. During the read operation and even in the IDLE mode of the bitcell, the leakage current through the substrate acts as a noise source to the bitcell. Due to multiple ports, there are many access transistors. As the body bias of access transistor is changed, the substrate leakage current is also changed and it can impact the RSNM. Figure 5.12 also shows that RSNM decreases linearly as body bias is increased from 0 to 0.6 V.

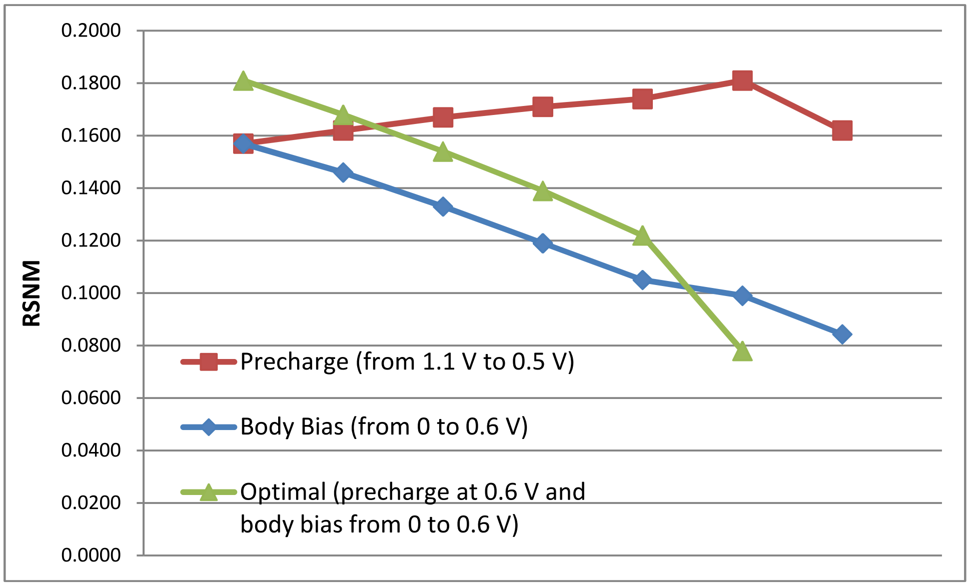


Figure 5.12. Impact on RSNM by varying the precharge voltage and body bias voltage.

The graph for optimal RSNM is generated by keeping the precharge voltage at the 0.6 V and varying the body bias from 0.0 to 0.5 V. Figure 5.12 clearly shows that for a precharge voltage of 0.6 V and body bias of 0.2 V the RSNM is 154 mV, which is the same RSNM as in the normal operating case. Thus, using these values for precharge and body bias, a reasonable balance between speed and leakage can be obtained. The actual impact of these precharge and body bias values on read access time and power consumption using HSPICE simulation will be evaluated in future work.

Chapter 6

Results and Accuracy Analysis

Before making any design decision, it is important to analyze its effect on area, power and clock frequency, It is also important to know how the area, access time and energy consumption vary when the structure sizes are varied, number of ports are varied and implementation is varied. The following sections give a detailed analysis on the variation in area, access time and energy consumption by varying the different parameters of the memory structure.

6.1 Area

Area is an important parameter and needs to be analyzed in multiported memory structures. Multiported memory structures feature large bitcells with many transistors and have significant impact on overall chip area. In addition, as the number of ports increases, the number of peripheral circuits also increases linearly. Figure 6.1 shows how area increases with the number of ports. Figure 6.1 shows the area for 128 X 32 and 256 X 32 SRAMs by varying the number of ports from 3 to 24 (i.e., from 2R1W to 16R8W) and the area for 64 X 16 and 32 X 16 CAMs by varying the number ports from 2 to 16 (i.e., from 1R1W to 8R8W). Table 6.1 shows the effect on area by varying the degree of column multiplexing from none to 4:1. As we fold the memory and add column multiplexer, the number of row decoders and word line drivers are halved, but the numbers of other peripheral circuits (precharge circuit, write drivers, column multiplexers) and interconnections increase.

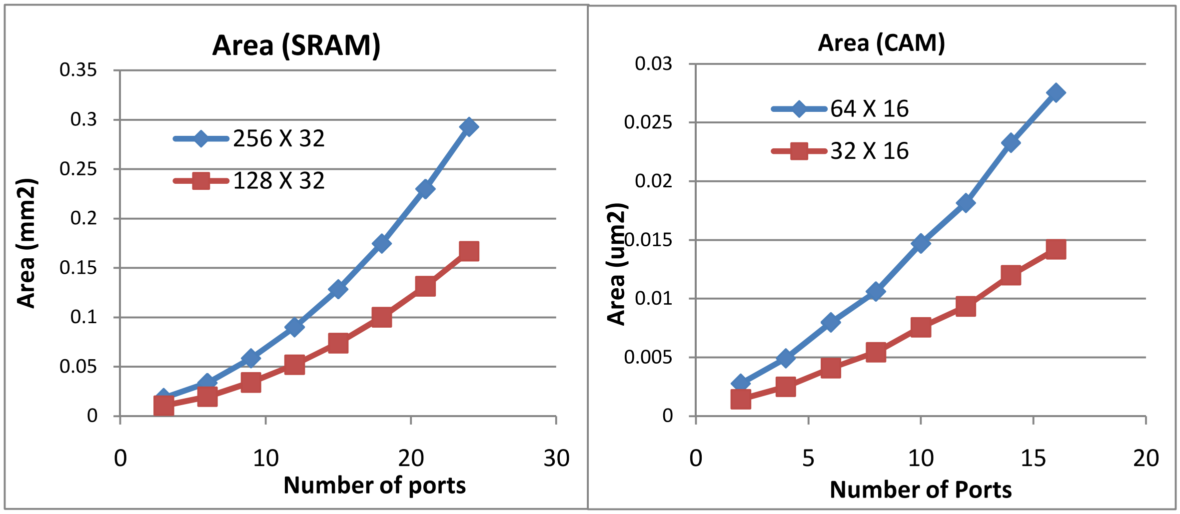


Figure 6.1. Area vs. number of ports in SRAM and CAM.

Table 6.1 Variation in area by folding the SRAM

64 X 32 4R4W SRAM	Area (mm^2)
No column multiplexer	0.01465
2:1 column multiplexer	0.01381
4:1 column multiplexer	0.01758

From figure 6.1, the area increases exponentially with an increase in the number of ports and an increase in size. The increase in area in each type of memory is approximately 70% for SRAM as we move from 128 X 32 to 256 X 32 and 95% for CAM as we move from 32 X 16 to 64 X 16.

6.2 Read Access Time

Figures 6.2 and 6.3 show the read access time vs. the number of ports and number of entries for both SRAM and CAM, respectively. In the previous section it was shown that the area increases exponentially with the increase in the number of ports and so it is expected that the read access time will increase exponentially or linearly with a high slope. But due to the placement of the extra pull-down transistor in the bitcell and other optimizations, rise in access time is milder. The figure also shows a linear increase in read access time with more entries.

Table 6.2 shows the effect of folding the memory on the read access time. As we fold the memory and add the column multiplexer, the length of the bitline becomes half, but wordline length doubles. When the wordline delay starts dominating the bitline delay, the gain in access time becomes negative.

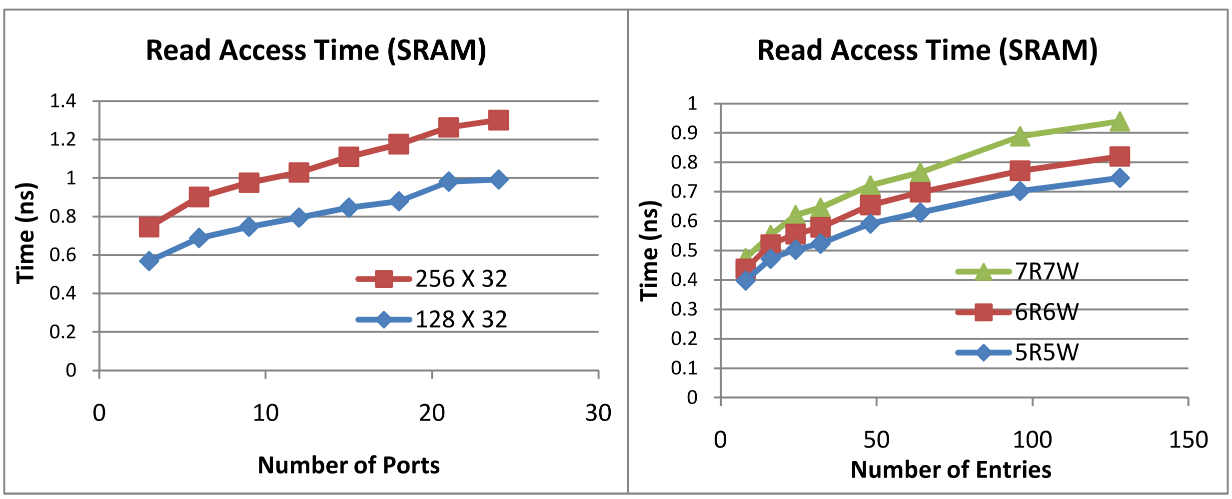


Figure 6.2. Read access time vs. number of ports and entries in SRAM.

Table 6.2. Variation in read access time by folding the SRAM.

64 X32 4R4W SRAM	Read Access Time (ns)
No column multiplexer	0.873500
2:1 column multiplexer	0.615300
4:1 column multiplexer	0.666700

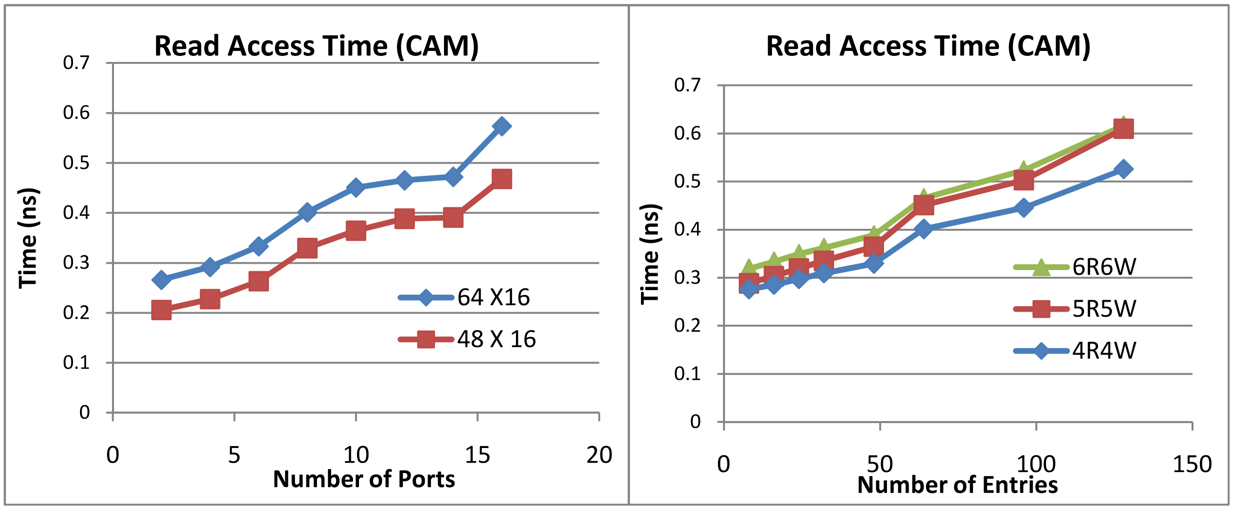


Figure 6.3. Read access time vs. number of ports and entries in CAM.

6.3 Read Energy

Figures 6.4 and 6.5 show the total read energy vs. number of ports for SRAM and CAM, respectively. If the Figures 6.2 and 6.3 are compared with the Figures 6.4 and 6.5, respectively, then it is clear that the read energy consumption increases in a same way as the read access time. For each type of SRAM bitcell, as we move from 128X32 to 256X32, on average read energy increases by 41%. For the CAM, as the size increases from 32X16 to 64X16, on average, the read energy increases by 100%. The Table 6.3 shows the variation in read energy by varying the degree of column multiplexing. As the column multiplexer is added, twice as many bits are read and twice as many bitlines are precharged. Thus, the read energy increases when the memory is folded to add the column multiplexer.

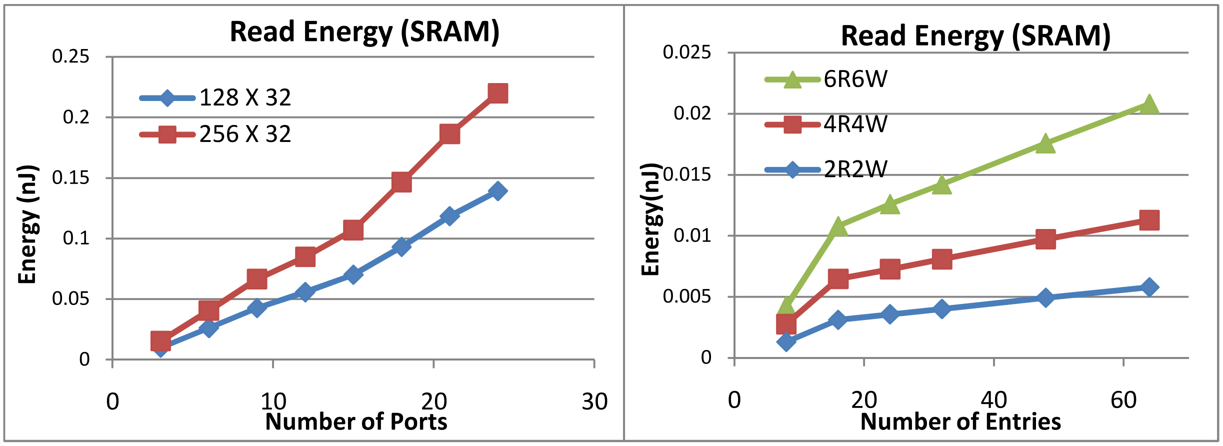


Figure 6.4. Total read energy in SRAM.

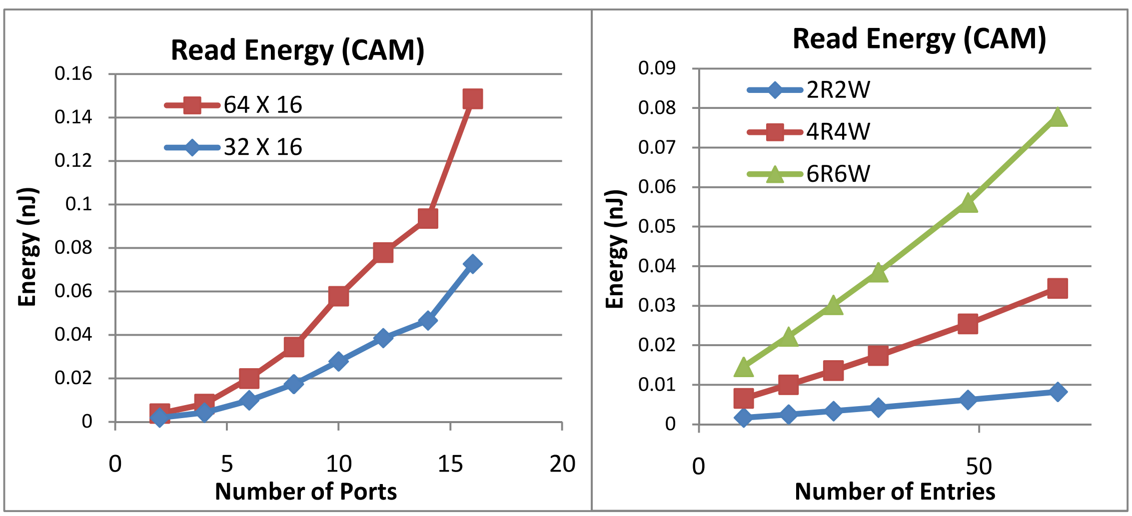


Figure 6.5. Total read energy in CAM.

Table 6.3 Variation in Read Energy by folding the SRAM

64 X 32 4R4W SRAM	Read Energy (pJ)
No column multiplexer	9.281132
2:1 column multiplexer	11.28335
4:1 column multiplexer	19.36744

6.4 Write Time

The “Write” operation is quite different from the “Read” operation. “Read” is a sequential operation. For “Read”, first the input address is decoded, then wordline is driven; when wordline reaches the threshold voltage of the access transistor, the read bitline starts reading by discharging, in case of reading “0”. All these steps are a chain of operations. In the case of “Write”, the address decoding and the wordline driving are done in parallel with the write bitline driving. When both the wordline and bitline reach their desired voltages, the bit value gets written into the bitcell. This yields less write latency as compared to the read latency. Figures 6.6 and 6.7 show write time for SRAM and CAM, respectively. The write time increases by approximately 5% from 32 X 32 to 64 X 32 for SRAM. For CAM, it increases by 18% from 32 X 16 to 64 X 16. Table 6.4 shows the variation in write time as the degree of column multiplexing is varied.

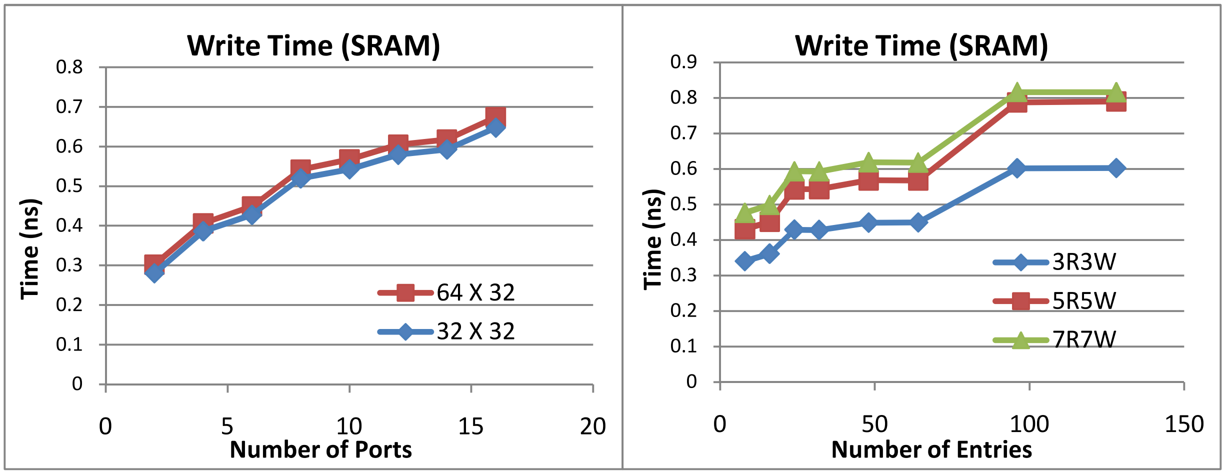


Figure 6.6. Write latency in SRAM.

Table 6.4. Variation in write time by folding the SRAM.

64 X 32 4R4W SRAM	Write Time (ns)
No column multiplexer	0.591700
2:1 column multiplexer	0.541900
4:1 column multiplexer	0.724300

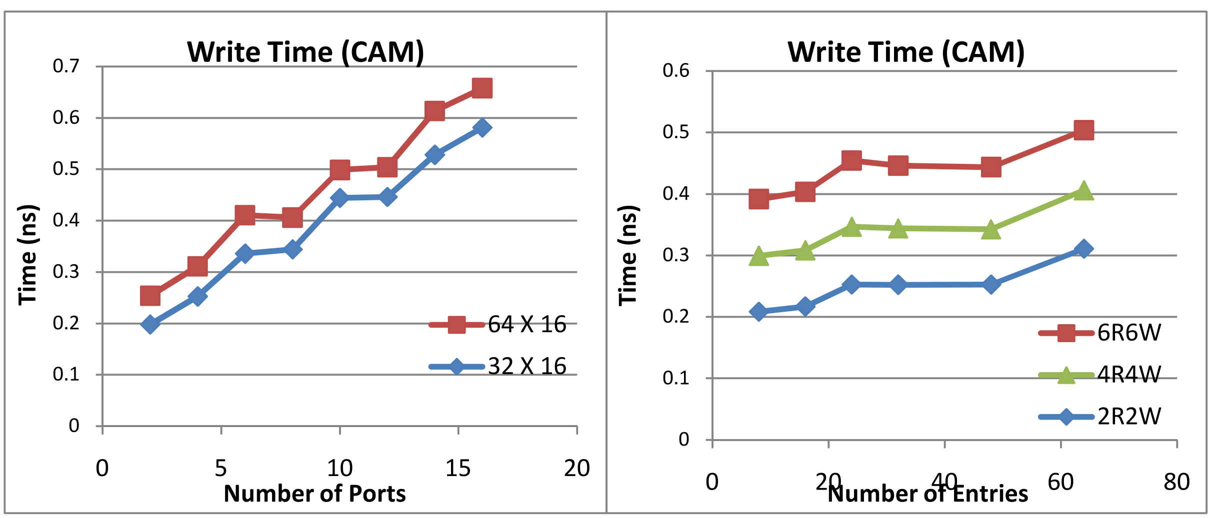


Figure 6.7. Write latency in CAM.

6.5 Write Energy

Figures 6.8 and 6.9 show the write energy in SRAM and CAM, respectively. It is observed that as the size of the SRAM is increased from 128X32 to 256X32, on an average, write energy increases by 33%, and as the size of the CAM is increased from 8X16 to 16X16, write energy increases by 14%. Table 6.5 shows write energy variation with respect to the degree of column multiplexing. As the column multiplexer is added, the number of write bitlines doubles, which increases the write energy.

Table 6.5. Variation in write energy by folding the SRAM.

64 X 32 4R4W SRAM	Write Energy (nJ)
No column multiplexer	4.745384
2:1 column multiplexer	6.404908
4:1 column multiplexer	13.637446

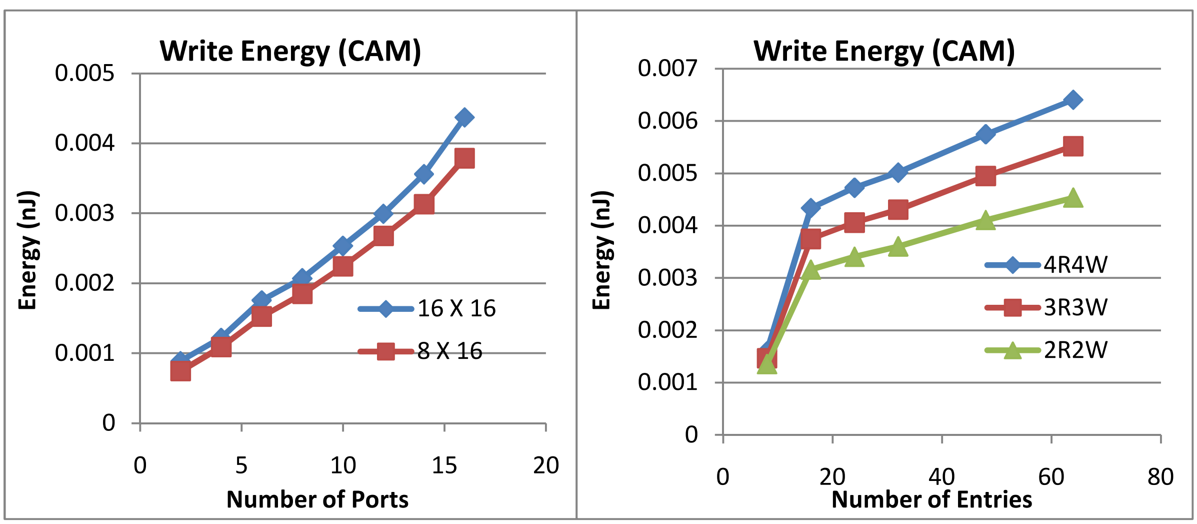


Figure 6.8. Write energy in SRAM.

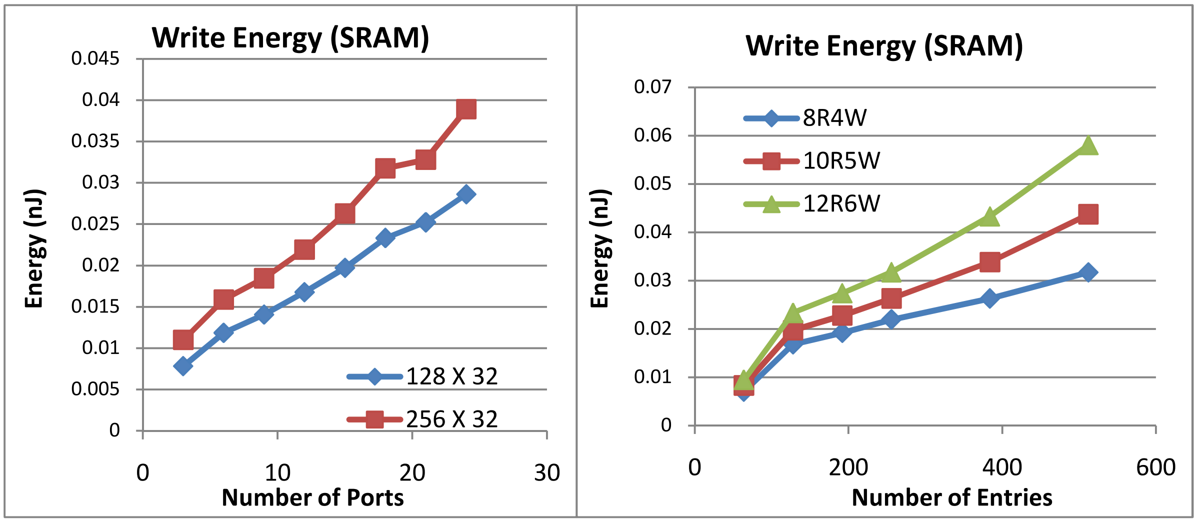


Figure 6.9. Write energy in CAM.

6.6 Clock Tree Power and Power Breakdown

The clock is a global signal like V_{dd} and GND, routed to many areas of the memory. In the current implementation of the multiported SRAM, the clock is routed to the decoder of each port, the precharge and the write driver circuits. The decoder is implemented using dynamic logic, so the clock is present in both the pull-up and pull-down paths. If the memory is 8R4W type, then there will be a total of 24 clock sinks for the decoder. This clock should be driven using a clock tree, which is a special circuit to drive the clock at each sink and also to handle clock skew. In the current implementation, a clock tree is not implemented and the clock is directly driven from the input. Thus clock tree power is not explicitly modeled. For the clock sinks in the row decoder, the length of that clock segment will be the same as the length of an address line and so the capacitance will be approximately the same. The clock segment for the precharge and write driver circuits will be as long as the wordline. So, the approximate clock power is calculated for each sink using the address line driver and wordline driver as a clock sink driver circuit.

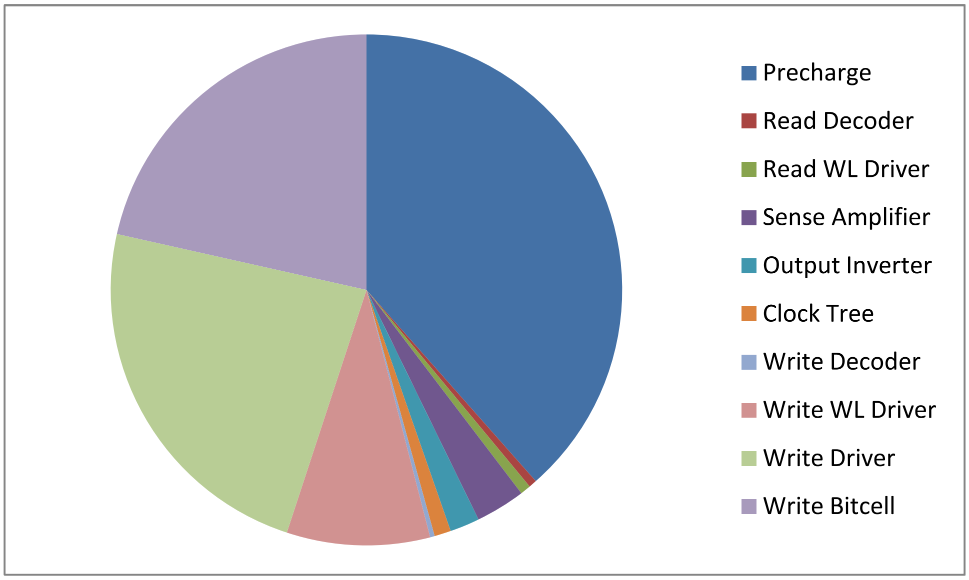


Figure 6.10. Pie chart showing the power dissipation in each part of the 32X32 8R4W SRAM.

The pie chart in Figure 6.10 shows the power consumption of each part of the SRAM as a percentage of total power consumption, assuming all read and write ports are active. As shown in the figure, the precharge consumes 38%, write driver consumes 23% and the bitcell during the "write" consumes 21% of the total power.

Chapter 7

SRAM Optimization

To fit the design in a given area, power or clock period budget, it may be required to optimize the design using microarchitectural techniques. This chapter proposes some microarchitectural optimizations for SRAM for better performance.

7.1 Pipelined SRAM

As the superscalar processor's issue width increases, the number of ports and the size of the physical register file (PRF) also increase, resulting in a large access time that may not fit into the cycle time of the processor. For example, the access time of a 256X32 16R8W PRF is 1.3 ns, which directly impacts the cycle time of the processor. Pipelining can divide up the propagation delay, but the pipelining of SRAM is a little different than pipelining of combinational logic. The major propagation delay present in the SRAM is due to the wire capacitance, gate capacitance and drain capacitance of the access transistor. Wordline and bitline signals propagate on wires only, while in combinational logic signals propagate through transistors and in between any transistors the signal will be either HIGH or LOW. So, combinational logic can be easily pipelined. To pipeline the SRAM, the block diagram shown in Figure 7.1 is followed. As shown in the figure, the data width (i.e., wordline) is divided into the pipeline depth. The remaining SRAM structure is folded using column multiplexing until the minimum access time is obtained. For a pipeline depth of 4, the 256X32 SRAM will be divided into four 256X8 SRAM modules. The wordline signals from the first SRAM module goes into a pipeline register and the output of that register is applied to the second SRAM module and so on. In this scheme, each cycle a quarter of the total data is obtained, which is latched into pipeline register. The multiplexers in the bypass network are controlled by comparing source register tags with bypassed tags and latched bypass tags of each port. Each multiplexer is implemented as a purely parallel multiplexer instead of using priority logic, because at any time a source register tag will match at most only one of the bypass tags. Address decoding takes place in only the first module of the SRAM. The Remaining SRAM modules are SRAMs without address decoders. After pipelining, the read access time will be 0.8ns. The proposed pipelined structure was implemented in RTL. Due to pipelining, the IPC also reduces. The above RTL code is integrated FabScalar toolset [1] and an IPC reduction was for the "parser" benchmark of 0.85 IPC to 0.61 IPC.

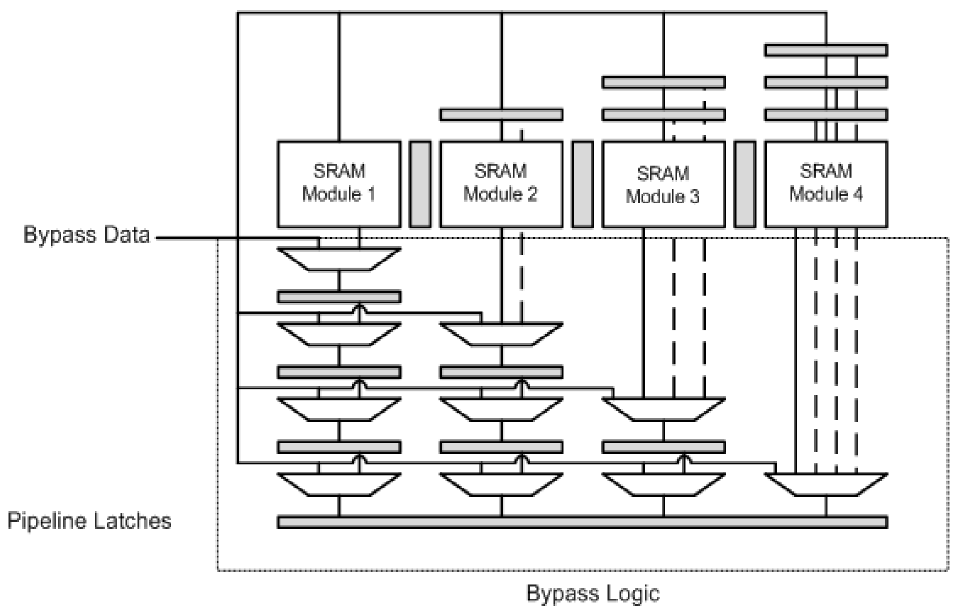


Figure 7.1. Pipelined SRAM.

7.2 Constraining Simultaneous Access

In Chapter 4, it was shown that the dominant factor in read access time for a multiported SRAM is simultaneous access to the same address. From Table 4.2, it is observed that if ports are constrained in such a way that two different ports will not access the same address in the SRAM, then the read access time will be much lower than the worst case. For this scheme, there will be overhead due to comparison logic among all the ports. Such comparison logic for four ports is shown in the Figure 7.2. If there is a match in two source operands then the comparator produces zero. Now, if the "valid" bit of any port is zero, then its address must be diverted. The logic shown in Figure 7.2 is not necessary to implement in the RR (register read) stage of the pipeline, but it can be implemented after the instructions are selected in the IS (Issue) stage. To divert the address for a particular port, one dummy row is allocated in the SRAM. Ports, whose addresses must be changed, are forced to access that dummy row and the values read by these ports are ignored. Read-after-read bypass multiplexers are required after the SRAM to steer shared value to the diverted ports. These multiplexers are also controlled by the comparison logic in Figure 7.2. If this scheme is implemented for a 128X32 8R4W PRF, then the read access time reduces from 0.795 ns to 0.67 ns. If we apply pipelining to this scheme then the read access time will be 0.478 ns. This gain increases as the size or number of ports of the SRAM increases. For example, using the above two schemes, the read access time reduces from 1.3 ns to 0.677 ns for a 256X32 16R8W SRAM.

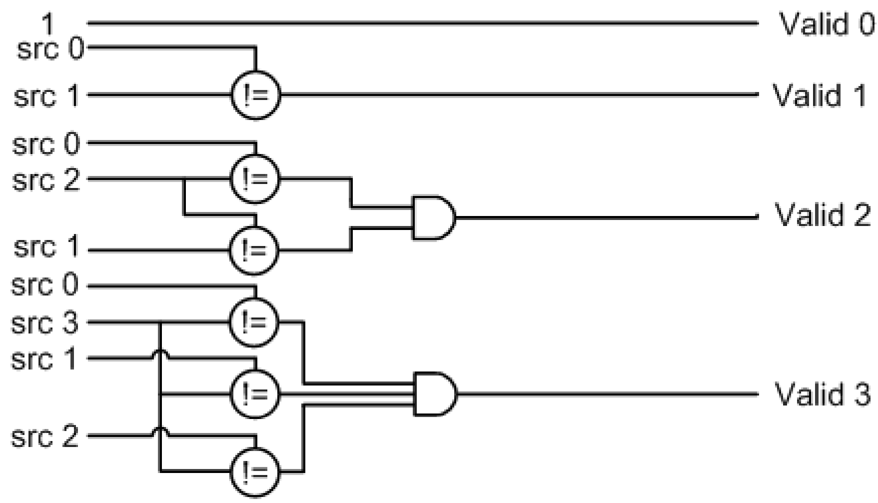


Figure 7.2. Comparison logic for constraining simultaneous accessed.

Chapter 8

Summary and Future Work

8.1 Summary

The multiported SRAM and CAM compiler was presented which can produce accurate results of area, access time and energy consumption for the required input configuration and can generate the layout of the whole memory. The SRAM compiler can generate a total of sixteen different port configurations (from 1R1W to 8R8W and from 2R1W to 16R8W). The CAM compiler is able to generate eight different port configurations (1R1W to 8R8W). Therefore, the tool can be used to explore the design space for arbitrary out-of-order superscalar processors. The detailed stability analysis of the bitcells shows that it produces robust memory structures. To optimize the performance of SRAM, pipelining and constrained access schemes are proposed.

8.2 Future Work

The SRAM and CAM compiler is tightly coupled with a particular 45 nm process technology. Using block assembly techniques with uniform physical data syntax, it might be possible to automatically adapt to different technology nodes. The current implementation is used for small to medium sized memory structures used in out-of-order superscalar processors and so it assumes one flat memory structure without any banking. Banking models can be implemented to generate cache memories or other larger memories. As said in Section 5.5, the optimal precharge voltage and body bias voltage will be evaluated to analyze their impact on read access time and energy consumption to generate optimal SRAM solution.

REFERENCES

[1] N. K. Chaudhary, S. V. Wadhavakar, T. A. Shah, S. S. Navada, H. Hashemi Najaf-abadi, E. Rotenberg, “FabScalar,”4th Workshop on Architectural Research Prototyping (WARP’09), in conjunction with ISCA-36, June 2009.

[2] J.E.Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W.R.Davis, P.D.Franzon, M. Bucher, S. Basavarjaiha, J. Oh, R. Jenkal, “FreePDK: An Open-Source Variation Aware Design Kit,” mse, IEEE International Conference on Microelectronic Systems Education (MSE’07), pp. 173-174, 2007

[3] P. Raghavan, A. Lambrechets, M. Jayapala, F. Kathoor, D. Verkest, “EMPIRE: Empirical power/area/timing models for register files”, Microprocessors and Microsystems (MICPRO), pp. 295-300, 2009.

[4] N. Muralimanohar, R. Balasubramonian, N. Jouppi, “CACTI 6.0: A Tool to Model Large Caches”, HP Laboratories, HPL-2009-85.

[5] Y. Liu, Z. Gao, X. He, “A Flexible Embedded SRAM Compiler”, Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA ’02).

[6] J. Rabaey, A. Chandrakasan, B. Nikolic, Digital Integrated Circuits, Prentice Hall, Second Edition, 2003.

[7] K. Pagiamatzis, A. Sheikholeslami, “Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey”, IEEE Journal of Solid State Circuits, Vol. 41, No. 3, pp. 712-727, March 2006.

[8] N. Delorme, M. Bellevile and J. Chilo, “Inductance and capacitance analytic formulas for VLSI interconnects”, IEEE Electronics Letters, Vol. 32, No. 11, May 1996.

[9] Z. Yan, X. Guo, H. Yang, R. Luo, H. Wang, “An Iterative Delay Model for Interconnects with Coupling Effects”, 5th International Conference on ASIC, pp. 311-314, 2003.

[10] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, N. P. Jouppi, “CACTI 5.1” HP Laboratories, HPL-2008-20, April 2008.

[11] Semiconductor Industries Association, “International Technology Roadmap for Semiconductors,” 2005. http://www.itrs.net

[12] R. E. Stallcup II, Z. Cross, W. James, P. Ngo, “Measuring Static Noise Margin of 65nm Node SRAMS using 7-Positioner SEM Nanopobing Technique”, Zyvex Instruments, Technical Paper 2007.

[13] J. Wang, S. Nalam, B. Calhoun, “Analyzing Static and Dynamic write margin for nanometer SRAMs,” International Symposium on Low Power Electronics and Design (ISLPED), pp. 129-134, August 2008.

[14] K. Agarwal, S. Nassif, “Statistical Analysis of SRAM Cell Stability”, The Design Automation Conference (DAC), pp. 57-62, 2006.

[15] T. May, M. Woods, “Alpha-Particle-Induced Soft Errors in Dynamic Memories,” IEEE Transactions on Electron Devices, ED-26, no. 1, pp. 2-9, January 1979.

[16] J. T. Kao, M. Miyazaki, “A 175-mV, Multiply-Accumulate Unit Using an Adaptive Supply Voltage and Body Bias Architecture,” IEEE Journal of Solid-State Circuits, Vol. 37, No. 11, pp. 1545-1554, November 2002.

[17] L. Chang, D. Fried, J. Hergenrother, J. Sleight, R. Dennard, R. Montoye, L. Sekaric, S. McNab, A. Topol, C. Adams, K. Guarini, W. Haensch, "Stable SRAM cell Design for the 32nm Node and Beyond," Symposium on VLSI Technology, Digest of Technical Papers, 2005.

[18] "Models for Assessment of CMOS Technologies and Roadmaps (MASTAR)," 2005. http://www.itrs.net/