

ECE 464 / ECE 520
Final 2007

Each question is worth 2 points. The correct answer earns 2 points, the incorrect 0. You have up to three hours to take this test. Answer all of the questions. Off-campus students, please return this test completed as specified in the syllabus. Version A.

The exam is open book, open notes.

Note that sometimes the answer is INTENTIONALLY E.

Clearly indicate your answers in the matrix below.

Name : _____

Student Number : _____

Question	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Question 1

What is wrong with the following code fragment?

```
always@(A or C)
    D = A & C;
always@(posedge clock)
    if (B) E <= C;
```

- A. The else is missing and unintentional latches are implied.
- B. Non blocking assignment should be used to obtain expected functionality.
- C. D should not be assigned like this. Unbuildable “wired-or” logic is implied.
- D. C is missing from the second sensitivity list.
- E. Nothing is wrong with this code fragment.

Question 2

What is wrong with the following code fragment?

```
always@(A or B or C)
begin
    F = A & C;
    A = F | B;
end
always@(B or D)
begin
    F = D ^ B;
end
```

- A. Non blocking assignment should be used to obtain expected functionality.
- B. F should not be assigned like this. Unbuildable “wired-or” logic is implied.
- C. There is combinational logic feedback.
- D. E is missing from the sensitivity list.
- E. Nothing is wrong with this code fragment.

Question 3

What is wrong with the following code fragment (note the use of non-blocking assignment)?

```
always@(A or B or C)
begin
    E <= C | B;
    F <= E ^ A;
end
```

- A. E is missing from the sensitivity list.
- B. Blocking assignment should be used to obtain expected functionality.
- C. E should not be assigned like this. Unbuildable “wired-or” logic is implied.
- D. Latches are implied.
- E. Nothing is wrong with this code fragment.

Question 4

Would you expect to achieve a 1 GHz clock with a 0.18 um standard cell part?

- A. True
- B. False

Question 5

You have to build a part that has numerous 256 Mbps inputs and you are required to perform real time processing on them. The volume is high. Which implementation style would you choose?

- A. A gate array
- B. A full custom chip.
- C. A standard cell chip.
- D. Leave it as an FPGA.
- E. None of the above.

Question 6

Is this a valid, synthesizable, use of a for loop?

```
reg [8:0] A, B;
integer i;
parameter N=8;
always@(B)
begin
    for (i=1; i<=N; i=i+1)
        A[i-1]=B[i];
    A[N] = A[N-1];
end
```

- A. True
- B. False

Question 7 (Does not apply to distance ed)

Assuming the code in question 6 is synthesizable, which of the following Verilog 2001 continuous assignment statements would have the closest meaning?

- A. assign A = B << 1;
- B. assign A = B <<< 1;
- C. assign A = B >> 1;
- D. assign A = B >>> 1;

Question 8

Is the following code fragment, that of a Mealy or Moore machine?

```
always@(posedge clock) state <= next_state;

always@(state or A)
begin
    out = 0;
    case (state)
        0 : if (A) begin
                    Out = 1;
                    next_state = S0;
                end
            else next_state = S1;
        1 : next_state = S0;
    end
```

- A. Mealy
- B. Moore
- C. Neither
- D. Both

Question 9

The code fragment in Question 8 is likely to implement unintended latches?

- A. True
- B. False

Question 10

Consider the following code fragment. Notice the use of non-blocking assignment.

```
always@(posedge clock)
begin
    B <= A^C;
    C <= B;
end
```

Which BLOCKING code version would lead to identical logic (after any potential logic gate sharing is taken into account)?

```
A. always@(posedge clock)
    begin
        B = A^C;
        C = B;
    end
```

```
B. always@(posedge clock)
    begin
        C = B;
        B = A^C;
    end
```

```
C. always@(posedge clock)
    begin
        C = B;
        B = A^CC;
    end
    assign CC=C;
```

```
D. always@(posedge clock)
    begin
        B = A^CC;
        C = B;
    end
    assign CC = C;
```

E. None of the above

Question 11

Which statement best describes the purpose of inserting scan chains into a design?

- A. Assist in verification.
- B. Assist in Built In Self Test.
- C. Enable stuck-at-faults to be emulated in the design..
- D. Enable test vectors to be loaded and read out.
- E. None of these are true

Question 12

If “enable” is high only 10% of the time, and the code below is synthesized, which of the following will have the lowest power consumption.

```
A.
wire [31:0] A, B, D;
reg [31:0] E, F, G;
```

```

always@(posedge clock)
  begin
    if (enable) E <= A * B;
  end

```

B.

```

wire [31:0] A, B, D;
reg [31:0] E, F, G;
always@(posedge clock)
  begin
    E <= AA * BB;
    If (enable) begin
      AA <=A; BB <= B;
    end
  end
end

```

C.

```

wire [31:0] A, B, D;
reg [31:0] E, F, G;
always@(posedge clock)
  begin
    E <= enable ? A*B : 0 *0;
  end
end

```

D.

```

wire [31:0] A, B, D;
reg [31:0] E, F, G;
always@(posedge clock)
  begin
    if (enable) E <= A * B;
    else E <= 0 * 0;
  end
end

```

E. These all consume the same power.

Question 13

If the following logic is built exactly as described, which test vector sensitizes a stuck-at-0 fault at e and propagates it to the output g.

```

wire a, b, c, d, e, f, g;

assign e = a & b;
assign f = c ^ e;
assign g = d | f;

```

A. {a, b, c, d} = 4'b0010;

- B. {a, b, c, d} = 4'b1100;
- C. {a, b, c, d} = 4'b1111;
- D. {a, b, c, d} = 4'b0101;
- E. None of the above

Question 14

What does the following primitive describe:

```
primitive PlanetX (A, B);
output F;
reg F;
input A, B;
table
0 0 : 0
0 1 : 1
1 1 : 1
1 0 : 1
endtable
endprimitive
```

- A. An XOR gate.
- B. An OR gate.
- C. An AND gate
- D. A D flip-flop
- E. None of the above

Questions 15 and 16

Both of these questions apply to the following code implemented exactly as described.

```
always@(posedge clock)
begin
A <= B;
B <= F;
E <= H;
end
assign C = A ^ B;
assign F = C | E;
assign H = F | A;
```

Each gate has a delay from its input to output as given as {1 : 2 : 3}, t_{Cp_Q} is {3 : 4 : 5} the clock skew is 1 ns, the flip-flop setup time is 1 ns and the hold time 2 ns. Format above is {min : typical : max}.

Question 15

The fastest possible viable clock period is:

- A. 9 ns

- B. 16 ns
- C. 18 ns
- D. 19 ns
- E. None of the above

Question 16

Is there potential for a hold violation in this logic?

- A. No, with a margin of 1 ns
- B. No, but with no margin.
- C. Yes, with a margin of 1 ns
- D. Yes, with a margin of 2 ns
- E. None of the above apply

Question 17

What best describes the purpose of the command `replace_synthetic -ungroup` in our standard class synopsys script?

- A. Allows design ware to be synthesized with the rest of the design.
- B. Allows designware to be mapped onto our standard cell library?
- C. Converts designware to be converted from one generation to another.
- D. Permits designware to be translated to another cell library.
- E. None of the above

Question 18 (ONLY Q 18, Does not apply to off campus students)

What advantage does SystemVerilog have over Verilog 2001 when it comes to writing assertions?

- A. Assertions can not be written in Verilog 2001.
- B. SystemVerilog permits assertions to be synthesized.
- C. SystemVerilog permits assertions to be written as part of a constrained random functional test.
- D. SystemVerilog permits a briefer syntax.
- E. More than one of these are true.

Question 19

For the following module, which is the best partition? Note the group names after the begin: statements. I am asking you which combinations of groups would be the best partition that would give the smallest area, contain all possible critical paths and the largest number of modules. Assume synthesis will find all possible sharable resources if the groups are in the same module.

```
module foo (clock, in1, in2, in3, out1, out2, out3, out4);
input clock,
input [3:0] in1, in2, in3;
output [3:0] out1, out2, out3, out4;
reg [3:0] who, stones, floyd, doors;

always@(in1 or in2)
begin : A
    who = in1 + in2;
end

always@(posedge clock)
begin : B
    out1 <= who;
end

always@(in1 or in2 or in3)
begin : C
    stones = (in1 ^ in2) | in3;
end

always@(posedge clock)
begin : D
    out2 <= stones;
end

always@(in2 or in2)
begin : E
    floyd = in1 ^ in2;
end

always@(posedge clock)
begin : F
    out3 <= floyd & out4;
end

always@(in3)
begin : G
    doors = in3 | out3;
end

always@(posedge clock)
begin : H
    out4 <= doors;
end
endmodule
```

- A. All one module
- B. Two modules: AB in one, C-H in the second.
- C. Three modules: AB in one, E,F in the second, and G,H in the third.
- D. Two modules : A-D in one, E-H in the second.
- E. Eight modules: All of A-H each in their own module.

Question 20

Generally, you would use an FSM over a counter in a controller when...

- A. The equivalent state sequence has many branches in it.
- B. The equivalent state sequence has no branches in it.
- C. You need to apply a reset.
- D. A and C
- E. None of the above

