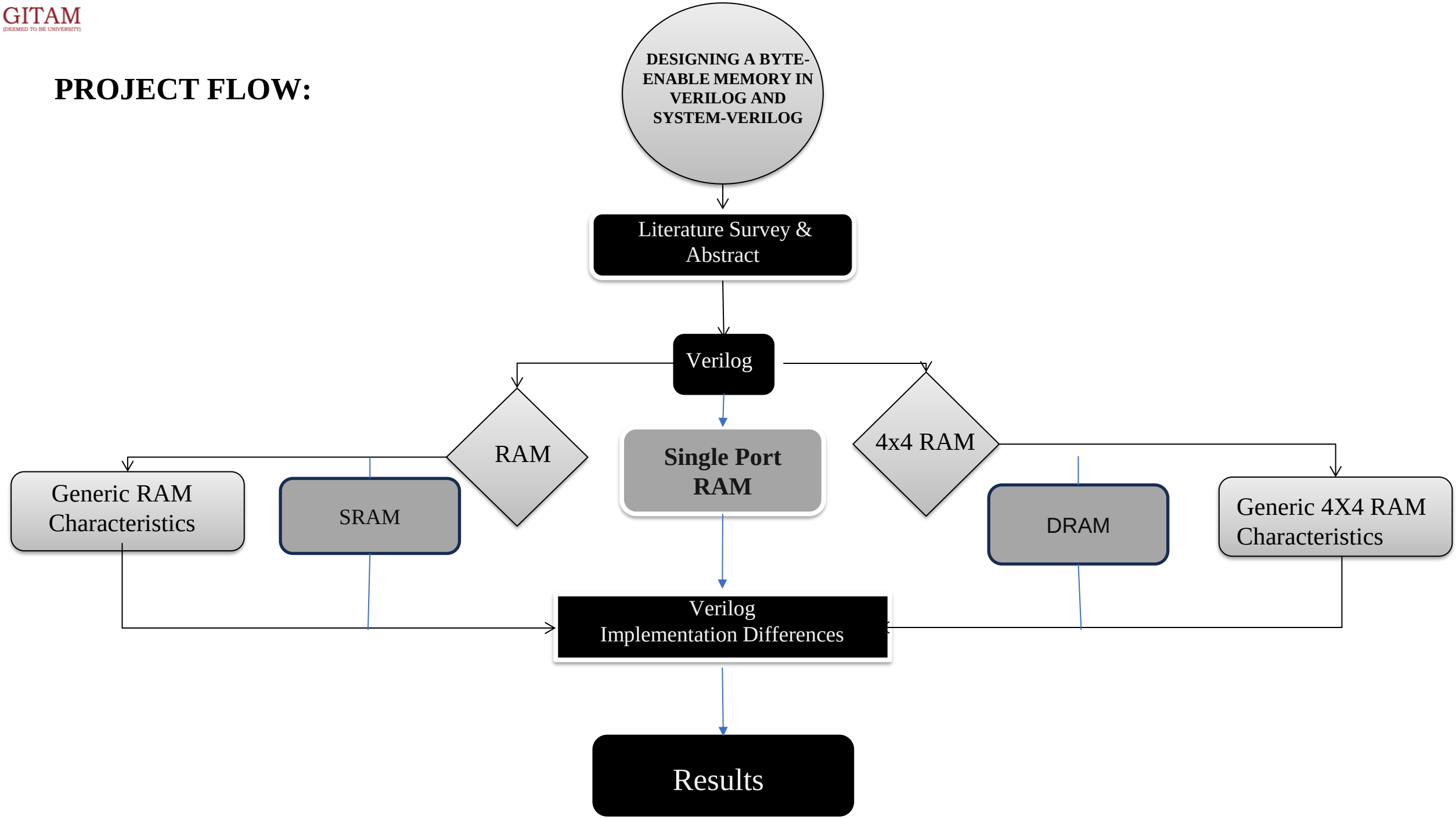
GITAM
(DEEMED TO BE UNIVERSITY)

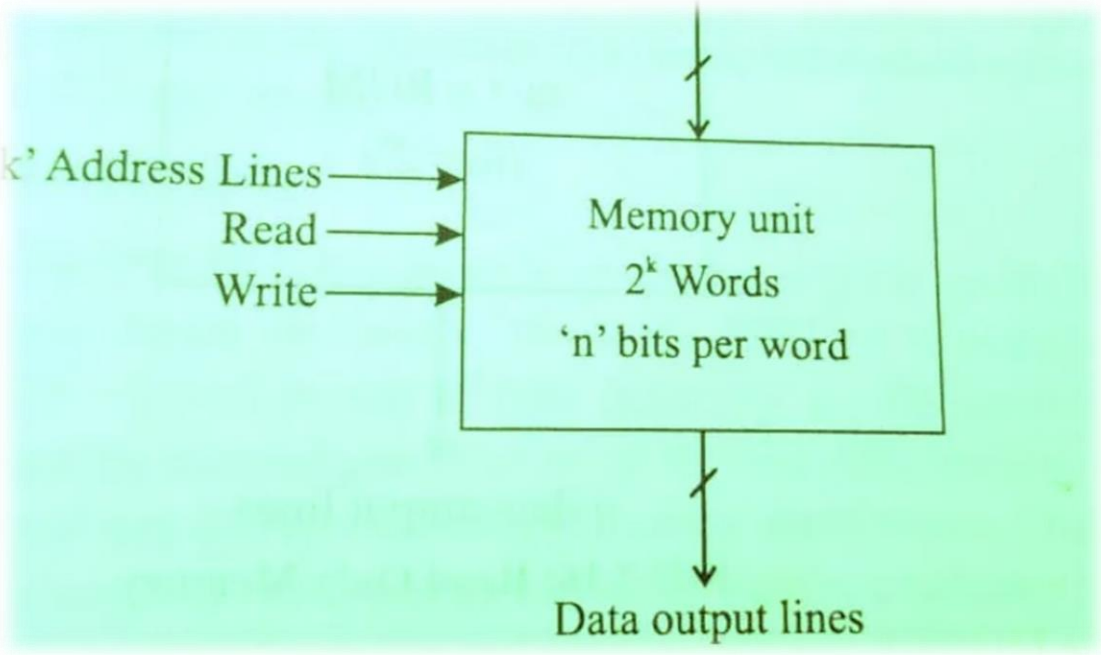
PROJECT FLOW:



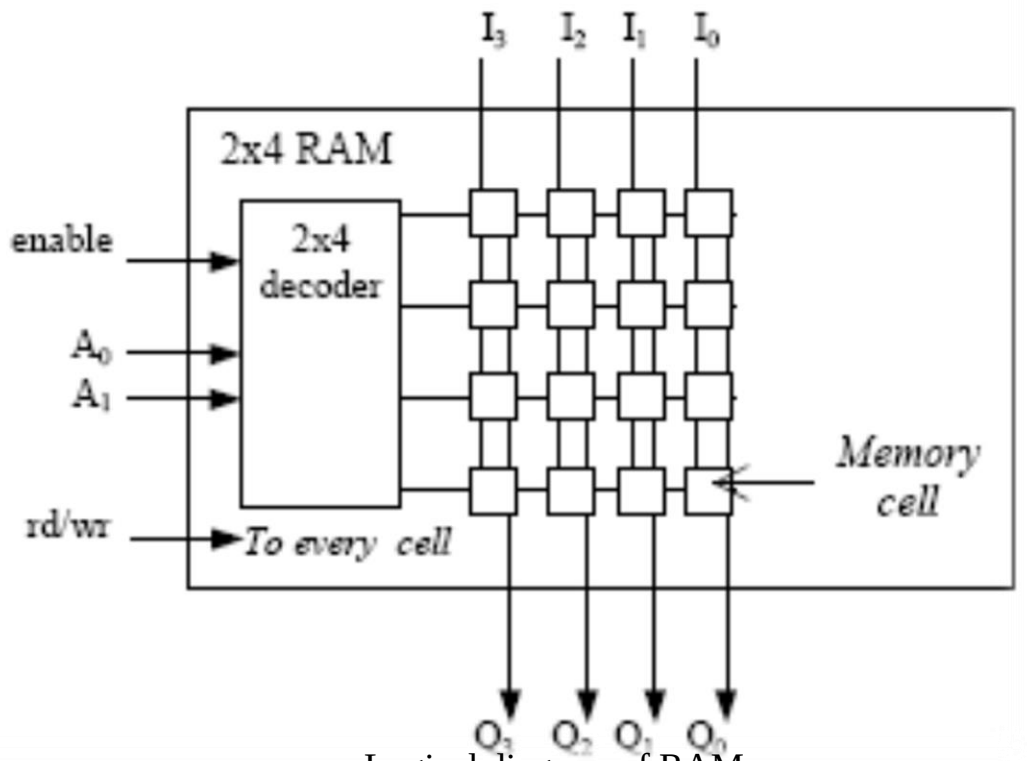
DESIGN SYSTEM:

RAM

RAM (Random Access Memory) is a type of computer memory that is used to store data that is actively being used or processed by the CPU. It is a volatile memory, meaning it loses its contents when the power is turned off, making it temporary storage. RAM plays a critical role in the performance of a computer or any digital system by providing quick access to data and instructions needed by the CPU.



Block diagram of RAM



Logical diagram of RAM

Verilog code for RAM:

C:/Users/Maria Punya/project_11/project_11.srcs/sources_1/new/ram6.v

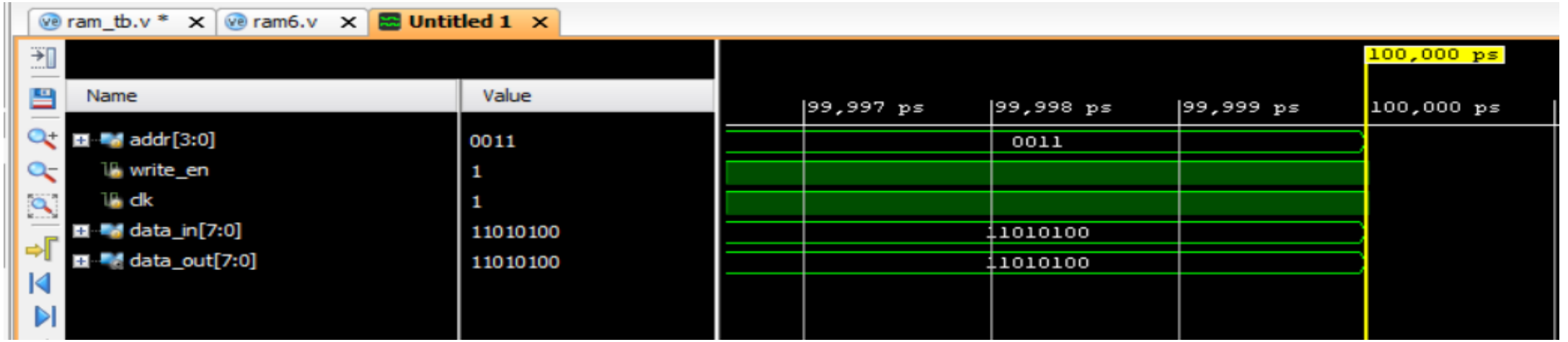
```
22
23  module ram6(
24      input clk,
25      input [3:0] addr,       // 4-bit address, can access 16 memory locations
26      input write_en,         // Write enable signal
27      input [7:0] data_in,    // 8-bit input data
28      output reg [7:0] data_out // 8-bit output data
29  );
30
31      reg [7:0] ram [15:0];  // 16x8 RAM
32
33      always @(posedge clk) begin
34          if (write_en) begin
35              ram[addr] <= data_in; // Write operation
36          end
37          data_out <= ram[addr];      // Read operation
38      end
39
40  endmodule
41
42
```

Project Summary | ram_tb.v *

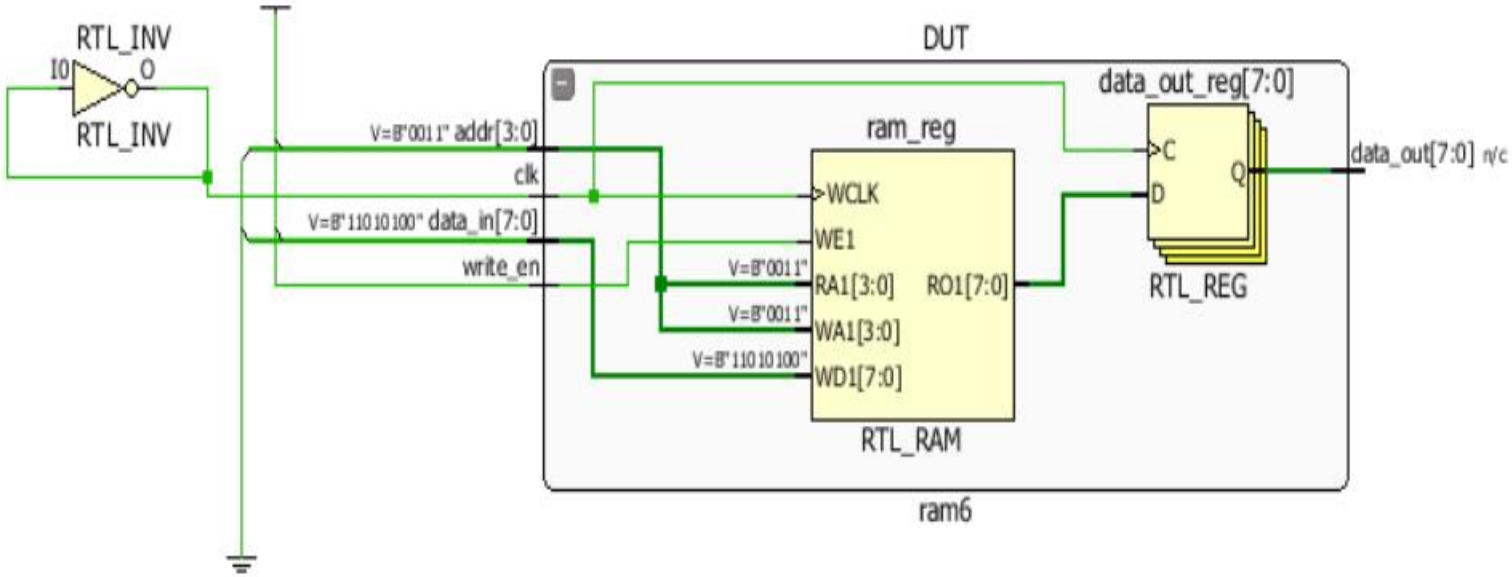
C:/Users/Maria Punya/project_11/project_11.srcs/sources_1/new/ram_tb.v

```
12  module ram_tb;
13      reg [3:0] addr;
14      reg write_en;
15      reg clk;
16      reg [7:0] data_in;
17      wire [7:0] data_out;
18      ram6 DUT(// Instantiate the RAM module
19          .clk(clk),
20          .addr(addr),
21          .write_en(write_en),
22          .data_in(data_in),
23          .data_out(data_out)
24      );
25      always #5 clk = ~clk;// Generate clock signal
26      initial begin
27          clk = 0;
28          write_en = 1;
29          data_in = 8'b00000000;
30          #10; // Write data to the RAM
31          write_en = 1;
32          addr = 4'b0000; data_in = 8'hA1; #10;
33          addr = 4'b0001; data_in = 8'hB2; #10;
34          addr = 4'b0010; data_in = 8'hC3; #10;
35          addr = 4'b0011; data_in = 8'hD4; #10;
36          write_en = 1;// Disable write
37          #10;// Read data from the RAM
38          addr = 4'b0000; #10;
39          addr = 4'b0001; #10;
40          addr = 4'b0010; #10;
41          addr = 4'b0011; #10;
42          $finish;
43      end
44      initial begin// Display output
45          $monitor("At time %t, addr = %h, data_out = %h", $time, addr, data_out);
46      end
```

RESULTS:



Schematic Diagram:



4X4 RAM

A 4x4 RAM is a small memory unit that can store 16 bits of data, organized into 4 locations (or memory cells), each holding 4 bits. It uses 2 address lines to select one of the 4 locations, and 4 data lines to read from or write to that location.

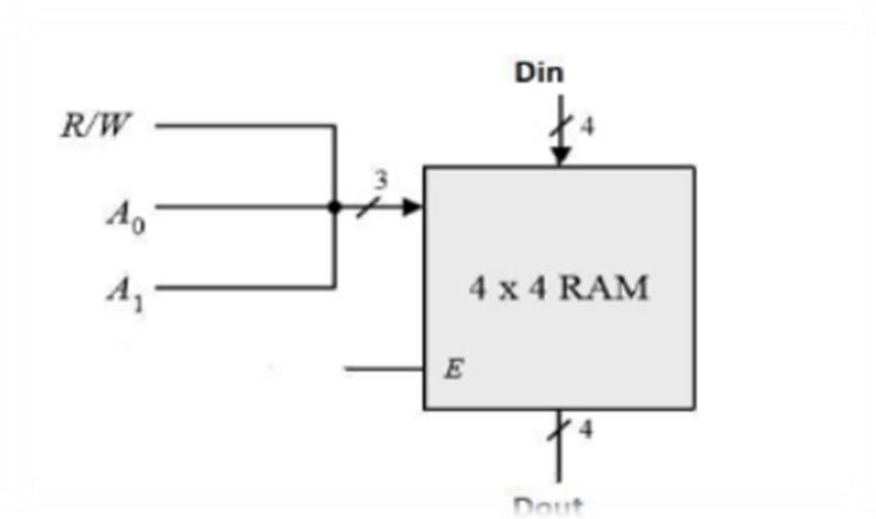
4 Locations: Each location stores 4 bits.

Addressing: 2 address lines (addr[1:0]) are used to select one of the 4 memory locations.

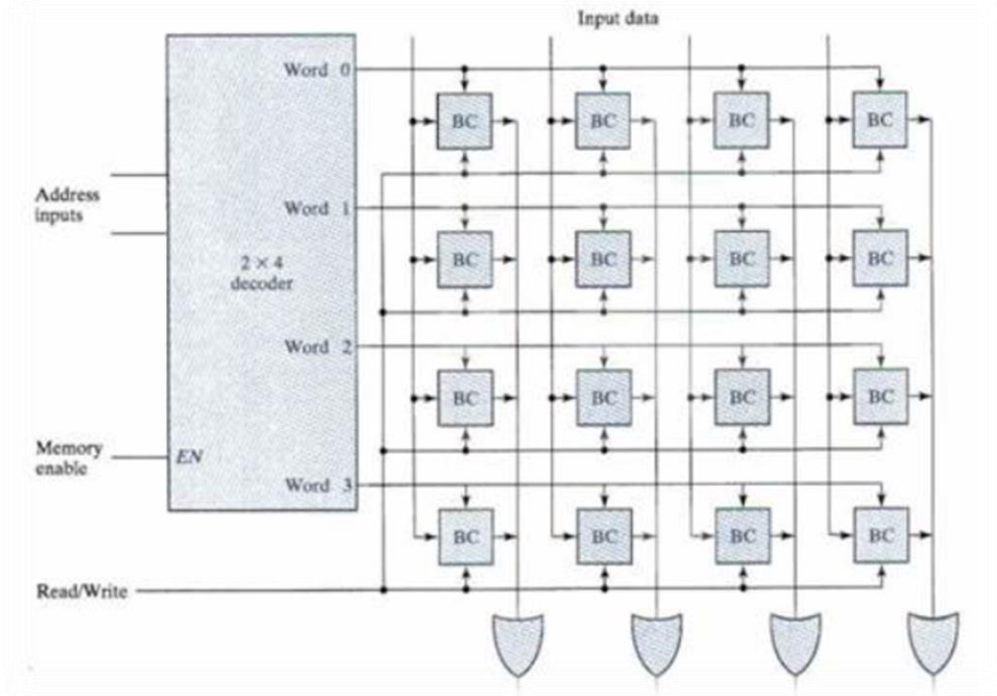
Data Handling: 4-bit data can be written to or read from the selected location.

Control Signal: A write enable (we) signal determines if data is written to memory (when high) or read from it (when low).

Clock: Operations are synchronized using a clock signal (clk).



Block diagram of 4X4 RAM



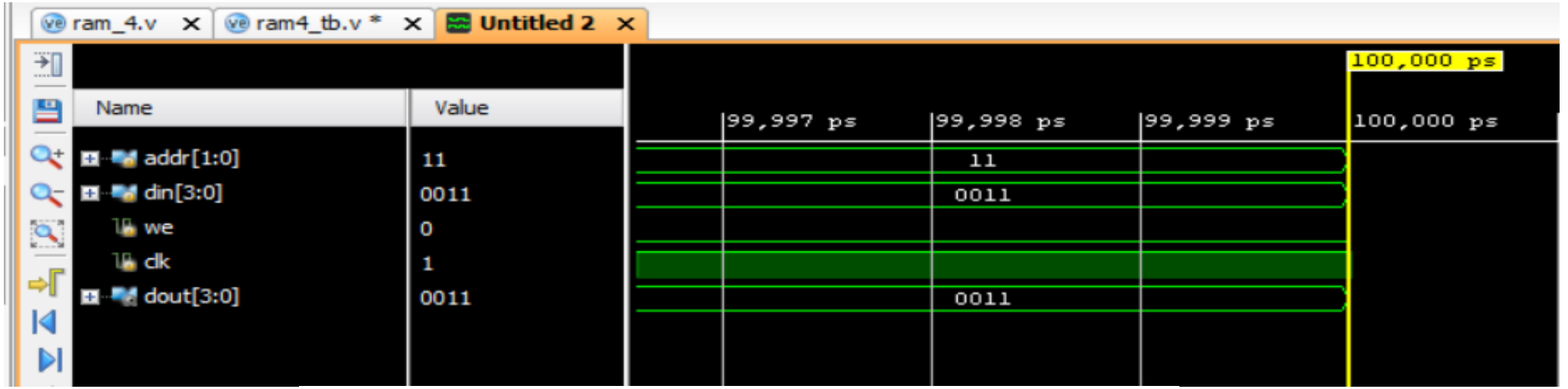
Logical diagram of 4X4 RAM

Verilog Code 4X4 RAM:

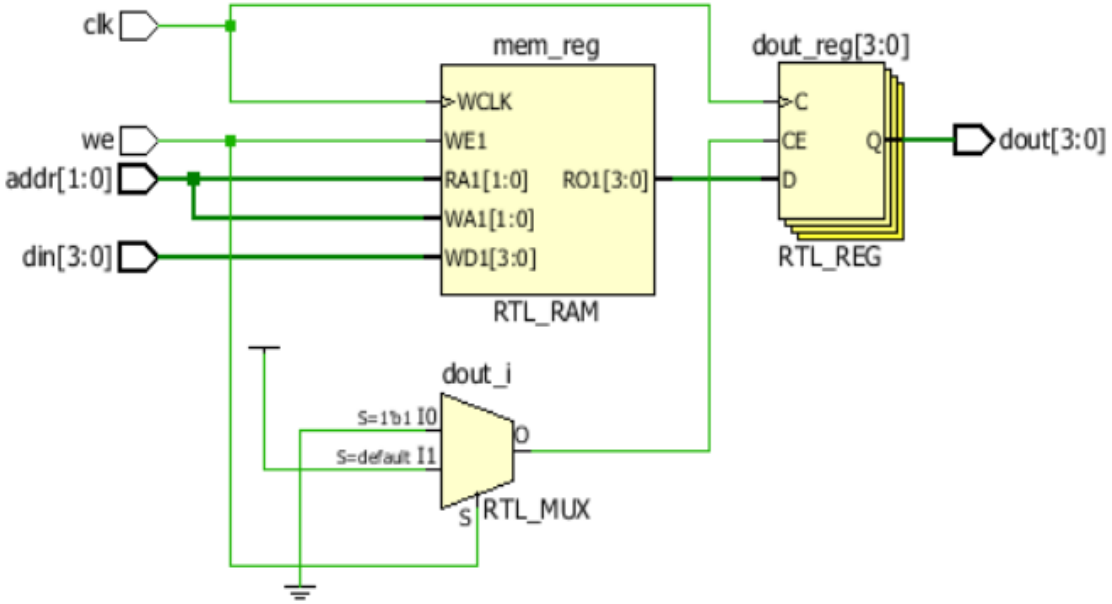
```
module ram_4(
    input wire [1:0] addr,    // 2-bit address input to select one of 4 locations
    input wire [3:0] din,     // 4-bit data input
    input wire we,            // Write enable signal
    input wire clk,           // Clock signal
    output reg [3:0] dout     // 4-bit data output
);

    // 4x4 memory array
    reg [3:0] mem [3:0];

    always @(posedge clk) begin
        if (we) begin
            mem[addr] <= din;  // Write data to memory at the selected address
        end else begin
            dout <= mem[addr]; // Read data from memory at the selected address
        end
    end
endmodule
```

```
module ram4_tb;
reg [1:0] addr;
    reg [3:0] din;
    reg we;
    reg clk;
    wire [3:0] dout;
    ram_4 uut (    // Instantiate the RAM module
        .addr(addr),
        .din(din),
        .we(we),
        .clk(clk),
        .dout(dout)
    );
    always #5 clk = ~clk;// Clock generation
    initial begin // Initialize signals
        clk = 0;
        we = 0;
        addr = 2'b00;
        din = 4'b0000;
        // Write some data into the memory
        #10 we = 1; addr = 2'b00; din = 4'b1010;  // Write 1010 at address 00
        #10 we = 1; addr = 2'b01; din = 4'b1100;  // Write 1100 at address 01
        #10 we = 1; addr = 2'b10; din = 4'b1111;  // Write 1111 at address 10
        #10 we = 1; addr = 2'b11; din = 4'b0011;  // Write 0011 at address 11
        // Read data from memory
        #10 we = 0; addr = 2'b00;  // Read from address 00
        #10 we = 0; addr = 2'b01;  // Read from address 01
        #10 we = 0; addr = 2'b10;  // Read from address 10
        #10 we = 0; addr = 2'b11;  // Read from address 11
        #20 $finish;// End simulation
    end
endmodule
```

RESULTS:



Schematic Diagram:



SRAM(Static Random Access Memory):

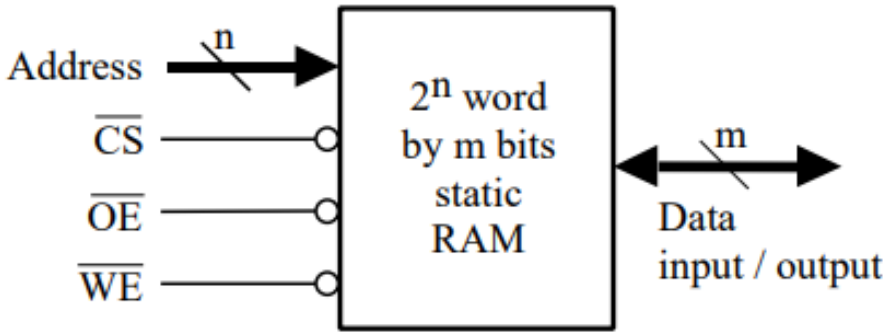
SRAM (Static Random Access Memory) is a type of memory that stores data using flip-flop circuits. Each memory cell in SRAM consists of a few transistors (typically 6) that can hold data as long as power is supplied. Unlike DRAM, SRAM doesn't require refreshing to retain data, which makes it faster but more expensive due to its higher transistor count.

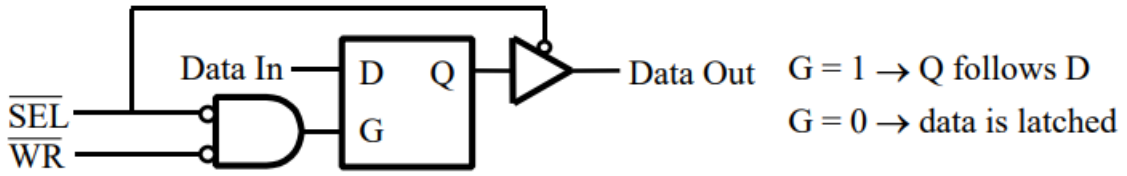
Working Principle:

Read Operation: When the memory cell's address is provided and the read control signal is active, the data stored in the cell is transferred to the output.

Write Operation: When the write control signal is active, data is written into the selected memory cell at the given address.

Key features include fast access times and lower power consumption when idle, but SRAM is typically used for small amounts of high-speed memory, such as CPU cache, due to its size and cost.

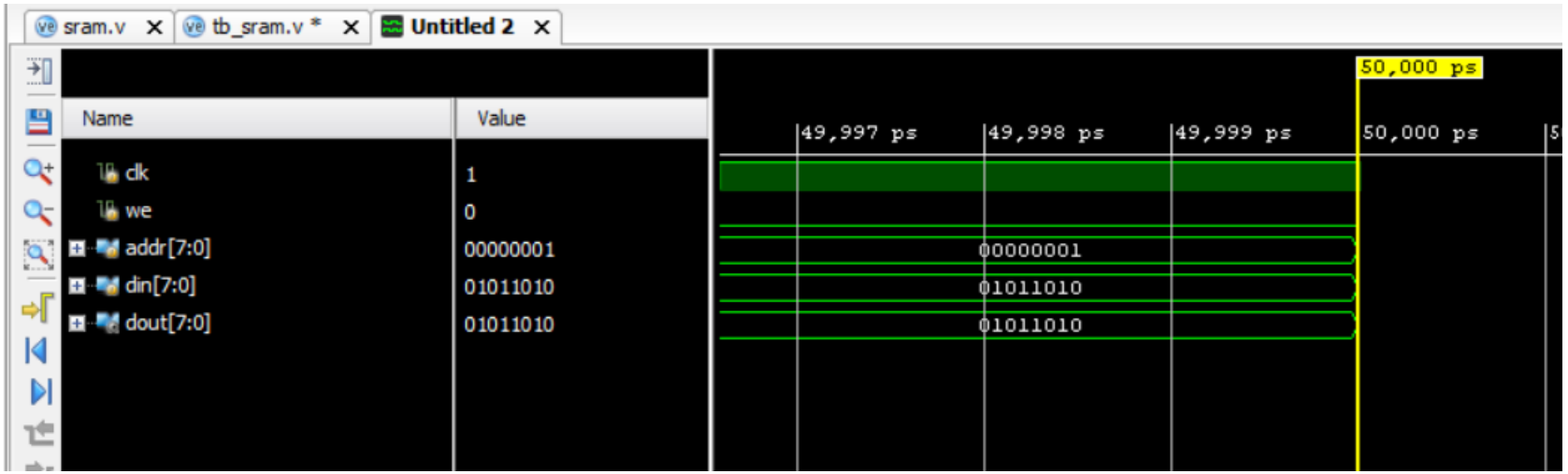




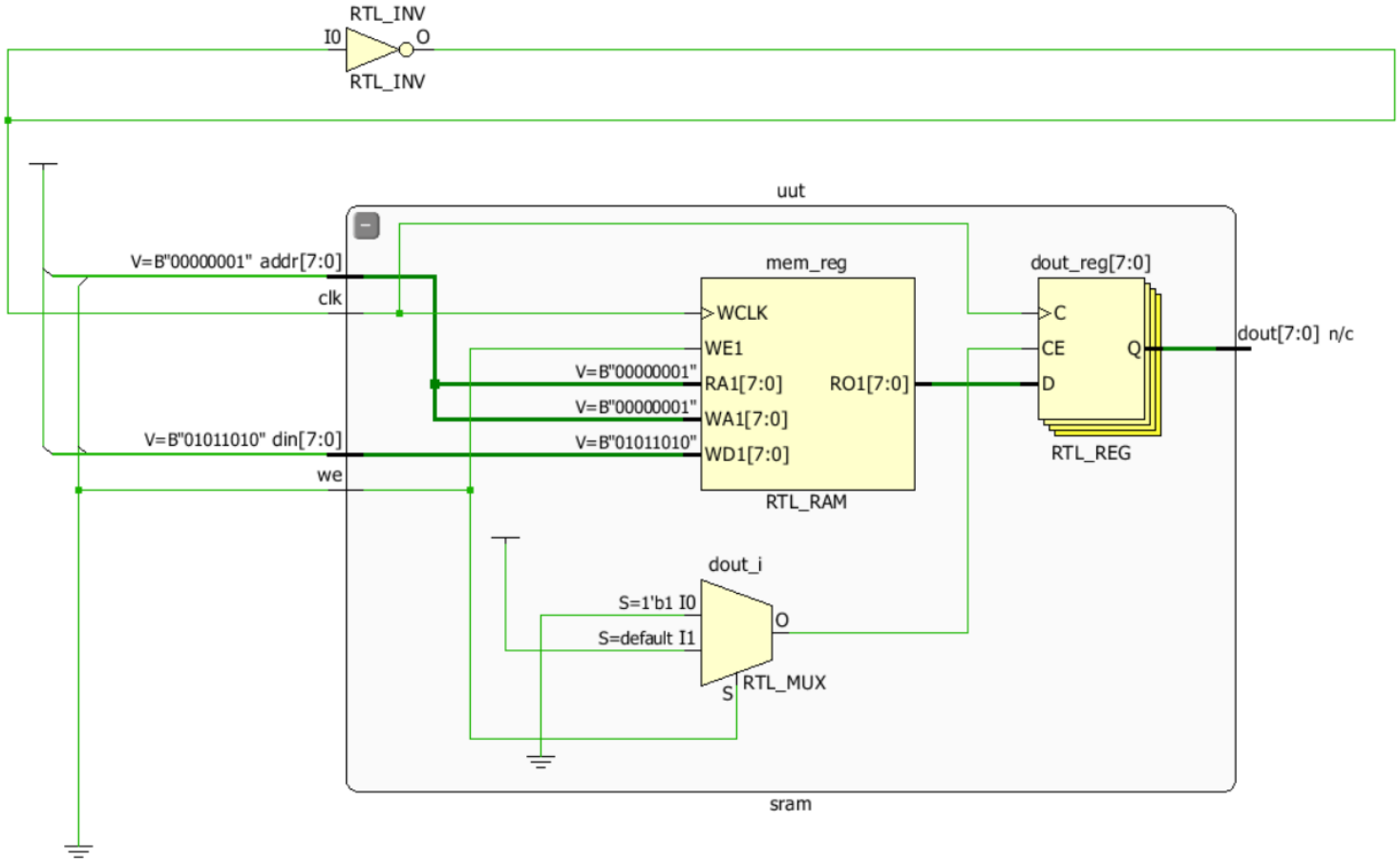
```
////////////////////////////////////////////////////////////////////////////////


module sram(
    input wire clk,           // Clock signal
    input wire we,            // Write Enable
    input wire [7:0] addr,    // Address bus (8-bit for 256 locations)
    input wire [7:0] din,     // Data input (8-bit)
    output reg [7:0] dout     // Data output (8-bit)
);

// Memory declaration (256 x 8-bit)
reg [7:0] mem [255:0];

always @(posedge clk) begin
    if (we) begin
        // Write operation
        mem[addr] <= din;
    end
    else begin
        // Read operation
        dout <= mem[addr];
    end
end
endmodule
```

```
module tb_sram();

reg clk;
reg we;
reg [7:0] addr;
reg [7:0] din;
wire [7:0] dout;
sram uut (
    .clk(clk),
    .we(we),
    .addr(addr),
    .din(din),
    .dout(dout)
);
// Clock generation
always #5 clk = ~clk;
initial begin
    // Initialize signals
    clk = 0;
    we = 0;
    addr = 8'd0;
    din = 8'd0;
    // Write 8'hA5 to address 0
    #10 we = 1; addr = 8'd0; din = 8'hA5;
    // Write 8'h5A to address 1
    #10 addr = 8'd1; din = 8'h5A;
    // Disable write, perform read from address 0
    #10 we = 0; addr = 8'd0;
    // Read from address 1
    #10 addr = 8'd1;
    #10 $stop;
end
endmodule
```

Results:


sram.v
tb_sram.v *
Untitled 2
Name
Value
clk
1
we
0
addr[7:0]
00000001
din[7:0]
01011010
dout[7:0]
01011010
50,000 ps
49,997 ps
49,998 ps
49,999 ps
50,000 ps
00000001
01011010
01011010

Schematic Diagram:



DRAM(Dynamic Random Access memory):

It is a type of volatile memory used in computers and other digital systems. Unlike SRAM, which stores data in flip-flops, DRAM stores each bit of data in a tiny capacitor within a memory cell. Due to the nature of capacitors, DRAM requires periodic refreshing to maintain its data, as the charge in the capacitors leaks over time.

•**Memory Cells**: Each memory cell in DRAM consists of a capacitor and a transistor. The capacitor stores electrical charge, representing a bit of data (either a '1' or '0'). If the capacitor is charged, the bit is '1'; if discharged, it is '0'.

•**Read Operation**:

•The memory controller selects a **row** and **column** in the DRAM matrix.

•The charge in the capacitor is read through a transistor.

•Reading data from a DRAM cell discharges the capacitor, so it needs to be recharged (refreshed) immediately afterward.

•**Write Operation**:

•A specific row and column are selected, and data (as a charge or lack of charge) is written into the corresponding memory cell's capacitor.

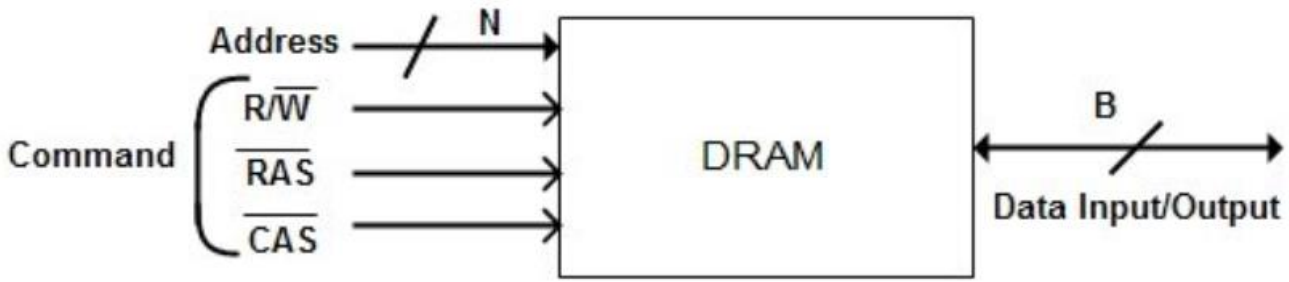
•**Refresh**:

•DRAM cells gradually lose their charge, so they need to be refreshed periodically (typically every few milliseconds) to retain data. The memory controller handles this refresh process.

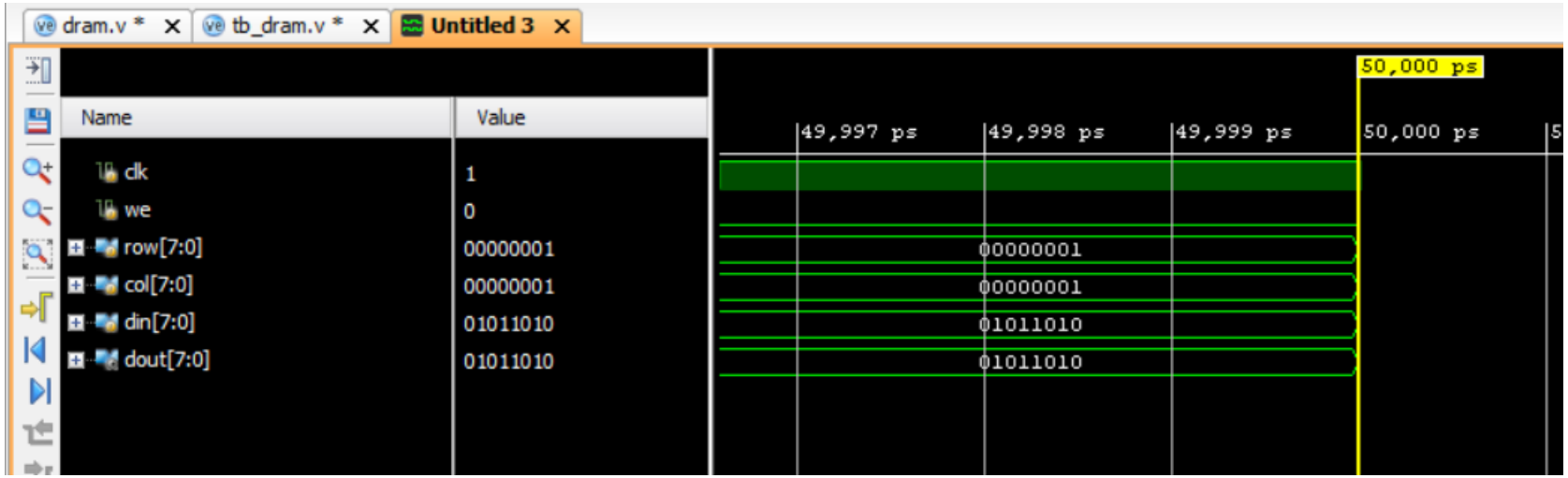
Verilog code for DRAM:

```
module dram(
    input wire clk,          // Clock signal
    input wire we,           // Write Enable
    input wire [7:0] row,    // Row address (8-bit)
    input wire [7:0] col,    // Column address (8-bit)
    input wire [7:0] din,    // Data input (8-bit)
    output reg [7:0] dout    // Data output (8-bit)
);

// Memory declaration (256 rows x 256 columns, 8-bit data)
reg [7:0] mem [255:0][255:0];

// Internal signals for storing the row and column address
reg [7:0] current_row;
reg [7:0] current_col;

always @(posedge clk) begin
    if (we) begin
        // Write operation to memory (select row and column)
        mem[row][col] <= din;
    end
    else begin
        // Read operation from memory (select row and column)
        dout <= mem[row][col];
    end
end
endmodule
```

```
module tb_dram();
reg clk;
reg we;
reg [7:0] row;
reg [7:0] col;
reg [7:0] din;
wire [7:0] dout;
// Instantiate DRAM module
dram uut (
    .clk(clk),
    .we(we),
    .row(row),
    .col(col),
    .din(din),
    .dout(dout)
);
always #5 clk = ~clk;// Clock generation
initial begin
    clk = 0;// Initialize signals
    we = 0;
    row = 8'd0;
    col = 8'd0;
    din = 8'd0;
    // Write 8'hA5 to row 0, column 0
    #10 we = 1; row = 8'd0; col = 8'd0; din = 8'hA5;
    // Write 8'h5A to row 1, column 1
    #10 row = 8'd1; col = 8'd1; din = 8'h5A;
    // Disable write, perform read from row 0, column 0
    #10 we = 0; row = 8'd0; col = 8'd0;
     // Read from row 1, column 1
    #10 row = 8'd1; col = 8'd1;
    #10 $stop;
end
endmodule
```


Address
N
Command
R/W
RAS
CAS
DRAM
B
Data Input/Output

Results:


dram.v *
tb_dram.v *
Untitled 3
Name
Value
clk
1
we
0
row[7:0]
00000001
col[7:0]
00000001
din[7:0]
01011010
dout[7:0]
01011010
50,000 ps
49,997 ps
49,998 ps
49,999 ps
50,000 ps
00000001
00000001
01011010
01011010

Single Port RAM:

Single-Port RAM (Random Access Memory) is a memory module that allows access to only one memory location (either read or write) at a time through a single address and data bus. Unlike dual-port RAM, where two different addresses can be accessed simultaneously, single-port RAM has a single address line, meaning it performs either a read or a write operation during a single clock cycle.

Components:

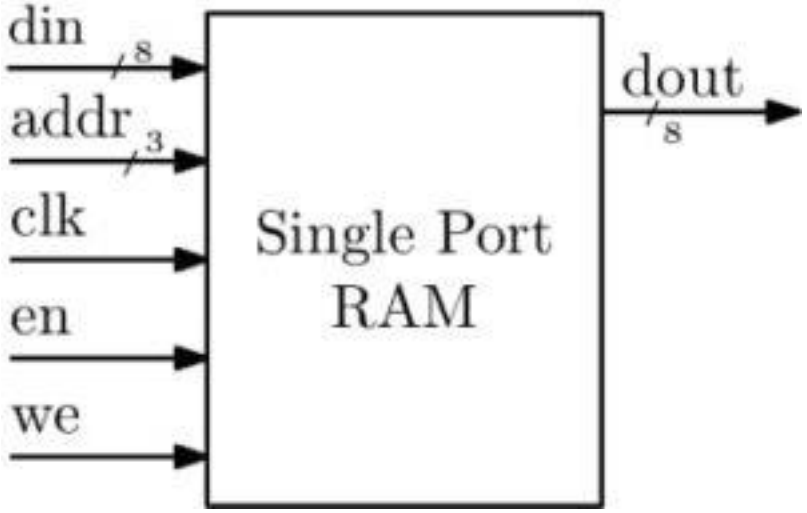
Clock (clk): Controls when data is read or written.

Write Enable (we): Determines if data is written to memory during a clock cycle.

Address Bus (addr): Specifies the memory location for reading or writing.

Data Input (data_in): Carries the data to be written into memory.

Data Output (data_out): Holds the data being read from memory.



Block Diagram Single-Port RAM

Working of Single-Port RAM:

1.Clock Cycle Synchronization:

•The operations are synchronized with the clock. On every positive edge of the clock, either a read or write operation takes place.

2.Write Operation:

•If the we (write enable) signal is high during a clock cycle, the data present on data_in is stored in the memory at the address specified by the addr bus.

3.Read Operation:

•If the we signal is low, the data stored at the address on the addr bus is placed on the data_out bus, allowing it to be read.

Applications:

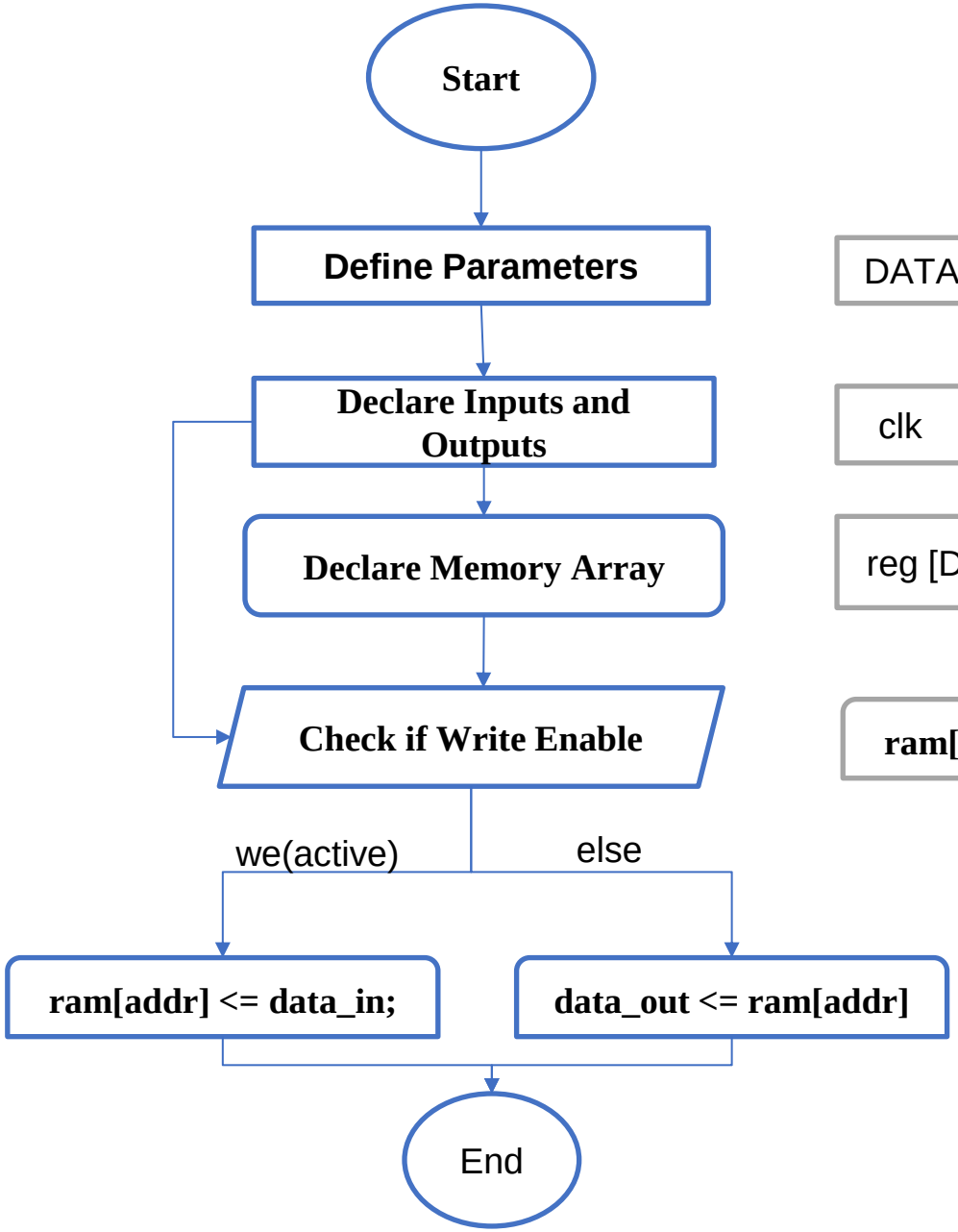
•**Cache Memory**: Used in simple cache memory implementations where only one cache line can be accessed at a time.

•**Embedded Systems**: Common in microcontroller and FPGA-based designs to store data.

•**Buffer Storage**: Used to store temporary data in small processing systems.

Verilog Code:

```
module single_port_ram #(
    parameter DATA_WIDTH = 8,    // width of data in bits
    parameter ADDR_WIDTH = 4     // width of address bus
)(
    input wire clk,                 // clock signal
    input wire we,                  // write enable
    input wire [ADDR_WIDTH-1:0] addr,  // address bus
    input wire [DATA_WIDTH-1:0] data_in, // input data
    output reg [DATA_WIDTH-1:0] data_out // output data
);

    // Declare the memory array (2^ADDR_WIDTH locations)
    reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];

    always @(posedge clk) begin
        if (we) begin
            // Write operation
            ram[addr] <= data_in;
        end
        // Read operation
        data_out <= ram[addr];
    end
endmodule
```



DATA_WIDTH

ADDR_WIDTH

clk

we

addr

data_in

data_out

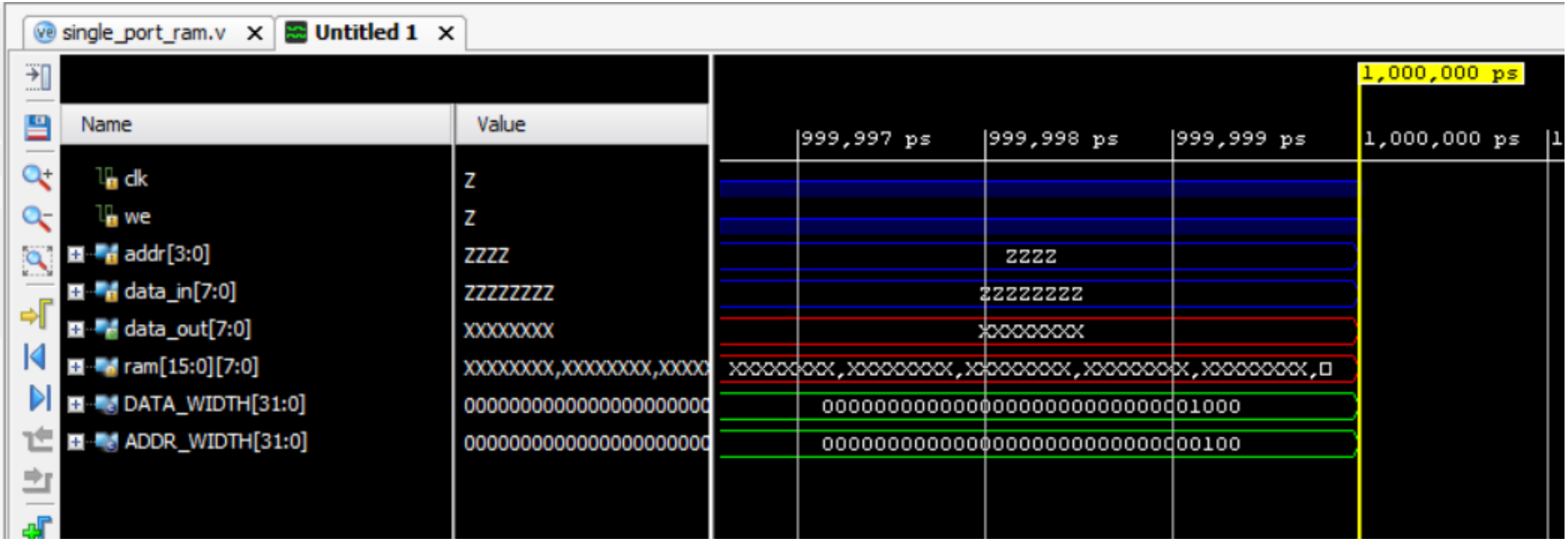
reg [DATA_WIDTH-1:0]

ram [2**ADDR_WIDTH-1:0]

ram[addr] <= data_in;

ram [2ADDR_WIDTH-1:0];**

Results:


single_port_ram.v
Untitled 1
Name
Value
clk
Z
we
Z
addr[3:0]
ZZZZ
data_in[7:0]
ZZZZZZZZ
data_out[7:0]
XXXXXXXX
ram[15:0][7:0]
DATA_WIDTH[31:0]
ADDR_WIDTH[31:0]
1,000,000 ps
999,997 ps
999,998 ps
999,999 ps
1,000,000 ps
ZZZZ
ZZZZZZZZ
XXXXXXXX
01000
00100

Schematic Diagram: