Design and Verification of Dual-Port RAM using SystemVerilog

(by CH.Anirudh)

1. Introduction:

In modern digital systems, efficient memory access is crucial for performance. **Dual-Port RAM (DPRAM)** is a type of memory that allows simultaneous read and write operations on two independent ports. Unlike single-port RAM, where only one operation (read/write) can occur at a time, **DPRAM enhances parallel processing** by enabling two processes to access memory concurrently.

This project implements a **synthesizable Dual-Port RAM in Verilog/SystemVerilog**, supporting simultaneous **read/write operations** with independent address buses, data buses, and control signals. The design is highly efficient for **high-speed computing applications, FPGA designs, and embedded systems.**

Through this implementation, we explore **memory arbitration techniques, conflict resolution strategies, and FPGA-based synthesis** to optimize DPRAM performance. The project includes **a fully functional testbench** for verification, ensuring correct read/write operations and data consistency.

2. Overview

Dual-Port RAM is essential in **multithreading, parallel processing, video processing, and networking hardware** where multiple processes need fast memory access. This project provides a parameterized DPRAM module, making it scalable for different memory sizes and architectures.

Key Features:

- Simultaneous Read/Write on two independent ports
- Asynchronous and Synchronous Support
- Conflict Resolution for Simultaneous Writes
- Scalable Design with Parameterized Address and Data Width
- Fully Verified Testbench in SystemVerilog

3. Difference Between Single-Port RAM and Dual-

3.1 Port RAM in Design and Verification

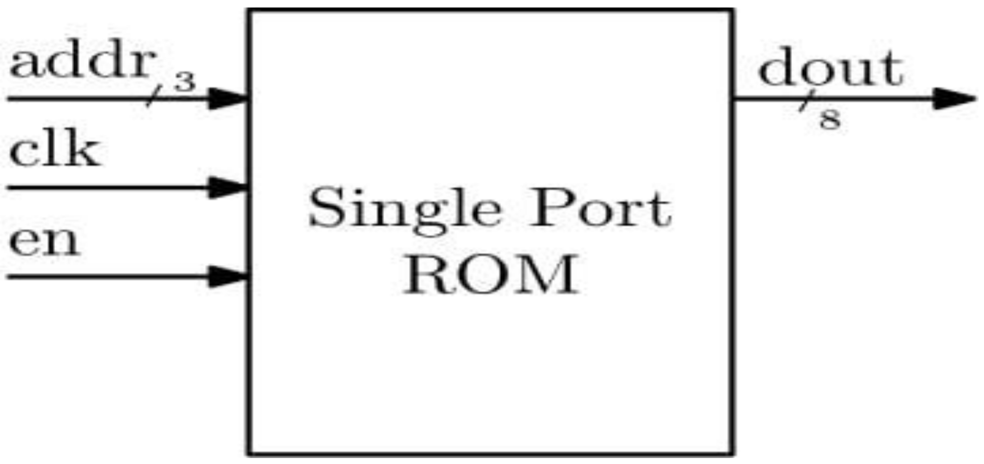
Single-Port RAM Design and Verification

A **Single-Port RAM** allows only one operation (either read or write) at a time. It consists of a **single address bus**, a **single data bus**, and a **control signal** (read/write enable). When a read or write request is made, the RAM processes it sequentially, meaning no two operations can occur at the same time.

From a **design perspective**, single-port RAM is simpler, requiring fewer control signals and arbitration logic. It is typically implemented using **a single register array** and controlled using a clock-driven read/write mechanism.

During **verification**, the testbench primarily checks sequential access to memory, ensuring that:

- Data is written correctly and can be read back without corruption.
- The read and write operations do not overlap.
- Memory integrity is maintained throughout multiple clock cycles.



Fig(a): Single-Port RAM

3.2 Dual-Port RAM Design and Verification

A **Dual-Port RAM**, unlike single-port RAM, allows simultaneous **read and write** operations on two independent address buses. This means one port can be **writing data** while the other port is **reading from** or writing to a different location in the **memory**. This is useful in high-speed applications where parallelism is required.

From a design perspective, dual-port RAM includes **two address buses, two data buses, and two independent control signals**. Additional logic is required for handling simultaneous writes to the same memory **location** (conflict resolution). Arbitration techniques such as **priority**-based write handling or **clock cycle** separation are often implemented.

During verification, the testbench must ensure:

- Both ports can read and write without interfering with each other.
- Conflict resolution logic properly handles simultaneous writes.

- Data consistency is maintained even when accessed from two different locations.
- Performance under high-speed access patterns is validated.

Thus, single-port RAM is simpler but has access limitations, while dual-port RAM is more complex but significantly improves parallel memory operations.

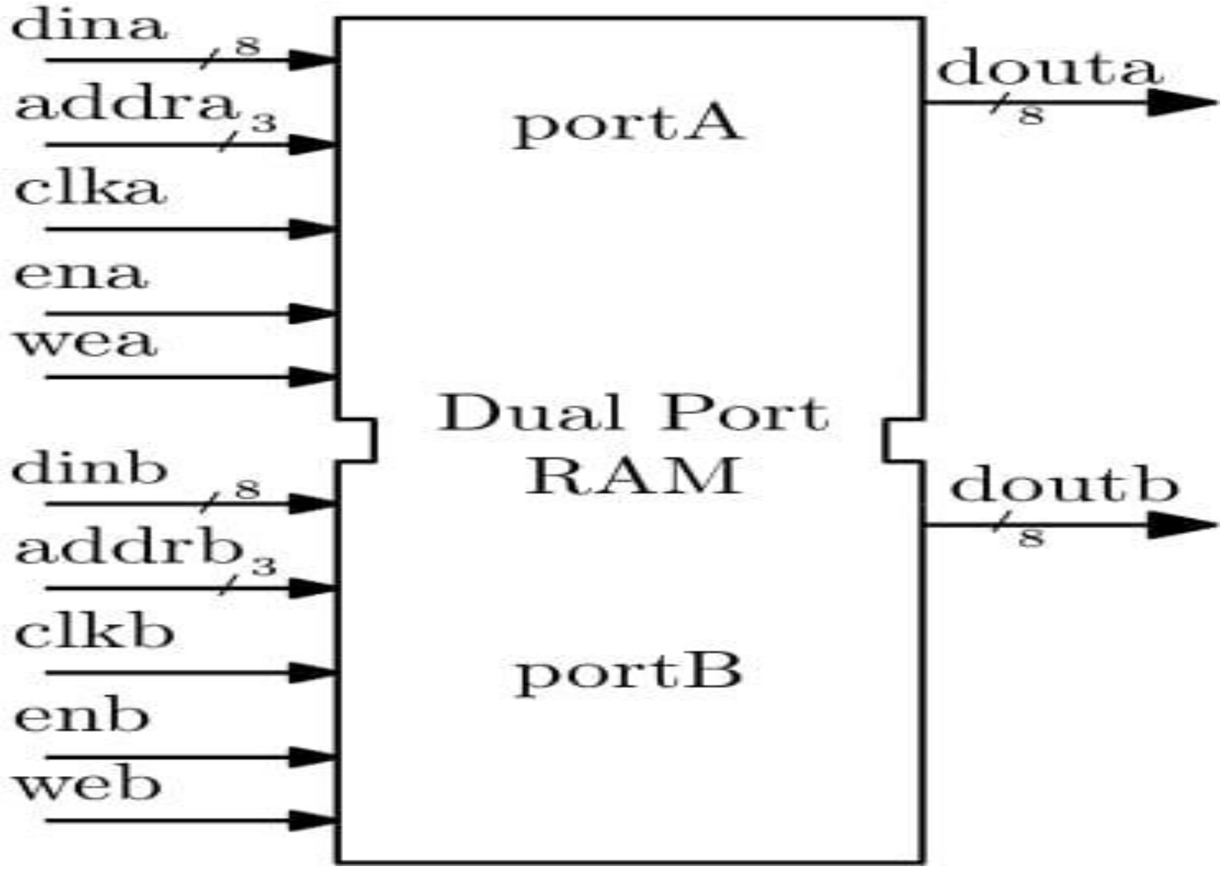


Fig (b): Dual-Port RAM

4.Design & Implementation of Dual-Port RAM

The **Dual-Port RAM** is designed using Verilog/SystemVerilog, with two separate **read/write ports** that can access memory simultaneously. The design includes:

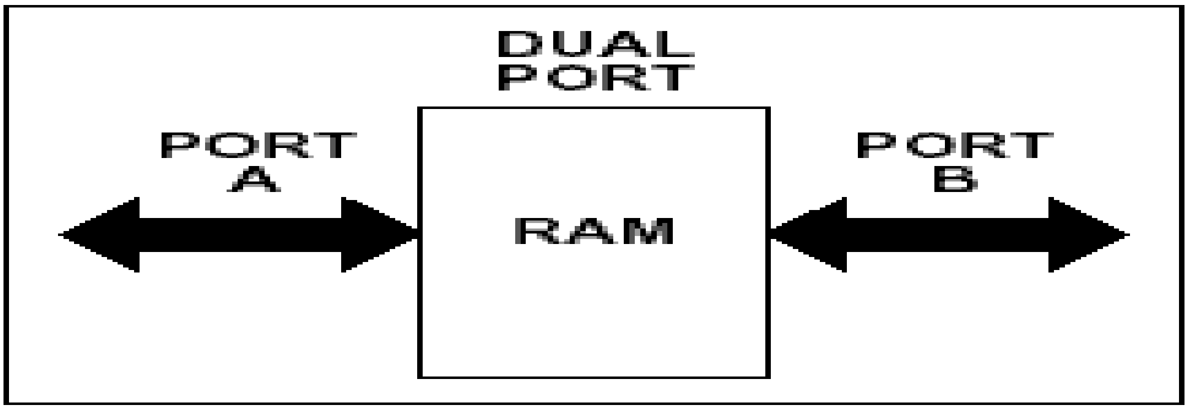
Implementation Highlights:

- **Memory Array Implementation:** reg [DATA_WIDTH-1:0] memory [0:MEM_DEPTH-1];
- **Two Independent Ports:** Separate address, data, and control line
- **Conflict Handling:** Priority-based arbitration for write conflicts

- **Synchronous & Asynchronous Read Support**

Functional Block Diagram:

1□**Port A** → Read/Write Access
2□**Port B** → Read/Write Access
3□**Arbitration Logic** → Resolves conflicts
4□**Memory Array** → Stores data



Fig(c): Dual-Port

5. SystemVerilog Implementation:

RTL code:

```
module dual_port_ram2#(

    parameter int DATA_WIDTH = 8,  // data width per memory allocation

    parameter int ADDR_WIDTH = 4)  // address width (total locations = 2^ ADDR_WIDTH)

    (

 input logic clk,    //clock signal

 input logic rst,   // reset signal

 input logic we_a,we_b, // write enable signals for port A and PORT B

 input logic [ADDR_WIDTH-1:0] addr_a,addr_b,    // adrress inputs

 input logic [DATA_WIDTH-1:0] din_a,din_b,      // data inputs for write operations

 output logic [DATA_WIDTH-1:0] dout_a,dout_b    // dtat inputs for read operations

 );

  // memory declaration: 2D array with 2^ ADDR_WIDTH locations, each storing DATA_WIDTH bits


  logic [DATA_WIDTH-1:0] mem [0:(1<<ADDR_WIDTH)-1];

  //pipe line registers for read operations(improves the performance)
```

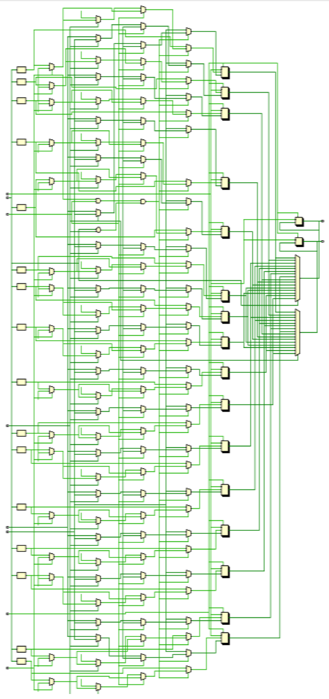
```
logic [DATA_WIDTH-1:0] dout_a_reg,dout_b_reg;

//conflict resolution flag

logic conflict;

assign dout_a = mem[addr_a];

assign dout_b = mem[addr_b];

//conflict detection (both ports writing the same address)

assign conflict = we_a && we_b && (addr_a == addr_b);

//sequential logic for write operation and conflict resolution


always_ff @(posedge clk or posedge rst) begin

 if (rst) begin

  // Initialize memory and outputs on reset

      for (int i = 0; i < (1 << ADDR_WIDTH); i++)

          mem[i] <= '0;

  dout_a_reg <= '0;

  dout_b_reg <= '0;

 end else begin

   if (conflict) begin  //if both ports writes a same address, prioritize portA

    mem[addr_a] <= din_a;

   end else begin

    if(we_a) mem[addr_a] <= din_a; // write by port A

    if(we_b) mem[addr_b] <= din_b; // write by port B

   end

    dout_a_reg <= mem[addr_a]; //pipelined read for port A

    dout_b_reg <= mem[addr_b]; //pipelined read for port b

 end

end

//assign pipelined outputs

assign dout_a = dout_a_reg;

assign dout_b = dout_b_reg;



endmodule
```

6. Testbench Simulation:

```
module dual_port_ram2_tb;
  //parameters
      parameter int DATA_WIDTH = 8;
      parameter int ADDR_WIDTH = 4;
      // signals
   logic clk;
   logic rst;
   logic we_a, we_b;
   logic [ADDR_WIDTH-1:0] addr_a,addr_b;
   logic [DATA_WIDTH-1:0] din_a,din_b;
   logic [DATA_WIDTH-1:0] dout_a,dout_b;
   //instantiate the dual port ram module
    dual_port_ram2 #(
    .DATA_WIDTH(DATA_WIDTH),
    .ADDR_WIDTH(ADDR_WIDTH)
    )uut(
    .clk(clk),
    .rst(rst),
    .we_a(we_a),
    .we_b(we_b),
    .addr_a(addr_a),
    .addr_b(addr_b),
    .din_a(din_a),
    .din_b(din_b),
    .dout_a(dout_a),
    .dout_b(dout_b)
    );
    //clock generation
        always begin
     #5 clk =~clk;   //toggle clk every 5 time units
    end
    //test sequence
    initial begin
      clk = 0;
      rst = 0;
      we_a = 0;
      we_b = 0;
      addr_a = 0;
      addr_b = 0;
          din_a = 0;
      din_b = 0;
     //apply reset
     $display("applying reset...");
     rst = 1;  //asserting the reset
     #10;
      rst = 0;  //deasserting the reset
      //test write operation-port A
     $display("writing to port A...");
     we_a = 1; we_b = 0; //enable port A only
```

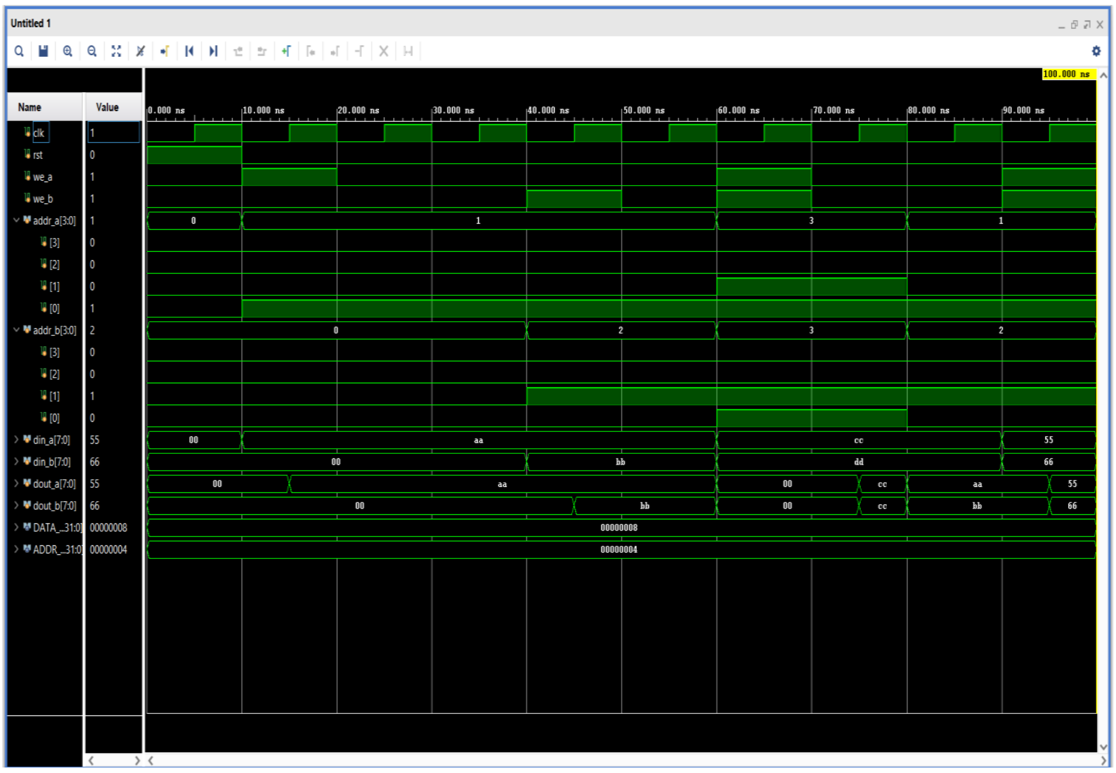
```
        addr_a = 4'b0001;   // write address
        din_a = 8'hAA;     // data to write
        #10;
        we_a = 0;         //disable write
        #10;
        $display ("dout_a = %h (expected:00)",dout_a);    //read the value back, should be zero initially
        #10;
        //test write operation port-B
        $display ("writing to port B...");
              we_a = 0; we_b = 1;     // enable write for port B only
              addr_b = 4'b0010;       //write address
              din_b = 8'hBB;         //data to write
              #10;
              we_b = 0;              //disable write
              #10;
         $display ("dout_b = %h (expected:00)", dout_b);  //read the value back, should be zero initially

         //test the simultaneous write(conflict resolution)
         $display("simultaneous write to same adress...");
         we_a = 1; we_b = 1;   //enable write to both ports
         addr_a = 4'b0011;     // same address for both
                  addr_b = 4'b0011;     //same address for both
         din_a = 8'hCC;      //port A wires to hCC
         din_b = 8'hDD;      //port B wires to hDD
         #10;
         we_a = 0; we_b = 0;  //disable wires
         #10;
         $display("memory at addr 3 = %h (expected: CC)", uut.mem[3]); //check value at adress 3,should
becc at prioriy of port A
          //test asynchronous read-port A and B
          $display("reading asynchronously...");
          addr_a = 4'b0001;   //set address for port A
          addr_b = 4'b0010; //set address for port B
          #10;
          $display("dout_a = %h (expected: AA), dout_b = %h(expected: BB)", dout_a,dout_b); //should
dhoe data earlier to those adsress
          //test pipelined access
          $display ("testing pipelined access...");
          addr_a = 4'b0001;
          addr_b = 4'b0010;
          din_a = 8'h55;
          din_b = 8'h66;
          we_a = 1; we_b  = 1;
          #10;
          $display("dout_a = %h (expected: 55),dout_b = %h (expected: 66)", dout_a,dout_b); //output
should the data written with one clock delay
          //end simulation
          $finish;
      end
endmodule
```

7.Elaboration Design:



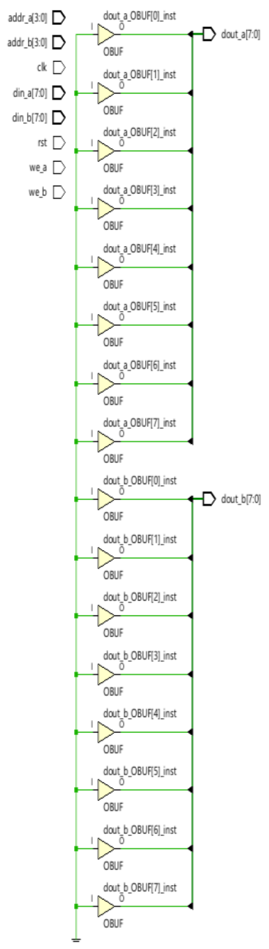
Fig(d): elaborated design

8.Simulation:



Fig(e): Simulation Design

9.Synthesis:



Fig(F): Synthesis design

10. Advantages:

- **High Performance** – Enables parallel processing by supporting concurrent memory accesses.
- **Reduced Latency** – Eliminates bottlenecks by allowing independent read/write operations.
- **Improved FPGA Resource Utilization** – Optimized for real-time embedded applications.

Disadvantages:

- **Increased Complexity** – Requires additional logic for arbitration and conflict resolution.

- **Higher Power Consumption** – Simultaneous operations increase dynamic power usage.
- **Resource Intensive** – Utilizes more FPGA resources compared to single-port RAM.

11. Applications:

- **FPGA-Based Processors** – Used in multi-core architectures for fast data sharing.
- **Networking & Communication Systems** – Enhances router and switch performance.
- **Video Processing** – Supports parallel pixel processing for real-time applications.
- **Embedded Systems** – Efficient memory management for microcontroller-based designs.

12. Conclusion:

The **Dual-Port RAM** is a powerful memory design that enhances system efficiency by enabling simultaneous data access. Its implementation in Verilog/SystemVerilog makes it suitable for **FPGA-based applications, multi-threaded processors, and high-speed digital circuits**. By incorporating conflict resolution strategies and parameterized configurations, this project provides a **scalable, optimized, and functional memory solution**.

---The End---