NMJ31804 – PRINCIPLES OF COMPUTER ARCHITECTURE
Semester 2, 2021/2022

LAB 3: Design of Memory Module RAM and ROM

Introduction

Computer memory is commonly classified as either internal or external memory. Internal memory, also known as "main or primary memory" refers to memory that stores small amounts of data that can be accessed quickly while the computer is running. External memory, also known as "secondary memory" refers to a storage device that can retain or store data persistently. They could be embedded or removable storage devices. The examples of external memory include hard disk or solid-state drives, USB flash drives, and compact discs.

There are basically two kinds of internal memory: Read Only Memory (ROM) and Random-Access Memory (RAM).

ROM is non-volatile, which means it can retain data even without power. It is used mainly to start or boot up a computer. Once the operating system is loaded, the computer uses RAM, which temporarily stores data while the central processing unit (CPU) is executing other tasks. With more RAM on the computer, the less the CPU has to read data from the external or secondary memory (storage device), and this allows the computer to run faster. RAM is fast but it is volatile, which means it will not retain data if there is no power. It is therefore important to save data to the storage device before the system is turned off.

Objectives

i. To design and model RAM and ROM in Verilog.
ii. To design a simple system using RAM and ROM.

Tasks

Prelab:

i. Design of ROM

Figure 1 shows the module of romtest. This module has 2-bit address and 8-bit data. Write Verilog code as shown in Figure 2 and simulate the result. You should get the simulation result similarly in Figure 3.

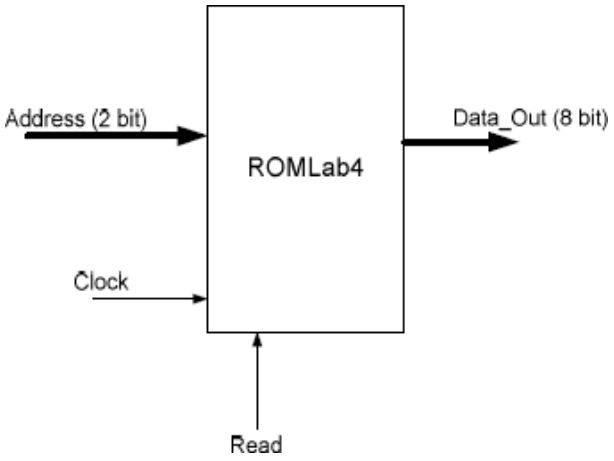

Figure 1: Block Diagram of romtest Module

```
module romtest (Clock,read, addr, data);
        input Clock, read;
        input [1:0] addr;
        output [7:0] data;
        reg [7:0] data;
        always @(posedge Clock)
                if(read)
                  begin
                        case (addr)
                        2'b00: data <= 8'b10000001;
                        2'b01: data <= 8'b10000010;
                        2'b10: data <= 8'b10000011;
                        2'b11: data <= 8'b10000100;
                        endcase
                   end
                else
                        data <= 8'bz;
endmodule
```

Figure 2: Verilog Code for romtest

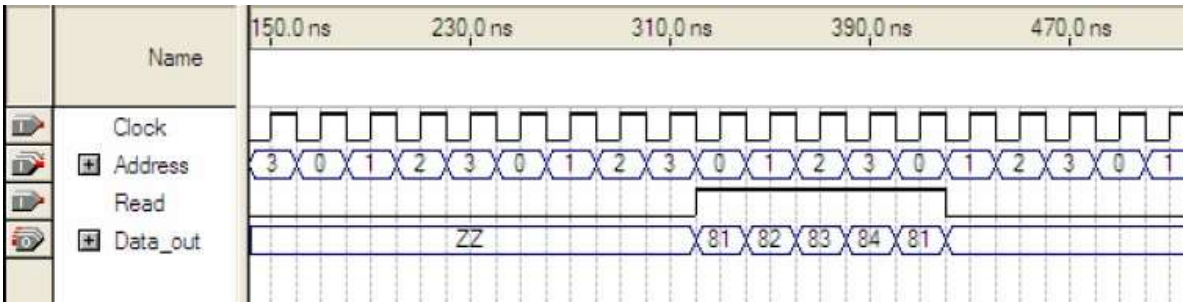

Figure 3: Simulation Result of romtest Module

ii. Design of RAM

Figure 4 is a module of romtest. This module has 2-bit address, 8-bit data input and 8-bit data outputs. Verilog code for this module is shown in Figure 5. Write this code and simulate the result. The simulation result should be similar to Figure 6. The first process involves reading data from Data_In and store in the RAM. Then the data is read out again from this module.

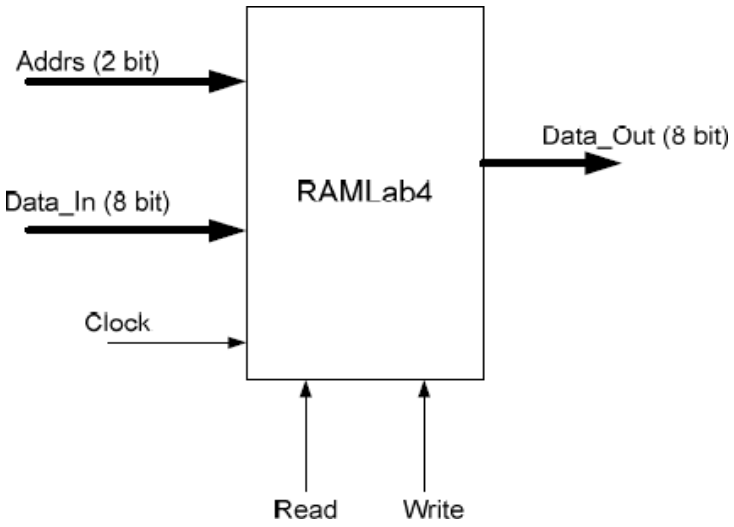

Figure 4: Block diagram of ramtest module

```
module ramtest (Clock, Write, Read, Addrs, Data_Out, Data_In);
        input [1 : 0] Addrs;
        input [7 : 0] Data_In;
        input Clock, Read,
        Write;output [7 : 0]
        Data_Out;
        reg [7:0] memory[3:0];
        reg [7:0] Data_Out;
        always @(posedge
        Clock)
          begin
                if (Write)
                        memory[Addrs] <= Data_In;
                if (Read)
                        Data_Out <= memory[Addrs];
                else
                        Data_Out <= 8'bz;
           end
endmodule
```

Figure 5: Verilog code for ramtest module

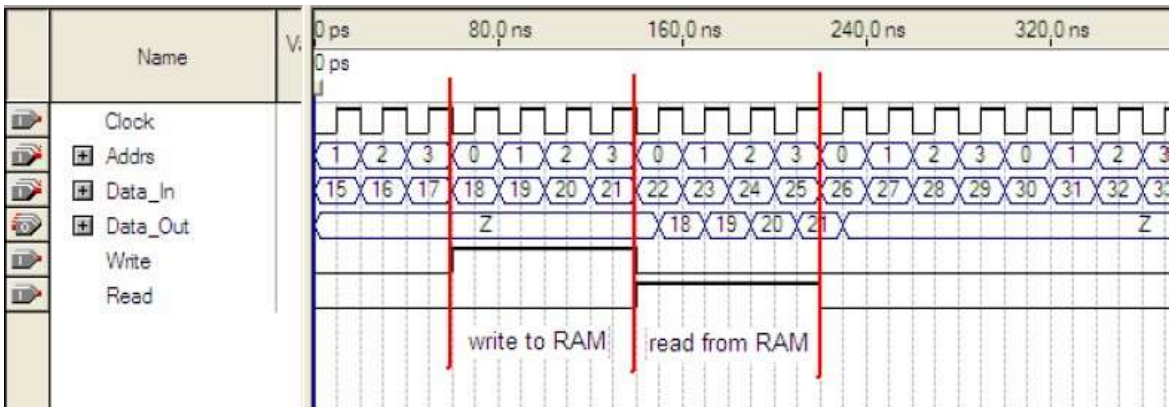

Figure 6: Simulation result of ramtest module

iii. **Design of control Unit to transfer data from ROM to RAM**

Figure 7 shows the entity module of control unit and named as FSMLab4. This module will control the overall process of the system (process transfer data from RAM to ROM). It consists of 1-bit clock signal and reset signal. The output for this module is Address value, read signal for RAM, write signal for RAM and read signal for ROM. This module is constructed by using seven states as shown in Table 1. For each state it produces a certain control signal. (refer Table 1). Write Verilog code in Figure 8 and simulate the result.

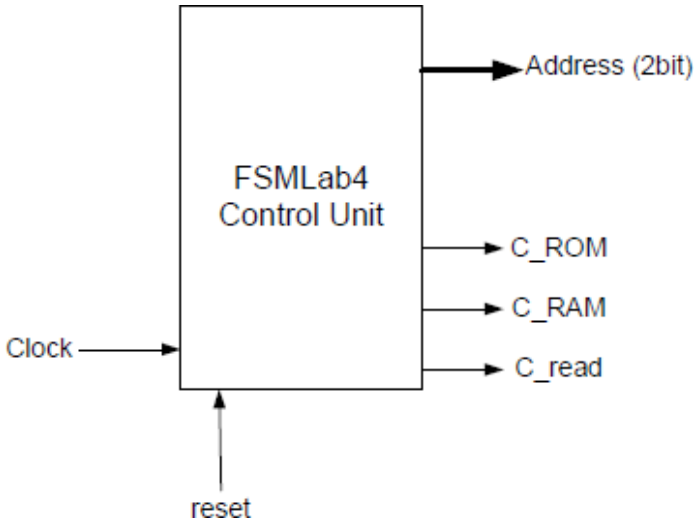

Figure 7: Block Diagram of Control Unit/ FSMlab4 Module

State	Address (Address)	Read ROM (c_ROM)	Write RAM (c_RAM)	Read RAM (c_read)	Operation
S1	00	1	1	0	Transfer data from ROM to RAM
S2	01	1	1	0	
S3	10	1	1	0	
S4	11	1	1	0	
S5	00	0	0	1	Read data from RAM
S6	01	0	0	1	
S7	10	0	0	1	
S8	11	0	0	1	

Table 1: State Diagram and Control Signal for Control Unit / FSMlab4 module

```
module FSMlab4
(clock,reset,Address,c_ROM,c_RAM,c_read);input
clock, reset;
output [1:0] Address;
output c_ROM, c_RAM, c_read;
parameter [2:0] s1 = 3'b000,s2 = 3'b001,s3 = 3'b010,s4 =
3'b011,s5 =3'b100,s6 = 3'b101,s7 = 3'b110,s8 = 3'b111;
reg [2:0] y_present,
y_next;reg [1:0]
Address;
reg c_ROM, c_RAM, c_read;

        always@(y_present)
        begin
                case
                (y_present)s1:
                begin
                                y_next <= s2;
                                Address <=
                                2'b00;c_ROM
                                <= 1'b1;
                                c_RAM <=
                                1'b1;
                                c_read <= 1'b0;
                        end
                s2: begin
                                y_next <= s3;
                                Address <=
                                2'b01;c_ROM
                                <= 1'b1;
                                c_RAM <=
                                1'b1;
                                c_read <= 1'b0;
                        end
                s3: begin
                                y_next <= s4;
                                Address <=
                                2'b10;c_ROM
                                <= 1'b1;
                                c_RAM <=
                                1'b1;
                                c_read <= 1'b0;
                        end
                s4: begin
                                y_next <= s5;
```

```
                s5: begin
                                y_next <= s6;
                                Address <=
                                2'b00;c_ROM
                                <= 1'b0;
                                c_RAM <=
                                1'b0;
                                c_read <= 1'b1;
                        end
                s6: begin
                                y_next <= s7;
                                Address <=
                                2'b01;c_ROM
                                <= 1'b0;
                                c_RAM <=
                                1'b0;
                                c_read <= 1'b1;
                        end
                s7: begin
                                y_next <= s8;
                                Address <= 2'b10;
                                c_ROM <= 1'b0;
                                c_RAM <= 1'b0;
                                c_read <= 1'b1;
                        end
                s8: begin
                                y_next <= s1;
                                Address <= 2'b11;
                                c_ROM <= 1'b0;
                                c_RAM <= 1'b0;
                                c_read <= 1'b1;
                        end
                endcase
        end

        always @ (posedge
        clock)begin
                if (reset == 0)
                        y_present <= s1;
                else
                        y_present <= y_next;
        end
endmodule
```

Figure 8: Verilog Code for Control Unit / FSMlab4 Module