

ECE 464 / ECE 520/ DS 510P
Final 2003

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. No notes written below are taken into account or even read. Note that sometimes the answer is INTENTIONALLY E.

NTU/VBEE students ONLY : Clearly indicate your answers in the matrix above.

Name : _____

Student Number : _____

NOTE. The following table is ONLY for off campus student use.

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 B or C)
begin
    F = A & C;
    G = E | B;
    F = E ^ 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. Latches are implied.
- D. E is missing from the sensitivity list.
- E. Nothing is wrong with this code fragment.

Question 2

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 3

What is wrong with the following code fragment?

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

In which scenario would an FPGA be more appropriate than a standard cell ASIC?

- A. A high performance design is required.
- B. NRE costs are not important.
- C. The design team is relatively small and inexperienced.
- D. Time to market is not important.
- E. None of the above.

Question 5

What is most true for a gate array?

- A. A device that has a functional density in between that of an FPGA and a standard cell ASIC.
- B. A special type of FPGA.
- C. A type of standard cell ASIC wherein only two types of gate (AND and OR) are available.
- D. A and B both apply.
- E. None are true.

Question 6

Which code fragment correctly describes a logic function that sorts three unsigned numbers into ascending order. E.g. If In1, In2, and In3 are 2b01, 2b11 and 2b10 respectively, then Out1, Out2 and Out3 are 2b01, 2b10, and 2b11 respectively.

A.

```
always@(In1 or In2 or In3)
begin
    Out1 = In1 < In2 < In3;
    Out2 = In1 > In2 < In3;
    Out3 = In1 > In2 < In3;
end
```

B.

```
always@(In1 or In2 or In3)
begin
    Out1 = In1; Out2 = In2; Out3 = In3;
    if (Out1 > In2) Out1 = In2;
    if (Out1 > In3) Out1 = In3;
    if (Out2 > Out1) Out2 = In1;
    if (Out2 > In3) Out2 = In3;
    if (Out3 > Out1) Out3 = In2;
    if (Out3 > Out2) Out3 = In3;
end
```

C.

```
always@(In1 or In2 or In3)
begin
    Out1 = In1; Out2 = In2; Out3 = In3; A=x; B=x;
    if (Out1 > Out2) begin Out1 = In2; Out2 = In1; end
    if (Out2 > Out3) begin A=Out2; Out2=Out3; Out3=A; end
    if (Out1 > Out2) begin B=Out1; Out1=Out2; Out2=B; end
end
```

D.

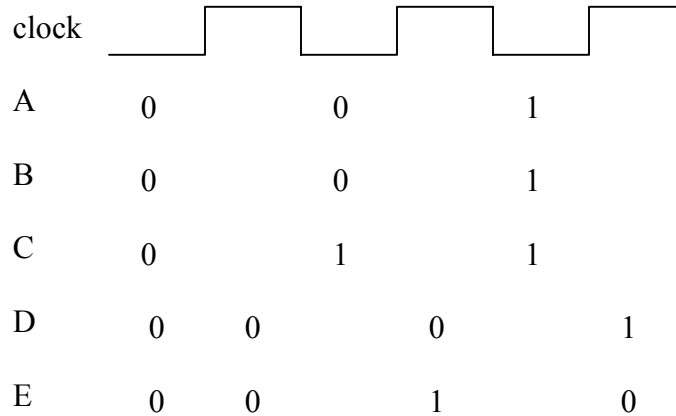
```
assign Out1
=(In3>In1)?((In1>In2)?In1:In2):((In3>In2)?In3:In2));
assign Out2
=(In3>In1)?((In1>In2)?In2:In1):((In3>In2)?In2:In3));
assign Out3
=(In3<In1)?((In1>In2)?In1:In2):((In3>In2)?In3:In2));
```

E.

None of the above.

Question 7

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



A.

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

B.

```
always@(posedge clock)
begin
    D = A & B;
    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 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;
    else E <= F * G;
    F <= A;
    G <= 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
    if (enable) E <= A * B;
    else E <= F * G;
    if (enable) F <= A;
    if (enable) G <= B;
end
```

D.

```
wire [31:0] A, B, D;
reg [31:0] E, F, G;
always@(posedge clock)
begin
    if (enable) E <= A * B;
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 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'b1100;

B. {a, b, c, d} = 4'b1110;

C. {a, b, c, d} = 4'b1111;

D. {a, b, c, d} = 4'b0100;

E. None of the above

Question 10

Which statement best describes why a high coverage of stuck-at-0 and stuck-at-1 faults is desirable.

A. It minimizes time on the test equipment.

B. It leads to an acceptably low number of flawed devices passing the test stage.

C. The overhead of scan chains can be avoided.

D. It minimizes the probability of a customer receiving a part with a design bug.

E. None of A-D.

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 {1 : 3 : 5} the clock skew is 1 ns, the flip-flop setup time is 1 ns and the hold time 2 ns.

Question 11

The fastest possible viable clock period is:

- A. 16 ns
- B. 17 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;
    #10 A = 1'b1;
    #delay1 A = 1'b0;
    #delay2 B = 1'b1;
  end
```

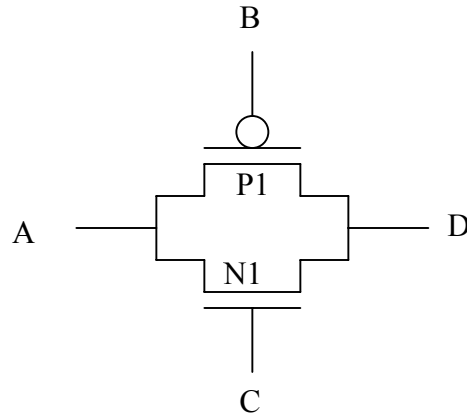
```
//fixture B
initial
  fork
    B = 1'b0;
    #10 A = 1'b1;
    #40 A = 1'b0;
    #60 B = 1'b1;
  join
```

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

What is a self-consistent set of states for the following CMOS design:



- A. A=1; B=1; C=0; D=1; N1 on; P1 on;
- B. A=1; B=0; C=1; D=1; N1 on; P1 on;
- C. A=1; B=0; C=1; D=z; N1 off; P1 off;
- D. A=1; B=0; C=0; D=z; N1 off; P1 off;
- E. None of the above.

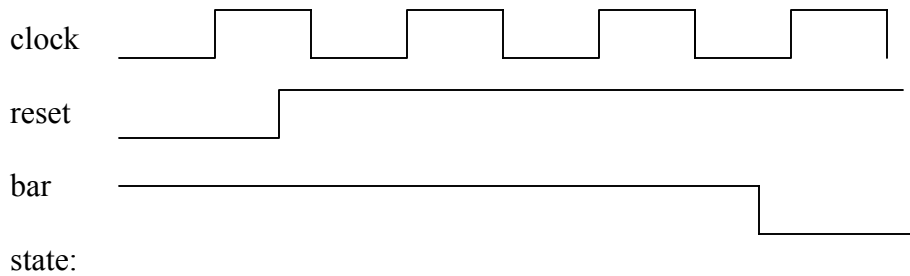
Question 15

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

```



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

Question 16

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 : 1 ;
  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 17

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

```
assign out = {&In, In[In[1:0]], In[1:0] >> 1};
```

- A. $out = 4'b0000;$
- B. $out = 4'b0001;$
- C. $out = 4'b1111;$
- D. $out = 4'b00x0;$
- E. None of the above.

Question 18

In this standard class synthesis script, the overall objective of the script was?

- A. To maximize the clock frequency.
- B. To simultaneously meet timing and area constraints.
- C. To minimize the area while meeting the timing constraints.
- D. To minimize how long it takes to do the compile.
- E. None of the above are true.

Question 19

For the following module, which is the best partition for ease-of-synthesis purposes? Note the group names after the begin: statements. I am asking you which combinations of groups would be the best partition. 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 = in2 + 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.

Question 20

Consider the following specify block:

```
specify
  specparam A0spec = 1 : 2 : 3;
  specparam A1spec = 2 : 3 : 4;
  (a => b) = (A0spec, A1spec);
endspecify
```

This is defining the following:

- A. Rising, falling and steady delay from input a to output b of 1, 2, and 3 ns respectively when a is 0, and 2, 3, and 4 ns when a is 1.
- B. Minimum, typical and maximum delay from input a to output b of 1, 2, and 3 ns on a rising edge at B, and 2, 3 and 4 ns on a falling edge.
- C. Non-blocking assignment of a to b with minimum, typical and maximum delay of 1, 2 and 3ns.
- D. Setup time requirements for the flip-flop with output B.
- E. None of the above.