FINAL EXAM

0764 - Digital Logic Design Using Verilog

Jagadeesh Vasudevamurthy
jvasudev@ucsc.edu

NOV 22 2014

Student IDNO # J5289433
Student Name John Hubbard

Problem	MAX	Student score
1	25	
2	25	
3	25	
4	25	
TOTAL	100	

P1: Job Interview Questions 25 Pts

Define precisely the difference and similarities. Be as precise as possible. You must write in the space provided. No extra spaces allowed. Each question carries 2 points. One extra point will be given for neatness.

1. Combinational and sequential logic Combinational logic does not maintain any state; the output is entirely a function of input. Sequential logic does maintain state — the output depends both upon current state, and on inputs.
2. Latch and Flip-flop A latch is level triggered, while a flip-flop is edge-triggered.
3. Nets and Registers(in verilog) A register represents storage, while a net represents a connection (wire).
4. Operator >> and >>> >> is a "shift right and fill the upper bits with zeros". >>> is an arithmetic shift right, so the upper bits will be filled with zeroes for unsigned variable, or ones for signed variable.
5. Operator == and ===
== is exactly like the C language equality test. === includes all four bit states: 0, 1, x, z. This must be an exact match.
6. Continuous and Procedural statement
A continuous assignment happens all the time, and in parallel with other assignment. procedural statements happen in the order they are written.

John Hubbard

7. Blocking and Nonblocking assignment Blocking assignments happen in order. Non-blocking happen in parallel with each other. You should not mix those two in the same begin-end block.

8. Decoder and encoder A decoder has N inputs and 2^N outputs. An encoder has 2^N inputs (plus N control lines) and one output, also called a MUX.

9. Mealy and Moore machine A Moore machine is an FSM where next state depends only upon the current state. A Mealy machine depends upon both current state and input, in order to choose its next state.

10. Positive edge and Negative edge D flip-flop

A positive edge D FF changes state on the rising edge of the clock. A negative edge D FF does on the falling edge of the clock.

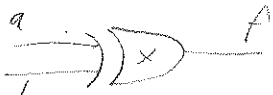
11. Asynchronous and synchronous D flip-flop A synchronous D FF does its reset and/or clear on the clock edge. An asynchronous D FF clears or resets immediately.

12. D flip-flop and D flip-flop with enable A D FF with an enable input will hold its current state, even through clock cycles, until enable is asserted. A D FF without enable has no such input.

P2- A Understanding structural, data flow and behavioral Verilog 10 Pts

Implement a 2-input XOR gate using structural, data flow and behavioral Verilog.

The structural and data flow must use only NAND gate as primitive. Draw the diagram and then write the Verilog code.




```
// structural
module xor2(a, b, f);
  input a, b;
  output f;
  xor x(f, a, b);
endmodule
```

```
// data flow
module xor2(a, b, f);
  input a, b;
  output f;
  assign f = a ^ b;
endmodule
```

The behavioral cannot use any equation. Write the code below.

```
module xor2(a, b, f);
  input a, b;
  output f;
  reg f;
  always @(*)
    f = a ^ b;
```


John Hubbard

P2- B Understanding blocking and non blocking assignment 15Pts

Given a memory of size 64 words of 8 bits wide, write Verilog code to swap the contents of memory in reverse order. That is transfer word 0 to word 63, word 1 to 62 etc. You cannot use any temp variables to swap ☺

```
reg [7:0] mem_In [63:0];
reg [7:0] mem_Out [63:0];
integer i;  integer j = 63;
for (i = 0; i < 64; i = i+1, j = j-1)
begin
        mem_Out [i] <= mem_In [j];
end
```

non-blocking assignment

P3-A: Design using Verilog : 15 points

Implement the counter below in Verilog. All inputs and outputs are in italics and underlined.

Inputs:

1. Synchronous signal *e* which enables the counter to increment/decrement by one.
2. Clock *clk.*
3. Asynchronous signal *r* which resets the counter to 0 when *r* = 1.
4. Synchronous load signal *load.* When load is '1', input data [N-1:0] *d* is loaded.
4. *updown* signal: 0 means up, 1 means down

Output:

1. Signal [N-1:0] *count*, which provides the current value of the counter.

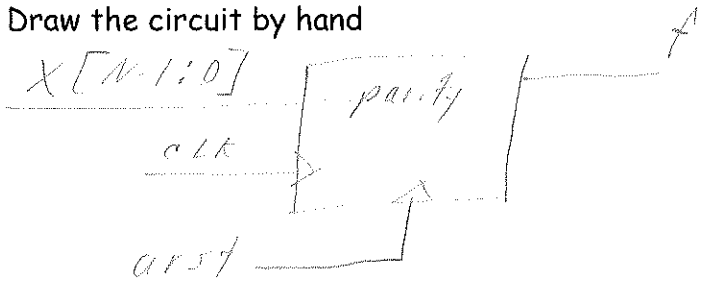
```
module counter (clk, d, e, r, load, updown, count);
    parameter N = 3.  input [N-1:0] d;
    input clk,  e, r, load, updown;
    output wire [N-1:0] count;
    reg [N-1:0] ic; // internal count
    assign count = ic;
    always @(posedge clk or posedge r)
    begin
        if (r == 1)
            ic = 0;
        else if (load == 1)
            ic = d;
        else if (e == 1)
            ic = (updown == 0 ?
                    (ic + 1) : (ic - 1));
    end
end module
```


John Hubbard

P3-B: Design using Verilog : 10 points

Design a parity checker that asserts '1' on its output **f**, if it has received an odd number of 1's on the input **X**. The parity checker has an asynchronous reset, **arst** and clock **clk**

1. Draw the circuit by hand



2. Implement using Verilog.

```
module parity (clk, arst, x, f);
    parameter n=3;
    input clk, arst;
    input wire [n-1:0] x;
    output f;
    reg f;
    always @(posedge clk or posedge arst)
    begin
        if (arst == 1)
            f = 0;
        else
            f = (x % 2 == 1);
    end
endmodule
```


P4: Decomposition strategy 25 points

No Verilog code is required. Draw clean diagrams and mark all the inputs and outputs correctly. Someone should be able take your drawing and write structural Verilog code without any difficulties.

1. Compose a 20 inputs XOR gates, using only 2-inputs XOR gates
2. Compose a 16x1 mux using only 2x1 muxs
3. Compose a 4x16 decoder using only 2x4 decoders
4. Compose a 1024x8 RAM using only 512x8 RAMS