

**ECE 464 / ECE 520/ DS 510P**  
**Final 2002**

Each question is worth 2 points. The correct answer earns 2 points, the incorrect 0. You have up to two hours to take this test. Answer all of the questions. Off-campus students, please return this test completed by the end of May 15.

The exam is open book, open notes.

NCSU students : Write your name and student number on the bubble sheets in the specified locations. I would like to display final scores against your student number. If you do NOT wish this for your test results do not fill in your number on the bubble sheets.

IN ADDITION, mark 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					
21					
22					

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

### Question 1

What is wrong with the following code fragment?

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

- A. D is missing from the sensitivity list.
- 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. Latches are implied.
- 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
    D = A & C;
    D = C | B;
    E = D ^ B;
end
```

- A. D is missing from the sensitivity list.
- 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. Latches are implied.
- 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;
    F = C | B;
end
```

- A. D is missing from the sensitivity list.
- 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. Latches are implied.
- E. Nothing is wrong with this code fragment.

**Question 4**

Out of the following choices, what is the most important advantage of an FPGA over a standard cell ASIC?

- A. It will result in a faster, smaller chip.
- B. It saves on mask costs, reducing NRE.
- C. Production costs are lower for large production runs.
- D. Writing RTL code is much easier for an FPGA.
- E. More than one of A-D are important advantages.

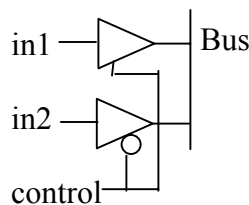
**Question 5**

Moore’s law will have the following impact on your career over the next ten years.

- A. Chips you will design will be able to implement novel highly sophisticated functions.
- B. You will be designing chips containing over 100 trillion transistors.
- C. Design teams will either have to grow, increased design reuse or even higher level tools will be required.
- D. A and B both apply.
- E. A and C both apply.

**Question 6**

Which code fragment correctly synthesizes the following logic.



- A.
 

```

trireg [3:0] bus;
always@(control or in1 or in2)
begin
  if (control) bus = in1;
  else if (!control) bus = in2;
  else bus = 4'hz;
end

```

B.  

```
tri [3:0] bus;
assign bus = (control) ? in1 : 4'hz;
assign bus = (control) ? 4'hz : in2;
```

C.  

```
tri [3:0] bus;
assign bus = (control) ? in2 : 4'hz;
assign bus = (control) ? 4'hz : in2;
```

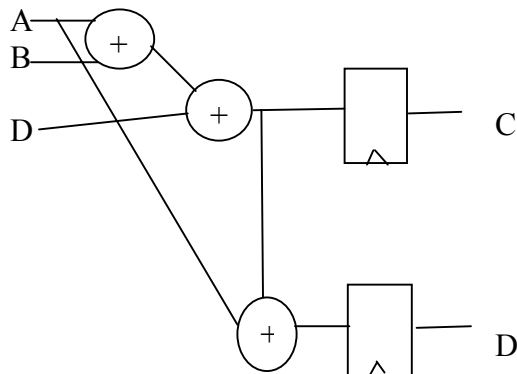
D.  

```
tri [3:0] bus1, bus2;
assign bus1 = (control) ? in1 : 4'hz;
assign bus2 = (control) ? 4'hz : in1;
```

E.  
None of the above.

### Question 7

Which code fragment synthesizes the following logic.



A.  

```
reg A, B, C, D;
always@(posedge clock)
begin
    C <= A + B + D;
    D <= C + A;
end
```

B.  

```
reg A, B, C, D;
always@(posedge clock)
begin
```

```

    C = A + B + D;
    D = C + A;
end

```

C.

```

reg A, B, C, D;
always@(posedge clock)
begin
    C <= A + B + D;
    D <= A + D;
end

```

D.

```

reg A, B, C, D;
always@(posedge clock)
begin
    C = A + B + D;
    D = D + A;
end

```

E. None of the above

### Question 8

If “enable” is high only 10% of the time, and the logic is built exactly as described, which of the following will have the lowest power consumption.

A.

```

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

```

B.

```

wire [31:0] A, B, D;
reg [31:0] E;
always@(posedge clock)
    if (enable) E <= D;
    else E <= A;
assign D = (~enable) ? A : A * B;

```

C.

```

wire [31:0] A, Aint, Bint, B, D;
reg[31:0] E;
always@(posedge clock)

```

```

        if (enable) E <= D;
        else E <= A;
assign D = Aint * Bint;
assign Aint = (enable) ? A : 32'h0;
assign Bint = (enable) ? B : 32'h0;

```

D.

```

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

```

E. These all assume about 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 c = a | b;
assign e = c & d;
assign g = e | f;

```

- A. {a, b, d, f} = 4'b0110;
- B. {a, b, d, f} = 4'b0010;
- C. {a, b, d, f} = 4'b0111;
- D. {a, b, d, f} = 4'b1111;
- E. None of the above

### Question 10

Which statement best describes the difference between design for test (DFT) and verification.

- A. There is no difference. They both permit you to detect faults in the design.
- B. DFT permits you to detect and analyze bugs in the design, while complete verification is important as the chip comes off the fab line.
- C. DFT allows you to work out if a manufactured chip has no faults, while verification ensures there are no bugs in the design before it goes to manufacturing.
- D. The vectors used for DFT and verification should be very similar.

E. None of A-D describe the difference very well.

### Questions 11 and 12

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

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

Each gate has a delay from its input to output as given as {1 : 2 : 4},  $t_{Cp_Q}$  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. 13 ns
- B. 14 ns
- C. 15 ns
- D. 16 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;
        #20 A = 1'b0;
        #40 B = 1'b1;
    join

```

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

- A. delay1 = 20; delay2 = 40;
- B. delay1 = 10; delay2 = 10;
- C. delay1 = 20; delay2 = 10;
- D. delay1 = 10; delay2 = 20;
- E. None of these are correct

#### Question 14

In verification, “code coverage” usually refers to the following:

- A. The percentage of the modules that are exercised during system level verification.
- B. The percentage of state sequences exercised in all the FSMs and the percentage of combinational logic and sequential logic exercised.
- C. The percentage of statements exercised in the code during the total verification effort.
- D. The percentage of statements in which all possible paths to the statement in question are exercised during verification.
- E. None of the above.

#### Question 15

During high level design, it is important to clearly separate control and datapath. Which statement best states the reason why?

- A. It clearly separates cause and effect. This makes it easier both to produce the design and debug it.
- B. This usually results in registers at the output of every module, simplifying synthesis.
- C. There is no other viable coding alternative.
- D. A controller is usually an FSM or counter and they must always be synthesized separately.
- E. None of these are close to being a good reason.

#### 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 : 0 ;
  1 0 : 1 ;
endtable;
endprimitive

```

- A. A toggle flip-flop.
- B. A mux.
- C. A NOR gate.
- D. A XOR gate
- E. None of the above.

**Question 17**

In verification, most of the effort should be applied at the system (complete chip) level. Which of the following statements gives the best reason as to why?

- A. Most of the bugs in a design are in the netlist wiring it together.
- B. Most of the bugs in a design are due to poorly understood interactions between different modules.
- C. This is the fastest way to verify the individual modules that make up the design.
- D. Most of the bugs in a design occur because of poorly designed interfaces, e.g. buses.
- E. None of the above are remotely a good reason.

**Question 18**

In this standard class synthesis script, we did another worst-case timing check after the doing an incremental compile to fix hold violations because?

- A. We want to make sure that any changes introduced during the incremental compile did not increase the delay in the critical path, causing a set-up violation.
- B. We want to make sure that any changes introduced during the incremental compile don't result in a hold violation elsewhere in the design.
- C. We want to make sure that any changes introduced during the incremental compile don't cause critical paths to be swapped.
- D. We want to make sure that any changes introduced during the incremental compile don't result in the removal of flip-flops.
- E. None of the above are true.

**Question 19**

You are designing a JPEG (a standard still image format) decoder chip for use in a cell (mobile) phone. Your design goal is best stated as follows:

- A. You want to maximize the clock frequency so that the image is decoded quickly.

- B. You want to minimize the maximum power consumed by the chip.
- C. You want to minimize the product of power and the time it takes to decode an image.
- D. You want to take maximum advantage of the available memory bandwidth by using a systolic array.
- E. None of these are a good goal.

**Question 20**

What does the following code fragment implement:

```
always@(B or C)
  if (B) D = C;
```

- A. A mux.
- B. A tri-state buffer.
- C. A latch.
- D. A OR gate.
- E. None of the above.

**Question 21**

How many flip-flops are in the following design:

```
reg A, B, D, E;
always@(posedge clock)
  begin
    if (B) A = D;
    if (A) D = A ^ B;
    A = A | E;
  end
```

- A. One
- B. Two
- C. Three
- D. Four
- E. None of the above

**Question 22**

Consider the following specify block:

```
specify
  specparam A0spec = 1 : 2 : 3;
  (a => b) = A0spec;
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.

- B. Minimum, typical and maximum delay from input a to output b of 1, 2, and 3 ns respectively.
- 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.