Volatile and NonVolatile Memory Design

Project by

Pooja Solanki-21ECUSF048

&

Monil Jariwala-21ECUBG065

Department of Electronics & Communication Engineering
Faculty of Technology
Dharmsinh Desai University,Nadiad-387001

Abstract

This document presents a comprehensive guide to designing Random Access Memory (RAM) and Read-Only Memory (ROM) modules using the Verilog hardware description language. It details the specification, definition, and implementation of these memory components, focusing on both the functional and optimization aspects necessary for effective integration into larger digital designs. The document also emphasizes the importance of verification, outlining the creation of test benches to ensure the reliability and efficiency of the memory modules. Through detailed explanations and practical examples, readers will gain a thorough understanding of how to design and implement RAM and ROM in Verilog, facilitating their application in advanced digital systems.

A significant component of this project is the visualization and analysis of the designed memory modules. We will demonstrate the Register Transfer Level (RTL) view to provide a graphical representation of the module's architecture and logic. Additionally, waveform outputs will be shown to illustrate the dynamic behavior of the memory modules during simulation, highlighting key operations such as read and write cycles for RAM and data retrieval for ROM.

Contents

CHAPTER 1

Introduction

1.1 Random Access Memory - RAM

RAM, or Random Access Memory, is a type of computer memory that is used to store data temporarily while a computer is running. It allows data to be read and written in almost the same amount of time, irrespective of the physical location of the data inside the memory. RAM can be categorized based on the number of ports available for reading and writing data such as,
1. Single-Port RAM
2. Dual-Port RAM
3. Multi-Port RAM

1.1.1 What is Single-Port RAM?

A single-port RAM (Random Access Memory) is a type of digital memory component that allows data to be read from and written to a single memory location (address) at a time. It is a simple form of memory that provides a basic storage mechanism for digital systems. Each memory location in a single-port RAM can store a fixed number of bits (usually a power of 2, such as 8, 16, 32, etc.).

During a read operation, the data stored at a specific address is retrieved.During a write operation, new data is stored at a specific address, replacing the previous data.

1.1.2 Why is it called single port ?

A single-port RAM has only one data port, which means that read and write operations cannot occur simultaneously at different addresses. If a write operation is in progress, a read operation must wait,and vice versa.

1.2 Read Only Memory - ROM

Read-Only Memory (ROM) is a type of non-volatile memory that is used to store data permanently. The data in ROM is written during the manufacturing process or through a special programming procedure and is primarily read-only during normal operation.ROM retains its data even when the power is turned off,This makes ROM a reliable storage medium for firmware, boot loaders, and other critical software components that must persist across reboots.

1.2.1 Types of ROM

1. Masked ROM (MROM)
2. Programmable ROM (PROM)
3. Erasable Programmable ROM (EPROM)
4. Electrically Erasable Programmable ROM (EEPROM)
5. Flash Memory

Based on number of Ports:

1. Single-Port ROM:
Single-port ROM has only one access port through which data can be read.
It allows only one read operation at a time.

2. Dual-Port ROM:
Dual-port ROM has two independent access ports that allow simultaneous read operations. Each port can be accessed independently, doubling the data throughput.
Dual port ROM Provides higher data throughput and allows concurrent access, making it suitable for high-performance and real-time applications.

CHAPTER 2

Design

2.1 Random Access Memory - RAM

2.1.1 Specification

We have define the size of the RAM, such as 256kb, and the type of RAM, such as a Static RAM Array. These parameters establish the fundamental characteristics of the memory module.

2.1.2 Module Definition

For Verilog module we need to specify the input and output ports, including address lines, data lines, control signals (read/write enable), and a clock signal.This defines the external interface of the RAM component.

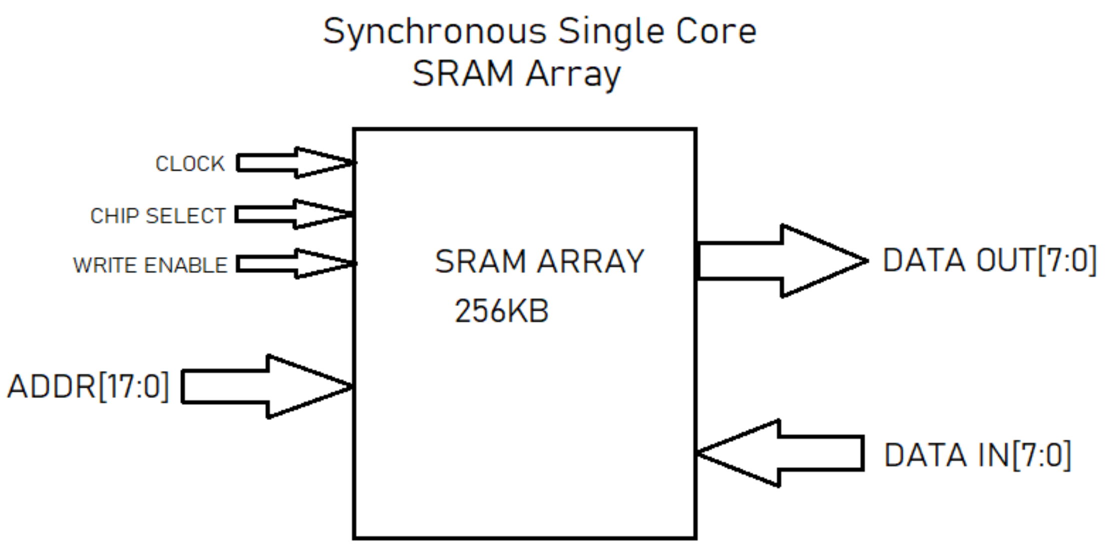


Figure 2.1: block diagram

2.1.3 Behavioral Modeling

We used Verilog constructs like always blocks to describe the read and write operations. We also Used a register array to model the memory storage.

2.2 Read Only Memory - ROM

2.2.1 Specification

We have define the size of the ROM as 1KB and the type, such as Dual-Core Read-Only Memory, Also predetermined the initial data sequence to be stored into ROM.

2.2.2 Module Definition

For a dual-port ROM (Read-Only Memory) in Verilog involves specifying the behavior and structure of the ROM, we included two separate sets of address inputs allow independent addressing of memory locations for each port and two separate data outputs provide the data read from the memory locations specified by the address inputs.Typically, a clock signal is used to synchronize read operations.Enable signals control whether the read operation for the respective port is active.

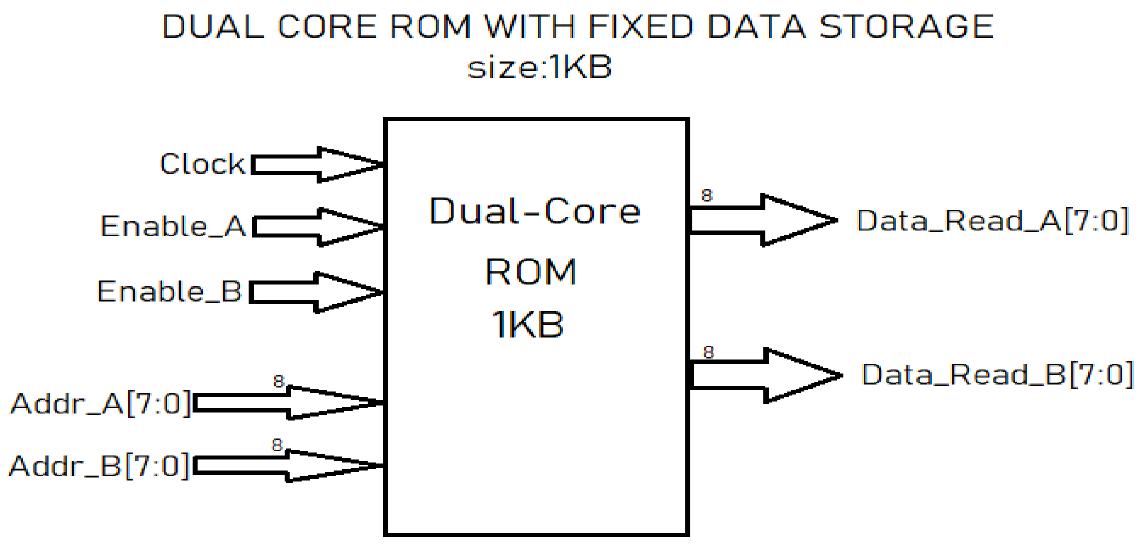


Figure 2.2: block diagram

2.2.3 Initialization

We Used initial blocks to initialize the ROM content and then Store the fixed data in a register array.We have our predefined data sequence with us.

2.3 Sense Amplifier

Sense amplifiers are integral components in memory devices, such as DRAM, SRAM, and ROM. They play a critical role in reading stored data by detecting and amplifying small voltage differences to full logic levels.

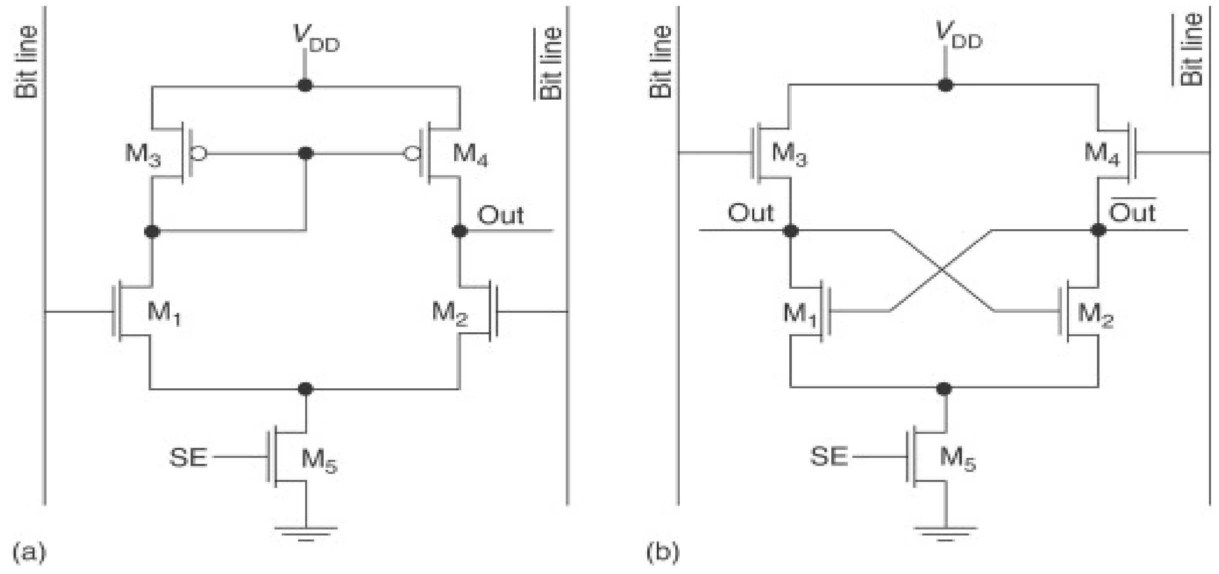


Figure 2.3: Sense amplifier

2.3.1 Function

Amplification: Sense amplifiers boost the small voltage difference between bit lines to a full logic level, necessary for reliable data readout.
Speed: They enable rapid read operations, crucial for the overall performance of the memory device.
Accuracy: Sense amplifiers ensure accurate data reading by precisely distinguishing small voltage differences.

2.3.2 Operation

Precharge Phase:
Bit lines are precharged to a reference voltage (typically Vdd/2) to ensure they start from the same voltage level.

Row Activation:
The word line (WL) is activated, connecting the memory cell to the bit lines. This creates a small voltage difference between the bit lines based on the stored data.

Sensing and Amplification:
The sense amplifier detects the small voltage difference between the bit lines and amplifies it to full logic levels (0V or Vdd).

Latching and Output:
The amplified signal is latched and output as either a logic '0' or logic '1', representing the stored data.

Restoring Phase:
Bit lines are precharged again to Vdd/2 in preparation for the next read operation.

CHAPTER 3

RTL Diagrams

1. 256 kb Synchronous RAM

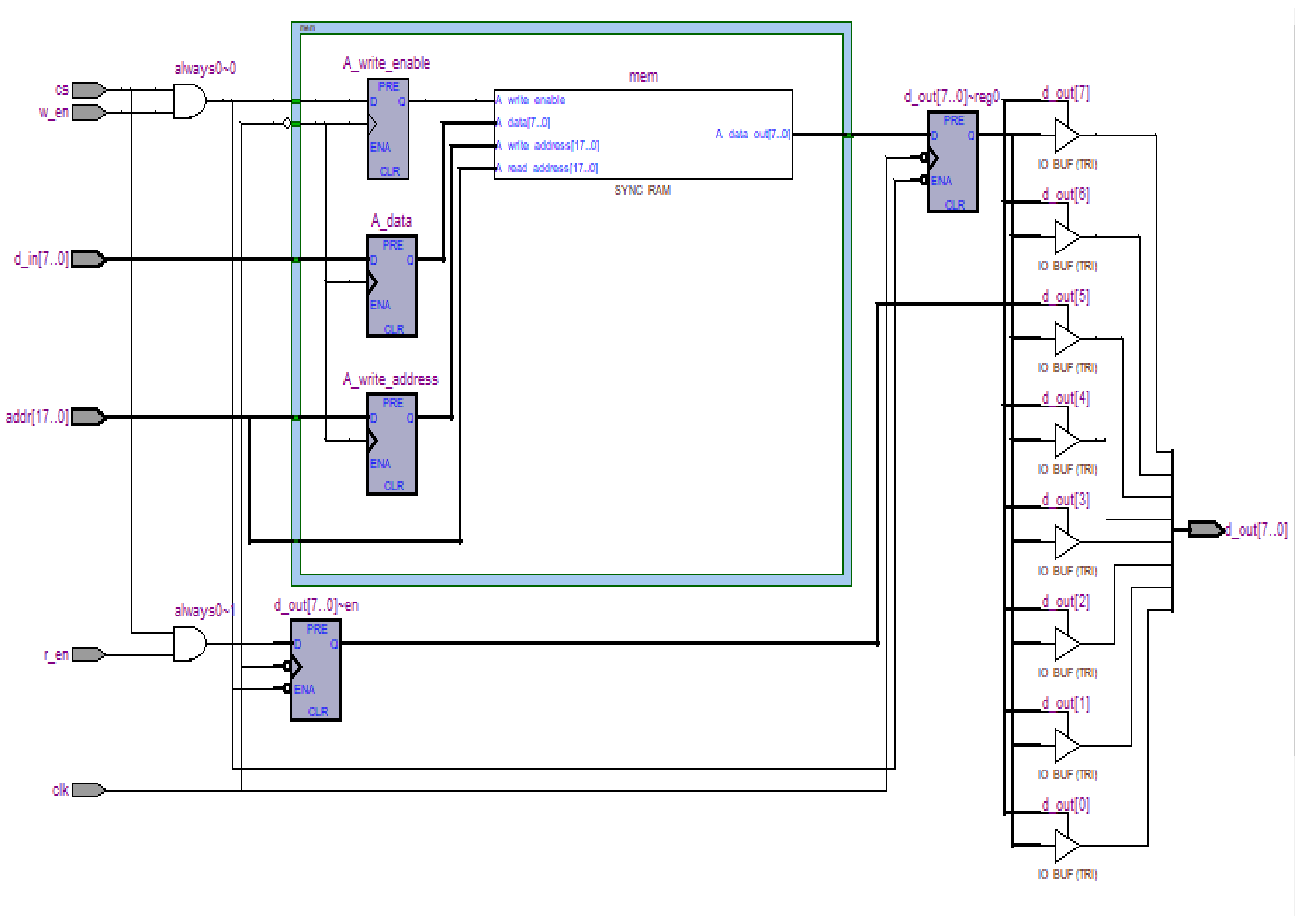


Figure 3.1: RTL view of 256kb REM

2. 1 kb Dual Core ROM

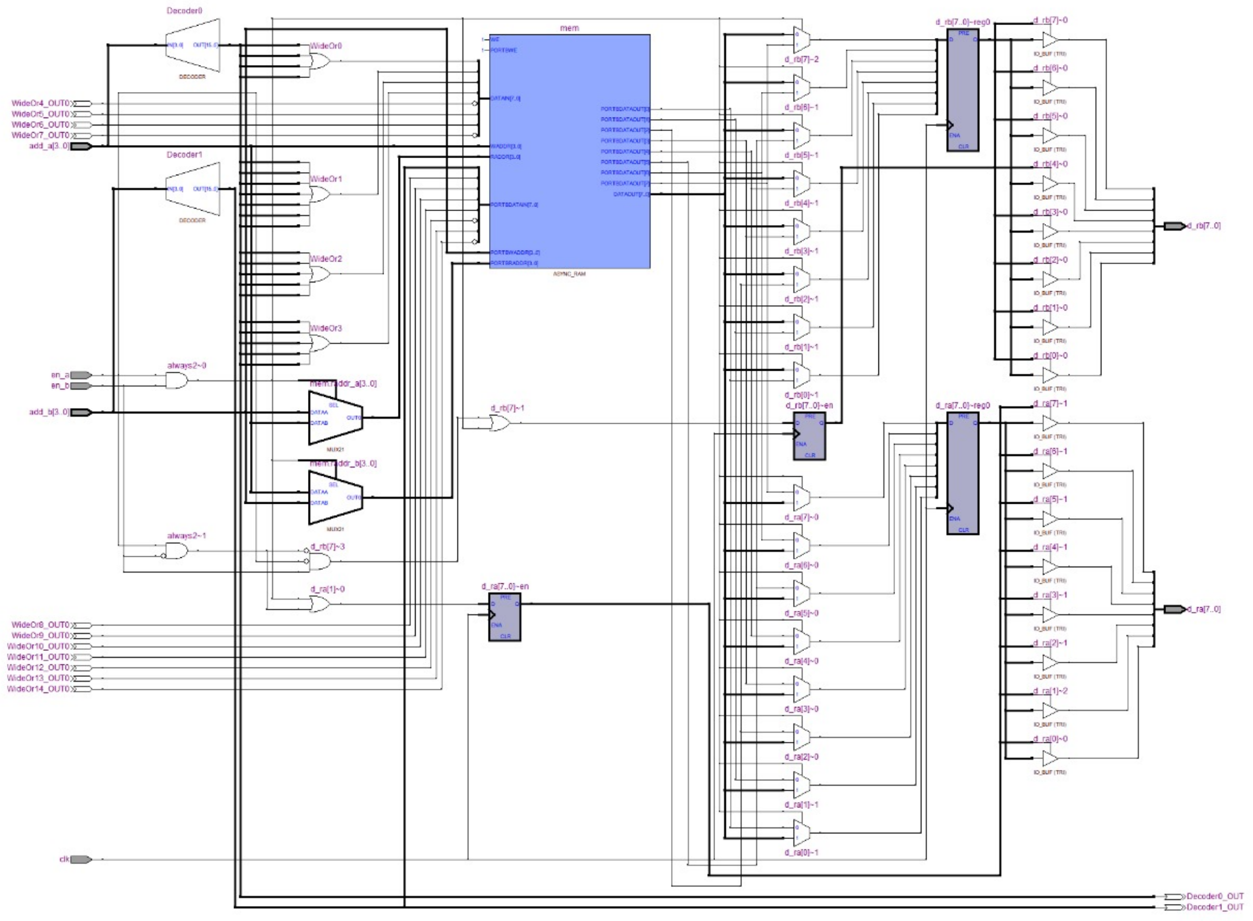


Figure 3.2: RTL view of Dual Core ROM

3. Sense Amplifier

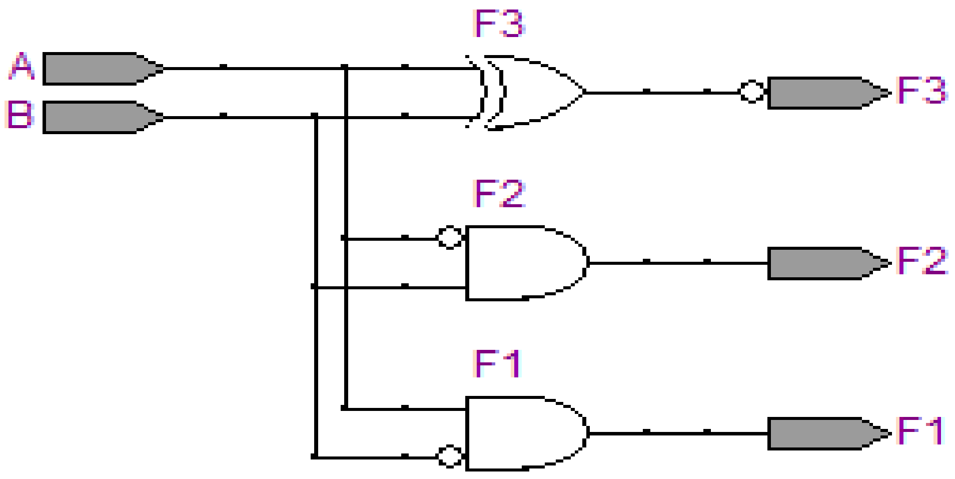


Figure 3.3: RTL view of Sense amplifier

CHAPTER 4

Results and Discussion

4.1 Waveforms

4.1.1 RAM

When Write enable is active we can write our data into Memory, after write if we want to read that data we can read it by Read enable signal, we get same data as input.

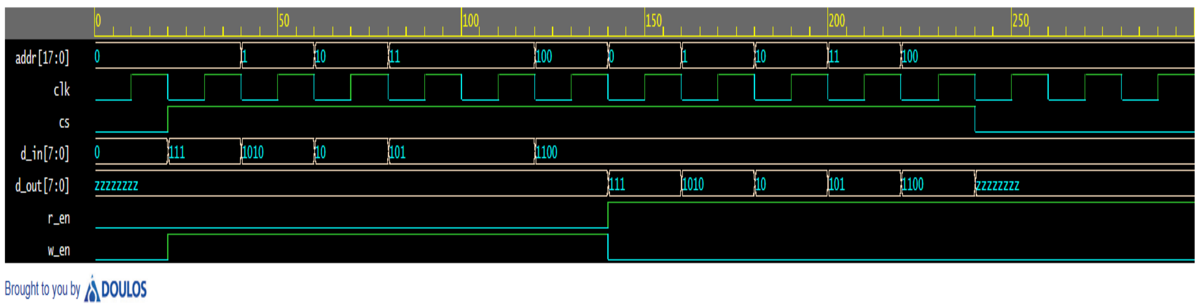


Figure 4.1: RAM Waveform1

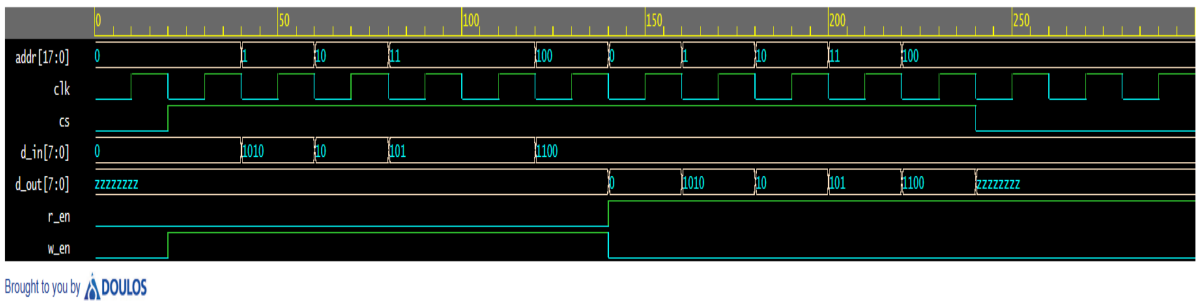
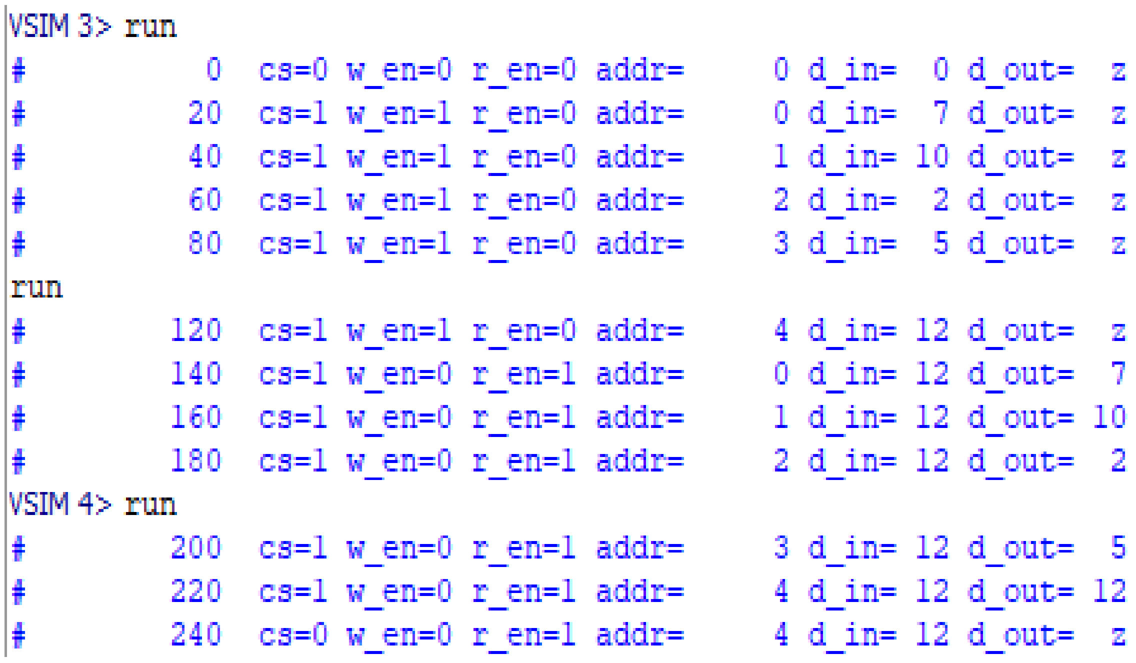


Figure 4.2: RAM Waveform2

4.1.2 RAM Verification

For verification of Design,
We have applied different test case

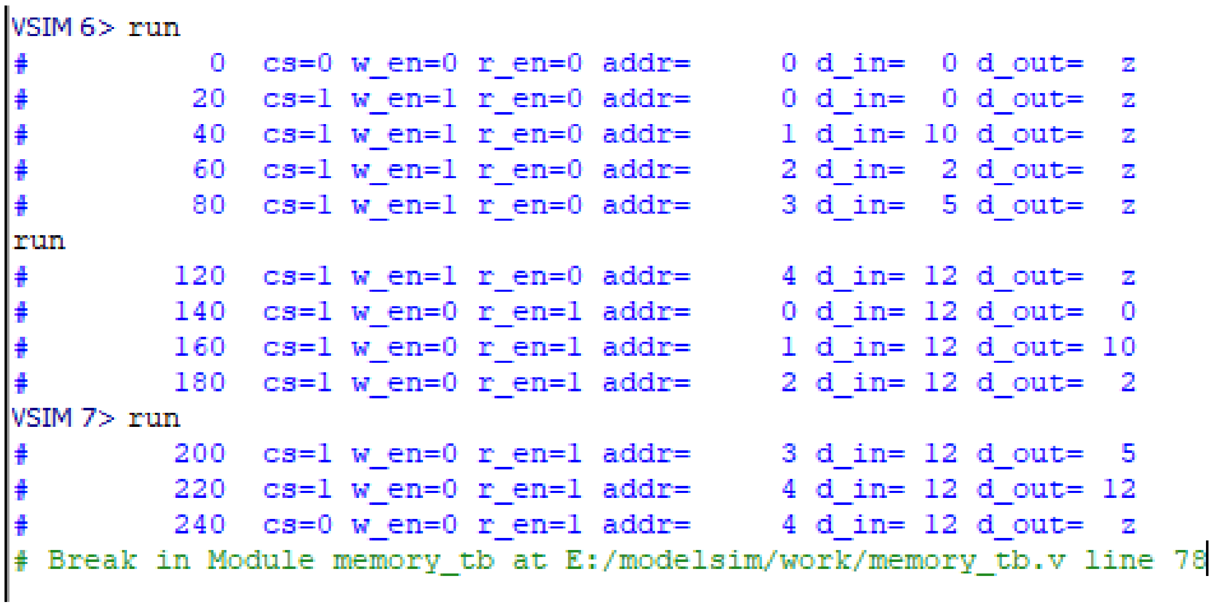
1. Test cases file 1



```
VSIM 3> run
#          0  cs=0 w_en=0 r_en=0 addr=     0 d_in=  0 d_out=  z
#         20  cs=1 w_en=1 r_en=0 addr=     0 d_in=  7 d_out=  z
#         40  cs=1 w_en=1 r_en=0 addr=     1 d_in= 10 d_out=  z
#         60  cs=1 w_en=1 r_en=0 addr=     2 d_in=  2 d_out=  z
#         80  cs=1 w_en=1 r_en=0 addr=     3 d_in=  5 d_out=  z
run
#        120  cs=1 w_en=1 r_en=0 addr=     4 d_in= 12 d_out=  z
#        140  cs=1 w_en=0 r_en=1 addr=     0 d_in= 12 d_out=  7
#        160  cs=1 w_en=0 r_en=1 addr=     1 d_in= 12 d_out= 10
#        180  cs=1 w_en=0 r_en=1 addr=     2 d_in= 12 d_out=  2
VSIM 4> run
#        200  cs=1 w_en=0 r_en=1 addr=     3 d_in= 12 d_out=  5
#        220  cs=1 w_en=0 r_en=1 addr=     4 d_in= 12 d_out= 12
#        240  cs=0 w_en=0 r_en=1 addr=     4 d_in= 12 d_out=  z
```

Figure 4.3: Test case_1

2. Test cases file 2



```
VSIM 6> run
#          0  cs=0 w_en=0 r_en=0 addr=     0 d_in=  0 d_out=  z
#         20  cs=1 w_en=1 r_en=0 addr=     0 d_in=  0 d_out=  z
#         40  cs=1 w_en=1 r_en=0 addr=     1 d_in= 10 d_out=  z
#         60  cs=1 w_en=1 r_en=0 addr=     2 d_in=  2 d_out=  z
#         80  cs=1 w_en=1 r_en=0 addr=     3 d_in=  5 d_out=  z
run
#        120  cs=1 w_en=1 r_en=0 addr=     4 d_in= 12 d_out=  z
#        140  cs=1 w_en=0 r_en=1 addr=     0 d_in= 12 d_out=  0
#        160  cs=1 w_en=0 r_en=1 addr=     1 d_in= 12 d_out= 10
#        180  cs=1 w_en=0 r_en=1 addr=     2 d_in= 12 d_out=  2
VSIM 7> run
#        200  cs=1 w_en=0 r_en=1 addr=     3 d_in= 12 d_out=  5
#        220  cs=1 w_en=0 r_en=1 addr=     4 d_in= 12 d_out= 12
#        240  cs=0 w_en=0 r_en=1 addr=     4 d_in= 12 d_out=  z
# Break in Module memory_tb at E:/modelsim/work/memory_tb.v line 78
```

Figure 4.4: Test case_2

4.1.3 ROM Verification

For ROM we have initially load data into memory with the help of different testbench file.We have tested design with different data, when read enable signal is active of respective port,data can be read at data_out pin of respective port.

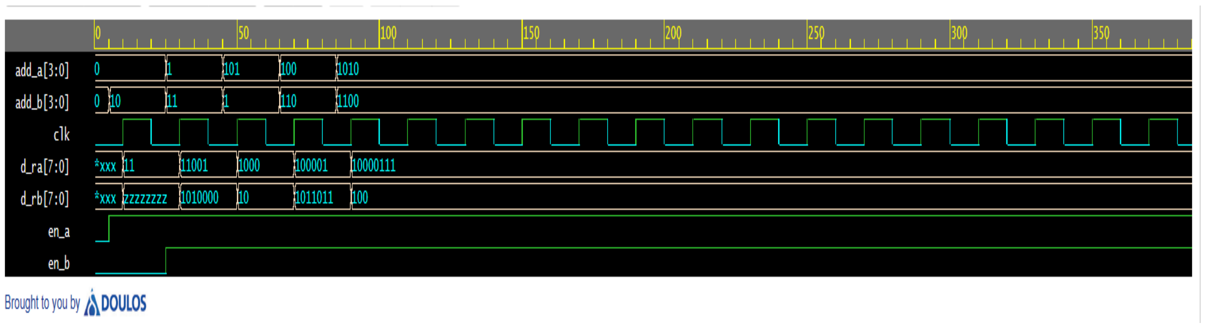


Figure 4.5: ROM testbench waveform1

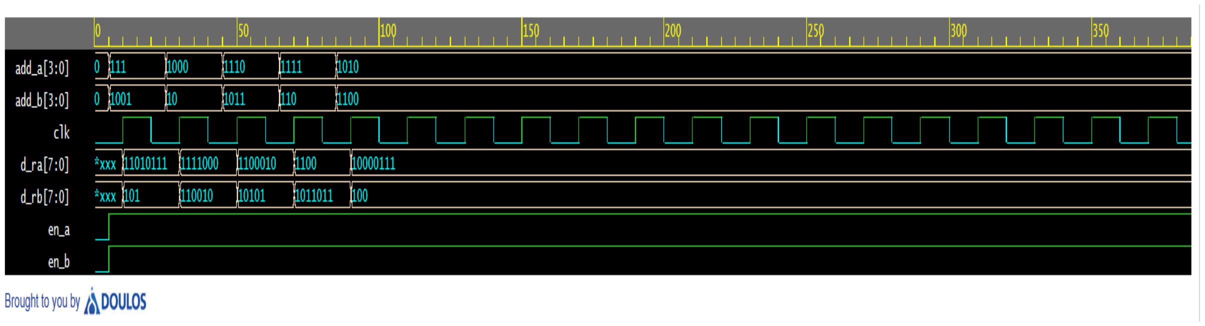


Figure 4.6: ROM testbench waveform2

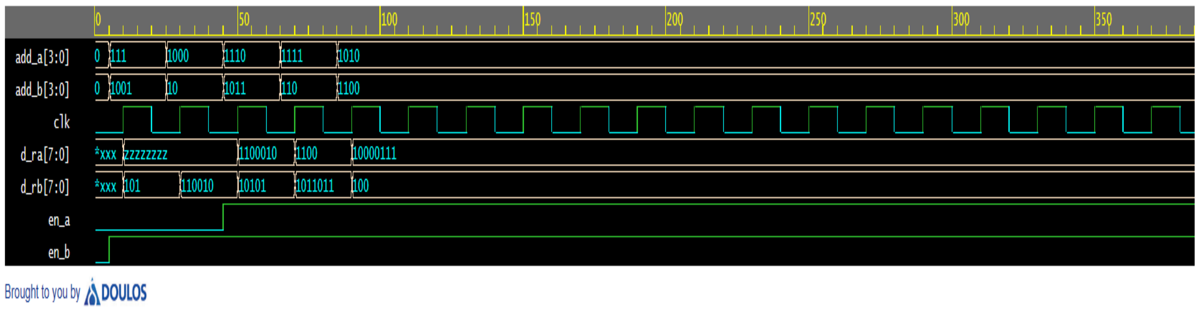
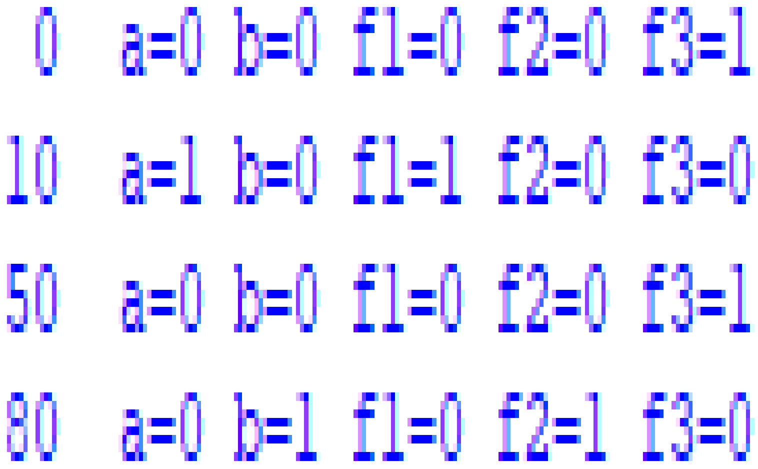


Figure 4.7: ROM testbench waveform3

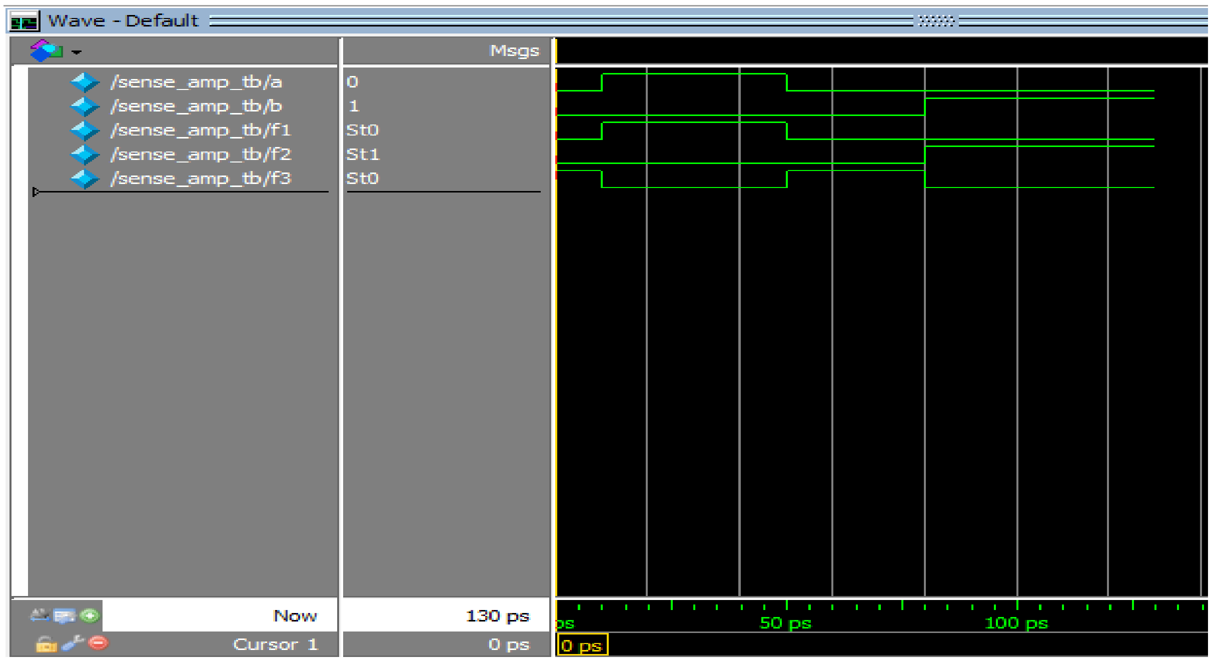
4.1.4 Sense amplifier Verification

Output states of Sense amplifier is obtained based on different states of input A & B.



```
 0  a=0 b=0 f1=0 f2=0 f3=1
10  a=1 b=0 f1=1 f2=0 f3=0
50  a=0 b=0 f1=0 f2=0 f3=1
80  a=0 b=1 f1=0 f2=1 f3=0
```

Figure 4.8: Sense amplifier test cases



```
Wave - Default
                      Msgs
/sense_amp_tb/a       0
/sense_amp_tb/b       1
/sense_amp_tb/f1      St0
/sense_amp_tb/f2      St1
/sense_amp_tb/f3      St0
Now                   130 ps
Cursor 1              0 ps
```

Figure 4.9: waveform

CHAPTER 5

Conclusion

In summary, this project provided a comprehensive understanding of RAM, ROM, and sense amplifier design in Verilog.The design of RAM and ROM modules in Verilog is a crucial aspect of digital system development,a these memory components serve as the foundation for efficient data storage and retrieval. By following the step-by-step process outlined in this document, we can effectively design, implement, and integrate these essential memory building blocks into digital systems, ensuring optimal performance, reliability, and security.By implementing these modules and simulating them with testbenches, we can ensure their functionality and reliability in various computing applications. This project serves as a foundation for further exploration and development in digital design and memory systems.

References

[1] https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-examples/horizontal/ver-dual-clock-syncram.html

[2] https://www.intel.com/content/www/us/en/docs/programmable/683082/22-1/simple-dual-port-dual-clock-synchronous-ram.html

[3] https://youtu.be/yN_1LAKmR0g

Appendix

256kb RAM : VeriLog code

```
module memory(clk,cs,addr,d_in,w_en,r_en,d_out);
input clk;
input cs;
input [17:0] addr;
input [7:0] d_in;
input w_en;
input r_en;
output  reg [7:0] d_out ;

reg [7:0] mem [0:262143];

always @ (negedge clk)
begin
if (w_en && cs)
mem[addr] <= d_in;
else if(r_en && cs)
d_out <= mem[addr];
else
d_out <= 8'dz;
end
endmodule
```

256kb RAM : testbench1

```
module memory_tb;
reg clk;
reg cs;
reg w_en;
reg r_en;
reg [7:0]d_in;
reg [17:0]addr;
wire [7:0]d_out;

memory mp(clk,cs,addr,d_in,w_en,r_en,d_out);

initial begin
$dumpfile("dump.vcd");
$dumpvars(1,memory_tb);

clk<=1'b0;
forever #10 clk<=~clk;
end

initial begin

cs=1'b0;
w_en=1'b0;
r_en=1'b0;
addr=18'd0;
d_in=8'd0;

#20;
cs=1'b1;
w_en=1'b1;
addr=18'd0;
```

```
d_in=8'd7;

#20;
addr=18'd1;
d_in=8'd10;

#20;
addr=18'd2;
d_in=8'd2;

#20;
addr=18'd3;
d_in=8'd5;

#40;
addr=18'd4;
d_in=8'd12;

#20;
w_en=1'b0;
r_en=1'b1;
addr=18'd0;

#20;
addr=18'd1;

#20;
addr=18'd2;

#20;
addr=18'd3;
```

```
#20;
addr=18'd4;

#20;
cs=1'b0;

end

initial begin

$monitor($stime,,,"cs=%b w_en=%b r_en=%b addr=%d d_in=%d
d_out=%d",cs,w_en,r_en,addr,d_in,d_out);

end
initial begin
#300 $stop;
end
endmodule
```

256kb RAM : testbench2

```
module memory_tb;
reg clk;
reg cs;
reg w_en;
reg r_en;
reg [7:0]d_in;
reg [17:0]addr;
wire [7:0]d_out;

memory mp(clk,cs,addr,d_in,w_en,r_en,d_out);
```

```
initial begin
$dumpfile("dump.vcd");
$dumpvars(1,memory_tb);

clk<=1'b0;
forever #10 clk<=~clk;
end

initial begin

cs=1'b0;
w_en=1'b0;
r_en=1'b0;
addr=18'd0;
d_in=8'd0;

#20;
cs=1'b1;
w_en=1'b1;
addr=18'd0;
d_in=8'd0;

#20;
addr=18'd1;
d_in=8'd10;

#20;
addr=18'd2;
d_in=8'd2;

#20;
```

```
addr=18’d3;
d_in=8’d5;

#40;
addr=18’d4;
d_in=8’d12;

#20;
w_en=1’b0;
r_en=1’b1;
addr=18’d0;

#20;
addr=18’d1;

#20;
addr=18’d2;

#20;
addr=18’d3;

#20;
addr=18’d4;

#20;
cs=1’b0;

end

initial begin
```

```
$monitor($stime,,,"cs=%b w_en=%b r_en=%b addr=%d d_in=%d
d_out=%d",cs,w_en,r_en,addr,d_in,d_out);

end
initial begin
#300 $stop;
end
endmodule
```

1kb ROM : VeriLog code

```
module ROM(clk,en_a,en_b,add_a,add_b,d_ra,d_rb);
input clk;
input en_a;
input en_b;
input [3:0] add_a,add_b;
output reg [7:0] d_ra,d_rb;

reg [7:0] mem [15:0] ;

always @ (add_a)
begin
case (add_a)

4'd0:
mem[add_a]=8'd3;

4'd1:
mem[add_a]=8'd25;

4'd2:
mem[add_a]=8'd65;
```

```
4'd3:
mem[add_a]=8'd8;

4'd4:
mem[add_a]=8'd33;

4'd5:
mem[add_a]=8'd8;

4'd6:
mem[add_a]=8'd90;

4'd7:
mem[add_a]=8'd215;

4'd8:
mem[add_a]=8'd120;

4'd9:
mem[add_a]=8'd45;

4'd10:
mem[add_a]=8'd135;

4'd11:
mem[add_a]=8'd200;

4'd12:
mem[add_a]=8'd155;

4'd13:
```

```
mem[add_a]=8'd99;

4'd14:
mem[add_a]=8'd98;

4'd15:
mem[add_a]=8'd12;

default:
mem[add_a]=8'd0;

endcase
end

always @ (add_b)
begin
case (add_b)

4'd0:
mem[add_b]=8'd47;

4'd1:
mem[add_b]=8'd2;

4'd2:
mem[add_b]=8'd50;

4'd3:
mem[add_b]=8'd80;

4'd4:
mem[add_b]=8'd37;
```

```
4’d5:
mem[add_b]=8’d88;

4’d6:
mem[add_b]=8’d91;

4’d7:
mem[add_b]=8’d199;

4’d8:
mem[add_b]=8’d1;

4’d9:
mem[add_b]=8’d5;

4’d10:
mem[add_b]=8’d100;

4’d11:
mem[add_b]=8’d21;

4’d12:
mem[add_b]=8’d4;

4’d13:
mem[add_b]=8’d19;

4’d14:
mem[add_b]=8’d68;

4’d15:
```

```
mem[add_b]=8'd52;

default:
mem[add_b]=8'd0;

endcase
end

always @ (posedge clk)
begin
if(en_a && en_b)
begin
d_ra=mem[add_a];
d_rb=mem[add_b];
end
else if(en_a && ~en_b)
begin
d_ra=mem[add_a];
d_rb=8'dz;
end
else if(~en_a && en_b)
begin
d_ra=8'dz;
d_rb=mem[add_b];
end
else
begin
d_ra=8'dz;
d_rb=8'dz;
end
end
endmodule
```

1kb ROM : testbench1

```
module ROM_tb;
reg clk;
reg en_a;
reg en_b;
reg [3:0]add_a;
reg [3:0]add_b;
wire [7:0]d_ra;
wire [7:0]d_rb;

ROM mp(clk,en_a,en_b,add_a,add_b,d_ra,d_rb);

initial begin

$dumpfile("dump.vcd");
$dumpvars(1,ROM_tb);
clk<=1'b0;
forever #10 clk<=~clk;
end

initial begin
en_a<=1'b0;
en_b<=1'b0;
add_a<=4'd0;
add_b<=4'd0;

#5;
en_a<=1'b1;
en_b<=1'b0;
add_a<=4'd0;
add_b<=4'd2;
```

```
#20;
en_b<=1'b1;
add_a<=4'd1;
add_b<=4'd3;

#20;
add_a<=4'd5;
add_b<=4'd1;

#20;
add_a<=4'd4;
add_b<=4'd6;

#20;
add_a<=4'd10;
add_b<=4'd12;

#300;
$finish;
end

initial begin
$monitor($stime,,,"en_a=%b en_b=%b add_a=%d add_b=%d
d_ra=%d d_rb=%d",en_a,en_b,add_a,add_b,d_ra,d_rb);

end

endmodule
```

1kb ROM : testbench2

```
module ROM_tb;
reg clk;
reg en_a;
reg en_b;
reg [3:0]add_a;
reg [3:0]add_b;
wire [7:0]d_ra;
wire [7:0]d_rb;

ROM mp(clk,en_a,en_b,add_a,add_b,d_ra,d_rb);

initial begin

$dumpfile("dump.vcd");
$dumpvars(1,ROM_tb);
clk<=1'b0;
forever #10 clk<=~clk;
end

initial begin
en_a<=1'b0;
en_b<=1'b0;
add_a<=4'd0;
add_b<=4'd0;

#5;
en_a<=1'b1;
en_b<=1'b1;
add_a<=4'd7;
add_b<=4'd9;
```

```
#20;
add_a<=4'd8;
add_b<=4'd2;

#20;
add_a<=4'd14;
add_b<=4'd11;

#20;
add_a<=4'd15;
add_b<=4'd6;

#20;
add_a<=4'd10;
add_b<=4'd12;

#300;
$finish;
end

initial begin

$monitor($stime,,,"en_a=%b en_b=%b add_a=%d  add_b=%d
d_ra=%d d_rb=%d",en_a,en_b,add_a,add_b,d_ra,d_rb);

end

endmodule
```

1kb ROM : testbench3

```
module ROM_tb;
reg clk;
reg en_a;
reg en_b;
reg [3:0]add_a;
reg [3:0]add_b;
wire [7:0]d_ra;
wire [7:0]d_rb;

ROM mp(clk,en_a,en_b,add_a,add_b,d_ra,d_rb);

initial begin

$dumpfile("dump.vcd");
$dumpvars(1,ROM_tb);
clk<=1'b0;
forever #10 clk<=~clk;

end

initial begin
en_a<=1'b0;
en_b<=1'b0;
add_a<=4'd0;
add_b<=4'd0;

#5;
en_a<=1'b0;
en_b<=1'b1;
add_a<=4'd7;
add_b<=4'd9;
```

```
#20;
add_a<=4’d8;
add_b<=4’d2;

#20;
en_a<=1’b1;
add_a<=4’d14;
add_b<=4’d11;

#20;
add_a<=4’d15;
add_b<=4’d6;

#20;
add_a<=4’d10;
add_b<=4’d12;

#300;
$finish;
end

initial begin

$monitor($stime,,,"en_a=%b en_b=%b add_a=%d  add_b=%d
d_ra=%d d_rb=%d",en_a,en_b,add_a,add_b,d_ra,d_rb);

end

endmodule
```

Sense amplifier : VeriLog code

```
/***single bit comparator or Sense amplifier
Where A=Bit line
  B=~Bit line
  F1=A>B-->Read 1 operation
  F2=A<B-->Read 0 operation
  F3=A=B-->Both bit & ~bit line have same
           value which is invalid
***/
module sense_amp(A,B,F1,F2,F3);
input A,B;
output F1,F2,F3;
assign F1=A&~B; //read 1 operation
assign F2=~A&B; //read 0 operation
assign F3=A~^B;
endmodule
```

Sense amplifier : testbench1

```
module sense_amp_tb;
reg a,b;
wire f1,f2,f3;
sense_amp s(a,b,f1,f2,f3);
initial begin
a=1'b0;
b=1'b0;
#10 a=1'b1;
#10 b=1'b0;
#30 a=1'b0;
#30 b=1'b1;
#50 $stop;
end
```

```
initial begin
$monitor($stime,,,"a=%b b=%b f1=%b f2=%b f3=%b"
,a,b,f1,f2,f3);
end
endmodule
```