Design and Implementation of a RAM and ROM Using Verilog

Samuel Singaram

CONTENTS

1. Introduction

This project demonstrates the implementation of Single-Port RAM and Dual-Port RAM using Verilog HDL. RAM (Random Access Memory) is a type of volatile memory used to store data that can be accessed or modified. The design includes both single-port and dual-port implementations, providing insight into how memory systems handle read and write operations. The project also utilizes test benches to validate the functionality of the RAM modules through simulations.

RAM and ROM

Random Access Memory (RAM):

Definition:

RAM is a type of volatile memory that temporarily stores data for quick access by a processor. It loses its data when power is turned off.

Types of RAM:

Static RAM (SRAM):

Data is stored using flip-flops.

Faster but more expensive than DRAM.

Dynamic RAM (DRAM):

Data is stored as charges in capacitors.

Slower but cheaper and used in most computer systems.

Key Characteristics:

High-speed read and write operations.

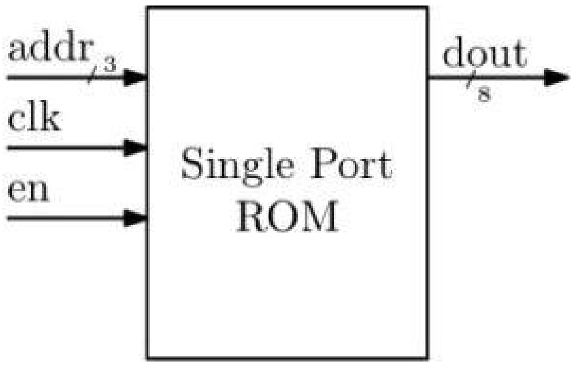
Addressable memory locations for random access.

Typically used as primary memory in systems like computers, microcontrollers, and embedded devices.

Read-Only Memory (ROM):

Definition:

ROM is non-volatile memory that stores data permanently. Data cannot be modified after it is written (or only with special programming tools).



Types of ROM:

PROM (Programmable ROM): Data is written once and cannot be changed.

EPROM (Erasable PROM): Data can be erased using UV light and reprogrammed.

EEPROM (Electrically Erasable PROM): Data can be erased and rewritten electrically.

Applications:

ROM is used to store firmware, boot loaders, and critical system data.

Differences Between RAM and ROM:

Feature	RAM	ROM
Volatility	Volatile (data lost on power-off)	Non-volatile (data retained)
Usage	Temporary data storage	Permanent data storage
Write Capability	Read and Write	Mostly Read-only

Feature	RAM	ROM
Speed	Faster	Slower
Cost	Expensive	Cheaper

Applications in Digital Systems:

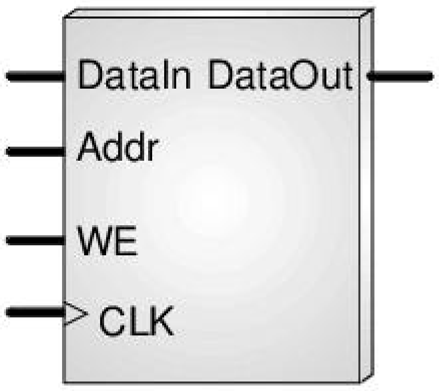
RAM is used for temporary storage in operations like caching, buffering, and processing.

ROM is used for permanent storage of boot firmware, lookup tables, and configuration data.

2. Block Diagram

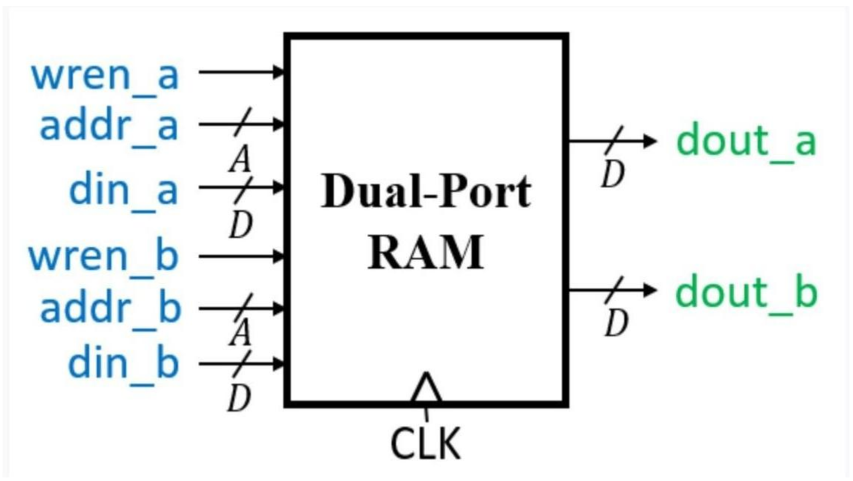
The RAM design can be visualized as follows:

Single-Port RAM:



A single port handles both read and write operations based on a write-enable signal (we).

Dual-Port RAM:



Two independent ports (Port A and Port B) handle simultaneous read/write operations.

3. Functional Operation

· Single-Port RAM:

· Read/Write operations occur on a single port.

Data is written to the memory location specified by the address (addr) when we is high.

Data is read from the memory when we is low.

· Dual-Port RAM:

· Operates with two ports, Port A and Port B.

Each port can independently read or write data based on its write-enable signal (we_a or we_b).

4. About Language Used

The project is implemented using **Verilog HDL (Hardware Description Language)**, a popular language for designing and simulating digital circuits. Verilog provides a concise and flexible framework for defining hardware behavior, enabling the modeling of combinational and sequential logic with ease.

Key Features of Verilog:

Structural and Behavioral Modeling:

Verilog supports both low-level structural descriptions and high-level behavioral descriptions, making it suitable for a wide range of applications.

Event-Driven Simulation:

Verilog uses an event-driven simulation model, allowing for the precise analysis of circuit behavior over time.

Concurrency:

It allows concurrent execution of statements, reflecting real-world hardware behavior where multiple components operate simultaneously.

Parameterized Designs:

Verilog allows parameterization, making designs more reusable and scalable.

5. Verilog Design Code

```
//single port RAM

 module single_port_ram(

    input [7:0] data,

    input [5:0] addr,

    input clk,

    input we,

    output reg [7:0] dout

);

    reg [7:0] mem[63:0];

    always @(posedge clk) begin

        if (we)

            mem[addr] <= data;
        else
            dout <= mem[addr];
    end
endmodule
//dual port RAM
module dual_port_ram(
    input [7:0] data_a, data_b,
    input [5:0] addr_a, addr_b,
    input we_a, we_b, clk,
    output reg [7:0] q_a, q_b
);
    reg [7:0] mem[63:0];
    always @(posedge clk) begin
        if (we_a)
            mem[addr_a] <= data_a;
        else
            q_a <= mem[addr_a];
    end
    always @(posedge clk) begin
        if (we_b)
            mem[addr_b] <= data_b;
        else
            q_b <= mem[addr_b];
    end
endmodule
```

```
//ROM
module rom (
   input [5:0] addr,   // 6-bit address (64 locations)
   output reg [7:0] data // 8-bit data output
);
   reg [7:0] mem[63:0]; // ROM memory with 64 locations of 8 bits each

   initial begin
      // Preload ROM with data
      mem[0] = 8'hAA;
      mem[1] = 8'hBB;
      mem[2] = 8'hCC;
      mem[3] = 8'hDD;
      mem[4] = 8'hEE;
      mem[5] = 8'hFF;
      // Initialize other locations as needed
   end

   always @(*) begin
      data = mem[addr]; // Continuous read operation
   end
endmodule
```

6. Test Bench Code

```
//single port RAM tb
module single_port_ramtb;
   reg [7:0] data;
   reg [5:0] addr;
   reg we, clk;
   wire [7:0] dout;

   single_port_ram spr1(.data(data), .addr(addr), .we(we), .clk(clk), .dout(dout));

   initial begin
      $dumpfile("dump.vcd");
      $dumpvars(1, single_port_ramtb);
      clk = 1'b1;
      forever #5 clk = ~clk;
   end

   initial begin
      data = 8'h01; addr = 5'd0; we = 1'b1; #10;
      data = 8'h02; addr = 5'd1; #10;
      data = 8'h03; addr = 5'd2; #10;
      addr = 5'd0; we = 1'b0; #10;
      addr = 5'd1; #10;
```

```
    end

endmodule



//Dual port RAM tb

module dual_port_ram_tb;

    reg [7:0] data_a, data_b;

    reg [5:0] addr_a, addr_b;

    reg we_a, we_b, clk;

    wire [7:0] q_a, q_b;

    dual_port_ram
uut(.data_a(data_a), .data_b(data_b), .addr_a(addr_a), .addr_b(addr_b), .we_a(we_a), .we_b(w
e_b), .clk(clk), .q_a(q_a), .q_b(q_b));

    initial begin

        $dumpfile("dump.vcd");

        $dumpvars(1, dual_port_ram_tb);

        clk = 1'b1;

        forever #5 clk = ~clk;

    end



    initial begin

        data_a = 8'h33; addr_a = 6'h01; data_b = 8'h44; addr_b = 6'h02; we_a = 1'b1; we_b = 1'b1;
#10;

        data_a = 8'h55; addr_a = 6'h03; addr_b = 6'h01; we_b = 1'b0; #10;

        addr_a = 6'h02; addr_b = 6'h03; we_a = 1'b0; #10;

    end

endmodule
```

```
//ROM tb
module rom_tb;
    reg [5:0] addr; // 6-bit address input
    wire [7:0] data; // 8-bit data output

    rom uut (
        .addr(addr),
        .data(data)
    );

    initial begin
        $dumpfile("rom.vcd");
        $dumpvars(1, rom_tb);

        // Test cases
        addr = 6'd0; #10; // Read from address 0
        addr = 6'd1; #10; // Read from address 1
        addr = 6'd2; #10; // Read from address 2
        addr = 6'd3; #10; // Read from address 3
        addr = 6'd4; #10; // Read from address 4
        addr = 6'd5; #10; // Read from address 5

        $finish;
    end
endmodule
```

7. References

· Samir Palnitkar, *"Verilog HDL: A Guide to Digital Design and Synthesis."*

· IEEE Standard for Verilog Hardware Description Language.

· Online resources on FSM design and Verilog simulation(youtube etc..).