ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KĨ THUẬT MÁY TÍNH

BÀI BÁO CÁO ASSIGNMENT LAB04
THIẾT KẾ HỆ THỐNG SỐ VỚI HDL

Sinh viên: Trương Thiên Quý
MSSV: 23521321
Lớp: CE213.P21
Giảng viên hướng dẫn: Hồ Ngọc Diễm

BÀI THỰC HÀNH 4

KIỂM TRA THIẾT KẾ SỬ DỤNG TESTBENCH

I. Mục tiêu

Sinh viên làm quen với việc kiểm tra thiết kế bằng phương pháp viết Testbench và sử dụng phần mềm ModelSim-Altera để kiểm tra thiết kế.

II. Chuẩn bị thực hành

- Sinh viên đọc trước và thực hành sử dụng phần mềm ModelSim-Altera trong file "huong dan su dung phan mem ModelSim-Altera.pdf"
- Sinh viên phải chuẩn bị các phần được yêu cầu trong mỗi câu của bài Lab và nộp vào đầu buổi học.
- Điểm bài chuẩn bị được tính vào điểm bài báo cáo của Lab.

III. Nội dung thực hành

Câu 1.

Sử dụng ngôn ngữ Verilog HDL, thiết kế bộ ALU 32-bit có các chức năng như Hình 4-1.

- Viết testbench tạo giá trị cho các tín hiệu input và quan sát kết quả sử dụng các thủ tục **$display** và **$monitor**, chạy mô phỏng kiểm tra chức năng của thiết kế dùng phần mềm ModelSim-Altera.

M	S_1	S_0	ALU Operations
0	0	0	Complement A
0	0	1	AND
0	1	0	EX-OR
0	1	1	OR
1	0	0	Decrement A
1	0	1	Add
1	1	0	Subtract
1	1	1	Increment A

Hình 4-1. Chức năng của ALU

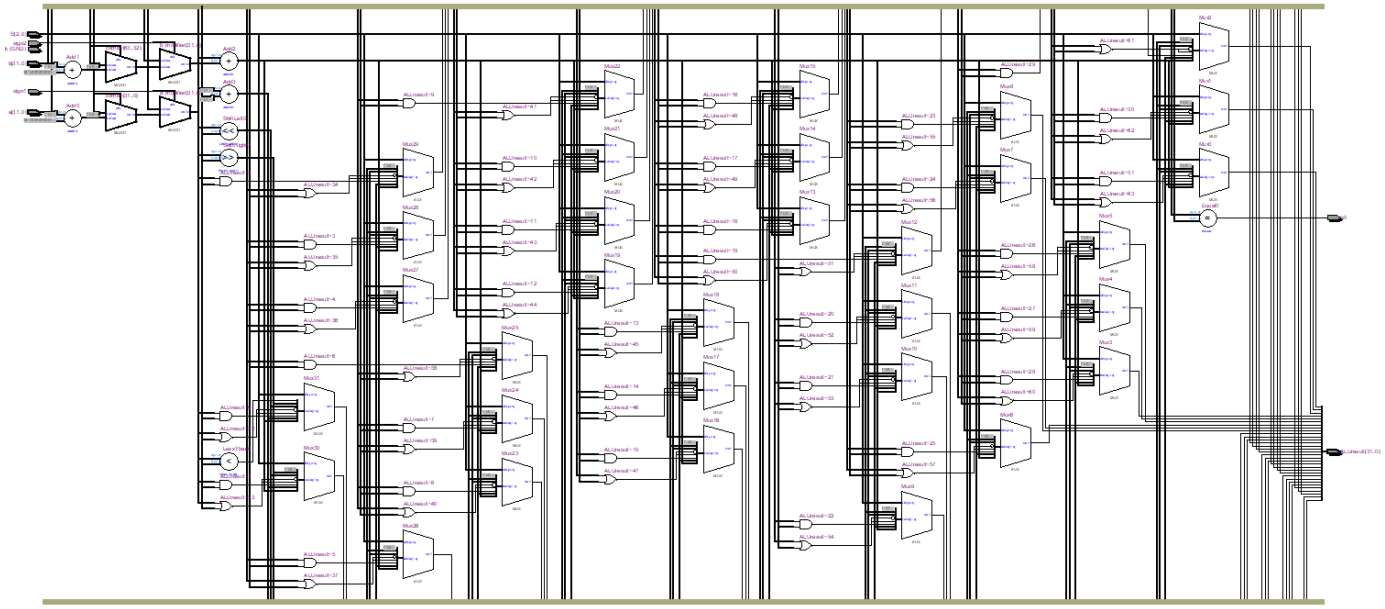
Code thực thi:

```
module ALU32bit(
    input clk,
    input [31:0] a,
    input [31:0] b,
    input sign1,
    input sign2,
    input [2:0] S,
    output reg [31:0] ALUresult,
    output reg is0
);
    // Biến a và b sẽ được tính toán lại dựa trên sign1, sign2
    wire [31:0] a_modified = (sign1 == 1'b1) ? (a[31] ? -a : a) : a;
    wire [31:0] b_modified = (sign2 == 1'b1) ? (b[31] ? -b : b) : b;

    always @(*) begin
        case(S)
            3'b000: ALUresult <= a_modified + b_modified;          // Cộng
            3'b001: ALUresult <= a_modified - b_modified;          // Trừ
            3'b010: ALUresult <= a_modified & b_modified;          // AND
            3'b011: ALUresult <= a_modified | b_modified;          // OR
            3'b100: ALUresult <= ~(a_modified | b_modified);       // NOR
            3'b101: ALUresult <= a_modified << b_modified[4:0];    // Dịch trái
            3'b110: ALUresult <= a_modified >> b_modified[4:0];    // Dịch phải
            3'b111: ALUresult <= (a_modified < b_modified) ? 32'd1 : 32'd0;  // Kiểm tra nhỏ hơn
            default: ALUresult <= 32'd0;                           // Mặc định là 0
        endcase

        is0 <= (a == b);  // Kiểm tra nếu a == b
    end
endmodule
```

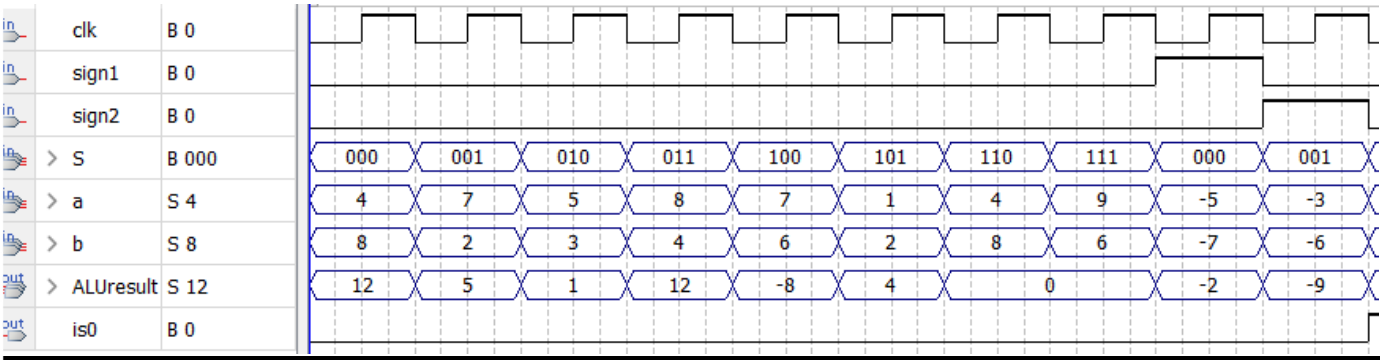
'estbench:

```
`timescale 1ns/1ps

module ALU32bit_tb;
    reg [31:0] a, b;
    reg [2:0] S;
    reg sign1, sign2;
    wire [31:0] ALUresult;
    wire is0;

    ALU32bit uut (
        .a(a),
        .b(b),
        .sign1(sign1),
        .sign2(sign2),
        .S(S),
        .ALUresult(ALUresult),
        .is0(is0)
    );

    initial begin
        $monitor("Time=%0t | S=%b | a=%0d | b=%0d | sign1=%b | sign2=%b | Result=%0d | is0=%b",
                 $time, S, a, b, sign1, sign2, ALUresult, is0);

        sign1 = 0; sign2 = 0;

        S = 3'b000; a = 4; b = 8; #10;
        S = 3'b001; a = 7; b = 2; #10;
        S = 3'b010; a = 8; b = 3; #10;
        S = 3'b011; a = 5; b = 4; #10;
        S = 3'b100; a = 7; b = 6; #10;
        S = 3'b101; a = 1; b = 2; #10;
        S = 3'b110; a = 4; b = 8; #10;
        S = 3'b111; a = 9; b = 6; #10;

       sign1 = 1; sign2 = 1;
       S = 3'b000; a = -5; b = -7; #10;
       S = 3'b001; a = -3; b = -6; #10;

       $finish;
    end
endmodule
```

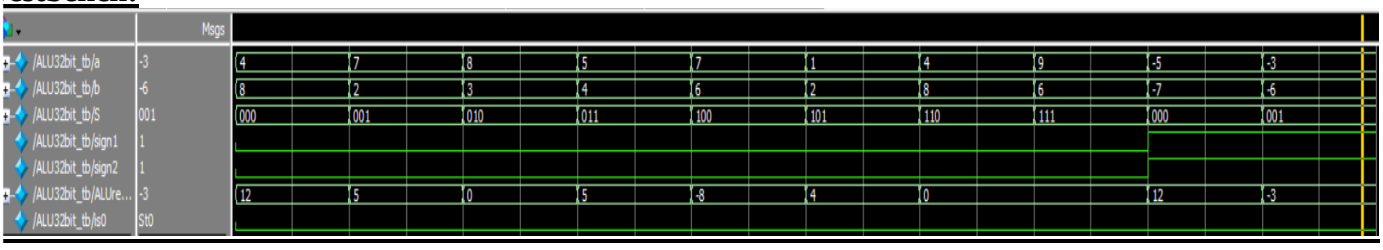
RTL viewer:



Mô phỏng Wareform:



Testbench:



Giải thích:

. Xử lý hai đầu vào a và b tùy theo yêu cầu unsigned

- sign1 điều chỉnh a
- sign2 điều chỉnh b

Nếu sign1 = 1, ta coi giá trị a là **không dấu** (unsigned):

- Nếu a đang âm → lấy giá trị dương tương ứng (đổi dấu).
- Nếu a đã dương → giữ nguyên.

Tương tự, **sign2** cũng xử lý cho b.

→ **Mục đích:** Cho phép linh hoạt dùng a hoặc b theo dạng signed hoặc unsigned tùy theo loại lệnh.

. Thực hiện các phép toán theo mã lệnh S

Khi bạn chọn giá trị S (3 bit), bạn quyết định ALU thực hiện phép gì:

S (giá trị)	Phép toán	Ý nghĩa
0	Cộng	a_modified + b_modified
1	Trừ	a_modified - b_modified
10	AND	a_modified AND b_modified
11	OR	a_modified OR b_modified
100	NOR	Phủ định (a OR b)
101	Dịch trái	a_modified << b_modified[4:0]
110	Dịch phải	a_modified >> b_modified[4:0]
111	So sánh nhỏ hơn	nếu a_modified < b_modified, kết quả là 1, ngược lại 0

Câu 2.

Sử dụng ngôn ngữ Verilog HDL, thiết kế một tập gồm 32 thanh ghi, mỗi thanh ghi 4 byte. Tập thanh ghi (Register File) có các tín hiệu sau: ReadAddress1[4:0], ReadAddress2[4:0], WriteAddress[4:0], WriteData[31:0], ReadData1[31:0], ReadData2[31:0], ReadWriteEn.

- Viết testbench tạo giá trị cho các tín hiệu input và chạy mô phỏng kiểm tra chức năng của thiết kế.

Code thực thi:

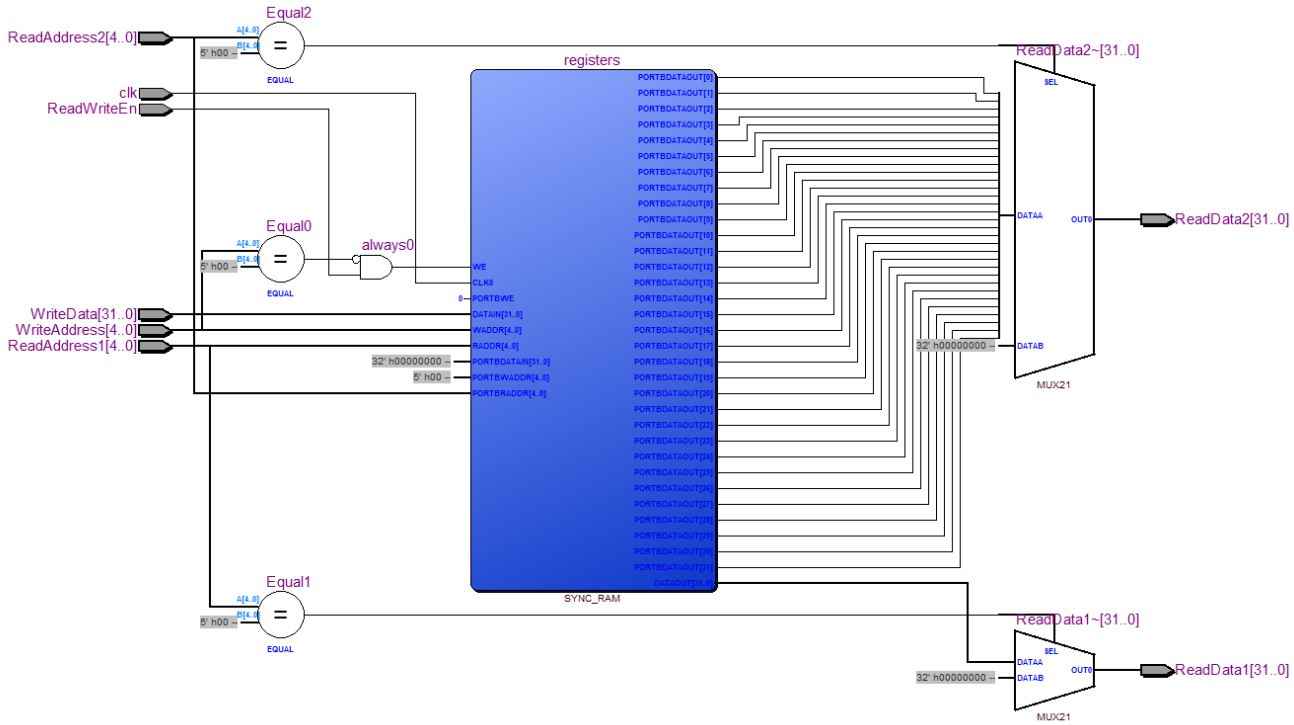
```
1  module RegisterFile(
2      input clk,
3      input ReadWriteEn,
4      input [4:0] ReadAddress1,
5      input [4:0] ReadAddress2,
6      input [4:0] WriteAddress,
7      input [31:0] WriteData,
8      output [31:0] ReadData1,
9      output [31:0] ReadData2
10 );
11     reg [31:0] registers [0:31];
12
13     // Write operation
14     always @(posedge clk) begin
15         if (ReadWriteEn && (WriteAddress != 5'd0)) begin
16             registers[WriteAddress] <= WriteData;
17         end
18     end
19
20     // Read operation
21     assign ReadData1 = (ReadAddress1 == 5'd0) ? 32'b0 : registers[ReadAddress1];
22     assign ReadData2 = (ReadAddress2 == 5'd0) ? 32'b0 : registers[ReadAddress2];
23 endmodule
```

Testbench:

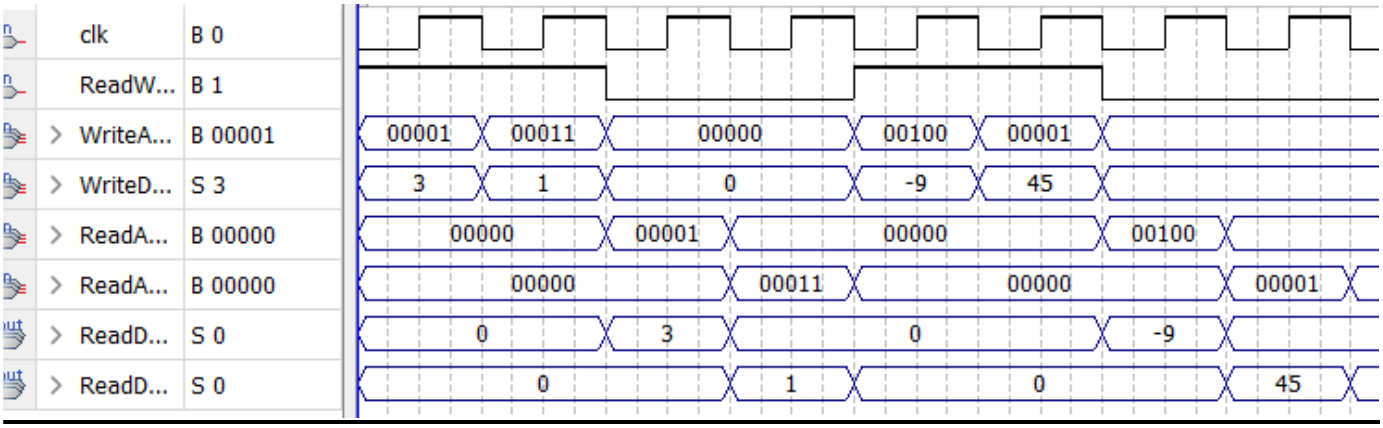
```
1   `timescale 1ns/1ps
2   module RegisterFile_tb;
3
4       reg clk;
5       reg ReadWriteEn;
6       reg [4:0] ReadAddress1;
7       reg [4:0] ReadAddress2;
8       reg [4:0] WriteAddress;
9       reg [31:0] WriteData;
10      wire [31:0] ReadData1;
11      wire [31:0] ReadData2;
12
13      RegisterFile uut (
14          .clk(clk),
15          .ReadWriteEn(ReadWriteEn),
16          .ReadAddress1(ReadAddress1),
17          .ReadAddress2(ReadAddress2),
18          .WriteAddress(WriteAddress),
19          .WriteData(WriteData),
20          .ReadData1(ReadData1),
21          .ReadData2(ReadData2)
22      );
23
24      always #5 clk = ~clk;
25
26      initial begin
27          $monitor("T=%0t | WE=%b | WAddr=%0d WData=%0d | RAddr1=%0d RData1=%0d | RAddr2=%0d RData2=%0d",
28                   $time, ReadWriteEn, WriteAddress, WriteData,
29                   ReadAddress1, ReadData1, ReadAddress2, ReadData2);
30      end
31
32      initial begin
33          clk = 0;
34          ReadWriteEn = 0;
35          WriteAddress = 0;
36          WriteData = 0;
37          ReadAddress1 = 0;
```

```
38          ReadAddress2 = 0;
39
40          #3;  ReadWriteEn = 1; WriteAddress = 5'd1; WriteData = 32'd3;
41          #10;
42
43          WriteAddress = 5'd3; WriteData = 32'd1;
44          #10;
45
46          ReadWriteEn = 0; ReadAddress1 = 5'd1;
47          #10;
48
49          ReadAddress2 = 5'd3; ReadAddress1 = 5'd0;
50          #10;
51
52          WriteAddress = 5'd4; WriteData = -9; ReadWriteEn = 1;
53          #10;
54
55          WriteAddress = 5'd1; WriteData = 32'd45; ReadWriteEn = 1;
56          #10;
57
58          ReadWriteEn = 0;
59          ReadAddress1 = 5'd4; ReadAddress2 = 5'd1;
60          #10;
61          $display("Check: R4 = %0d (expected -9), R1 = %0d (expected 45)", ReadData1, ReadData2);
62
63          $finish;
64      end
65
66  endmodule
```

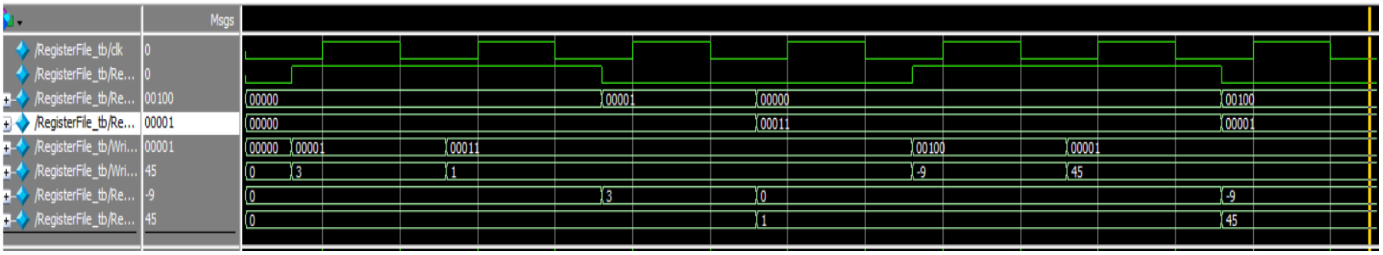
RTL viewer:



Mô phỏng Wareform:



TestBench:



Giải thích:

Module RegisterFile mà bạn viết có chức năng **quản lý 32 thanh ghi** (register), mỗi thanh ghi có kích thước **32 bit**.

Nó thực hiện hai việc chính:

- **Ghi dữ liệu** vào 1 thanh ghi khi có yêu cầu ghi.
- **Đọc dữ liệu** từ 2 thanh ghi cùng lúc.

. Ghi (Write):

- Chỉ ghi khi:
  - Có **tín hiệu clock** (posedge clk).
  - **ReadWriteEn = 1** (cho phép ghi).
  - **Địa chỉ ghi khác 0** (WriteAddress != 0) — vì register 0 phải luôn bằng 0 (theo quy định).

. Đọc (Read):

- **Luôn luôn có thể đọc** bất kỳ lúc nào.
- Đọc hai thanh ghi song song:
  - Thanh ghi ReadAddress1 ra ReadData1.

- Thanh ghi ReadAddress2 ra ReadData2.

- Nếu địa chỉ đọc là **0**, thì **luôn trả về 0**, không quan tâm bộ nhớ thật bên trong.

➔ **RegisterFile** giống như một tủ đựng 32 cái hộp, mỗi hộp 32-bit. Bạn có thể lấy ra 2 hộp bất kỳ cùng lúc, và mỗi lần nhét đồ vào 1 hộp thì phải bấm chuông đồng hồ (clock) và không được nhét vào hộp số 0.

Câu 3.

Sử dụng ngôn ngữ Verilog HDL, hiện thức thiết kế bộ nhớ dữ liệu (Data Memory) dung lượng 1024 bytes có các tín hiệu sau: Address[9:0], WriteData[7:0], ReadData[7:0], WriteEn, ReadEn và viết testbench kiểm tra chức năng trên trên phần mềm mô phỏng ModelSim.

- Viết testbench tạo giá trị cho các tín hiệu input và chạy mô phỏng kiểm tra chức năng của thiết kế.

Code thực thi:

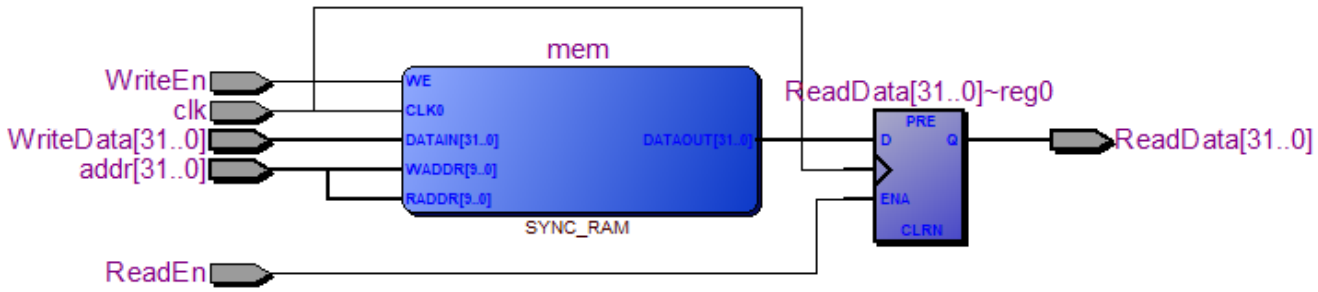
```
1  module DataMemory(
2      input clk,
3      input [31:0] addr,
4      input [31:0] WriteData,
5      output reg [31:0] ReadData,
6      input WriteEn,
7      input ReadEn
8  );
9      reg [31:0] mem [0:1023];
10 always @(posedge clk) begin
11     if(ReadEn) begin
12         ReadData <= mem[addr];
13     end
14     if(WriteEn) begin
15         mem[addr] <= WriteData;
16     end
17 end
18 endmodule
```

'estbench:

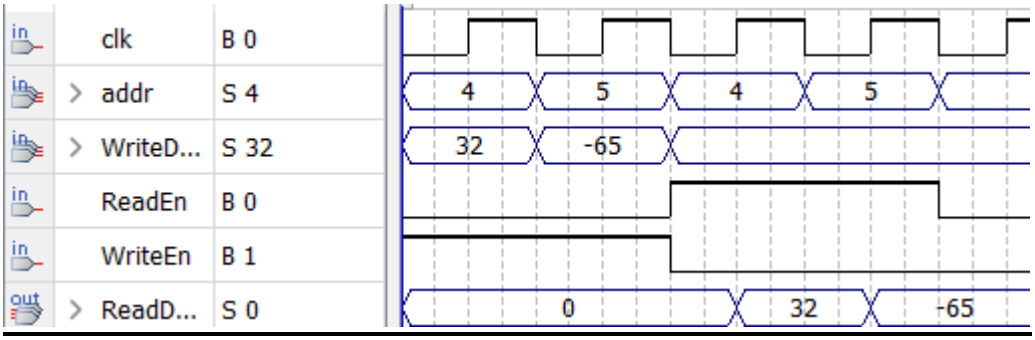
```
2
3   module DataMemory_tb();
4       reg clk;
5       reg [31:0] addr;
6       reg [31:0] WriteData;
7       wire [31:0] ReadData;
8       reg WriteEn;
9       reg ReadEn;
10
11      // G?i module chính
12      DataMemory uut(
13          .clk(clk),
14          .addr(addr),
15          .WriteData(WriteData),
16          .ReadData(ReadData),
17          .WriteEn(WriteEn),
18          .ReadEn(ReadEn)
19      );
20
21      // T?o clock 10ns (100MHz)
22      initial begin
23          clk = 0;
24          forever #5 clk = ~clk;
25      end
26
27      // Các thao tác ki?m tra
28      initial begin
29          // Kh?i t?o
30          WriteEn = 0;
31          ReadEn = 0;
32          addr = 0;
33          WriteData = 0;
34
35          // Ghi d? li?u
36          #10;
```

```
        addr = 4;
        WriteData = 32;
        WriteEn = 1;
        #10;
        WriteEn = 0;

        // Ghi d? li?u khác
        #10;
        addr = 5;
        WriteData = -65;
        WriteEn = 1;
        #10;
        WriteEn = 0;

        // ??c d? li?u t? ??a ch? 10
        #10;
        addr = 4;
        ReadEn = 1;
        #10;
        ReadEn = 0;

        // ??c d? li?u t? ??a ch? 20
        #10;
        addr = 5;
        ReadEn = 1;
        #10;
        ReadEn = 0;

        // K?t thúc mô ph?ng
        #20;
        $stop;
    end

    // Hi?n th? các giá tr? trong quá trình mô ph?ng
    initial begin
        $monitor("At time %t: addr=%d, WriteData=0x%h, ReadData=0x%h, WriteEn=%b, ReadEn=%b",
                 $time, addr, WriteData, ReadData, WriteEn, ReadEn);
    end
endmodule
```

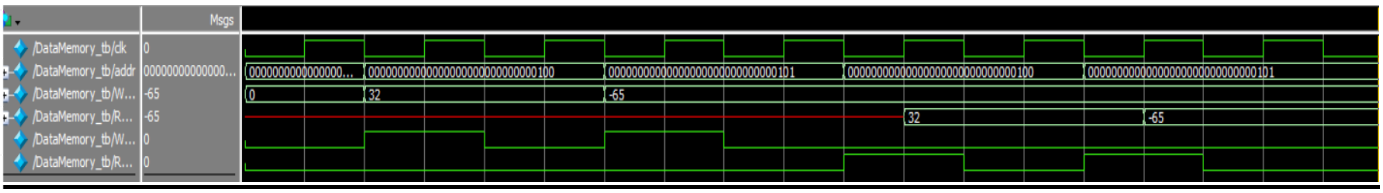
RTL viewer:



Mô phỏng Wareform:



TestBench:



Giải thích:

Module **DataMemory** này mô phỏng một **RAM** có **1024 ô nhớ**. Mỗi ô chứa **32 bit** (4 byte).

Bạn có thể:

- **Ghi dữ liệu** vào một địa chỉ cụ thể.
- **Đọc dữ liệu** từ một địa chỉ cụ thể.

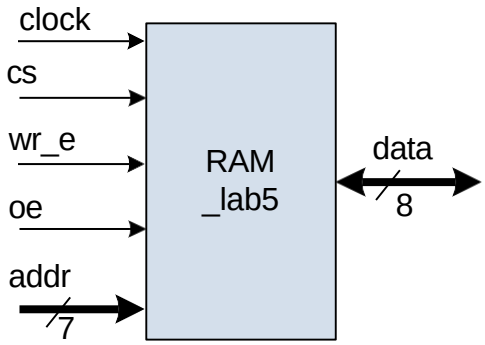
Khi có cạnh lên của clock (posedge clk):

- Nếu ReadEn = 1, thì lấy dữ liệu từ mem[addr] và gán vào ReadData.
- Nếu WriteEn = 1, thì lấy WriteData và lưu vào mem[addr].

Chú ý: Nếu vừa đọc vừa ghi cùng lúc, cả hai sẽ xảy ra **trong cùng 1 chu kỳ**, nhưng **ghi** thì sẽ ảnh hưởng đến lần đọc sau.

Câu 4.

Thiết kế một **single port RAM đồng bộ read/write** có sơ đồ như bên dưới:



Hình 4-2. Sơ đồ RAM

Biết rằng:

- **clock:** kích cạnh lên
- **cs**: chip_select
- **wr_e** = 1: cho phép ghi
  wr_e = 0: cho phép đọc
- **oe**: Output enable

- **addr**: address (7-bit → RAM 128 byte)
- **data**: kiểu inout 8-bit

Yêu cầu:

- Viết testbench để kiểm tra thiết kế theo mô hình quan sát dạng sóng bằng phần mềm ModelSim-Altera
- Viết testbench để kiểm tra thiết kế theo mô hình tự kiểm tra (Self-checking) bằng phần mềm ModelSim-Altera

Code thực thi:

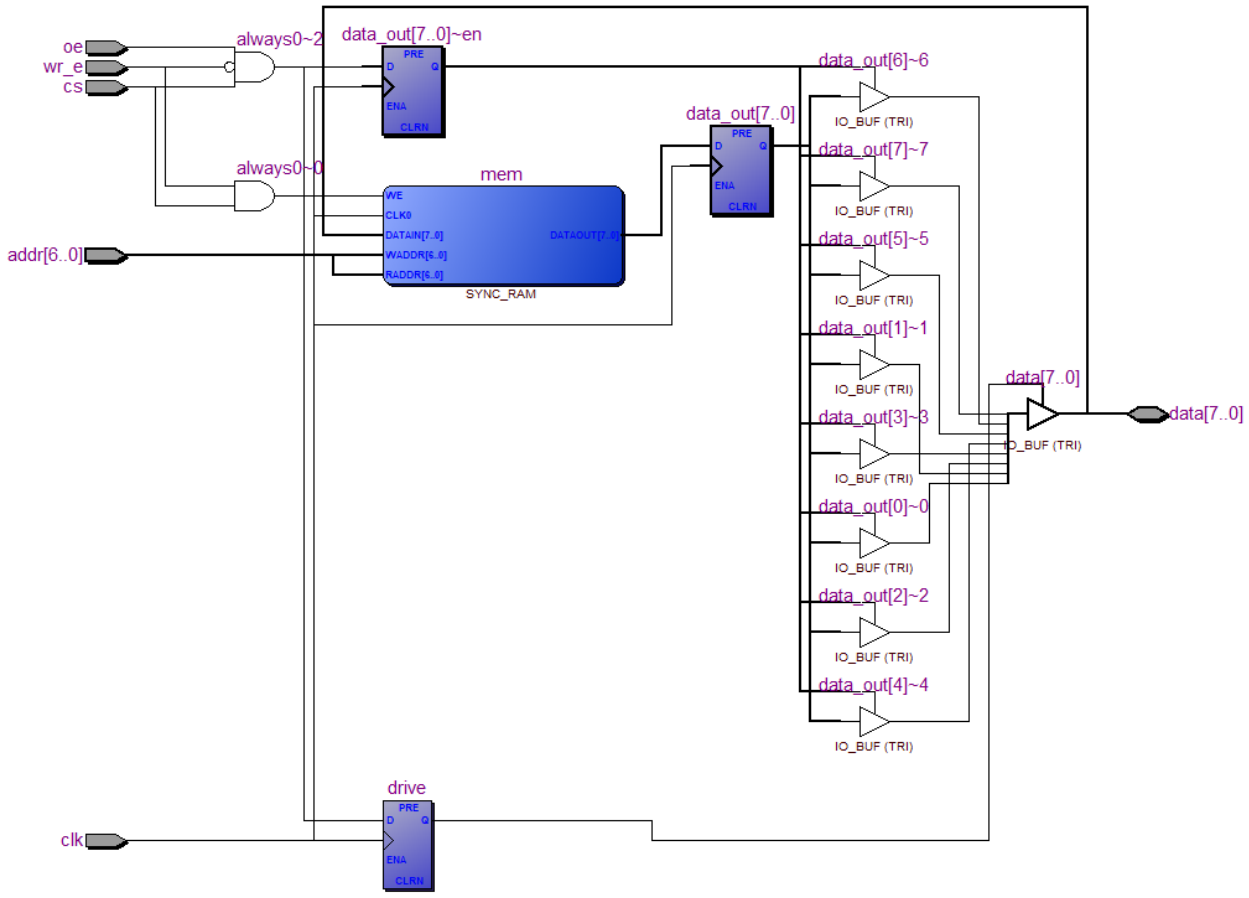
```
1  module Ram_lab5(
2      input clk,
3      input wr_e,
4      input oe,
5      input cs,
6      input [6:0] addr,
7      inout [7:0] data
8  );
9      reg [7:0] mem [0:127];
10     reg [7:0] data_out;
11     reg drive;
12 always @(posedge clk) begin
13     if(cs && wr_e) begin
14         mem[addr] <= data;
15     end
16     if(cs && !wr_e && oe) begin
17         data_out <= mem[addr];
18         drive <= 1;
19     end else begin
20         drive <= 0;
21         data_out <= 8'bz;
22     end
23 end
24 assign data = (drive) ? data_out : 8'bz;
25 endmodule
```

'estbench:

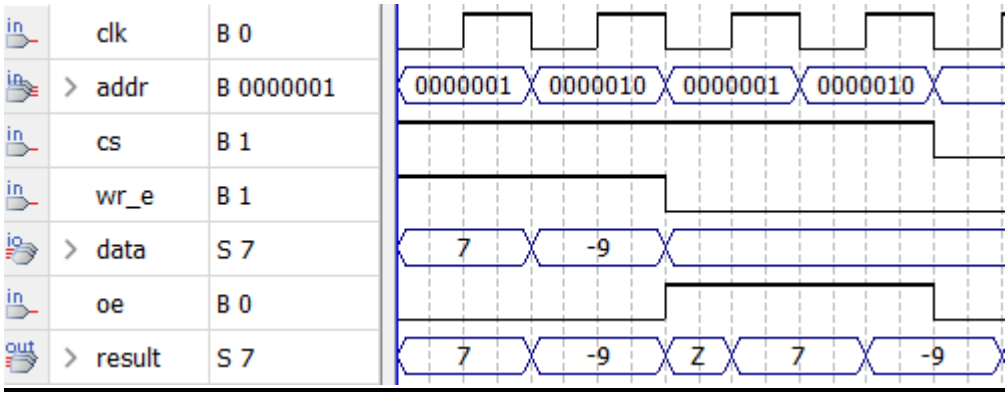
```
module tb_Ram_lab5;
  reg clk;
  reg wr_e;
  reg oe;
  reg cs;
  reg [6:0] addr;
  reg [7:0] data_in;
  wire [7:0] data;
  wire [7:0] result;

  // ?i?u khi?n h??ng truy?n trên bus d? li?u
  assign data = (cs && wr_e && !oe) ? data_in : 8'bz;
  assign result = data;

  // Instantiate the Unit Under Test (UUT)
  Ram_lab5 uut (
    .clk(clk),
    .wr_e(wr_e),
    .oe(oe),
    .cs(cs),
    .addr(addr),
    .data(data)
  );

  // Clock generation
  always #5 clk = ~clk;

  // Test scenario
  initial begin
    clk = 0;
    wr_e = 0;
    oe = 0;
    cs = 0;
```

```
34      addr = 7'b0;
35      data_in = 8'b0;
36
37      $monitor("T=%0t | clk=%b | addr=%h | cs=%b | wr_e=%b | oe=%b | data_in=%0d | result=%0d",
38               $time, clk, addr, cs, wr_e, oe, data_in, result);
39
40      // Write 7 to address 1
41      #10 cs = 1; wr_e = 1; addr = 7'd1; data_in = 8'd7;
42
43      // Write -9 to address 2
44      #10 cs = 1; wr_e = 1; addr = 7'd2; data_in = -8'd9;
45      #10 wr_e = 0;
46
47      // Read from address 1
48      #10 oe = 1; addr = 7'd1; cs = 1;
49      #10 oe = 0;
50
51      // Read from address 2
52      #10 oe = 1; addr = 7'd2; cs = 1;
53      #10 oe = 0;
54
55      // Done
56      #10 $finish;
57    end
58  endmodule
```

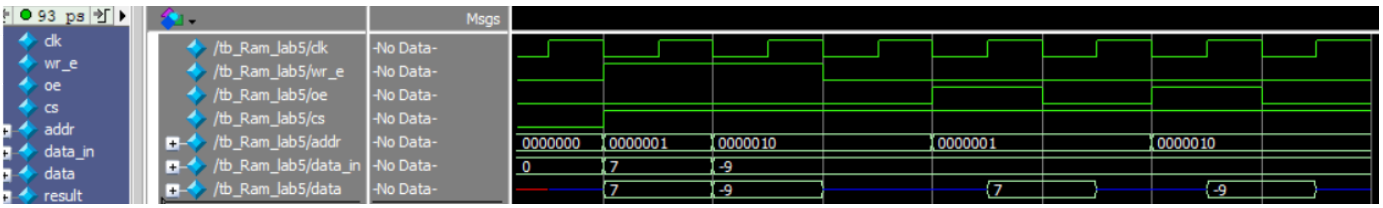
RTL viewer:



Mô phỏng Wareform:



TestBench:



Giải thích:

Mô-đun RAM trong Verilog mô phỏng bộ nhớ với khả năng đọc và ghi dữ liệu. Mô-đun này sử dụng một số tín hiệu điều khiển để xác định hành vi của bộ nhớ, gồm các tín hiệu như đồng hồ (clk), cho phép ghi (wr_e), cho phép đọc (oe), chọn chip (cs), địa chỉ bộ nhớ (addr), và đường dữ liệu (data).

Giải thích hoạt động của mô-đun:

1. **Ghi dữ liệu vào bộ nhớ:**
   - Khi tín hiệu cs (chip select) và wr_e (write enable) cùng có giá trị bằng 1, dữ liệu trên đường data sẽ được ghi vào bộ nhớ tại địa chỉ addr.
   - Dữ liệu này sẽ được lưu trữ trong mảng bộ nhớ mem.
2. **Đọc dữ liệu từ bộ nhớ:**
   - Khi cs = 1 (chip select được bật), wr_e = 0 (không ghi), và oe = 1 (cho phép đọc), mô-đun sẽ xuất dữ liệu từ bộ nhớ ra đường data.
   - Dữ liệu từ bộ nhớ tại địa chỉ addr sẽ được truyền qua biến data_out và đường dữ liệu data sẽ được lái (drive) với giá trị này.
   - Nếu oe = 0 (không cho phép đọc), hoặc nếu cs = 0 (không chọn chip), đường dữ liệu sẽ không bị điều khiển (ở trạng thái 'Z' - high impedance).
3. **Trạng thái 'Z':**
   - Khi không có phép đọc hoặc ghi, đường dữ liệu sẽ ở trạng thái 'Z', có nghĩa là nó không ảnh hưởng tới tín hiệu từ các mô-đun khác trong hệ thống (tín hiệu không được lái).

Các tín hiệu quan trọng:

- cs (Chip Select): Điều khiển việc kích hoạt bộ nhớ. Khi cs = 1, bộ nhớ sẽ hoạt động, còn khi cs = 0, bộ nhớ sẽ không thực hiện bất kỳ thao tác nào.
- wr_e (Write Enable): Điều khiển việc ghi dữ liệu vào bộ nhớ. Nếu wr_e = 1, dữ liệu từ đường data sẽ được ghi vào bộ nhớ tại địa chỉ addr.
- oe (Output Enable): Điều khiển việc xuất dữ liệu từ bộ nhớ ra đường data. Nếu oe = 1, dữ liệu tại địa chỉ addr sẽ được xuất ra đường data.