# \*Name:

**\*Registration:**

**Department of Computer Systems Engineering University of Engineering & Technology Peshawar**

**Digital System Design CSE 308**

**Finalterm Examination Spring 2021**

29 July 2021, Duration: 180 Minutes

**Start Time: 2:00 PM (Sharp) End Time: 5:00 PM (Sharp)**

**\*\*Exam Rules\*\***

**Please read carefully before proceeding.**

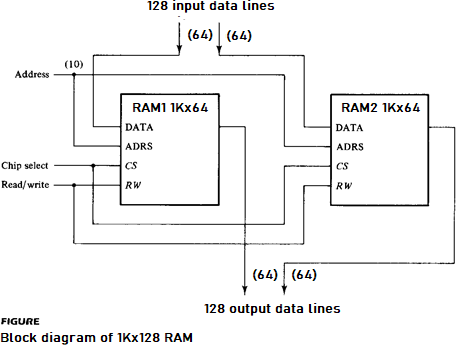
1. This exam is open books/notes/Internet.
2. It's good to share but sharing books, notes, and other materials during this exam is not permitted.
3. Answer all problems.
4. Problems will not be interpreted during the exam.

|  |  |  |
| --- | --- | --- |
| **Problem** | **Score** |  |
| **1** |  | **/15** |
| **2** |  | **/20** |
| **3** |  | **/15** |
| **Total** |  | **/50** |

# Good Luck!

**Problem 1. (15 pts.)**

As discussed in the class, it is possible to combine two chips to form a composite memory containing the same number of words but with twice as many bits in each word. Figure below shows the interconnection of two 1Kx64 RAM chips to form a 1Kx128 memory. The 128 input and output data lines are split between the two chips. Both receive the same 10-bit address and the common CS and RW control inputs. **(15 pts.)**

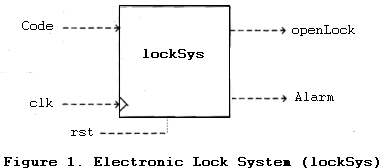


In this problem, write a top-level module to combine two 1Kx64 RAM chips (given below) to form a 1Kx128 memory.

|  |  |
| --- | --- |
| **module RAM1 (addr, CS, RW, idata, odata);**  **input CS, RW; input [9:0] addr;**  **input [63:0] idata;**  **output [63:0] odata;**  **reg [63:0] d\_out;**  **reg [63:0] Mem1 [0:1023];**  **assign odata = (CS && RW)?d\_out:64**’**bz; always @(addr or idata or CS or RW)**  **if (CS && !RW)**  **Mem1 [addr] = idata; always @(addr or CS or RW)**  **if (CS && RW)**  **d\_out = Mem1 [addr];**  **endmodule** | **module RAM2 (addr, CS, RW, idata, odata);**  **input CS, RW; input [9:0] addr;**  **input [63:0] idata;**  **output [63:0] odata;**  **reg [63:0] d\_out;**  **reg [63:0] Mem2 [0:1023];**  **assign odata = (CS && RW)?d\_out:64**’**bz; always @(addr or idata or CS or RW)**  **if (CS && !RW)**  **Mem2 [addr] = idata; always @(addr or CS or RW)**  **if (CS && RW)**  **d\_out = Mem2 [addr];**  **endmodule** |
| **Ans:**  **module TopLevelRAM (addr, CS, RW, idata, odata);**  **input CS, RW;**  **input [9:0] addr;**  **input [127:0] idata;**  **output [127:0] odata;**  **wire [63:0] odata1, odata2;**  **wire [63:0] idata1, idata2;**  **RAM1 ram1 (.CS(CS), .RW(RW), .addr(addr[9:0]), .idata(idata[63:0]), .odata(odata1));**  **RAM2 ram2 (.CS(CS), .RW(RW), .addr(addr[9:0]), .idata(idata[127:64]), .odata(odata2));**  **assign odata = {odata2, odata1};**  **assign idata1 = idata[63:0];**  **assign idata2 = idata[127:64];**  **endmodule** |  |

# Problem 2. (20 pts.)

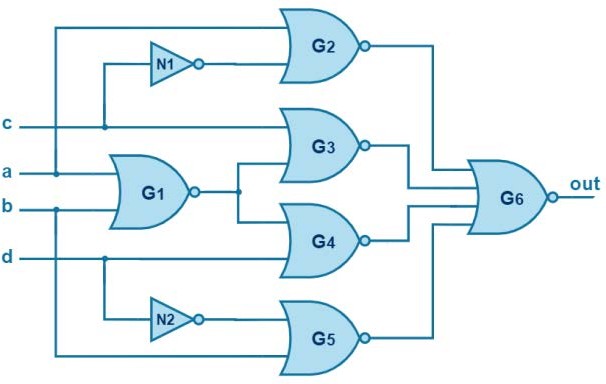
In this problem, design an electronic lock system (**lockSys**, see Figure 1) for a garage door lock. The electronic lock accepts a 4- digit user code input, one digit at a time. If the input code sequence exactly matches 0011, the electronic lock is opened (**openLock** is asserted). If any part of the 4-digit code input sequence is incorrect, the **Alarm** is activated and the user is forced to restart code entry i.e. all backward arcs go back to the initial state if a "wrong" digit is entered. When **openLock** is asserted after the correct code has been entered, the lock stays open. And the lock shuts itself (**openLock** is de-asserted) if the asynchronous input signal **rst** is asserted or a 0/1 is entered after the correct input code sequence until the correct input code is again entered.



1. Design a Mealy/Moore FSM for **lockSys**. **(10 pts.)**
2. Implement the FSM in **(a)** in Verilog. **(10 pts.)**

# Problem 3. (15 pts.)

Consider the below given circuit based on the single stuck-at fault assumption and answer the questions related to this.



1. What is the number of all possible faults? **(2 pts.)**
2. How many checkpoint faults are in the circuit? **(3 pts.)**
3. Write the reduced fault list using the method of fault equivalence reduction. What is the collapse ratio after you perform fault collapsing (based on the equivalent faults you find)? **(10 pts.)**