DESIGN OF 32-BIT 32-WORD 10-READ/WRITE PORT REGISTER FILE

By

SRIKANTH GOPI

Bachelor of Engineering

University of Madras

Tamil Nadu, India

2002

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2006

DESIGN OF 32-BIT 32-WORD 10-READ/WRITE

PORT REGISTER FILE

Thesis Approved:

Dr. Louis G. Johnson

Thesis Adviser

Dr. Weili Zhang

Dr. Yumin Zhang

Dr. Gordon Emslie

Dean of the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

RAM	Random Access Memory
SRAM	Static Random Access Memory
CMOS	Complementary Metal-Oxide Semiconductor
MOSFET	Metal Oxide Semiconductor Field-Effect Transistor
PMOS	p-channel Metal Oxide Semiconductor
NMOS	n-channel Metal Oxide Semiconductor
SPICE	Simulation Programs with Integrated Circuit Emphasis

NOMENCLATURES

V_{dd}	Supply Voltage (5V)
V_{inv}	Inverter Switching Voltage
V_{GS}	Gate-Source Voltage
V_{DS}	Drain-Source Voltage
V_{Tn}	n-channel MOSFET threshold voltage
V_{Tp}	p-channel MOSFET threshold voltage
t_{acc}	Access time of the register file

CHAPTER 1

INTRODUCTION

1.1 Register Files

Advancements in VLSI technology have lead to many architectural innovations in processor design. As more and more room became available in the chip, processors designers began to add many parallel instruction pipelines to improve the performance of the processor. These processors are capable of executing more than one instruction in each clock cycle and are called superscalar processor.

Register files are important components of Superscalar processor. They are on-chip data storage elements which are used to relay data between memory and the functional units. The access time of the registers is one of the important factors in the design of the register file since the register file has to run at the speed of the core processor. The processor operates much more efficiently when there are enough registers so that only occasionally does data need to be transferred between registers and memory.

Register files in superscalar processors need to be interfaced with more than one functional unit. One approach would be to partition the register file and dedicate each partition to one functional unit. The problem with this approach is that many machine cycles are wasted in moving the data between the registers. The second, more efficient approach would be to provide many input / output ports to each of the individual registers in the register file, and these register files are called multi-port register files.

Static RAM cells are used to implement on-chip register files. To implement multi-port register files, SRAM cells with more than one read/write port are required. The main focus of this thesis is to design a 32 bit x 32 word register file with 10-read/write ports using 0.6 micron technology.

1.2 Design Issues

Reliability is the main issue while designing a multi-port SRAM. The main figure of merit that defines the reliability of an SRAM is its noise margin. Noise Margin represents the immunity of an SRAM against external noise induced on the bit lines during both the read and write cycles. The noise margin of the SRAM has to be high to ensure proper working of SRAM. Both read and write margins are set by the sizes of the transistors in the SRAM cell. If the transistors are not sized appropriately, SRAM may not work correctly.

Read access time of the SRAM is another important design issue to be considered. The read access time determines the speed of the microprocessor. In the worst case, the 10-port SRAM will have to drive ten bit lines during a read operation. The transistors have to be sufficiently sized up so that the read access is not too large.

Area of the SRAM is another major concern. Since the SRAM cells are stacked to form the register file, an increase in the cell area of the SRAM will lead to significant increase in the area of the register file.

The design issues listed above are all interdependent. For example, sizing up the transistors to decrease the read access delay will significantly increase the area of the register file. If the transistors are not appropriately sized, the reliability of the SRAM cell is greatly reduced. Therefore in this design, arriving at the right tradeoffs to obtain acceptable performance benefits for each of these design parameters is a major challenge.

1.3 Thesis Organization

This thesis is organized into five chapters starting with coverage of the literature available about the various memory cell topologies and their pros and cons in chapter 2. Chapter 3 starts with a detailed description of the proposed memory cell. It then goes on to describe in detail the various design issues and trade-offs and arrives at specific values for the various parameters of the memory cell. Finally the chapter ends with a discussion of the design methodology of the decoder and sense amplifier. A testing methodology is developed and the various simulation results for this methodology are detailed in Chapter 4. The last chapter concludes by summarizing the design parameters and lists the strengths of the proposed design and gives a direction for future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Many journal papers have been published about the design of Static RAM cells. Researchers are still coming up with new designs to reduce the read access time and improve the noise margin of the memory cell. This chapter summarizes some of the design styles adapted while designing an SRAM.

2.2 6-Transistor (6T) SRAM Cell

The 6-Transistor (6T) SRAM is one of the most commonly used SRAM cells. This SRAM cell uses a single word line and two data lines (bit and bitbar). The cell contains a pair of cross coupled inverters and an access transistor for each data line.

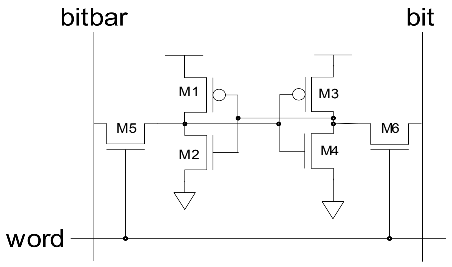

Figure 1: 6-Transistor (6T) SRAM cell

Dr. Louis G. Johnson, in his lecture notes [1], has discussed in detail about the design of 6T-SRAM cell. He has provided mathematical equations to find the optimum ratio between cell transistors (M1, M2, M3 and M4) and pass transistors (M5 and M6) to maximize noise margin and reduce access time delay.

During the read cycle, the bit lines are pre-charged. While reading a stored ‘0’, the transistor M4 (as shown in Figure 1) has to discharge the charge stored in the bit lines during the pre-charge cycle. Hence the transistor M4 has to be sufficiently sized up for proper working of the SRAM. Writing into the SRAM is double ended i.e. data and inverted data is provided at the bit and bitbar lines respectively. The transistors of the memory cell are sized up so that only a ‘0’ on the bit (or bitbar) lines can overwrite a ‘1’ in the memory cell. A ‘1’ on the bit (or bitbar) line cannot overwrite a ‘0’ stored in the memory cell. If this condition is not satisfied, a ‘1’ may be written into the memory cell while reading a stored ‘0’.

The main advantage of 6T SRAM cell is that it occupies small area and dissipates low power. The main limitation of this memory cell is that it’s read and write noise margin is very low. A register file constructed using 6T SRAM cell can be integrated with only one functional unit. Hence multi-port SRAM cells are preferred in superscalar processors.

2.3 Quad Ported SRAM

The authors of [2] have designed a quad ported SRAM which allows four concurrent read/write accesses simultaneously. It was designed in 0.35 μm CMOS process with three metal layers and a VDD voltage of 3.3V. The schematic of the one bit quad ported cell is shown in Figure 2.

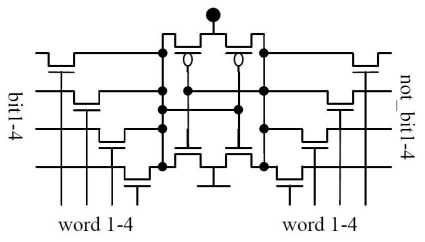

Figure 2: Quad Ported SRAM

The NMOS transistors of the two inverters are sized up to threefold area of a minimum sized transistor. The PMOS transistors are kept at minimum size. The bit lines are pre-charged to 2.8V at every read cycle. A differential amplifier with current source load, as shown in Figure 3, is employed as the sense amplifier.

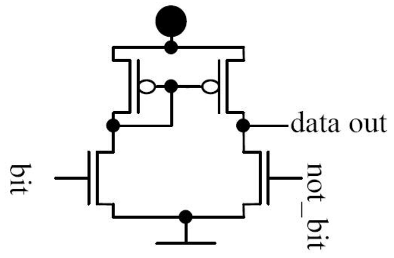

Figure 3: Sense Amplifier

It is mentioned that the optimum size for the PMOS transistor in the sense amplifier is 15 times minimum size and that of NMOS is 4 times minimum size. The sense amplifier takes about 1.7 ns to drive the output to 0.9VDD (while reading a 1) or 0.1VDD (while reading a 0). The complete read cycle takes 2.1ns from applying the address to the output of the digital data.

2.4 32–Word by 32–bit 3–Port Register File

A 32–Word by 32–bit 3–Port Register File has been designed by the authors of [3]. Figure 4 shows how the register file is organized.

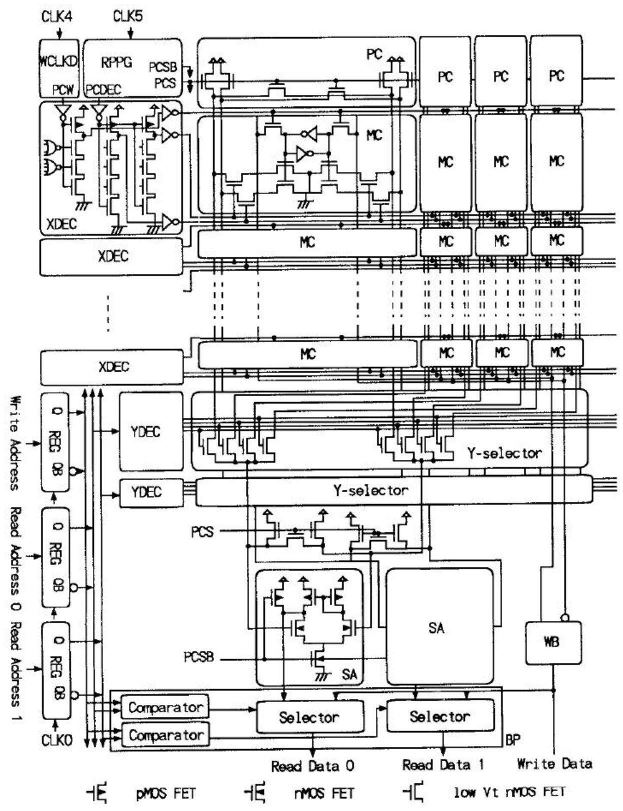

Figure 4: 32–Word by 32–bit 3–Port Register File

The register file consist of a write clock driver (WCLKD), a read pre-charge pulse generator (RPPG), pre-charge circuits (PCs), decoders (XDECs, YDECs), memory cells (MCs), Y selectors, sense amplifiers (SAS), a write buffer (WB), a bypass circuit (BP), and address registers (REGS). Low threshold voltage n channel MOSFET was used in the design of decoders, memory cell read & write ports and pre-charge circuits to improve the read access time. The register file was fabricated using 0.4 μm CMOS process technology and occupied an area of 2.78mm x 0.72mm on chip. The measured delay time was 1.88ns.

In the memory cells discussed in section 2.2, 2.3 and 2.4, during the read cycle, if the transistors are not appropriately sized up, there is a possibility that a wrong value gets stored in the memory cell. This situation is overcome by using different read and write busses as discussed in section 2.5 and 2.6. But these architectures require more area than the conventional multi-port SRAM.

2.5 SRAM with three address word lines and data bit lines

In conventional two port static RAM cells, a read-time error can take place if the transition of write/read mode and the transition of register address occur simultaneously. To overcome this problem, the authors of [4] have designed a new SRAM as shown in Figure 5. In this topology, two simultaneous single ended read operations are possible.

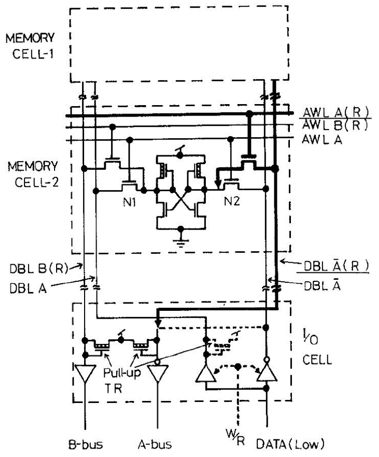

Figure 5: SRAM with three address word lines and data bit lines

The two complementary data bit lines, $DBL.A$ and $DBL.\overline{A}$ and the address word line $AWL.A$ are used for a write operation. One of the data bit lines, $DBL.\overline{A}(R)$ or $DBL.B(R)$ and one address word line $AWL.A(R)$ or $AWL.B(R)$ are used for two simultaneous read operations. The authors have mentioned that the cell occupies an area of 61 x 68 μm^2 which is about 30 percent larger than a conventional two-port RAM.

2.6 Register File on the Itanium-2 Microprocessor

As discussed in [5], the Itanium-2 microprocessor contains a 20-ported, 128-entry, 65-bit register file with 12 read and 8 write ports. Each register has 12 word lines instead of 20 word lines as in a standard register file. There is a shared control wire (WRITEH) for every register which determines the read/write directionality of the word line. Figure 6 shows the schematic of a SRAM cell.

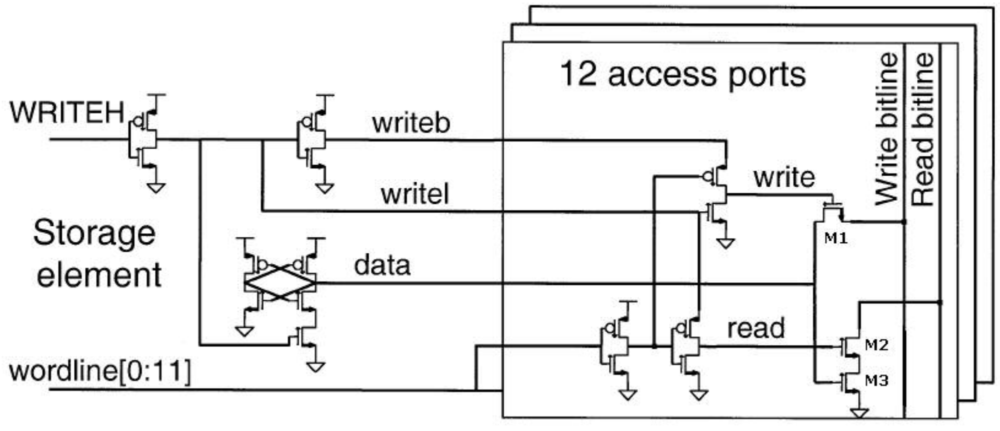

Figure 6: Schematic of 20-port SRAM in Itanium-2 microprocessor

Different read and write bit lines are used in this SRAM cell. During the write cycle, the data to be written is available on the write bit line and the WRITEH signal is asserted high. The gate of the transistor M1 is asserted high and the transistor M1 is turned ON. The data on the write bit line is now available at the data node. While reading a 1 from the memory cell, the transistors M2 and M3 are ON and the charge stored on the bit lines during the pre-charge cycle is discharged. While reading a 0 from the memory cell, the transistor M3 is cut-off and the read bit line is not discharged.

2.7 Summary

The 6T SRAM cell occupies much less area when compared to other multi-port SRAM cells, but it can be integrated to only one functional unit. Hence multi-port SRAM cells are preferred to design register files for superscalar processors. The transistors of multi-port SRAM cells have to be appropriately sized up to avoid a wrong value being written into the memory cell during the read cycle. This situation can be avoided by using different read and write busses as discussed in section 2.5 and 2.6, but such designs occupy a larger area than conventional designs. A trade-off has to be achieved between the area, noise margin and access time of the memory cell. In this thesis, a new memory cell has been designed to achieve the right trade-off between these design parameters.

CHAPTER 3

DESIGN OF REGISTER FILE

3.1 Introduction

A register file is an array of processor registers in a central processing unit. Modern integrated circuit-based register files are usually implemented using fast static RAMs with multiple ports. A register file consists of three main components:

1. Memory cell array
2. Address Decoder
3. Sense Amplifier

Memory cell arrays are constructed by arranging static RAM cells in the form of a matrix. Each column of this matrix shares common bit lines and each row shares common word lines. An address decoder decodes a valid address and asserts only one of the word lines high. A sense amplifier quickly detects the value on the bit line and helps in reducing the access time of the register file. This chapter describes in detail the design of these three components.

3.2 Register file organization

Figure 7 shows how the register file is organized. The register file that is designed can store 32 words where each word is 32 bits wide. 10 port SRAM cells are used to construct the register file and words can be simultaneously written on any of the ten ports or simultaneously read from any of the 10 ports. This word addressable register file requires 5-to-32 decoder to address every word in the register file. Since there are 10 ports in the register file, we require 10 decoders for addressing all ports. Each of the bit lines is connected to a sense amplifier. In this register file, an inverter is used as a sense amplifier. By using an inverter as a sense amplifier, the complexity of circuit and the chip area required are greatly reduced. In register files where read access time is critical, single ended analog sense amplifier like [6] is recommended.

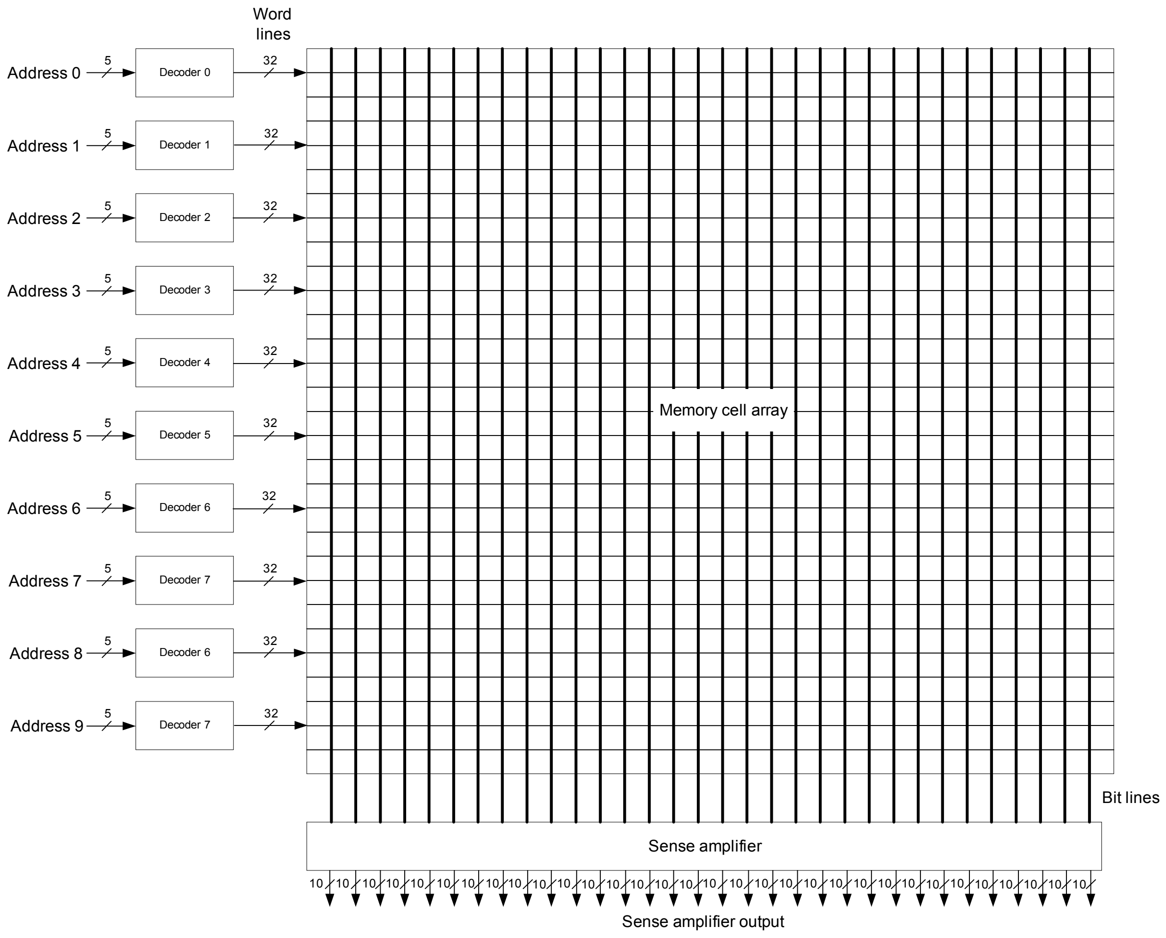

Figure 7: Register File

The timing sequence of write and read operations are shown in Figure 8 and Figure 9. A write operation occurs when both the clock and word signal is asserted high. A read operation occurs when the clock is asserted low and word is asserted high. Multiple simultaneous read operations from a register array are possible. But multiple simultaneous write operations to the same register array should not be done because this would result in unpredictable data being stored in the register array.

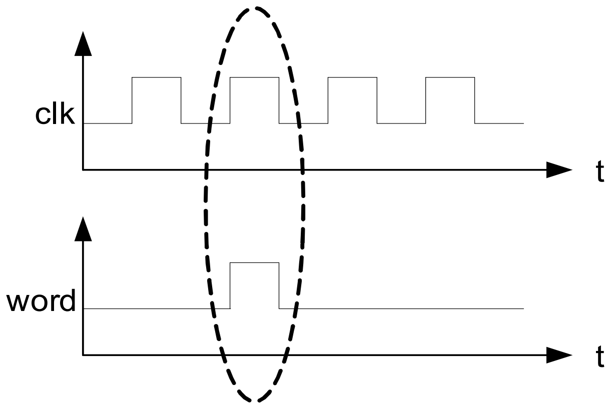

Figure 8: Write cycle

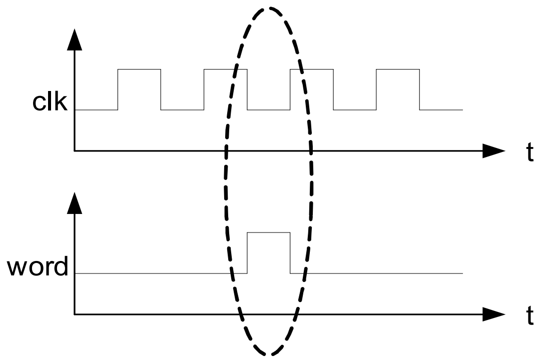

Figure 9: Read cycle

3.3 Implementation of Data Storage Element

Static RAM is used as the data storage elements in a register file. Dynamic RAM is not usually used for register files for the following reasons [7]:

1. The contents of the cell would fade away with time and must be refreshed at regular intervals. Additional refresh circuitry is required if dynamic RAM cells are used for implementing the data storage elements in a register file.

2. The Dynamic RAM stores the charge on the floating node which is used to drive the column sense amplifiers connected to each bit line. The storage capacitance has to be very large to produce a change on the bit lines. But these large storage capacitances increase the capacitive load on the bit lines and which in turn increase the settling time of the bit lines.

3. Dynamic RAM cells are implemented using specialized fabrication processes. Implementing DRAM cells using fabrication processes used for normal digital circuits will reduce its performance tremendously. Thus implementing register files using DRAM will not be a good idea especially when they need to be integrated with other components of a processor.

3.4 Design of 10 port SRAM

The 10-port SRAM cell described in this thesis was inspired by the SRAM cell of the Itanium-2 microprocessor discussed in section 2.6. Both these memory cells use single ended write and single ended read. In this thesis, an attempt was made to reduce the number of transistors involved in the design of SRAM when compared to that in the Itanium-2 microprocessor. As discussed in section 2.6, in the Itanium-2 microprocessor, a separate read/write signal (WRITEH) is used to write into or read from the SRAM cell. Here, we use clock and word signals to identify read and write operations. In the Itanium-

2 microprocessor, there are separate read and write bit lines. On the other hand, in this design, the same bit line is used for both read and write operations. In the Itanium-2 microprocessor, separate transistors drive each bit line. But in this design a single large transistor is used to drive the capacitance associated with all the 10 bit lines.

In the register file used in the Itanium-2 microprocessor, since there are separate read and write bit lines, read and write operations to different memory locations can take place simultaneously. But in the register file proposed in this thesis, since only one bit line is used per port for both read and write operation, simultaneous read and write operations are not possible. By using the same bit line for both read and write operation, we achieve a huge reduction in the area occupied by the memory cell array when compared to that in the Itanium-2 microprocessor.

The schematic of the 10 port SRAM is shown in Figure 10. It has ten word lines (word0 to word9) and ten bit lines (bit0 to bit9). During the write cycle, data can be written from only one of the ten ports to an individual memory cell and during the read cycle, data can be read from a single memory cell simultaneously from all the ten ports.

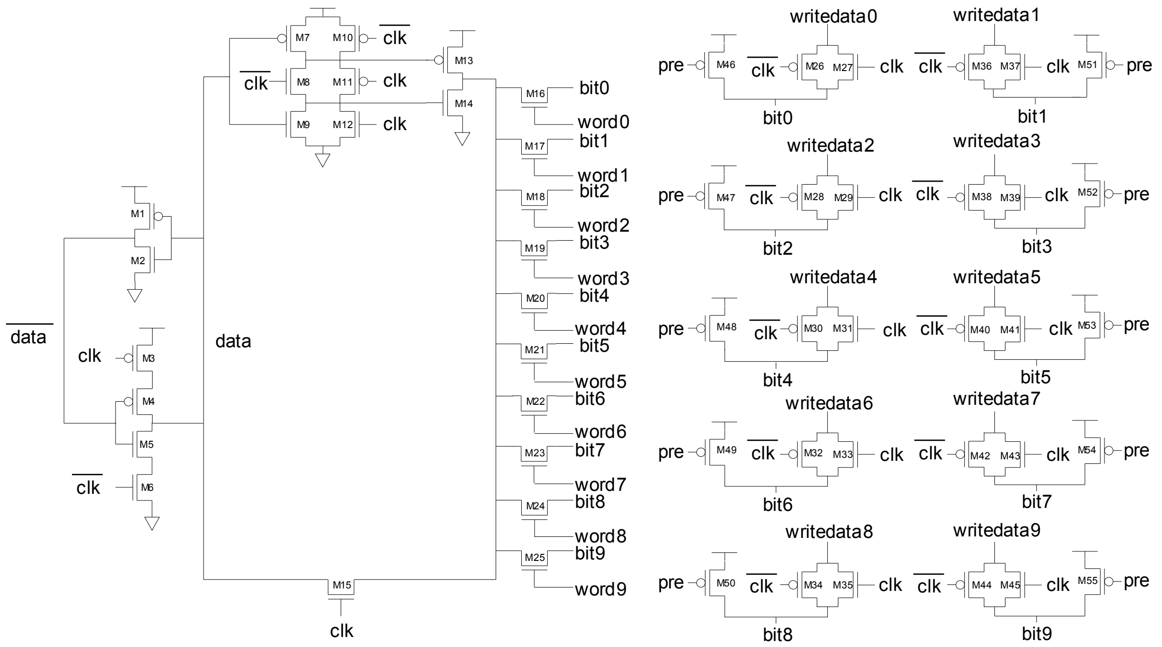

Figure 10: Ten port SRAM

3.4.1 Writing into the memory cell

During the write cycle, the word lines of one of the ports through which data is written will be asserted high. The data to be written is applied to the writedata pin of the corresponding port. Since clock is asserted high during the write cycle, the data to be written will be available at the data node after passing through the NMOS pass transistors (M15 to M25). Since the transistors M3 and M6 are OFF, writing the data into the memory cell is much easier. The transistors M8 and M11 are OFF and transistors M10 and M12 are ON. Hence the drain of transistors M13 and M14 are in a high impedance state during the write cycle, irrespective of the value at the data node.

Figure 11 shows the condition of each transistor when data is written through port 4.

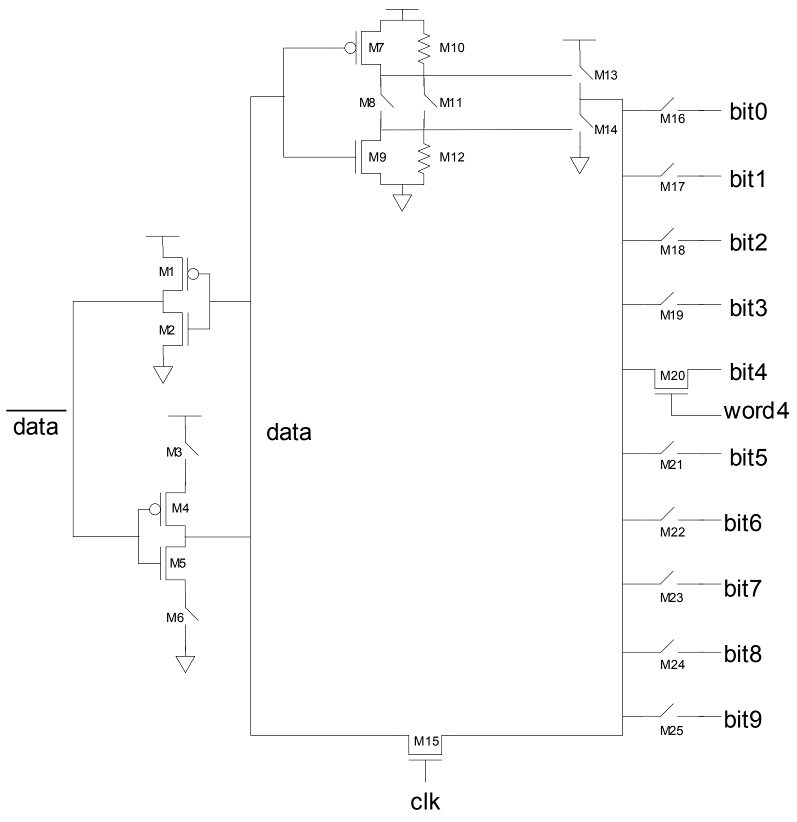

Figure 11: Ten port SRAM – Write cycle

3.4.2 Reading 0 from the memory cell

While reading a 0 from the memory cell, the bit lines are pre-charged high for a time duration equal to the decoder delay. After the pre-charge cycle, the word line corresponding to the port through which data is read will be asserted high. The transistors M7, M8, M11 and M14 are ON, and the transistors M9, M10, M12 and M13 are OFF. The charge stored in the bit line during the pre-charge cycle, is discharged through the transistor M14 and the bit line is pulled low.

Figure 12 shows the condition of each transistor when 0 is read through port 2, port 4 and port 6.

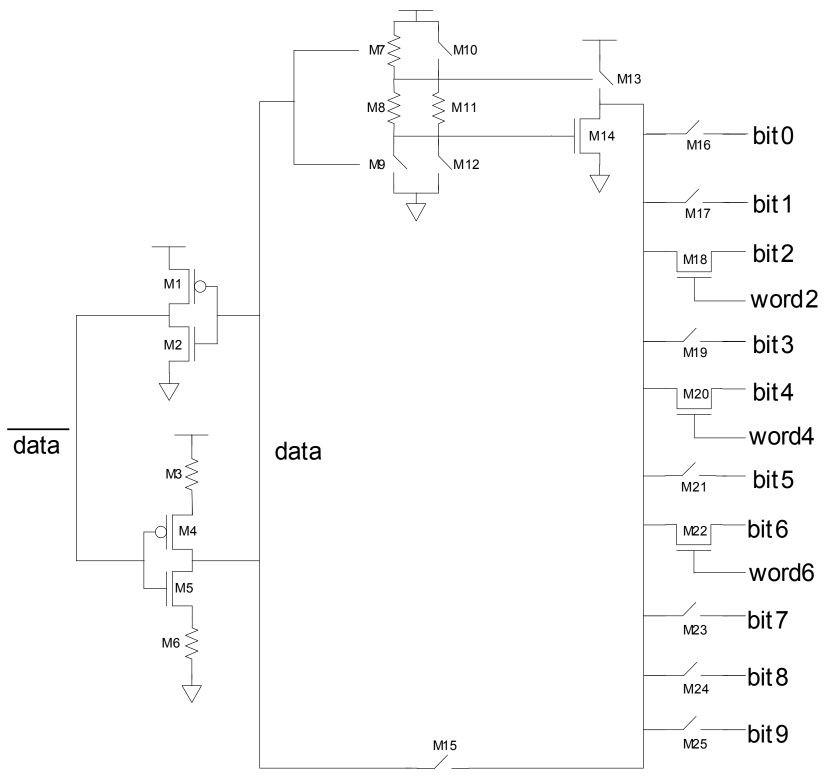

Figure 12: Read 0 from the memory cell

3.4.3 Reading 1 from the memory cell

While reading a 1 from the memory cell, the bit lines are pre-charged high. After the pre-charge cycle, the word line corresponding to the port through which data is read will be asserted high. The transistors M8, M9, M11 and M13 are ON, and the transistors M7, M10, M12 and M14 are OFF. Since the bit lines are already pre-charged high, the transistor M13 need not drive the bit lines high.

Figure 13 shows the condition of each transistor when 0 is read through port 2, port 4 and port 6.

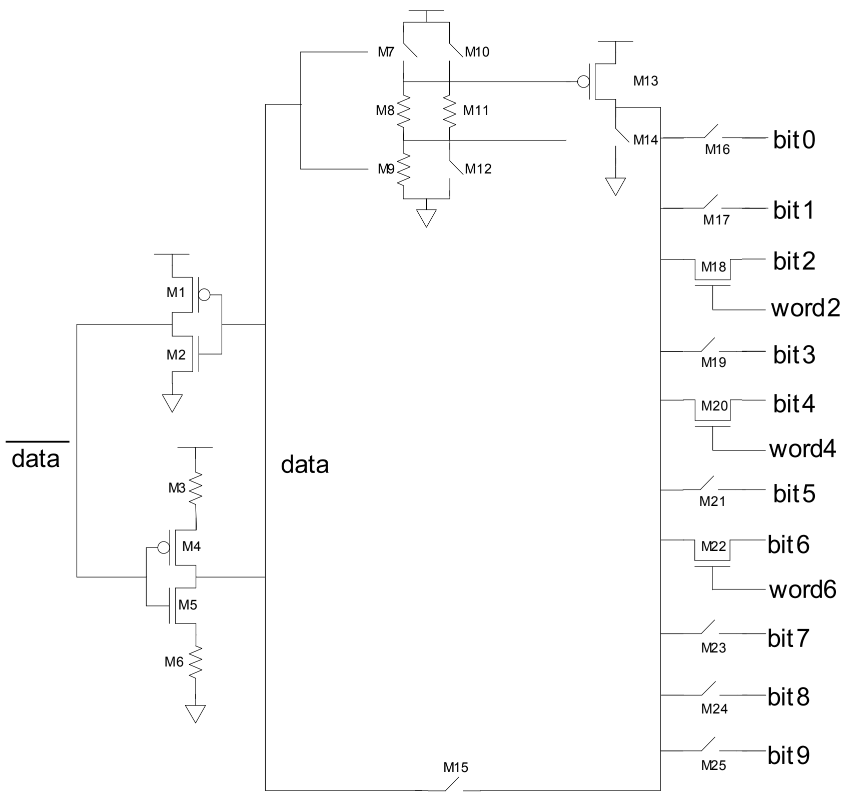

Figure 13: Read 1 from the memory cell

3.4.4 Transistor sizing

During the read cycle, while reading a stored 0, the transistor M14 has to discharge the charge stored in the large bit line capacitance. In the worst case, it may have to discharge all the ten bit line capacitances. If the transistors are not appropriately sized, the memory cell will not work correctly.

While reading a stored 1, as discussed in section 3.4.3, the transistor M13 need not drive the bit lines high. Hence the transistor M13 is kept at minimum width to minimize the area occupied by the SRAM cell.

Read access time is inversely proportional to the transistor size of M14. But increasing the transistor size considerably will increase the area of the register file. Hence a trade-off has to be made between the read access time and the memory cell area.

Simulation studies were done to determine the time taken to discharge bit line capacitance for various widths of n-transistors. The following graph shows the simulation results.

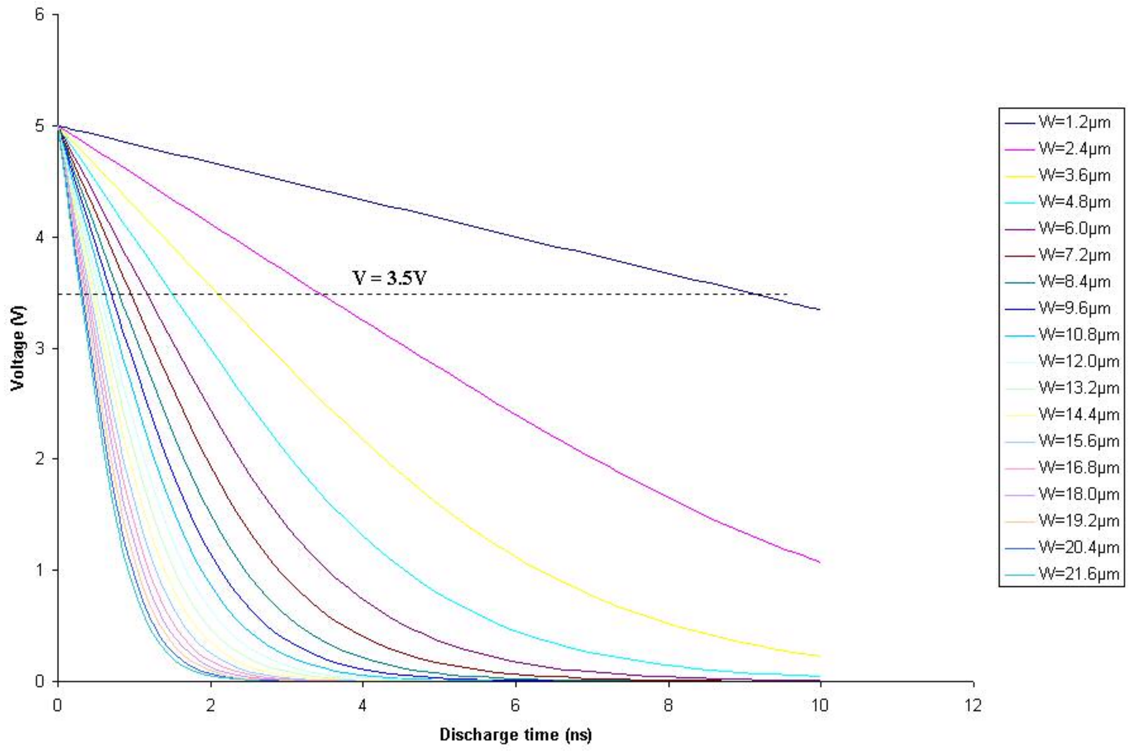

Figure 14: Discharge rate of bit line capacitance for various transistor widths

The time taken to discharge the bit line capacitance from 5V to 3.5 V is given in Table 1. 3.5V represents the switching voltage (V_{inv}) of the sense amplifier. The reason for choosing 3.5 V to be the switching voltage of sense amplifier is explained in section 3.6.

Table 1: Discharge rate of bit line capacitance for various transistor widths

Width of n-transistor (µm)	Time taken to discharge capacitance from 5V to 3.5V (ns)
21.6	0.30
20.4	0.31
19.2	0.33
18	0.36
16.8	0.39
15.6	0.42
14.4	0.45
13.2	0.50
12	0.55
10.8	0.61
9.6	0.70
8.4	0.81
7.2	0.95
6	1.17
4.8	1.50
3.6	2.10
2.4	3.46
1.2	9.15

Figure 15 shows a plot of the data represented in Table 1. As the width of the transistor increases, the time taken to discharge the bit line capacitance decreases. But if the transistor width is increased beyond 14.4µm, the percentage decrease in discharge time is less than 9% and we are now in the region of diminishing returns. These simulation results are used to determine the width of the transistor M14. The width of transistor M14 is set to 14.4 µm.

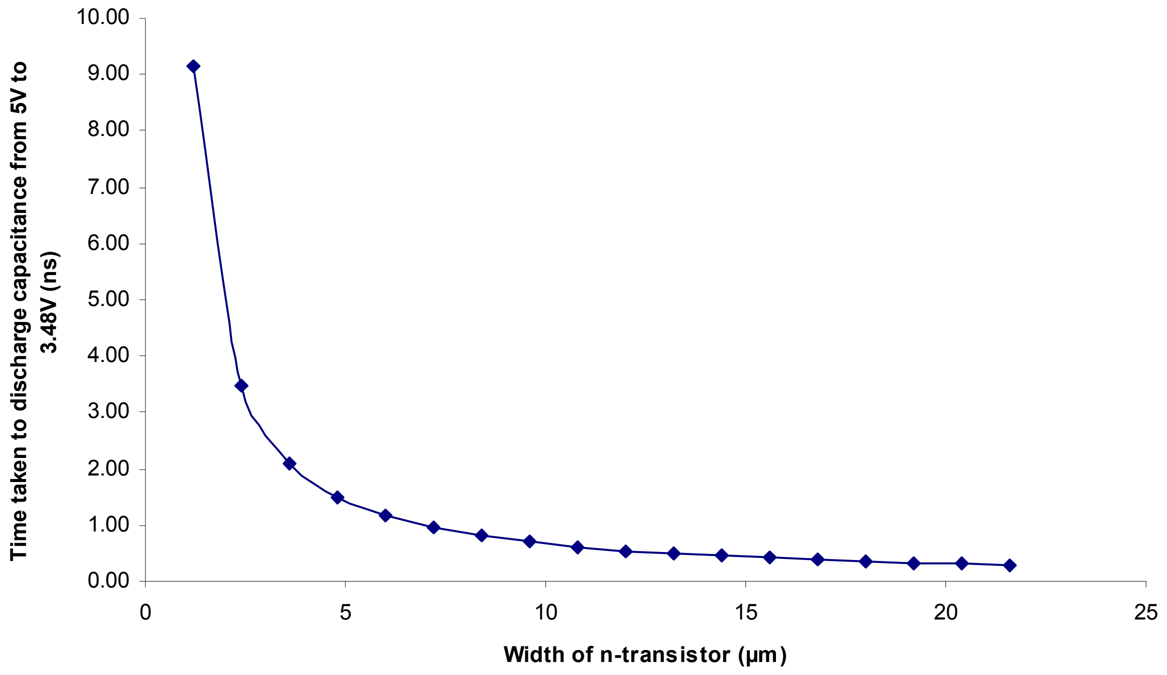

Figure 15: Time taken to discharge from 5V to 3.5V for various transistor widths

3.4.5 Estimating the bit line capacitance

The bit line capacitance consists of two components.

1. The diffusion capacitance (C_{diff}) of the pass transistors connected to the bit lines.
2. The metal capacitance (C_m) on the bit lines.

The diffusion capacitance consists of two components: bottom-plate junction capacitance (C_{bottom}) and the side wall junction capacitance (C_{sw})[8].

$$\begin{aligned} C_{diff} &= C_{bottom} + C_{sw} \\ &= CJ.W.L + CJSW.(W + 2L) \end{aligned}$$

where

CJ = Bottom plate junction capacitance per unit area

CJSW = Side wall junction capacitance per unit perimeter

L = Length of the drain region

W = Width of the drain region

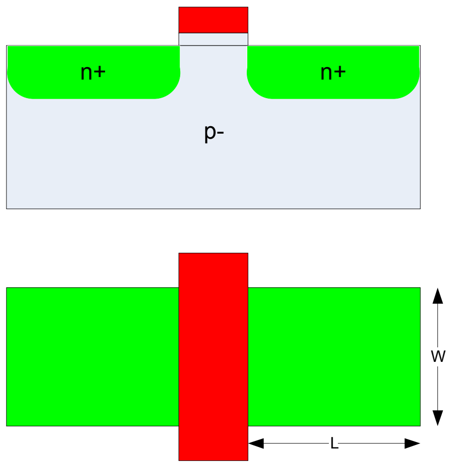

Figure 16: Dimension of drain region

CJ and CJSW are process parameters. The values for CJ and CJSW for 0.6 micron technology are given below.

$$CJ = 4.234 \times 10^{-4}\ F.m^{-2}$$

$$CJSW = 3.826 \times 10^{-10}\ F.m^{-1}$$

3.5 Design of Decoder

Each memory cell has 10 word lines. Hence the register file needs totally 10 address decoders. Each decoder has 5 input lines and 32 output lines. The decoder can accept address ranging from 00000 to 11111. According to the input to the decoder, one of the output lines is asserted high and all the other output lines remain low.

The behavioral code for the address decoder is presented in Appendix 1a which was synthesized using the OSU standard cell library. The synthesized structural code is presented in Appendix 1b. The test bench used to test the decoder and the simulation results are presented in Appendix 1c and 1d respectively. Figure 17 shows the schematic of the address decoder.

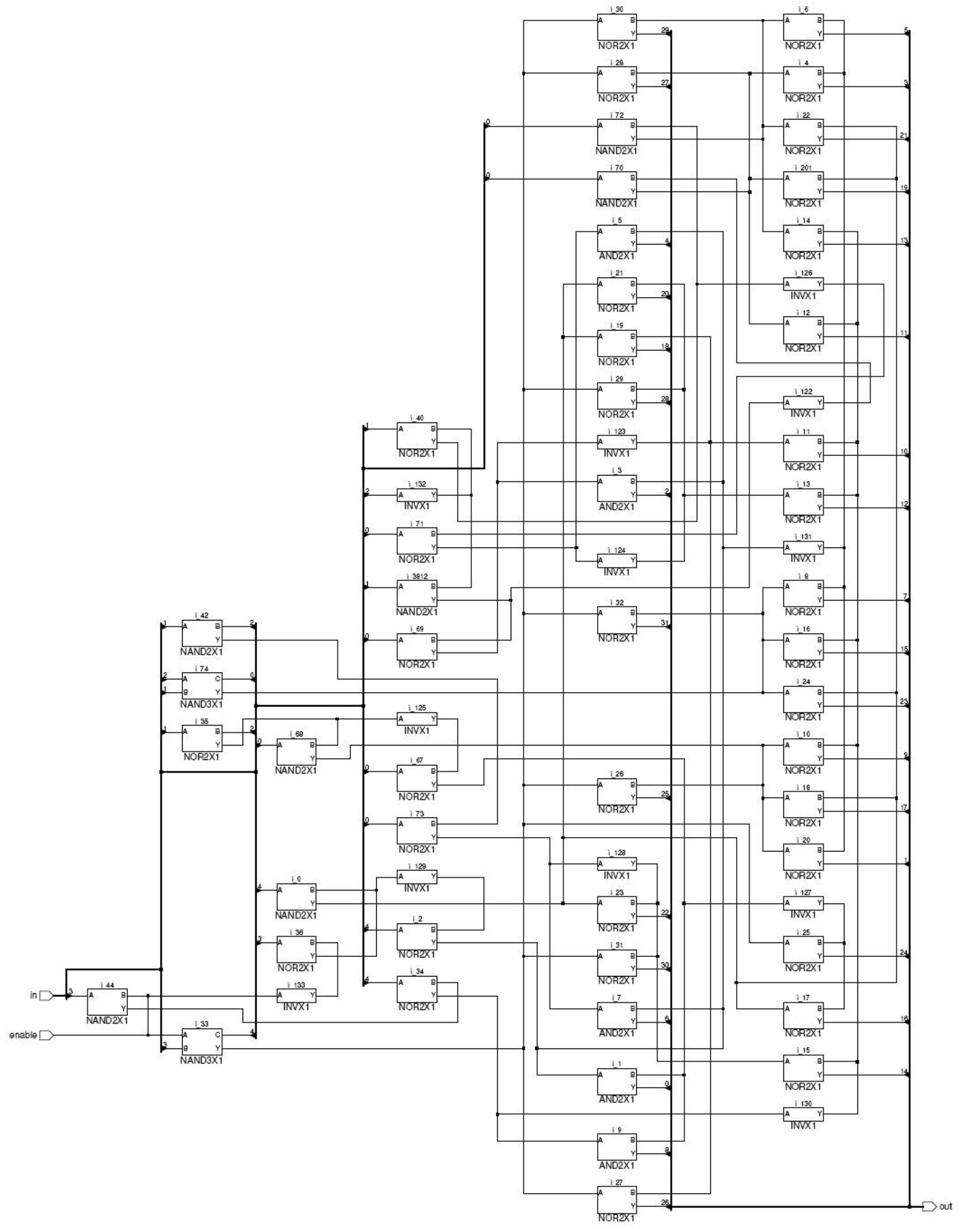

Figure 17: Schematic of Address Decoder

The decoder delay* was found to be 0.6ns. Address decoding and pre-charging of bit lines are done simultaneously to reduce the access time of the register file. The pre-charge transistors are sized up so that the bit lines are pre-charged within 0.6ns.

3.6 Design of sense amplifiers

In memory design, it is a common practice to read information from the memory cells by using an analog differential amplifier to sense the difference between the bit and bitbar lines quickly. In this design since there is only one bit line for every word line, a specially designed inverter is used as the sense amplifier for single ended reads as shown in Figure 18. As the focus of this thesis is mainly on the design of the SRAM cell, this design uses an inverter as a sense amplifier although analog sense amplifiers are industry standard nowadays.

Since the bit lines are pre-charged high, the inverters are designed to quickly detect a zero in the bit line. The switching voltage V_{inv} of the inverter has to be greater than $V_{dd}/2$ to minimize the read access time. This is achieved by using wider pFETs in the inverter sense amplifier [7].

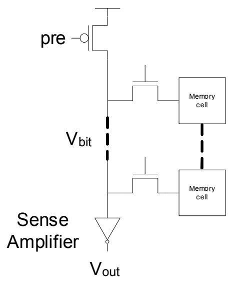

Figure 18: Inverter as Sense Amplifier

* Pre-layout delay calculated from simulation in Cadence Simvision. The post-layout back annotated delay was not calculated which is beyond the scope of this thesis.

Simulations were done to determine the switching voltage (Vinv) of a CMOS inverter for various widths of p-transistor. The n-transistors of inverters are kept at minimum width because sizing it up will not enhance the performance of sense amplifier. The Voltage Transfer Characteristics (VTC) of inverters with different p-transistor widths as shown in Figure 19.

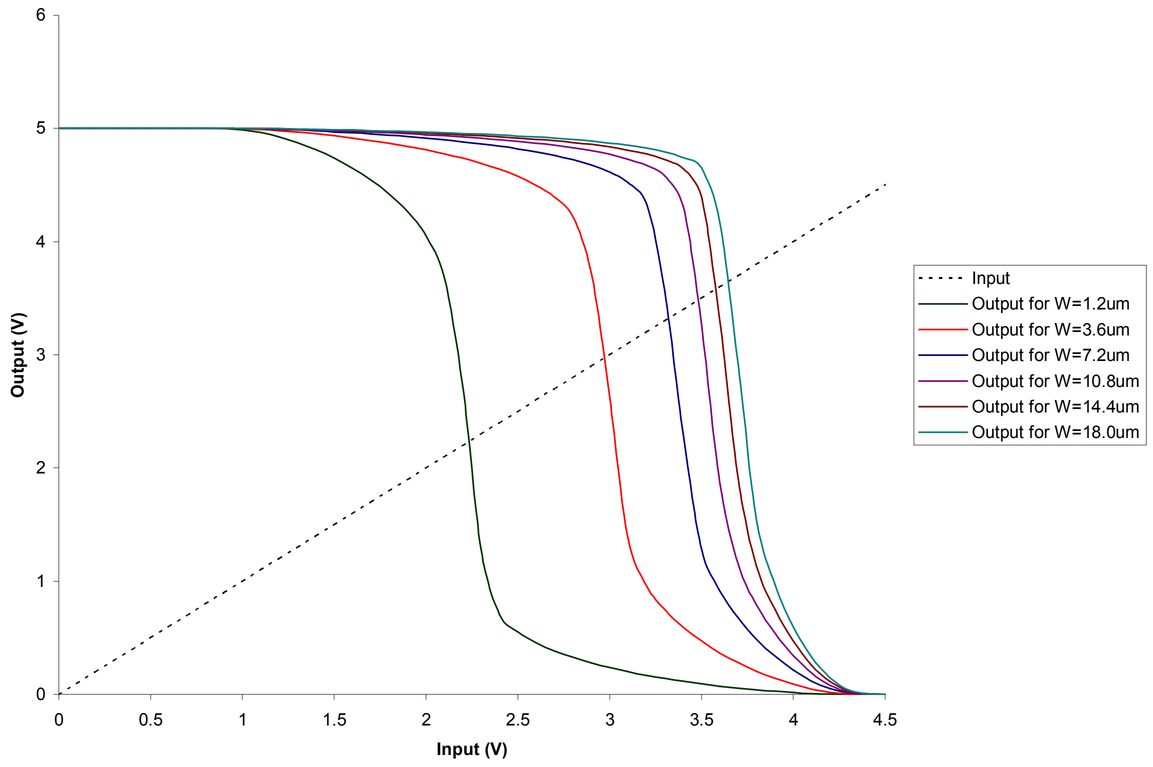

Figure 19: Voltage Transfer Characteristics (VTC) of inverter

Table 2 shows the switching voltage of inverters for different PMOS width.

Figure 20 shows a plot of the data represented in Table 6.

Table 2: Switching voltage for various p-transistor widths

Width of PMOS (µm)	**Switching Voltage V_{inv} (V)**
1.2	2.23
3.6	2.97
7.2	3.32
10.8	3.48
14.4	3.58
18.0	3.64

As the width of the PMOS increases, the switching voltage V_{inv} value also increases. But when the width of PMOS increases beyond 10.8 µm, the rise in the switching voltage becomes negligible. Hence width of the PMOS is chosen to be 10.8 µm.

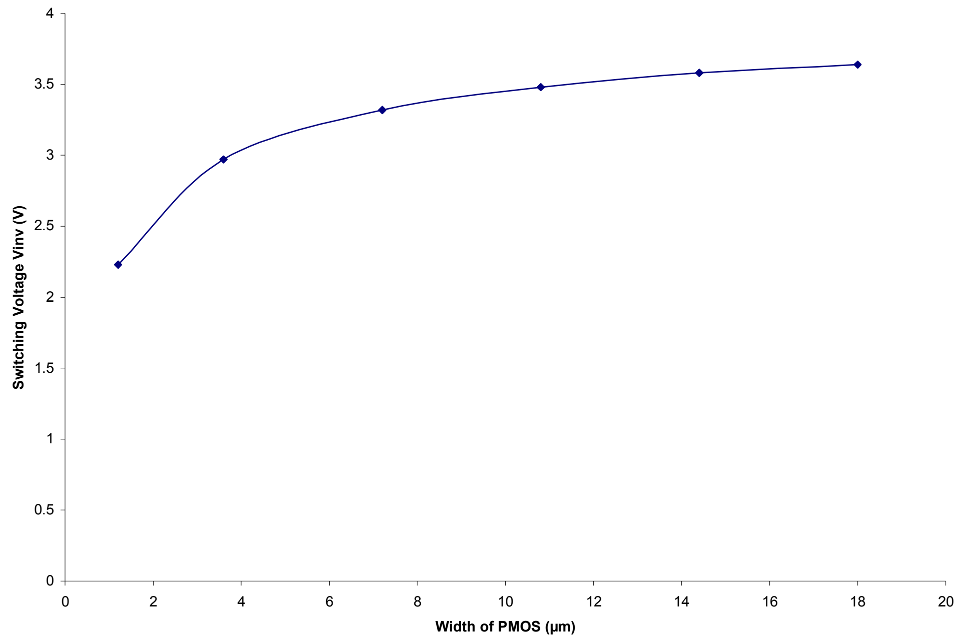

Figure 20: Inverter switching voltage for various p-transistor widths

Figure 21 shows the timing diagram during the read cycle. During the read cycle, the word line is asserted high and clock is asserted low. The pre-charge signal is asserted low for duration equal to decoder delay. This will turn ON the pre-charge transistor and the bit lines are pre-charged high. After the pre-charge cycle, if zero is read from the memory cell, the transistor M12 discharges the bit line capacitance. Once the voltage at the bit line reaches the switching voltage (*V_{inv}*) of the sense amplifier, the output of the sense amplifier (*V_{out}*) begins to rise.

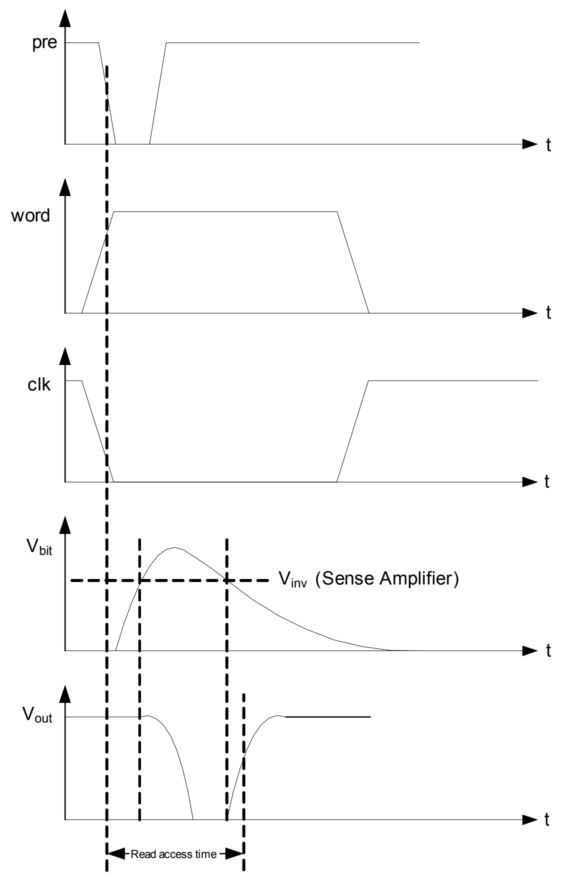

Figure 21: Timing diagram – Read cycle

3.7 Summary

An address decoder, a sense amplifier and a memory cell array were designed. The address decoder was designed using OSU standard cells. The delay of the decoder was found to be 0.6ns. An inverter with wider pFET was used as the sense amplifier. The switching voltage (V_{inv}) of the sense amplifier is 3.5 V and the width of the p-transistor used in sense amplifier is 10.8μm. The memory cell array was constructed using 10 port SRAM cells. All the pass transistors of the SRAM cell are kept at minimum width (1.2 μm) to minimize the bit line capacitance. The width transistor M14 of the SRAM is 14.4μm.

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 Introduction

In chapter 3, the design of memory cell, address decoder and the sense amplifier is explained in detail. The focus of this chapter would be to explain the implementation of the memory cell, the testing methodology of the memory cell & register array, and to tabulate the simulation results obtained.

4.2 10-port SRAM layout

Figure 22 shows the layout of the 10-port SRAM cell. All the word lines are laid out in metal 2 and the bit lines are laid in metal 3. Most of the routing between transistors in the memory cell is achieved using metal 1. The area occupied by one memory cell is 1198.8 μm^2. The tools and process used in making the layout is listed below.

- ➢ EDA (Electronic Design Automation) tool: Cadence
- ➢ Layout editor: Virtuoso
- ➢ Design rules: MOSIS Sub-micron, Scalable CMOS N-well technology (SCN_SUBM)
- ➢ Process: AMI 0.6 micron
- ➢ Waveform viewing tool : Analog artist waveform window (AWD)

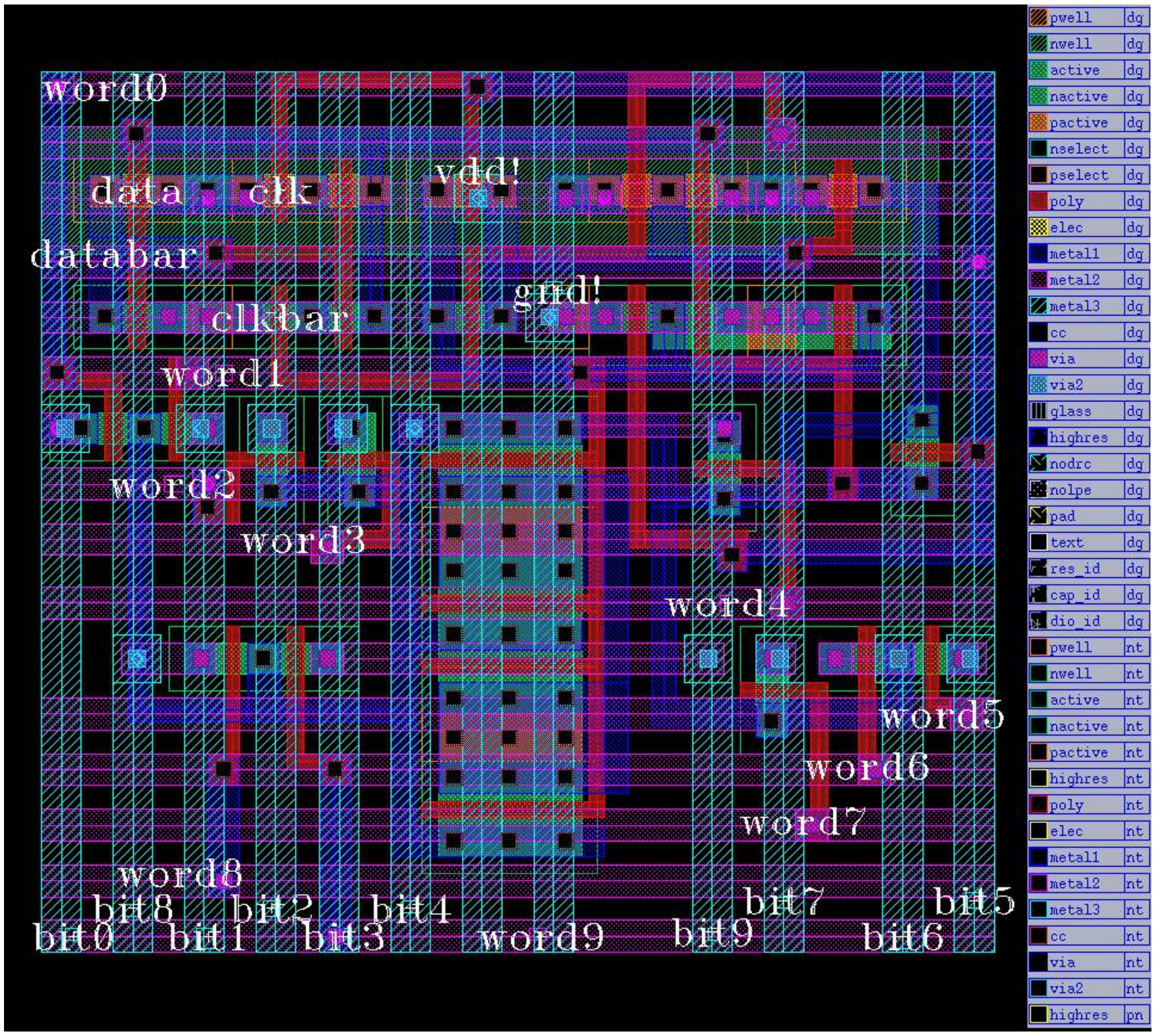

Figure 22: Layout of 10 port memory cell

4.3 Testing 10-port SRAM cell

The memory cell is tested by writing data through each port of the memory cell and reading the same from all the ports simultaneously. If the data being read matches with the data written, then the SRAM cell is working fine. As discussed in the previous chapter, reading a stored 0 through all the ports simultaneously represents the worst case scenario and hence has to be tested.

The layout is tested using a simulation tool called Spectre. Spectre is a detailed circuit simulator similar to spice that accurately determines node voltages and currents as functions of time. The parasitic capacitances are extracted and a netlist with parasitic capacitance is generated. The netlist file is however not a complete Spectre file. We need to add the MOSFET transistor models and the input stimulus to perform the simulation. The pre-charge transistors, write drivers and sense amplifiers were simulated in the Spectre file. The Spectre file used to test the 10 port SRAM is presented in Appendix 2.

The following sequences of operations were done to test the SRAM cell.

1. Writing a 1 through port 0.
2. Reading from all the ten ports simultaneously.
3. Writing a 0 through port 1.
4. Reading from all the ten ports simultaneously.
5. Writing a 1 through port 2.
6. Reading from all the ten ports simultaneously.
7. Writing a 0 through port 3.
8. Reading from all the ten ports simultaneously.
9. Writing a 1 through port 4.
10. Reading from all the ten ports simultaneously.
11. Writing a 0 through port 5.
12. Reading from all the ten ports simultaneously.
13. Writing a 1 through port 6.
14. Reading from all the ten ports simultaneously.

15. Writing a 0 through port 7.
16. Reading from all the ten ports simultaneously.
17. Writing a 1 through port 8.
18. Reading from all the ten ports simultaneously.
19. Writing a 0 through port 9.
20. Reading from all the ten ports simultaneously.

The storage node voltages, V_{data} and $V_{databar}$, are monitored during each read and write cycle. The simulation waveforms are plotted using the AWD waveform viewing tool. Figure 23 shows the simulation results obtained while testing the 10 port SRAM cell using Spectre.

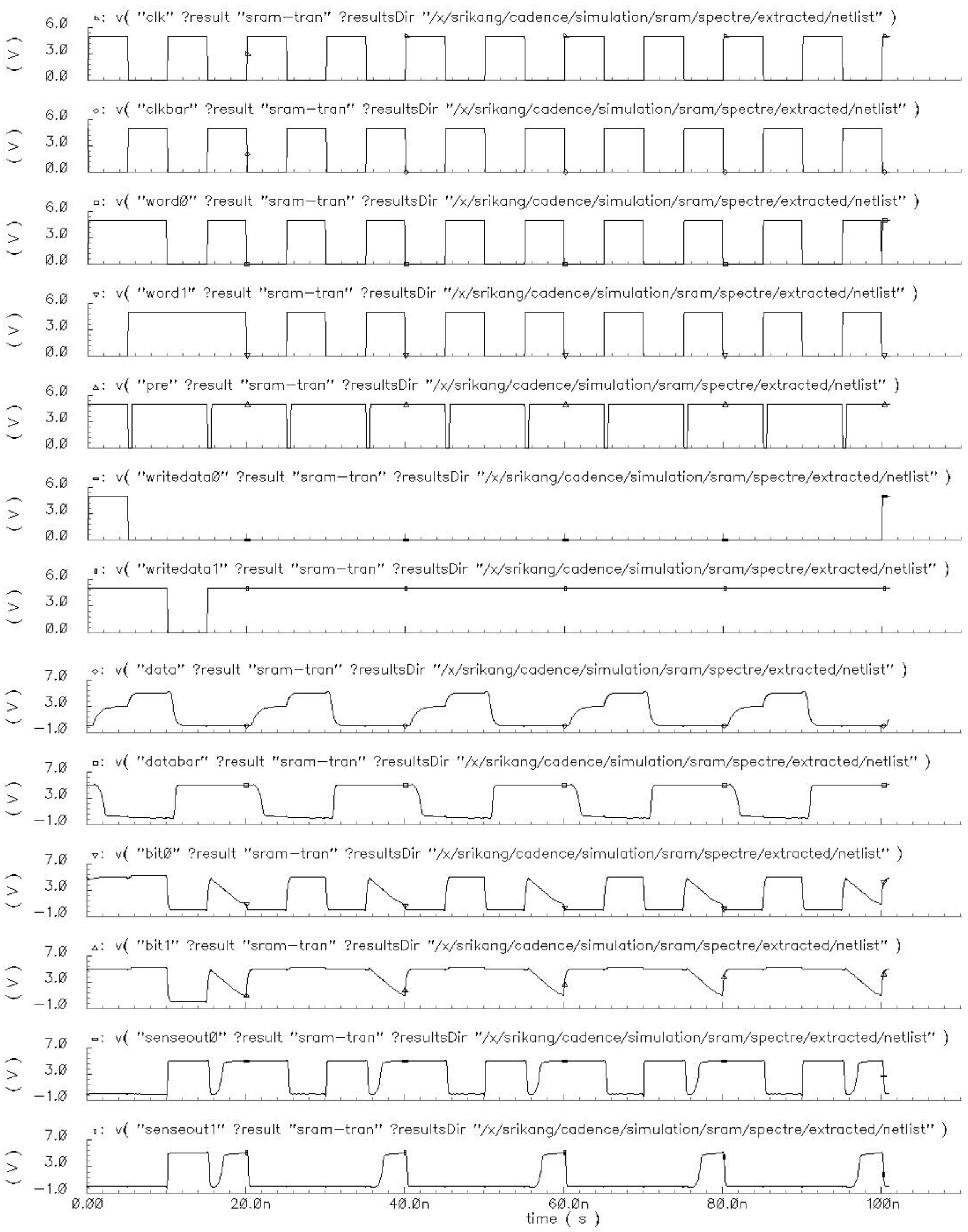

Figure 23: 10 port SRAM simulation results

4.4 Noise Margin

While writing a '1' into the memory cell, the maximum voltage achieved at the data node is 3.04 V and while writing a '0' the maximum voltage achieved at the data node is 25.18 mV. The storage inverter (M1 and M2) are designed using minimum width transistors. Hence the switching voltage (V_{inv}) of the storage inverter is 2.23 V as shown in Table 2. The write noise margin of the memory cell is

$$NM_{write} = 3.04 - 2.23$$
$$= 0.81\ V.$$

While reading the data from the memory cell, since the bit line is not directly driven by the storage inverter pair, the data stored in the memory cell is never destroyed. The read cycle is non-destructive and hence the read noise margin is infinite.

$$NM_{read} = \infty$$

4.5 Access time

Access time is the interval between the time when an address is provided to the address decoder and the time when stable data is available at the sense amplifier output. It is the sum of delays of the address decoder, the memory cell array and the sense amplifier. Figure 24 explains how the access time is calculated. The access time is measured between 50% of the pre-charge signal transition and 50% of the sense amplifier output transition. From the Spectre simulations, the access time was found to be 1.96ns.

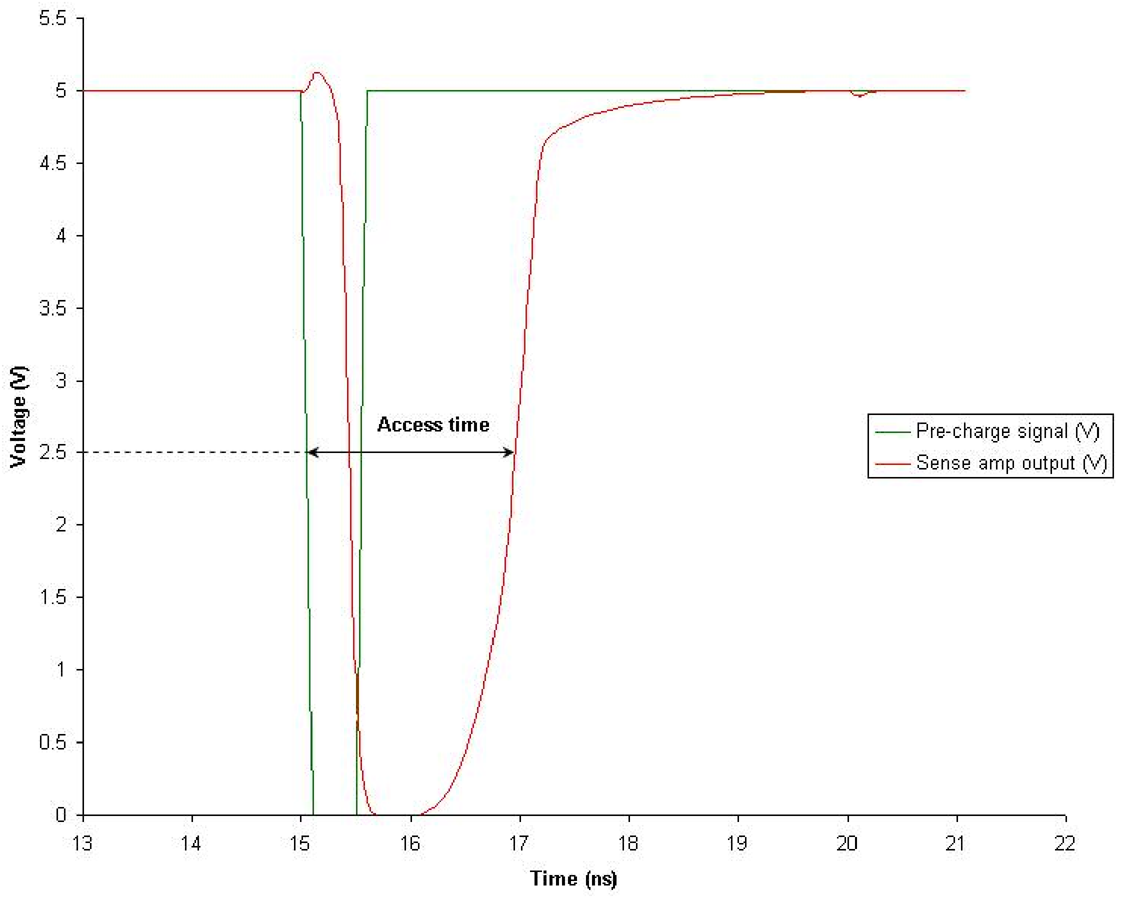

Figure 24: Access time delay

4.6 Testing the memory cell array

The register file is simulated using the IRSIM simulator. IRSIM [9] is an event driven logic level simulator which can handle the RC delays on nets. Logic level simulation is a coarser simulation than SPICE. Signals are represented in one of the following states: 0, 1, X (undefined). IRSIM simulations are much faster and use less memory than SPICE simulations.

The IRSIM file used to test the memory cell array is represented in Appendix 3a and the simulation results are shown in Appendix 3b. In these tests, a 32 bit word is being written to each row in the register file and is read back in the next consecutive cycle. The data read from the register file is compared with the expected data and mismatches are asserted.

4.7 Summary

The design of a 32 x 32 memory cell array was laid out using AMI 0.6 micron technology and simulated using Spectre and IRSIM. The design was found to meet the following specifications:

- Read Noise Margin of Memory cell (NM_{write}) = ∞
- Write Noise Margin of memory cell (NM_{write}) = 0.81V.
- Access time of the register file t_{acc} = 1.96ns.
- Pre-charge duration of bit lines = 0.6ns.
- Decoder Delay = 0.6ns.
- Area of SRAM cell = 1198.8 μm^2.
- Area of memory cell array = 1.294mm^2

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The objective of this thesis is to design a 32-bit 32-word 10-read/write port register file using AMI 0.6 μm technology. The design issues involved in this design include reliability of the memory cell, access time of the register file and area occupied by the memory array. Arriving at the right tradeoffs to obtain acceptable performance benefits for each of these design parameters is a major challenge.

The following design approach was followed to design the register file: Simulation studies were done to arrive at an optimum width for the p-transistor in the sense amplifier. The width of the p-transistor was fixed to 14.4μm and the corresponding switching voltage of the sense amplifier was approximately 3.5V. Simulations were done to determine the time taken to discharge the bit line capacitance from 5V (V_{dd}) to 3.5V (switching voltage of sense amplifier). This simulation result was used to find the width of the transistor M12 of the SRAM cell.

The layout of the SRAM cell was made using the Cadence Virtuoso Layout editor. Behavioral code for the address decoder was written in Verilog, synthesized and its worst case delay was found to be 0.6ns. The memory cell was simulated with the Spectre simulator. The write noise margin was found to be 0.81V and the access time of the register file was 1.96ns. The memory cell array occupies an area of 1.294mm^2. The memory cell array was simulated using the IRSIM simulator and was checked for correct functionality.

The following are the strengths of this register file design:

1. There is only one bit line associated with each word line. In traditional memory cell designs, two bit lines are associated with every word line. The reduction in the number of bit lines will reduce the total area occupied by the memory cell.
2. Since there are only half the number of bit lines when compared to that of traditional memory cells, the number of bit lines that needs to be pre-charged is reduced to half. This tremendously decreases the wastage of power during the pre-charge cycle.
3. During the read cycle, the bit line is not directly coupled to the inverter pair of the memory cell. Hence the read cycle will not destroy the contents of the memory cell.

5.2 Future Work

Once the architecture of the memory cell has been characterized fully, a more generic analysis using models such as the Piece-Wise Linear Transistor Model [1] could be done. Also once a complete model for the memory cell and the bit-line delays is obtained, the entire process of designing the entire register file could be automated to meet a given specification. This automation of memory cell design and layout is a hot area of research.

The access time of the register file can be improved by using single ended analog sense amplifiers like [6] instead of using inverters as the sense amplifier. The multi-port register file can be fabricated and tested on chip. This will enable us to observe the read access time and noise margin of the SRAM cell with real transistors.

REFERENCES

[1] Louis.G.Johnson, "Memory", Lecture notes for ECEN 6263, 2005, http://lgjohn.okstate.edu/6263/lectures/memory.pdf

[2] F. L. Christian Reichling, Vloker Lindenstruth, Markus Schulz, "*A quad ported SRAM.*"

[3] M. Nomura, M. Yamashina, K. Suzuki, M. Izumikawa, H. Igura, H. Abiko, K. Okabe, A. Ono, T. Nakayama, and H. Yamada, "*A 500-MHz, 0.4µm CMOS, 32-word by 32-bit 3-port register file*," 1995,Vol 151

[4] H. Kadota, S. Ozawa, K. Kawakami, and E. Ichinohe, "A new register file structure for the high-speed microprocessor," *Solid-State Circuits, IEEE Journal of*, vol. 17, pp. 892, 1982.

[5] E. S. Fetzer, M. Gibson, A. Klein, N. Calick, Z. Chengyu, E. Busta, and B. Mohammad, "A fully bypassed six-issue integer datapath and register file on the Itanium-2 microprocessor," *Solid-State Circuits, IEEE Journal of*, vol. 37, pp. 1433, 2002.

[6] C. Papaix and J. M. Daga, "*A new single ended sense amplifier for low voltage embedded EEPROM non volatile memories*," 2002,Vol 149

[7] Louis.G.Johnson, "Register File Design," Lecture notes for ECEN 6263, 2000.

[8] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd ed.

[9] A. Yulius and R. Lethin, "MAGIC/IRSIM Notes," 2001. http://www.reservoir.com/extra/ee425/roadmaps/irsim.html

APPENDIX

Appendix 1a: Behavioral code – Address Decoder

```
module decoder (
                        in ,       // 5 bit binary input
                        out ,      // 32-bit output
                        enable  // Enable for the decoder
                  );

        input [4:0] in;
        input enable;
        output [31:0] out;

        wire [31:0] out;

        assign out = (enable) ? (1 << in) : 32'b0 ;

endmodule
```

Appendix 1b: Structural code – Address Decoder

```
module decoder (in, out, enable);

        input [4:0] in;
        output [31:0] out;
        input enable;

        NOR2X1 i_8(.A(n_37), .B(n_51), .Y(out[7]));
        AND2X1 i_7(.A(n_36), .B(n_27), .Y(out[6]));
        NOR2X1 i_6(.A(n_34), .B(n_51), .Y(out[5]));
        AND2X1 i_5(.A(n_33), .B(n_27), .Y(out[4]));
        NOR2X1 i_4(.A(n_31), .B(n_51), .Y(out[3]));
        AND2X1 i_3(.A(n_30), .B(n_27), .Y(out[2]));
        NOR2X1 i_20(.A(n_28), .B(n_51), .Y(out[1]));
        AND2X1 i_1(.A(n_27), .B(n_25), .Y(out[0]));
        NAND3X1 i_33(.A(enable), .B(in[3]), .C(in[4]), .Y(n_41));
        NAND2X1 i_0(.A(in[4]), .B(n_26), .Y(n_40));
        NOR2X1 i_34(.A(in[4]), .B(n_38), .Y(n_39));
        NAND2X1 i_44(.A(in[3]), .B(enable), .Y(n_38));
        NAND3X1 i_74(.A(in[2]), .B(in[1]), .C(in[0]), .Y(n_37));
        NOR2X1 i_73(.A(in[0]), .B(n_35), .Y(n_36));
        NAND2X1 i_42(.A(in[1]), .B(in[2]), .Y(n_35));
        NAND2X1 i_72(.A(in[0]), .B(n_32), .Y(n_34));
        NOR2X1 i_71(.A(in[0]), .B(n_46), .Y(n_33));
        NOR2X1 i_40(.A(in[1]), .B(n_52), .Y(n_32));
        NAND2X1 i_70(.A(in[0]), .B(n_42), .Y(n_31));
        NOR2X1 i_69(.A(in[0]), .B(n_29), .Y(n_30));
        NAND2X1 i_3812(.A(in[1]), .B(n_52), .Y(n_29));
        NAND2X1 i_68(.A(in[0]), .B(n_24), .Y(n_28));
        NOR2X1 i_2(.A(in[4]), .B(n_49), .Y(n_27));
        NOR2X1 i_36(.A(in[3]), .B(n_53), .Y(n_26));
        NOR2X1 i_67(.A(in[0]), .B(n_45), .Y(n_25));
        NOR2X1 i_35(.A(in[1]), .B(in[2]), .Y(n_24));
        AND2X1 i_9(.A(n_39), .B(n_25), .Y(out[8]));
        NOR2X1 i_10(.A(n_28), .B(n_50), .Y(out[9]));
        NOR2X1 i_11(.A(n_43), .B(n_50), .Y(out[10]));
        NOR2X1 i_12(.A(n_31), .B(n_50), .Y(out[11]));
        NOR2X1 i_13(.A(n_44), .B(n_50), .Y(out[12]));
        NOR2X1 i_14(.A(n_34), .B(n_50), .Y(out[13]));
        NOR2X1 i_15(.A(n_48), .B(n_50), .Y(out[14]));
        NOR2X1 i_16(.A(n_37), .B(n_50), .Y(out[15]));
        NOR2X1 i_17(.A(n_40), .B(n_47), .Y(out[16]));
        NOR2X1 i_18(.A(n_28), .B(n_40), .Y(out[17]));
        NOR2X1 i_19(.A(n_40), .B(n_43), .Y(out[18]));
        NOR2X1 i_201(.A(n_31), .B(n_40), .Y(out[19]));
        NOR2X1 i_21(.A(n_40), .B(n_44), .Y(out[20]));
        NOR2X1 i_22(.A(n_34), .B(n_40), .Y(out[21]));
        NOR2X1 i_23(.A(n_40), .B(n_48), .Y(out[22]));
        NOR2X1 i_24(.A(n_37), .B(n_40), .Y(out[23]));
        NOR2X1 i_25(.A(n_41), .B(n_47), .Y(out[24]));
        NOR2X1 i_26(.A(n_41), .B(n_28), .Y(out[25]));
        NOR2X1 i_27(.A(n_41), .B(n_43), .Y(out[26]));
        NOR2X1 i_28(.A(n_41), .B(n_31), .Y(out[27]));
        NOR2X1 i_29(.A(n_41), .B(n_44), .Y(out[28]));
        NOR2X1 i_30(.A(n_41), .B(n_34), .Y(out[29]));
        NOR2X1 i_31(.A(n_41), .B(n_48), .Y(out[30]));
        NOR2X1 i_32(.A(n_41), .B(n_37), .Y(out[31]));
        INVX1 i_122(.A(n_29), .Y(n_42));
        INVX1 i_123(.A(n_30), .Y(n_43));
        INVX1 i_124(.A(n_33), .Y(n_44));
        INVX1 i_125(.A(n_24), .Y(n_45));
        INVX1 i_126(.A(n_32), .Y(n_46));
        INVX1 i_127(.A(n_25), .Y(n_47));
        INVX1 i_128(.A(n_36), .Y(n_48));
        INVX1 i_129(.A(n_26), .Y(n_49));
        INVX1 i_130(.A(n_39), .Y(n_50));
        INVX1 i_131(.A(n_27), .Y(n_51));
        INVX1 i_132(.A(in[2]), .Y(n_52));
        INVX1 i_133(.A(enable), .Y(n_53));

endmodule
```

Appendix 1c: Test bench for testing Address Decoder

```
`timescale 1ns/10ps
module decoder_test();
reg [4:0] in1[31:0], in;
reg [31:0] out1[31:0];
reg enable;
wire [31:0] out;
integer i, error;

decoder dut (in, out, enable);

initial begin

enable = 1;

in1[0]  = 5'b00000;
in1[1]  = 5'b00001;
in1[2]  = 5'b00010;
in1[3]  = 5'b00011;
in1[4]  = 5'b00100;
in1[5]  = 5'b00101;
in1[6]  = 5'b00110;
in1[7]  = 5'b00111;
in1[8]  = 5'b01000;
in1[9]  = 5'b01001;
in1[10] = 5'b01010;
in1[11] = 5'b01011;
in1[12] = 5'b01100;
in1[13] = 5'b01101;
in1[14] = 5'b01110;
in1[15] = 5'b01111;
in1[16] = 5'b10000;
in1[17] = 5'b10001;
in1[18] = 5'b10010;
in1[19] = 5'b10011;
in1[20] = 5'b10100;
in1[21] = 5'b10101;
in1[22] = 5'b10110;
in1[23] = 5'b10111;
in1[24] = 5'b11000;
in1[25] = 5'b11001;
in1[26] = 5'b11010;
in1[27] = 5'b11011;
```

```
in1[28] = 5'b11100;
in1[29] = 5'b11101;
in1[30] = 5'b11110;
in1[31] = 5'b11111;

out1[0]   = 32'b00000000000000000000000000000001;
out1[1]   = 32'b00000000000000000000000000000010;
out1[2]   = 32'b00000000000000000000000000000100;
out1[3]   = 32'b00000000000000000000000000001000;
out1[4]   = 32'b00000000000000000000000000010000;
out1[5]   = 32'b00000000000000000000000000100000;
out1[6]   = 32'b00000000000000000000000001000000;
out1[7]   = 32'b00000000000000000000000010000000;
out1[8]   = 32'b00000000000000000000000100000000;
out1[9]   = 32'b00000000000000000000001000000000;
out1[10]  = 32'b00000000000000000000010000000000;
out1[11]  = 32'b00000000000000000000100000000000;
out1[12]  = 32'b00000000000000000001000000000000;
out1[13]  = 32'b00000000000000000010000000000000;
out1[14]  = 32'b00000000000000000100000000000000;
out1[15]  = 32'b00000000000000001000000000000000;
out1[16]  = 32'b00000000000000010000000000000000;
out1[17]  = 32'b00000000000000100000000000000000;
out1[18]  = 32'b00000000000001000000000000000000;
out1[19]  = 32'b00000000000010000000000000000000;
out1[20]  = 32'b00000000000100000000000000000000;
out1[21]  = 32'b00000000001000000000000000000000;
out1[22]  = 32'b00000000010000000000000000000000;
out1[23]  = 32'b00000000100000000000000000000000;
out1[24]  = 32'b00000001000000000000000000000000;
out1[25]  = 32'b00000010000000000000000000000000;
out1[26]  = 32'b00000100000000000000000000000000;
out1[27]  = 32'b00001000000000000000000000000000;
out1[28]  = 32'b00010000000000000000000000000000;
out1[29]  = 32'b00100000000000000000000000000000;
out1[30]  = 32'b01000000000000000000000000000000;
out1[31]  = 32'b10000000000000000000000000000000;

$timeformat(-9,1,"ns",12);

end

initial begin

        error = 0;
        for(i=0; i<=31; i=i+1) begin
```

```
            in = in1[i];


            #0.58
            $display("TIME = %t",$realtime);
            $display ("    Test %d", i+1);
            $display ("    ****************\n");
            $display ("    Decoder Input =    %b", in);
            $display ("    Decoder Enable  =    %b", enable);
            $display ("    Your Output     =         %b", out);
            $display ("    Correct Output =         %b", out1[i]);

            if(out1[i] !== out) begin
            $display ("    -------------ERROR. A Mismatch Has Occured-----------");
            error = error + 1;
           end


           end

           if ( error !== 0) begin
           $display("-- SIMULATION UNSUCCESFUL - MISMATCHES HAVE
           OCCURED -");
           $display(" No. Of Errors = %d\n\n\n", error);
           end

           if ( error == 0) begin
           $display("--YOU DID IT!! SIMULATION SUCCESFULLY FINISHED---- \n");
           end

end

initial begin
$shm_open("decoder.shm");
$shm_probe ("AS");
end
endmodule
```

Appendix 1d: Simulation results of Address Decoder

```
TIME =       0.6ns
  Test   1
  ******
   Decoder Input          =   00000
   Decoder Enable         =   1
   Your Output            =  00000000000000000000000000000001
   Correct Output         =  00000000000000000000000000000001

TIME =       1.2ns
   Test   2
   ******
   Decoder Input          =   00001
   Decoder Enable         =   1
   Your Output            =  00000000000000000000000000000010
   Correct Output         =  00000000000000000000000000000010

TIME =       1.7ns
   Test   3
   ******
   Decoder Input          =   00010
   Decoder Enable         =   1
   Your Output            =   00000000000000000000000000000100
   Correct Output         =   00000000000000000000000000000100

TIME =       2.3ns
   Test   4
   ******
   Decoder Input          =   00011
   Decoder Enable         =   1
   Your Output            =   00000000000000000000000000001000
   Correct Output         =   00000000000000000000000000001000

TIME =       2.9ns
   Test   5
   ******
   Decoder Input          =   00100
   Decoder Enable         =   1
   Your Output            =   00000000000000000000000000010000
   Correct Output         =   00000000000000000000000000010000


TIME =       3.5ns
```

```
   Test   6
   ******
   Decoder Input           =   00101
   Decoder Enable          =   1
   Your Output             =  00000000000000000000000000100000
   Correct Output          =  00000000000000000000000000100000

TIME =       4.1ns
   Test   7
   ******
   Decoder Input           =   00110
   Decoder Enable          =   1
   Your Output             =   00000000000000000000000001000000
   Correct Output          =   00000000000000000000000001000000

TIME =       4.6ns
   Test   8
   ******
   Decoder Input           =   00111
   Decoder Enable          =   1
   Your Output             =   00000000000000000000000010000000
   Correct Output          =   00000000000000000000000010000000

TIME =       5.2ns
   Test   9
   ******
   Decoder Input           =   01000
   Decoder Enable          =   1
   Your Output             =   00000000000000000000000100000000
   Correct Output          =   00000000000000000000000100000000

TIME =       5.8ns
   Test   10
   *******
   Decoder Input           =   01001
   Decoder Enable          =   1
   Your Output             =   00000000000000000000001000000000
   Correct Output          =   00000000000000000000001000000000


TIME =       6.4ns
   Test   11
   *******
   Decoder Input           =   01010
   Decoder Enable          =   1
   Your Output             =   00000000000000000000010000000000
   Correct Output          =  00000000000000000000010000000000

TIME =     7.0ns
   Test   12
   *******
   Decoder Input           =  01011
   Decoder Enable          =  1
   Your Output             =  00000000000000000000100000000000
   Correct Output          =  00000000000000000000100000000000

TIME =     7.5ns
    Test   13
   *******
   Decoder Input           =  01100
   Decoder Enable          =  1
   Your Output             =  00000000000000000001000000000000
   Correct Output          =  00000000000000000001000000000000

TIME =     8.1ns
   Test   14
   *******
   Decoder Input           =  01101
   Decoder Enable          =  1
   Your Output             =  00000000000000000010000000000000
   Correct Output          =  00000000000000000010000000000000

TIME =     8.7ns
   Test   15
   *******
   Decoder Input           =  01110
   Decoder Enable          =  1
   Your Output             =  00000000000000000100000000000000
   Correct Output          =  00000000000000000100000000000000


TIME =     9.3ns
   Test   16
   *******
   Decoder Input           =  01111
   Decoder Enable          =  1
   Your Output             =  00000000000000001000000000000000
   Correct Output          =  00000000000000001000000000000000

TIME =     9.9ns
   Test   17
   *******
   Decoder Input          =  10000
   Decoder Enable         =  1
   Your Output            =  00000000000000010000000000000000
   Correct Output         =  00000000000000010000000000000000

TIME =      10.4ns
   Test   18
   *******
   Decoder Input          =  10001
   Decoder Enable         =  1
   Your Output            =  00000000000000100000000000000000
   Correct Output         =  00000000000000100000000000000000

TIME =      11.0ns
   Test   19
   *******
   Decoder Input          =  10010
   Decoder Enable         =  1
   Your Output            =  00000000000001000000000000000000
   Correct Output         =  00000000000001000000000000000000

TIME =      11.6ns
   Test   20
   *******
   Decoder Input          =  10011
   Decoder Enable         =  1
   Your Output            =  00000000000010000000000000000000
   Correct Output         =  00000000000010000000000000000000


TIME =      12.2ns
   Test   21
   *******
   Decoder Input          =  10100
   Decoder Enable         =  1
   Your Output            =  00000000000100000000000000000000
   Correct Output         =  00000000000100000000000000000000

TIME =      12.8ns
   Test   22
   *******
   Decoder Input          =  10101
   Decoder Enable         =  1
   Your Output            =  00000000001000000000000000000000
   Correct Output         =  00000000001000000000000000000000
```

```
TIME =      13.3ns
   Test   23
   *******
   Decoder Input          =   10110
   Decoder Enable         =   1
   Your Output            =   00000000010000000000000000000000
   Correct Output         =   00000000010000000000000000000000

TIME =      13.9ns
   Test   24
   *******
   Decoder Input          =   10111
   Decoder Enable         =   1
   Your Output            =   00000000100000000000000000000000
   Correct Output         =   00000000100000000000000000000000

TIME =      14.5ns
   Test   25
   *******
   Decoder Input          =   11000
   Decoder Enable         =   1
   Your Output            =   00000001000000000000000000000000
   Correct Output         =   00000001000000000000000000000000


TIME =      15.1ns
   Test   26
   *******
   Decoder Input          =   11001
   Decoder Enable         =   1
   Your Output            =   00000010000000000000000000000000
   Correct Output         =   00000010000000000000000000000000

TIME =      15.7ns
   Test   27
   *******
   Decoder Input          =   11010
   Decoder Enable         =   1
   Your Output            =   00000100000000000000000000000000
   Correct Output         =   00000100000000000000000000000000

TIME =      16.2ns
   Test   28
   *******
   Decoder Input          =   11011
   Decoder Enable         =   1
   Your Output              =  00001000000000000000000000000000
   Correct Output           =  00001000000000000000000000000000

TIME =     16.8ns
   Test  29
   *******
   Decoder Input            =  11100
   Decoder Enable           =  1
   Your Output              =  00010000000000000000000000000000
   Correct Output           =  00010000000000000000000000000000

TIME =     17.4ns
   Test  30
   *******
   Decoder Input            =  11101
   Decoder Enable           =  1
   Your Output              =  00100000000000000000000000000000
   Correct Output           =  00100000000000000000000000000000


TIME =     18.0ns
   Test  31
   *******
   Decoder Input            =  11110
   Decoder Enable           =  1
   Your Output              =  01000000000000000000000000000000
   Correct Output           =  01000000000000000000000000000000

TIME =     18.6ns
   Test  32
   *******
   Decoder Input            =  11111
   Decoder Enable           =  1
   Your Output              =  10000000000000000000000000000000
   Correct Output           =  10000000000000000000000000000000

---------YOU DID IT!! SIMULATION SUCCESFULLY FINISHED----------

7 warnings
0 simulation events (use +profile or +listcounts option to count) + 1201 accelerated
events + 62 timing check events
CPU time: 0.7 secs to compile + 0.1 secs to link + 0.2 secs in simulation
End of Tool:   VERILOG-XL        05.30.007-s   Mar 29, 2006  13:31:45
```

Appendix 2: Spectre file used to test the 10 port SRAM

```
simulator lang=spectre

model ami06N bsim3v3 type = n
+version = 3.1          tnom    = 27             tox     = 1.41E-8
+xj      = 1.5E-7       nch     = 1.7E17         vth0    = 0.7086
+k1      = 0.8354582    k2      = -0.088431      k3      = 41.4403818
+k3b     = -14          w0      = 6.480766E-7    nlx     = 1E-10
+dvt0w   = 0            dvt1w   = 5.3E6          dvt2w   = -0.032
+dvt0    = 3.6139113    dvt1    = 0.3795745      dvt2    = -0.1399976
+u0      = 533.6953445  ua      = 7.558023E-10   ub      = 1.181167E-18
+uc      = 2.582756E-11 vsat    = 1.300981E5     a0      = 0.5292985
+ags     = 0.1463715    b0      = 1.283336E-6    b1      = 1.408099E-6
+keta    = -0.0173166   a1      = 0              a2      = 1
+rdsw    = 2.268366E3   prwg    = -1E-3          prwb    = 6.320549E-5
+wr      = 1            wint    = 2.043512E-7    lint    = 3.034496E-8
+xl      = 0            xw      = 0              dwg     = -1.446149E-8
+dwb     = 2.077539E-8  voff    = -0.1137226     nfactor = 1.2880596
+cit     = 0            cdsc    = 1.506004E-4    cdscd   = 0
+cdscb   = 0            eta0    = 3.815372E-4    etab    = -1.029178E-3
+dsub    = 2.173055E-4  pclm    = 0.6171774      pdiblc1 = 0.185986
+pdiblc2 = 3.473187E-3  pdiblcb = -1E-3          drout   = 0.4037723
+pscbe1  = 5.998012E9   pscbe2  = 3.788068E-8    pvag    = 0.012927
+delta   = 0.01         mobmod  = 1              prt     = 0
+ute     = -1.5         kt1     = -0.11          kt1l    = 0
+kt2     = 0.022        ua1     = 4.31E-9        ub1     = -7.61E-18
+uc1     = -5.6E-11     at      = 3.3E4          wl      = 0
+wln     = 1            ww      = 0              wwn     = 1
+wwl     = 0            ll      = 0              lln     = 1
+lw      = 0            lwn     = 1              lwl     = 0
+capmod  = 2            xpart   = 0.4            cgdo    = 1.99E-10
+cgso    = 1.99E-10     cgbo    = 0              cj      = 4.233802E-4
+pb      = 0.9899238    mj      = 0.4495859      cjsw    = 3.825632E-10
+pbsw    = 0.1082556    mjsw    = 0.1083618      pvth0   = 0.0212852
+prdsw   = -16.1546703  pk2     = 0.0253069      wketa   = 0.0188633
+lketa   = 0.0204965

model ami06P bsim3v3 type = p
+version = 3.1          tnom    = 27             tox     = 1.41E-8
+xj      = 1.5E-7       nch     = 1.7E17         vth0    = -0.9179952
+k1      = 0.5575604    k2      = 0.010265       k3      = 14.0655075
+k3b     = -2.3032921   w0      = 1.147829E-6    nlx     = 1.114768E-10
+dvt0w   = 0            dvt1w   = 5.3E6          dvt2w   = -0.032
```

```
+dvt0    = 2.2896412        dvt1    = 0.5213085        dvt2    = -0.1337987
+u0      = 202.4540953      ua      = 2.290194E-9      ub      = 9.779742E-19
+uc      = -3.69771E-11     vsat    = 1.307891E5       a0      = 0.8356881
+ags     = 0.1568774        b0      = 2.365956E-6      b1      = 5E-6
+keta    = -5.769328E-3     a1      = 0                a2      = 1
+rdsw    = 2.746814E3       prwg    = 2.34865E-3       prwb    = 0.0172298
+wr      = 1                wint    = 2.586255E-7      lint    = 7.205014E-8
+xl      = 0                xw      = 0                dwg     = -2.133054E-8
+dwb     = 9.857534E-9      voff    = -0.0837499       nfactor = 1.2415529
+cit     = 0                cdsc    = 4.363744E-4      cdscd   = 0
+cdscb   = 0                eta0    = 0.11276          etab    = -2.9484E-3
+dsub    = 0.3389402        pclm    = 4.9847806        pdiblc1 = 2.481735E-5
+pdiblc2 = 0.01             pdiblcb = 0                drout   = 0.9975107
+pscbe1  = 3.497872E9       pscbe2  = 4.974352E-9      pvag    = 10.9914549
+delta   = 0.01             mobmod  = 1                prt     = 0
+ute     = -1.5             kt1     = -0.11            kt1l    = 0
+kt2     = 0.022            ua1     = 4.31E-9          ub1     = -7.61E-18
+uc1     = -5.6E-11         at      = 3.3E4            wl      = 0
+wln     = 1                ww      = 0                wwn     = 1
+wwl     = 0                ll      = 0                lln     = 1
+lw      = 0                lwn     = 1                lwl     = 0
+capmod  = 2                xpart   = 0.4              cgdo    = 2.4E-10
+cgso    = 2.4E-10          cgbo    = 0                cj      = 7.273568E-4
+pb      = 0.9665597        mj      = 0.4959837        cjsw    = 3.114708E-10
+pbsw    = 0.99             mjsw    = 0.2653654        pvth0   = 9.420541E-3
+prdsw   = -231.2571566     pk2     = 1.396684E-3      wketa   = 1.862966E-3
+lketa   = 5.728589E-3
```

```
// Library name: sramV4
// Cell name: sram
// View name: extracted
\+26 (_30 clkbar vdd! vdd!) ami06P w=1.2e-06 l=6e-07 as=1.8e-12 \
      ad=1.08e-12 ps=4.2e-06 pd=1.8e-06 m=1 region=sat
\+25 (_30 clk _29 vdd!) ami06P w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.8e-12 \
      ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+24 (data databar _31 vdd!) ami06P w=1.2e-06 l=6e-07 as=1.08e-12 \
      ad=1.8e-12 ps=1.8e-06 pd=4.2e-06 m=1 region=sat
\+23 (_31 clk vdd! vdd!) ami06P w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.08e-12 \
      ps=4.2e-06 pd=1.8e-06 m=1 region=sat
\+22 (vdd! data databar vdd!) ami06P w=1.2e-06 l=6e-07 as=1.8e-12 \
      ad=1.8e-12 ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+28 (_27 _30 vdd! vdd!) ami06P w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.8e-12 \
      ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+27 (vdd! data _30 vdd!) ami06P w=1.2e-06 l=6e-07 as=1.08e-12 ad=1.8e-12 \
        ps=1.8e-06 pd=4.2e-06 m=1 region=sat
\+13 (_27 _29 0 0) ami06N w=5.4e-06 l=6e-07 as=8.1e-12 ad=8.1e-12 \
        ps=8.4e-06 pd=8.4e-06 m=1 region=sat
\+12 (0 _29 _27 0) ami06N w=5.4e-06 l=6e-07 as=4.86e-12 ad=8.1e-12 \
        ps=1.8e-06 pd=8.4e-06 m=1 region=sat
\+11 (_27 _29 0 0) ami06N w=5.4e-06 l=6e-07 as=8.1e-12 ad=4.86e-12 \
        ps=8.4e-06 pd=1.8e-06 m=1 region=sat
\+10 (0 _29 _27 0) ami06N w=5.4e-06 l=6e-07 as=8.1e-12 ad=8.1e-12 \
        ps=8.4e-06 pd=8.4e-06 m=1 region=sat
\+18 (_29 data 0 0) ami06N w=1.8e-06 l=6e-07 as=2.7e-12 ad=2.7e-12 \
        ps=4.8e-06 pd=4.8e-06 m=1 region=sat
\+15 (0 data _29 0) ami06N w=1.8e-06 l=6e-07 as=1.53e-12 ad=2.7e-12 \
        ps=2.1e-06 pd=4.8e-06 m=1 region=sat
\+20 (data clk _27 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.8e-12 \
        ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+17 (_27 word7 bit7 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.8e-12 \
        ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+16 (_27 word4 bit4 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.8e-12 \
        ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+8 (_27 word3 bit3 0) ami06N w=1.2e-06 l=6e-07 as=2.16e-12 ad=1.8e-12 \
        ps=4.8e-06 pd=4.2e-06 m=1 region=sat
\+4 (_27 word2 bit2 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.8e-12 \
        ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+21 (bit5 word5 _27 0) ami06N w=1.2e-06 l=6e-07 as=1.08e-12 ad=1.8e-12 \
        ps=1.8e-06 pd=4.2e-06 m=1 region=sat
\+19 (_27 word6 bit6 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.08e-12 \
        ps=4.2e-06 pd=1.8e-06 m=1 region=sat
\+14 (_29 clk 0 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.53e-12 \
        ps=4.2e-06 pd=2.1e-06 m=1 region=sat
\+9 (_30 clkbar _29 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.8e-12 \
        ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+7 (data databar _28 0) ami06N w=1.2e-06 l=6e-07 as=1.08e-12 ad=1.8e-12 \
        ps=1.8e-06 pd=4.2e-06 m=1 region=sat
\+6 (bit9 word9 _27 0) ami06N w=1.2e-06 l=6e-07 as=1.08e-12 ad=1.8e-12 \
        ps=1.8e-06 pd=4.2e-06 m=1 region=sat
\+5 (_28 clkbar 0 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.08e-12 \
        ps=4.2e-06 pd=1.8e-06 m=1 region=sat
\+3 (_27 word8 bit8 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.08e-12 \
        ps=4.2e-06 pd=1.8e-06 m=1 region=sat
\+2 (bit1 word1 _27 0) ami06N w=1.2e-06 l=6e-07 as=1.08e-12 ad=1.8e-12 \
        ps=1.8e-06 pd=4.2e-06 m=1 region=sat
\+1 (0 data databar 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.8e-12 \
        ps=4.2e-06 pd=4.2e-06 m=1 region=sat
\+0 (_27 word0 bit0 0) ami06N w=1.2e-06 l=6e-07 as=1.8e-12 ad=1.08e-12 \
        ps=4.2e-06 pd=1.8e-06 m=1 region=sat
M38 (senseout9 bit9 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M36 (senseout8 bit8 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M34 (senseout7 bit7 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M32 (senseout6 bit6 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M31 (senseout5 bit5 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M44 (senseout0 bit0 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M43 (senseout1 bit1 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M42 (senseout2 bit2 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M41 (senseout3 bit3 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M40 (senseout4 bit4 vdd! vdd!) ami06P w=9*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M29 (bit1 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M28 (bit0 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M27 (bit4 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M26 (bit3 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M25 (bit8 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M24 (bit9 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M23 (bit7 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M22 (bit6 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M21 (bit5 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M20 (bit2 pre vdd! vdd!) ami06P w=13*1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M12 (bit0 clkbar writedata0 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M6 (bit4 clkbar writedata4 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M1 (bit1 clkbar writedata1 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M3 (bit2 clkbar writedata2 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M7 (bit3 clkbar writedata3 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M14 (bit8 clkbar writedata8 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M15 (bit9 clkbar writedata9 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M16 (bit7 clkbar writedata7 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M17 (bit6 clkbar writedata6 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M19 (bit5 clkbar writedata5 vdd!) ami06P w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M39 (senseout9 bit9 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M37 (senseout8 bit8 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M35 (senseout7 bit7 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M33 (senseout6 bit6 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M30 (senseout5 bit5 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M49 (senseout0 bit0 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M48 (senseout1 bit1 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M47 (senseout2 bit2 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M46 (senseout3 bit3 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M45 (senseout4 bit4 0 gnd!) ami06N w=1.2u l=600n as=2.25e-12 ad=2.25e-12 \
        ps=6u pd=6u m=1 region=sat
M8 (writedata0 clk bit0 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M4 (writedata4 clk bit4 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M5 (writedata3 clk bit3 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M0 (writedata1 clk bit1 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M2 (writedata2 clk bit2 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M9 (writedata8 clk bit8 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M10 (writedata9 clk bit9 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M11 (writedata7 clk bit7 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M13 (writedata6 clk bit6 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat
M18 (writedata5 clk bit5 gnd!) ami06N w=10*1.2u l=600n as=2.25e-12 \
        ad=2.25e-12 ps=6u pd=6u m=1 region=sat



// simulate load cap on bit lines
crbit0 (bit0 0) capacitor c=200f
crbit1 (bit1 0) capacitor c=200f
crbit2 (bit2 0) capacitor c=200f
crbit3 (bit3 0) capacitor c=200f
crbit4 (bit4 0) capacitor c=200f
crbit5 (bit5 0) capacitor c=200f
crbit6 (bit6 0) capacitor c=200f
crbit7 (bit7 0) capacitor c=200f
crbit8 (bit8 0) capacitor c=200f
crbit9 (bit9 0) capacitor c=200f

// simulate load cap on word lines
crword0 (word0 0) capacitor c=100f
crword1 (word1 0) capacitor c=100f
crword2 (word2 0) capacitor c=100f
crword3 (word3 0) capacitor c=100f
crword4 (word4 0) capacitor c=100f
crword5 (word5 0) capacitor c=100f
crword6 (word6 0) capacitor c=100f
crword7 (word7 0) capacitor c=100f
crword8 (word8 0) capacitor c=100f
crword9 (word9 0) capacitor c=100f


//percharge

vpre(pre 0) vsource dc=5.0 type=pulse val0=0.0 val1=5.0\
  period=10n rise=0.1n fall=0.1n width=9.4n delay=-4.5n

//power supplies

VPWR(vdd! 0) vsource dc=5.0
VGND(gnd! 0) vsource dc=0.0

//clock and clockbar signals
```

```
vclk(clk 0) vsource dc=5.0 type=pulse val0=0.0 val1=5.0\
  period=10n rise=0.1n fall=0.1n width=4.9n

vclkbar(clkbar 0) vsource dc=5.0 type=pulse val0=5.0 val1=0.0\
  period=10n rise=0.1n fall=0.1n width=4.9n

// write 1 on bit0, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 0 on bit1, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 1 on bit2, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 0 on bit3, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 1 on bit4, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 0 on bit5, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 1 on bit6, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 0 on bit7, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 1 on bit8, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9
// write 0 on bit9, read from bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, bit9

vword0 (word0 0) vsource type=pwl wave=[0n 0 0.1n 5 9.9n 5 10n 0 15n 0 \
15.1n 5 19.9n 5 20n 0 25n 0 25.1n 5 29.9n 5 30n 0 35n 0 35.1n 5 39.9n 5 \
40n 0 45n 0 45.1n 5 49.9n 5 50n 0 55n 0 55.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 69.9n 5 70n 0 75n 0 75.1n 5 79.9n 5 \
80n 0 85n 0 85.1n 5 89.9n 5 90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

vword1 (word1 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 \
19.9n 5 20n 0 25n 0 25.1n 5 29.9n 5 30n 0 35n 0 35.1n 5 39.9n 5 \
40n 0 45n 0 45.1n 5 49.9n 5 50n 0 55n 0 55.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 69.9n 5 70n 0 75n 0 75.1n 5 79.9n 5 \
80n 0 85n 0 85.1n 5 89.9n 5 90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

vword2 (word2 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 9.9n 5 10n 0\
15n 0 15.1n 5 29.9n 5 30n 0 35n 0 35.1n 5 39.9n 5 \
40n 0 45n 0 45.1n 5 49.9n 5 50n 0 55n 0 55.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 69.9n 5 70n 0 75n 0 75.1n 5 79.9n 5 \
80n 0 85n 0 85.1n 5 89.9n 5 90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

vword3 (word3 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 9.9n 5 10n 0\
15n 0 15.1n 5 19.9n 5 20n 0 25n 0 25.1n 5 39.9n 5 \
40n 0 45n 0 45.1n 5 49.9n 5 50n 0 55n 0 55.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 69.9n 5 70n 0 75n 0 75.1n 5 79.9n 5 \
80n 0 85n 0 85.1n 5 89.9n 5 90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n
```

```
vword4 (word4 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 9.9n 5 10n 0\
15n 0 15.1n 5 19.9n 5 20n 0 25n 0 25.1n 5 29.9n 5 30n 0 35n 0 35.1n 5\
49.9n 5 50n 0 55n 0 55.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 69.9n 5 70n 0 75n 0 75.1n 5 79.9n 5 \
80n 0 85n 0 85.1n 5 89.9n 5 90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

vword5 (word5 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 9.9n 5 10n 0\
15n 0 15.1n 5 19.9n 5 20n 0 25n 0 25.1n 5 29.9n 5 30n 0 35n 0 35.1n 5\
39.9n 5 40n 0 45n 0 45.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 69.9n 5 70n 0 75n 0 75.1n 5 79.9n 5 \
80n 0 85n 0 85.1n 5 89.9n 5 90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

vword6 (word6 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 9.9n 5 10n 0\
15n 0 15.1n 5 19.9n 5 20n 0 25n 0 25.1n 5 29.9n 5 30n 0 35n 0 35.1n 5\
39.9n 5 40n 0 45n 0 45.1n 5 49.9n 5 50n 0 55n 0 55.1n 5 69.9n 5 \
70n 0 75n 0 75.1n 5 79.9n 5 \
80n 0 85n 0 85.1n 5 89.9n 5 90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

vword7 (word7 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 9.9n 5 10n 0\
15n 0 15.1n 5 19.9n 5 20n 0 25n 0 25.1n 5 29.9n 5 30n 0 35n 0 35.1n 5\
39.9n 5 40n 0 45n 0 45.1n 5 49.9n 5 50n 0 55n 0 55.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 79.9n 5 \
80n 0 85n 0 85.1n 5 89.9n 5 90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

vword8 (word8 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 9.9n 5 10n 0\
15n 0 15.1n 5 19.9n 5 20n 0 25n 0 25.1n 5 29.9n 5 30n 0 35n 0 35.1n 5\
39.9n 5 40n 0 45n 0 45.1n 5 49.9n 5 50n 0 55n 0 55.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 69.9n 5 70n 0 75n 0 75.1n 5 89.9n 5 \
90n 0 95n 0 95.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

vword9 (word9 0) vsource type=pwl wave=[0n 0 5n 0 5.1n 5 9.9n 5 10n 0\
15n 0 15.1n 5 19.9n 5 20n 0 25n 0 25.1n 5 29.9n 5 30n 0 35n 0 35.1n 5\
39.9n 5 40n 0 45n 0 45.1n 5 49.9n 5 50n 0 55n 0 55.1n 5 59.9n 5 \
60n 0 65n 0 65.1n 5 69.9n 5 70n 0 75n 0 75.1n 5 79.9n 5 80n 0 85n 0 \
85.1n 5 99.9n 5 100n 0] \
pwlperiod=100n

//bit lines

vwbit0 (writedata0 0) vsource type=pwl wave=[0n 0 0.1n 5 5n    5 5.1n 0] \
pwlperiod=100n
```

```
vwbit1 (writedata1 0) vsource type=pwl wave=[0n 5 10n  5 10.1n 0 15n  0 15.1n 5] \
pwlperiod=100n

vwbit2 (writedata2 0) vsource type=pwl wave=[0n 0 20n  0 20.1n 5 25n  5 25.1n 0] \
pwlperiod=100n

vwbit3 (writedata3 0) vsource type=pwl wave=[0n 5 30n  5 30.1n 0 35n  0 35.1n 5] \
pwlperiod=100n

vwbit4 (writedata4 0) vsource type=pwl wave=[0n 0 40n  0 40.1n 5 45n  5 45.1n 0] \
pwlperiod=100n

vwbit5 (writedata5 0) vsource type=pwl wave=[0n 5 50n  5 50.1n 0 55n  0 55.1n 5] \
pwlperiod=100n

vwbit6 (writedata6 0) vsource type=pwl wave=[0n 0 60n  0 60.1n 5 65n  5 65.1n 0] \
pwlperiod=100n

vwbit7 (writedata7 0) vsource type=pwl wave=[0n 5 70n  5 70.1n 0 75n  0 75.1n 5] \
pwlperiod=100n

vwbit8 (writedata8 0) vsource type=pwl wave=[0n 0 80n  0 80.1n 5 85n  5 85.1n 0] \
pwlperiod=100n

vwbit9 (writedata9 0) vsource type=pwl wave=[0n 5 90n  5 90.1n 0 95n  0 95.1n 5] \
pwlperiod=100n

//initial conditions

ic data=0 databar=5 bit0=5 bit1=5 bit2=5 bit3=5 bit4=5 bit5=5

sram tran stop=101n

save data databar bit0 bit1 bit2 bit3 bit4 bit5 bit6 bit7 bit8 bit9 pre
save word0 word1 word2 word3 word4 word5 word6 word7 word8 word9 clk clkbar
save writedata0 writedata1 writedata2 writedata3 writedata4 writedata5 writedata6
save writedata7 writedata8 writedata9
save senseout0 senseout1 senseout2 senseout3 senseout4 senseout5 senseout6 senseout7
save senseout8 senseout9
```

Appendix 3a: IRSIM File used to test the memory cell array

```
vector bit0v bit0_{31:0}
vector bit1v bit1_{31:0}
vector bit2v bit2_{31:0}
vector bit3v bit3_{31:0}
vector bit4v bit4_{31:0}
vector bit5v bit5_{31:0}
vector bit6v bit6_{31:0}
vector bit7v bit7_{31:0}
vector bit8v bit8_{31:0}
vector bit9v bit9_{31:0}

vector word0v word0_{31:0}
vector word1v word1_{31:0}
vector word2v word2_{31:0}
vector word3v word3_{31:0}
vector word4v word4_{31:0}
vector word5v word5_{31:0}
vector word6v word6_{31:0}
vector word7v word7_{31:0}
vector word8v word8_{31:0}
vector word9v word9_{31:0}

w word0v word1v word2v word3v word4v word5v word6v word7v word8v word9v
w bit0v bit1v bit2v bit3v bit4v bit5v bit6v bit7v bit8v bit9v
w clk clkbar


| TEST 1
| write 00000000000000000000000000000000

h clk
l clkbar

set word0v  00000000000000000000000000000001
set word1v  00000000000000000000000000000010
set word2v  00000000000000000000000000000100
set word3v  00000000000000000000000000001000
set word4v  00000000000000000000000000010000
set word5v  00000000000000000000000000100000
set word6v  00000000000000000000000001000000
set word7v  00000000000000000000000010000000
set word8v  00000000000000000000000100000000
set word9v  00000000000000000000001000000000
set bit0v  00000000000000000000000000000000
set bit1v  00000000000000000000000000000000
set bit2v  00000000000000000000000000000000
set bit3v  00000000000000000000000000000000
set bit4v  00000000000000000000000000000000
set bit5v  00000000000000000000000000000000
set bit6v  00000000000000000000000000000000
set bit7v  00000000000000000000000000000000
set bit8v  00000000000000000000000000000000
set bit9v  00000000000000000000000000000000

|Precharge bit lines

s 5
l clk
h clkbar

set word0v  00000000000000000000000000000000
set word1v  00000000000000000000000000000000
set word2v  00000000000000000000000000000000
set word3v  00000000000000000000000000000000
set word4v  00000000000000000000000000000000
set word5v  00000000000000000000000000000000
set word6v  00000000000000000000000000000000
set word7v  00000000000000000000000000000000
set word8v  00000000000000000000000000000000
set word9v  00000000000000000000000000000000

set bit0v  11111111111111111111111111111111
set bit1v  11111111111111111111111111111111
set bit2v  11111111111111111111111111111111
set bit3v  11111111111111111111111111111111
set bit4v  11111111111111111111111111111111
set bit5v  11111111111111111111111111111111
set bit6v  11111111111111111111111111111111
set bit7v  11111111111111111111111111111111
set bit8v  11111111111111111111111111111111
set bit9v  11111111111111111111111111111111


|read 00000000000000000000000000000000

s 5
l clk
h clkbar
```

```
x bit0v bit1v bit2v bit3v bit4v bit5v bit6v bit7v bit8v bit9v

set word0v  00000000000000000000000000000001
set word1v  00000000000000000000000000000010
set word2v  00000000000000000000000000000100
set word3v  00000000000000000000000000001000
set word4v  00000000000000000000000000010000
set word5v  00000000000000000000000000100000
set word6v  00000000000000000000000001000000
set word7v  00000000000000000000000010000000
set word8v  00000000000000000000000100000000
set word9v  00000000000000000000001000000000

s 5

assert bit0v  00000000000000000000000000000000
assert bit1v  00000000000000000000000000000000
assert bit2v  00000000000000000000000000000000
assert bit3v  00000000000000000000000000000000
assert bit4v  00000000000000000000000000000000
assert bit5v  00000000000000000000000000000000
assert bit6v  00000000000000000000000000000000
assert bit7v  00000000000000000000000000000000
assert bit8v  00000000000000000000000000000000
assert bit9v  00000000000000000000000000000000

| TEST 2
| write 00000000000000000000000000000000

h clk
l clkbar

set word0v  00000000000000000000010000000000
set word1v  00000000000000000000100000000000
set word2v  00000000000000000001000000000000
set word3v  00000000000000000010000000000000
set word4v  00000000000000000100000000000000
set word5v  00000000000000001000000000000000
set word6v  00000000000000010000000000000000
set word7v  00000000000000100000000000000000
set word8v  00000000000001000000000000000000
set word9v  00000000000010000000000000000000

set bit0v  00000000000000000000000000000000
set bit1v  00000000000000000000000000000000
set bit2v  00000000000000000000000000000000
set bit3v  00000000000000000000000000000000
set bit4v  00000000000000000000000000000000
set bit5v  00000000000000000000000000000000
set bit6v  00000000000000000000000000000000
set bit7v  00000000000000000000000000000000
set bit8v  00000000000000000000000000000000
set bit9v  00000000000000000000000000000000

|Precharge bit lines

s 5
l clk
h clkbar

set word0v  00000000000000000000000000000000
set word1v  00000000000000000000000000000000
set word2v  00000000000000000000000000000000
set word3v  00000000000000000000000000000000
set word4v  00000000000000000000000000000000
set word5v  00000000000000000000000000000000
set word6v  00000000000000000000000000000000
set word7v  00000000000000000000000000000000
set word8v  00000000000000000000000000000000
set word9v  00000000000000000000000000000000

set bit0v  11111111111111111111111111111111
set bit1v  11111111111111111111111111111111
set bit2v  11111111111111111111111111111111
set bit3v  11111111111111111111111111111111
set bit4v  11111111111111111111111111111111
set bit5v  11111111111111111111111111111111
set bit6v  11111111111111111111111111111111
set bit7v  11111111111111111111111111111111
set bit8v  11111111111111111111111111111111
set bit9v  11111111111111111111111111111111


|read 00000000000000000000000000000000

s 5
l clk
h clkbar

x bit0v bit1v bit2v bit3v bit4v bit5v bit6v bit7v bit8v bit9v
```

```
set word0v  00000000000000000000010000000000
set word1v  00000000000000000000100000000000
set word2v  00000000000000000001000000000000
set word3v  00000000000000000010000000000000
set word4v  00000000000000000100000000000000
set word5v  00000000000000001000000000000000
set word6v  00000000000000010000000000000000
set word7v  00000000000000100000000000000000
set word8v  00000000000001000000000000000000
set word9v  00000000000010000000000000000000

s 5

assert bit0v  00000000000000000000000000000000
assert bit1v  00000000000000000000000000000000
assert bit2v  00000000000000000000000000000000
assert bit3v  00000000000000000000000000000000
assert bit4v  00000000000000000000000000000000
assert bit5v  00000000000000000000000000000000
assert bit6v  00000000000000000000000000000000
assert bit7v  00000000000000000000000000000000
assert bit8v  00000000000000000000000000000000
assert bit9v  00000000000000000000000000000000

| TEST 3
| write 00000000000000000000000000000000

h clk
l clkbar

set word0v  00000000000100000000000000000000
set word1v  00000000001000000000000000000000
set word2v  00000000010000000000000000000000
set word3v  00000000100000000000000000000000
set word4v  00000001000000000000000000000000
set word5v  00000010000000000000000000000000
set word6v  00000100000000000000000000000000
set word7v  00001000000000000000000000000000
set word8v  00010000000000000000000000000000
set word9v  00100000000000000000000000000000

set bit0v  00000000000000000000000000000000
set bit1v  00000000000000000000000000000000
set bit2v  00000000000000000000000000000000
set bit3v  00000000000000000000000000000000
set bit4v  00000000000000000000000000000000
set bit5v  00000000000000000000000000000000
set bit6v  00000000000000000000000000000000
set bit7v  00000000000000000000000000000000
set bit8v  00000000000000000000000000000000
set bit9v  00000000000000000000000000000000

|Precharge bit lines

s 5
l clk
h clkbar

set word0v  00000000000000000000000000000000
set word1v  00000000000000000000000000000000
set word2v  00000000000000000000000000000000
set word3v  00000000000000000000000000000000
set word4v  00000000000000000000000000000000
set word5v  00000000000000000000000000000000
set word6v  00000000000000000000000000000000
set word7v  00000000000000000000000000000000
set word8v  00000000000000000000000000000000
set word9v  00000000000000000000000000000000

set bit0v  11111111111111111111111111111111
set bit1v  11111111111111111111111111111111
set bit2v  11111111111111111111111111111111
set bit3v  11111111111111111111111111111111
set bit4v  11111111111111111111111111111111
set bit5v  11111111111111111111111111111111
set bit6v  11111111111111111111111111111111
set bit7v  11111111111111111111111111111111
set bit8v  11111111111111111111111111111111
set bit9v  11111111111111111111111111111111


|read 00000000000000000000000000000000

s 5
l clk
h clkbar

x bit0v bit1v bit2v bit3v bit4v bit5v bit6v bit7v bit8v bit9v

set word0v  00000000000100000000000000000000
set word1v  00000000001000000000000000000000
set word2v  00000000010000000000000000000000
set word3v  00000000100000000000000000000000
set word4v  00000001000000000000000000000000
set word5v  00000010000000000000000000000000
set word6v  00000100000000000000000000000000
set word7v  00001000000000000000000000000000
set word8v  00010000000000000000000000000000
set word9v  00100000000000000000000000000000

s 5

assert bit0v  00000000000000000000000000000000
assert bit1v  00000000000000000000000000000000
assert bit2v  00000000000000000000000000000000
assert bit3v  00000000000000000000000000000000
assert bit4v  00000000000000000000000000000000
assert bit5v  00000000000000000000000000000000
assert bit6v  00000000000000000000000000000000
assert bit7v  00000000000000000000000000000000
assert bit8v  00000000000000000000000000000000
assert bit9v  00000000000000000000000000000000

| TEST 4
| write 00000000000000000000000000000000

h clk
l clkbar

set word0v  01000000000000000000000000000000
set word1v  10000000000000000000000000000000
set word2v  00000000000000000000000000000000
set word3v  00000000000000000000000000000000
set word4v  00000000000000000000000000000000
set word5v  00000000000000000000000000000000
set word6v  00000000000000000000000000000000
set word7v  00000000000000000000000000000000
set word8v  00000000000000000000000000000000
set word9v  00000000000000000000000000000000

set bit0v  00000000000000000000000000000000
set bit1v  00000000000000000000000000000000
set bit2v  00000000000000000000000000000000
set bit3v  00000000000000000000000000000000
set bit4v  00000000000000000000000000000000
set bit5v  00000000000000000000000000000000
set bit6v  00000000000000000000000000000000
set bit7v  00000000000000000000000000000000
set bit8v  00000000000000000000000000000000
set bit9v  00000000000000000000000000000000

|Precharge bit lines

s 5
l clk
h clkbar

set word0v  00000000000000000000000000000000
set word1v  00000000000000000000000000000000
set word2v  00000000000000000000000000000000
set word3v  00000000000000000000000000000000
set word4v  00000000000000000000000000000000
set word5v  00000000000000000000000000000000
set word6v  00000000000000000000000000000000
set word7v  00000000000000000000000000000000
set word8v  00000000000000000000000000000000
set word9v  00000000000000000000000000000000

set bit0v  11111111111111111111111111111111
set bit1v  11111111111111111111111111111111
set bit2v  11111111111111111111111111111111
set bit3v  11111111111111111111111111111111
set bit4v  11111111111111111111111111111111
set bit5v  11111111111111111111111111111111
set bit6v  11111111111111111111111111111111
set bit7v  11111111111111111111111111111111
set bit8v  11111111111111111111111111111111
set bit9v  11111111111111111111111111111111


|read 00000000000000000000000000000000

s 5
l clk
h clkbar

x bit0v bit1v bit2v bit3v bit4v bit5v bit6v bit7v bit8v bit9v

set word0v  01000000000000000000000000000000
set word1v  10000000000000000000000000000000
set word2v  00000000000000000000000000000000
set word3v  00000000000000000000000000000000
set word4v  00000000000000000000000000000000
set word5v  00000000000000000000000000000000
set word6v  00000000000000000000000000000000
set word7v  00000000000000000000000000000000
set word8v  00000000000000000000000000000000
set word9v  00000000000000000000000000000000

s 5

assert bit0v  00000000000000000000000000000000
assert bit1v  00000000000000000000000000000000
assert bit2v  11111111111111111111111111111111
assert bit3v  11111111111111111111111111111111
assert bit4v  11111111111111111111111111111111
assert bit5v  11111111111111111111111111111111
assert bit6v  11111111111111111111111111111111
assert bit7v  11111111111111111111111111111111
assert bit8v  11111111111111111111111111111111
assert bit9v  11111111111111111111111111111111


exit
```

Appendix 3b: Simulation results of memory cell array

```
Read Design.sim lambda:0.30u format:SU
7812 nodes; transistors: n-channel=22528 p-channel=7168
parallel txtors: n-channel=6144
bit9v=00000000000000000000000000000000
bit8v=00000000000000000000000000000000
bit7v=00000000000000000000000000000000
bit6v=00000000000000000000000000000000
bit5v=00000000000000000000000000000000
bit4v=00000000000000000000000000000000
bit3v=00000000000000000000000000000000
bit2v=00000000000000000000000000000000
bit1v=00000000000000000000000000000000
bit0v=00000000000000000000000000000000
word9v=00000000000000000000001000000000
word8v=00000000000000000000000100000000
word7v=00000000000000000000000010000000
word6v=00000000000000000000000001000000
word5v=00000000000000000000000000100000
word4v=00000000000000000000000000010000
word3v=00000000000000000000000000001000
word2v=00000000000000000000000000000100
word1v=00000000000000000000000000000010
word0v=00000000000000000000000000000001 clkbar=0 clk=1
time = 5.00ns
bit9v=11111111111111111111111111111111
bit8v=11111111111111111111111111111111
bit7v=11111111111111111111111111111111
bit6v=11111111111111111111111111111111
bit5v=11111111111111111111111111111111
bit4v=11111111111111111111111111111111
bit3v=11111111111111111111111111111111
bit2v=11111111111111111111111111111111
bit1v=11111111111111111111111111111111
bit0v=11111111111111111111111111111111
word9v=00000000000000000000000000000000
word8v=00000000000000000000000000000000
word7v=00000000000000000000000000000000
word6v=00000000000000000000000000000000
word5v=00000000000000000000000000000000
word4v=00000000000000000000000000000000
word3v=00000000000000000000000000000000
word2v=00000000000000000000000000000000
```

```
word1v=00000000000000000000000000000000
word0v=00000000000000000000000000000000 clkbar=1 clk=0
time = 10.00ns
bit9v=00000000000000000000000000000000
bit8v=00000000000000000000000000000000
bit7v=00000000000000000000000000000000
bit6v=00000000000000000000000000000000
bit5v=00000000000000000000000000000000
bit4v=00000000000000000000000000000000
bit3v=00000000000000000000000000000000
bit2v=00000000000000000000000000000000
bit1v=00000000000000000000000000000000
bit0v=00000000000000000000000000000000
word9v=00000000000000000000001000000000
word8v=00000000000000000000000100000000
word7v=00000000000000000000000010000000
word6v=00000000000000000000000001000000
word5v=00000000000000000000000000100000
word4v=00000000000000000000000000010000
word3v=00000000000000000000000000001000
word2v=00000000000000000000000000000100
word1v=00000000000000000000000000000010
word0v=00000000000000000000000000000001 clkbar=1 clk=0
time = 15.00ns
bit9v=00000000000000000000000000000000
bit8v=00000000000000000000000000000000
bit7v=00000000000000000000000000000000
bit6v=00000000000000000000000000000000
bit5v=00000000000000000000000000000000
bit4v=00000000000000000000000000000000
bit3v=00000000000000000000000000000000
bit2v=00000000000000000000000000000000
bit1v=00000000000000000000000000000000
bit0v=00000000000000000000000000000000
word9v=00000000000010000000000000000000
word8v=00000000000001000000000000000000
word7v=00000000000000100000000000000000
word6v=00000000000000010000000000000000
word5v=00000000000000001000000000000000
word4v=00000000000000000100000000000000
word3v=00000000000000000010000000000000
word2v=00000000000000000001000000000000
word1v=00000000000000000000100000000000
word0v=00000000000000000000010000000000 clkbar=0 clk=1
time = 20.00ns
```

```
bit9v=11111111111111111111111111111111
bit8v=11111111111111111111111111111111
bit7v=11111111111111111111111111111111
bit6v=11111111111111111111111111111111
bit5v=11111111111111111111111111111111
bit4v=11111111111111111111111111111111
bit3v=11111111111111111111111111111111
bit2v=11111111111111111111111111111111
bit1v=11111111111111111111111111111111
bit0v=11111111111111111111111111111111
word9v=00000000000000000000000000000000
word8v=00000000000000000000000000000000
word7v=00000000000000000000000000000000
word6v=00000000000000000000000000000000
word5v=00000000000000000000000000000000
word4v=00000000000000000000000000000000
word3v=00000000000000000000000000000000
word2v=00000000000000000000000000000000
word1v=00000000000000000000000000000000
word0v=00000000000000000000000000000000 clkbar=1 clk=0
time = 25.00ns
bit9v=00000000000000000000000000000000
bit8v=00000000000000000000000000000000
bit7v=00000000000000000000000000000000
bit6v=00000000000000000000000000000000
bit5v=00000000000000000000000000000000
bit4v=00000000000000000000000000000000
bit3v=00000000000000000000000000000000
bit2v=00000000000000000000000000000000
bit1v=00000000000000000000000000000000
bit0v=00000000000000000000000000000000
word9v=00000000000010000000000000000000
word8v=00000000000001000000000000000000
word7v=00000000000000100000000000000000
word6v=00000000000000010000000000000000
word5v=00000000000000001000000000000000
word4v=00000000000000000100000000000000
word3v=00000000000000000010000000000000
word2v=00000000000000000001000000000000
word1v=00000000000000000000100000000000
word0v=00000000000000000000010000000000 clkbar=1 clk=0
time = 30.00ns
bit9v=00000000000000000000000000000000
bit8v=00000000000000000000000000000000
bit7v=00000000000000000000000000000000
bit6v=00000000000000000000000000000000
```

```
bit5v=00000000000000000000000000000000
bit4v=00000000000000000000000000000000
bit3v=00000000000000000000000000000000
bit2v=00000000000000000000000000000000
bit1v=00000000000000000000000000000000
bit0v=00000000000000000000000000000000
word9v=00100000000000000000000000000000
word8v=00010000000000000000000000000000
word7v=00001000000000000000000000000000
word6v=00000100000000000000000000000000
word5v=00000010000000000000000000000000
word4v=00000001000000000000000000000000
word3v=00000000100000000000000000000000
word2v=00000000010000000000000000000000
word1v=00000000001000000000000000000000
word0v=00000000000100000000000000000000 clkbar=0 clk=1
time = 35.00ns
bit9v=11111111111111111111111111111111
bit8v=11111111111111111111111111111111
bit7v=11111111111111111111111111111111
bit6v=11111111111111111111111111111111
bit5v=11111111111111111111111111111111
bit4v=11111111111111111111111111111111
bit3v=11111111111111111111111111111111
bit2v=11111111111111111111111111111111
bit1v=11111111111111111111111111111111
bit0v=11111111111111111111111111111111
word9v=00000000000000000000000000000000
word8v=00000000000000000000000000000000
word7v=00000000000000000000000000000000
word6v=00000000000000000000000000000000
word5v=00000000000000000000000000000000
word4v=00000000000000000000000000000000
word3v=00000000000000000000000000000000
word2v=00000000000000000000000000000000
word1v=00000000000000000000000000000000
word0v=00000000000000000000000000000000 clkbar=1 clk=0
time = 40.00ns
bit9v=00000000000000000000000000000000
bit8v=00000000000000000000000000000000
bit7v=00000000000000000000000000000000
bit6v=00000000000000000000000000000000
bit5v=00000000000000000000000000000000
bit4v=00000000000000000000000000000000
bit3v=00000000000000000000000000000000
bit2v=00000000000000000000000000000000
```

```
bit1v=00000000000000000000000000000000
bit0v=00000000000000000000000000000000
word9v=00100000000000000000000000000000
word8v=00010000000000000000000000000000
word7v=00001000000000000000000000000000
word6v=00000100000000000000000000000000
word5v=00000010000000000000000000000000
word4v=00000001000000000000000000000000
word3v=00000000100000000000000000000000
word2v=00000000010000000000000000000000
word1v=00000000001000000000000000000000
word0v=00000000000100000000000000000000 clkbar=1 clk=0
time = 45.00ns
bit9v=00000000000000000000000000000000
bit8v=00000000000000000000000000000000
bit7v=00000000000000000000000000000000
bit6v=00000000000000000000000000000000
bit5v=00000000000000000000000000000000
bit4v=00000000000000000000000000000000
bit3v=00000000000000000000000000000000
bit2v=00000000000000000000000000000000
bit1v=00000000000000000000000000000000
bit0v=00000000000000000000000000000000
word9v=00000000000000000000000000000000
word8v=00000000000000000000000000000000
word7v=00000000000000000000000000000000
word6v=00000000000000000000000000000000
word5v=00000000000000000000000000000000
word4v=00000000000000000000000000000000
word3v=00000000000000000000000000000000
word2v=00000000000000000000000000000000
word1v=10000000000000000000000000000000
word0v=01000000000000000000000000000000 clkbar=0 clk=1
time = 50.00ns
bit9v=11111111111111111111111111111111
bit8v=11111111111111111111111111111111
bit7v=11111111111111111111111111111111
bit6v=11111111111111111111111111111111
bit5v=11111111111111111111111111111111
bit4v=11111111111111111111111111111111
bit3v=11111111111111111111111111111111
bit2v=11111111111111111111111111111111
bit1v=11111111111111111111111111111111
bit0v=11111111111111111111111111111111
word9v=00000000000000000000000000000000
word8v=00000000000000000000000000000000
```

```
word7v=00000000000000000000000000000000
word6v=00000000000000000000000000000000
word5v=00000000000000000000000000000000
word4v=00000000000000000000000000000000
word3v=00000000000000000000000000000000
word2v=00000000000000000000000000000000
word1v=00000000000000000000000000000000
word0v=00000000000000000000000000000000 clkbar=1 clk=0
time = 55.00ns
bit9v=11111111111111111111111111111111
bit8v=11111111111111111111111111111111
bit7v=11111111111111111111111111111111
bit6v=11111111111111111111111111111111
bit5v=11111111111111111111111111111111
bit4v=11111111111111111111111111111111
bit3v=11111111111111111111111111111111
bit2v=11111111111111111111111111111111
bit1v=00000000000000000000000000000000
bit0v=00000000000000000000000000000000
word9v=00000000000000000000000000000000
word8v=00000000000000000000000000000000
word7v=00000000000000000000000000000000
word6v=00000000000000000000000000000000
word5v=00000000000000000000000000000000
word4v=00000000000000000000000000000000
word3v=00000000000000000000000000000000
word2v=00000000000000000000000000000000
word1v=10000000000000000000000000000000
word0v=01000000000000000000000000000000 clkbar=1 clk=0
time = 60.00ns
```

VITA

Srikanth Gopi

Candidate for the Degree of

Master of Science

Thesis: DESIGN OF 32-BIT 32-WORD 10-READ/WRITE PORT REGISTER FILE

Major Field: Electrical and Computer Engineering

Biographical:

Education: Received Bachelor of Engineering degree in Electronics and Communication Engineering from the University of Madras, India in May 2002.

Experience: Worked as a Research Assistant in ACSEL & OCLNB Laboratory, Oklahoma State University from August 2004 to September 2005.

Worked as a Research Assistant in Mixed Signal VLSI Laboratory, Oklahoma State University from November 2005 to present.

Name: Srikanth Gopi Date of Degree: May, 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: DESIGN OF 32-BIT 32-WORD 10-READ/WRITE PORT REGISTER FILE

Pages in Study: 78 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study:

The objective of this research is to design a 32-bit 32-word 10-read/write port register file in the AMI 0.6 micron process and optimize the read access time, noise margin and the area occupied by the cell. The transistors are sized appropriately so that the memory cell works correctly even in the worst case scenario. Layout of the memory cell array was made in Cadence Virtuoso Layout Editor, and simulations were run for measuring the read access time using Cadence Spectre simulator.

Findings and Conclusions:

The design was found to meet the following specifications:

- Read Noise Margin of memory cell (NM_{write}) = ∞
- Write Noise Margin of memory cell (NM_{write}) = 0.81V.
- Access time of the register file t_{acc} = 1.96ns.
- Decoder delay = 0.6ns.
- Area of SRAM cell = 1198.8 μm^2.
- Area of memory cell array = 1.294mm^2.

ADVISER'S APPROVAL: Dr. Louis G. Johnson