

ECE 464 / ECE 520
Final 2006

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.

The exam is open book, open notes.

NCSU students : Write your name and student number on the bubble sheets in the specified locations. Note that the answers recorded on the bubble sheets are the final answer. **Also record your answers below.** No notes written below are taken into account or even read. Note that sometimes the answer is INTENTIONALLY E.

VBEE students ONLY : Clearly indicate your answers in the matrix below. **Only answer questions 1 to Y.**

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) D <= 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;
    F = F ^ 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?

```
always@(A or B or C)
begin
    if (C) E = C | B;
end
```

- A. E is missing from the sensitivity list.
- B. Non 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

You have a part which has been implemented in an FPGA. The manufacturing volume is becoming just large enough though, that the per-unit cost of the FPGA is becoming a concern. Thus, another implementation style is being considered. NO other changes are planned. Which 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 5

Which code fragment correctly describes a logic function that selects the largest of three unsigned numbers as an output. E.g. If In1, In2, and In3 are 2b01, 2b11 and 2b10 respectively, then Out = 2b11.

A.

```
always@(In1 or In2 or In3)
begin
    Out = In1 < In2 < In3;
end
```

B.

```
always@(In1 or In2 or In3)
begin
    Out = In1;
    if (In2 > Out) Out = In2;
    if (In3 > Out) Out = In3;
end
```

C.

```
always@(In1 or In2 or In3)
begin
    Out = In1; A=x;
    if (In1 > In2) begin A = In1; Out = A; end
    if (In2 > In3) begin A=In2; Out=A; end
    if (In3 > In1) begin A=In3; Out=A; end
end
```

D.

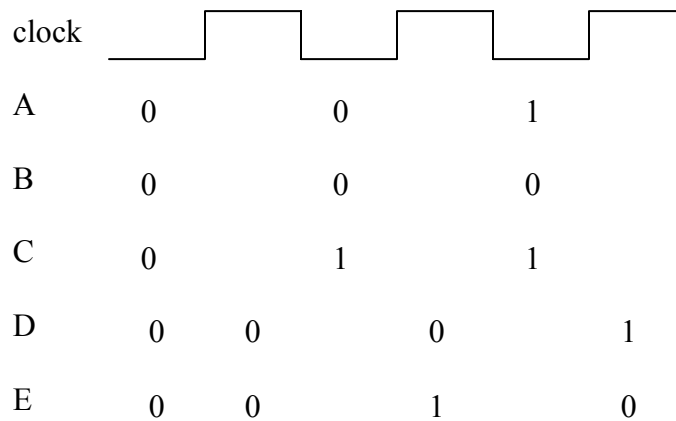
All of the above

E.

None of the above.

Question 6

Which code fragment synthesizes the logic that meets the following timing diagram.
Note the answers use **blocking assignment**.



A.

```
always@(posedge clock)
begin
    E = D ^ C;
    D = E;
end
```

B.

```
always@(posedge clock)
begin
    D = E;
    E = D ^ C;
end
```

C.

```
always@(posedge clock)
begin
    E = D ^ C;
    D = A & B;
end
```

D.

```
always@(posedge clock)
begin
    D = A & B;
    E = A | B;
end
```

E. None of the above

Question 7

Which of the following statements is most true concerning verification of a design?

- A. You can never know that a design is truly bug-free.
- B. Functional verification – testing most of the module functions known to the designer - will result in a bug-free design.
- C. Random testing is required to ensure high code coverage.
- D. Debugging does not matter – there are no moths in my computer.
- E. None of these are true

Question 8

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. Don’t forget about resource sharing in synthesis.

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 <= A * B;
end
```

C.

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

E. These all consume the same power.

Question 9

If the following logic is built exactly as described, which test vector sensitizes a stuck-at-0 fault at f 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'b1110;
- C. {a, b, c, d} = 4'b1111;
- D. {a, b, c, d} = 4'b0101;
- E. None of the above

Question 10

Which of the following is most true as it applies to design for test (DFT):

- A. Running your verification vectors on your chip to test it will ensure that there are no manufacturing defects in the chip.
- B. Scan-path testing is used to give access to the internal combinational logic from the chip I/O.
- C. Design of a good DFT strategy is critical to helping you debug the chip before sending it to fabrication.
- D. Random testing is always preferable to scan path testing.
- E. None of the above are true

Questions 11 and 12

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

```
always@(posedge clock)
  begin
    A <= C;
    B <= F;
    E <= H;
  end
assign C = A ^ B;
assign F = C | E;
assign G = F & C;
assign H = G | 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 11

The fastest possible viable clock period is:

- A. 9 ns
- B. 10 ns
- C. 18 ns
- D. 19 ns
- E. None of the above

Question 12

Is there potential for a hold violation in this logic?

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

Question 13

Consider the following two test fixtures.

```
// fixture A
parameter delay1 =
parameter delay2 =
initial
  begin
    B = 1'b0;
    #20 A = 1'b1;
    #delay1 A = 1'b0;
    #delay2 B = 1'b1;
  end
```

```
//fixture B
initial
  fork
    B = 1'b0;
    #20 A = 1'b1;
    #40 A = 1'b0;
    #60 B = 1'b1;
  join
```

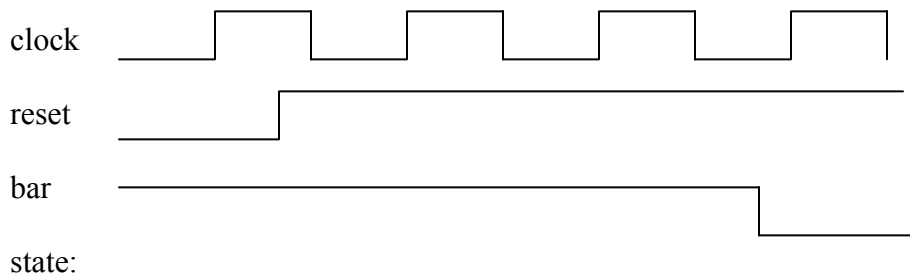
For these two fixtures to produce the same waveforms, delay1 and delay2 have to be set as follows:

- A. delay1 = 40; delay2 = 60;
- B. delay1 = 30; delay2 = 20;
- C. delay1 = 30; delay2 = 30;
- D. delay1 = 20; delay2 = 20;
- E. None of these are correct

Question 14

Which is a valid state sequence for the following FSM?

```
always@(posedge clock)
  if (!reset) state <= S0; else state <= next_state;
always@(state or bar)
  case (state)
    S0 : if (bar) next_state = S1; else next_state = S0;
    S1 : if (!bar) next_state = S0; else next_state = S1;
  endcase
```



- state:
- A. S0 S0 S0 S1
 - B. S0 S0 S1 S1
 - C. S0 S1 S0 S0
 - D. S0 S1 S1 S0
 - E. None of these are correct.

Question 15

The following primitive defines a what?

```
primitive foo (c, b, a);
output c;
reg c;
input a, b;
table
  // a b : c ;
  0 0 : 0 ;
  0 1 : 1 ;
  1 1 : 0 ;
  1 0 : 1 ;
endtable;
endprimitive
```

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

Question 16

If the input to this logic is $In = 4'b0101$ which is the correct output?

```
reg [3:0] out, In;
assign out = {&In, In[In[1:0]], {2{In[2]}}};
```

- A. $out = 4'b0011$;
- B. $out = 4'b0000$;
- C. $out = 4'b0111$;
- D. $out = 4'b1000$;
- E. None of the above.

Question 17

In this standard class synthesis script, the objective of the part that assigned port delays, drive strengths and loads was to?

- A. To allow Synopsys to maximize the clock frequency.
- B. Support timing calculations for nets connected to the module I/O.
- C. Determine if there were any unconstrained paths that did not go through flip-flops.
- D. All of the above.
- E. None of the above are true.

Question 18

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 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@(in2 or in3)
begin : E
    floyd = in1 ^ in3;
end

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

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

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

always@(in3)
begin : G
    doors = {4{&in3}};
end

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

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

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

The following questions apply to ON-campus students only

Question 19

Which is most true about the “interface” construct in SystemVerilog?

- A. It is not synthesizable, but is useful in building a transaction level design.
- B. It is used to replace port list specifications in module headers.
- C. It permits simpler interfaces to the test fixture.
- D. It simplifies coding whenever you have multiple modules sharing a common interface standard.
- E. None of the above are true.

Question 20

In a SystemVerilog assertion, the code extract “##[2:10]” means the following:

- A. A delay of 2-10 ns.
- B. A delay of 2 clock cycles and 10 ns
- C. A delay of 2-10 clock cycles.
- D. Entries 2-10 in the delay matrix.
- E. None of the above