

**ECE 464-001 / 520-001 -601**  
Midterm, 2009

Name: \_\_\_\_\_  
Student Number: \_\_\_\_\_  
Course/Section: \_\_\_\_\_

Each question is worth 1 point. The correct answer earns 1 point, the incorrect 0.

The exam is open book, open notes. No computers are permitted. **75 minutes.**

Write your name and student number on top of this sheet of paper and turn that in. Please write your answer in the boxes provided below.

<b>Q.</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

**Take care answering these questions. Work carefully through your answers and check them before turning them in. Silly errors can cost you a lot in a multiple choice exam. Don't rush the test.**

**Question 1**

You are designing a digital chip intended to be part of a battery-driven sensor. The chip has to do a lot of high performance digital signal processing and the battery life should be about a day, and several million units will be made. Which ASIC implementation style would you use?

- A. Full custom design.
- B. Standard Cell ASIC.
- C. Gate Array.
- D. FPGA.
- E. None of the above.

### Question 2

Which statement best describes the relevant impact of mask costs on a standard cell ASIC vs. a gate array ASIC. .

- A. A standard cell ASIC has more total masks layers, and that is why it costs more in low volumes.
- B. A gate array requires no customized mask layers and that is why it costs less in low volumes.
- C. A standard cell ASIC requires more customized mask layers than a gate array, and thus costs more in low volumes.
- D. A gate array is more specialized than a standard cell ASIC which is why it costs more in low volumes.

### Question 3

Which statement is most true about timing calculations to determine the clock period?

- A. Both setup and hold constraints determine the clock period.
- B. The clock period is set by using best timing conditions and a skew direction that gives a longer clock period.
- C. The clock period is set by using worst case timing conditions and a skew direction that gives a longer clock period.
- D. The clock period is set by using worst case timing conditions and a skew direction that gives a shorter clock period.
- E. None of the above

### Questions 4, 5, 6

The minimum and maximum delays between each set of successive gates are marked as #(min: typical: max) ns and are marked on the output node of the driving gate.

**(Remember the timing equations are  $\leq$  and  $\geq$  constraints).** You also need the following:

- $T_{\text{setup}} = \#(1 : 1 : 2)$  ns.
- $T_{\text{hold}} = \#(1 : 1 : 2)$  ns.
- $T_{\text{skew}} = \#(1 : 1 : 1)$  ns for the clock.
- $T_{\text{clock-Q}} = \#(2 : 3 : 5)$  ns.
- $T_{\text{logic}} = \#(1 : 2 : 3)$  ns for each and every logic gate.

Consider the following code (implemented exactly as described):

```
reg A, B, C, D, E;

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

**Question 4**

What is the fastest possible clock period that satisfies setup constraints across all variations?

- A. 5 ns
- B. 7 ns
- C. 17 ns
- D. 20 ns
- E. None of the above are correct.

**Question 5**

Is there a potential hold violation?

- A. Yes, the fastest logic is 2 ns too fast.
- B. Yes, the fastest logic is 1 ns too fast.
- C. No, but there is zero margin
- D. No, there is a 1 ns safety margin.
- E. No, there is a 2 ns safety margin.

**Question 6**

Which statement is most true about “cycle stealing”?

- A. It gives more flexibility to the designer, usually permitting a slightly faster clock.
- B. It reduces the potential for hold violations.
- C. It can only be enabled when using flip-flops.
- D. When enabled, it enables the design to go 50% faster.
- E. None of the above.

**Question 7**

Consider the following code. It is the only code in the module.

```
always@(posedge clock)
    C <= A+B;
always@(C)
    D = ^C;
```

If it is synthesized with the following constraints

```
Create_clock -period 8 -waveform {0 4} -name clock
set_clock_skew -uncertainty 1.0 clock

set_input_delay 2.0 -clock clock all_inputs() - clock
set_output_delay 2.0 -clock clock all_outputs() - clock
```

Flip flop tck-Q delay is 2 ns, setup time is 2 ns, and hold time 1 ns.

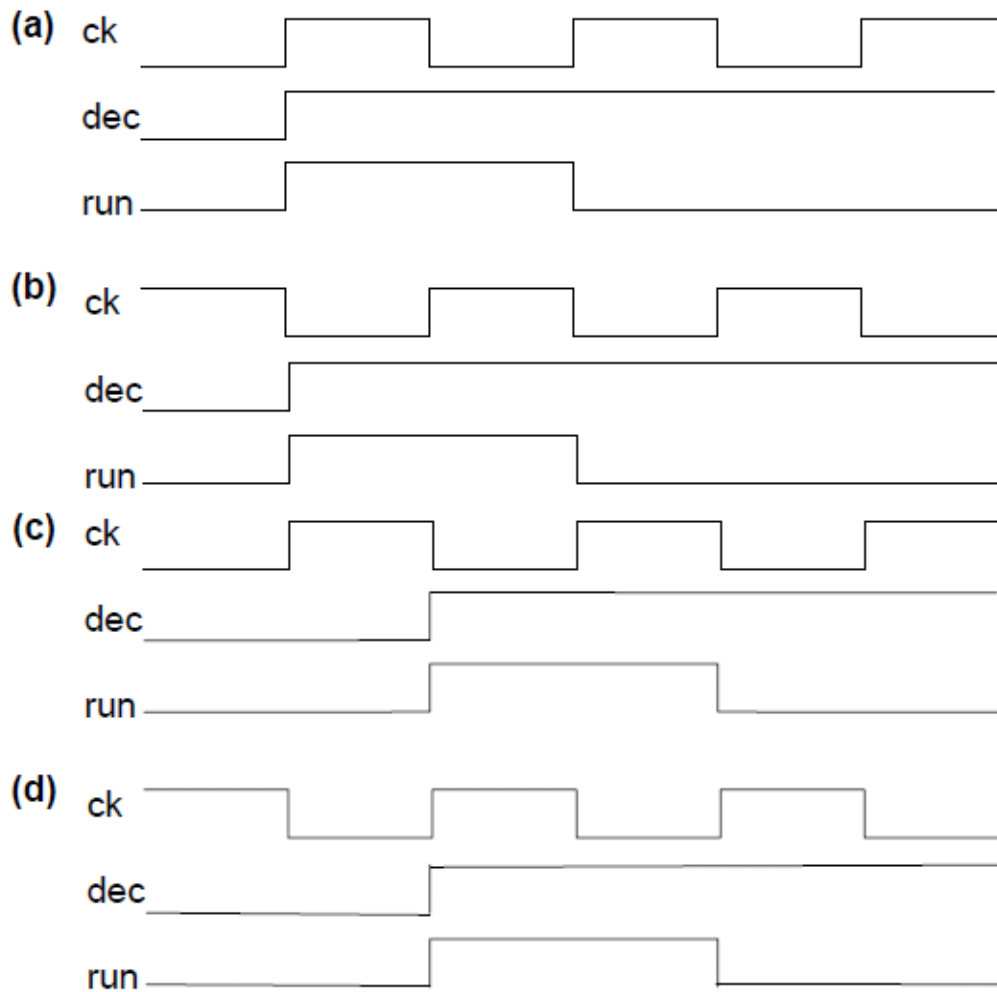
What is the maximum allowed delay for the logic A+B?

- A. 3 ns
- B. 4 ns
- C. 5 ns
- D. 6 ns
- E. None of the above

### Question 8

In the following test fixture, what is the timing diagram being specified?

```
initial
  dec = 0; run = 0; ck = 1;
  #10 dec =1; run = 1;
  #10 run = 0;
  #10 $finish;
end
always #5 ck = ~ck;
```



E. None of the above

**Question 9**

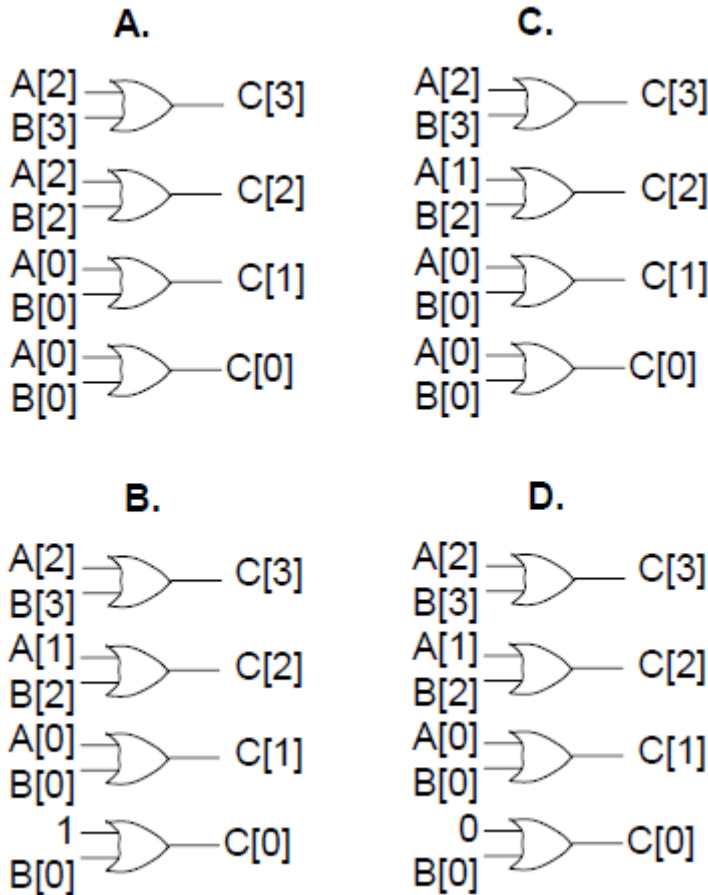
In the class synthesis script, the command “set\_fix\_hold clock” does what (precisely)?:

- A. Commands design compiler to change the design and insert delays as needed to fix timing.
- B. Specifies a rule to be applied when “compile -only\_design\_rule” is run.
- C. Checks the timing for hold violations and reports the results to the designer.
- D. Performs an incremental compile to fix any hold violations by inserting logic as need be.
- E. None of the above.

**Question 10**

Which alternative will the following code fragment synthesize to?

```
reg [3:0] A, B, C;  
always@(*)  
  C = {A << 1} | {B[3:2], {2{B[0]}}};
```



**Question 11**

Is the following code fragment an example of priority or non-priority logic?

```
always@(*)
  casex (A)
    {1x} : B=1;
    {00} : B=2;
    {01} : B= 3;
    Default : B = x;
  endcase
```

- A. Priority.
- B. Non Priority.

**Question 12**

Which is the most accurate statement about unintentional latches (latches with a clock being generated from logic) considered?

- A. If they simulate correctly pre-synthesis they can be accepted.
- B. They can be used to enable cycle stealing.
- C. Commands must be added to the synthesis script to handle them
- D. Since their clocks can be glitchy, they will not function correctly in the actual logic.
- E. None of the above are true.

**Question 13**

What is the correct binary representation of the number being assigned to B after the block is evaluated?

```
reg [3:0] A, B, C;

always@(*)
  begin
    B = 5'hFF;
  end
```

- A. 1111 1111;
- B. 1 1111;
- C. 1111;
- D. 0111;
- E. None of the above.

**Question 14**

What is the correct binary representation of the number being assigned to C after the block is evaluated?

```
reg [3:0] A, B, C;
```

```
always@(*)  
begin  
    C = 4'bx;  
end
```

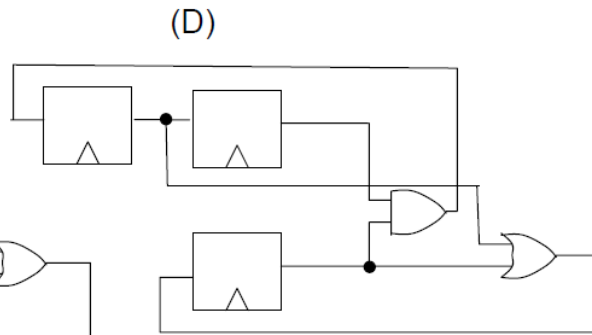
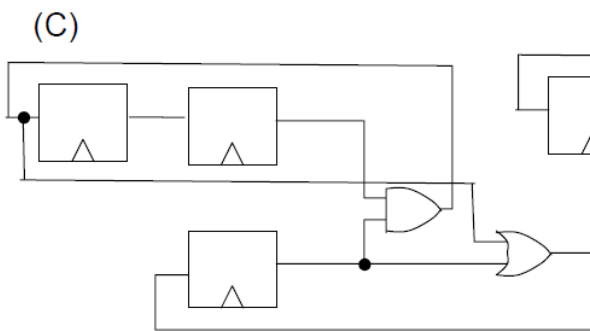
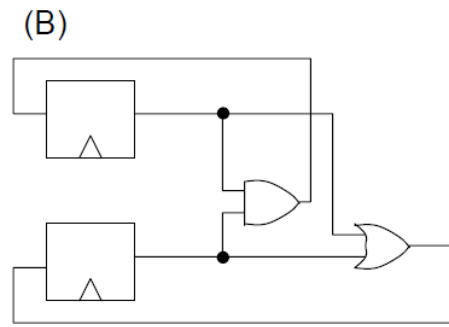
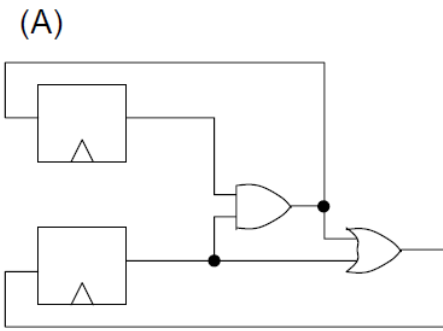
- A. 0000 0000 0000 0000 0000 0000 000x;
- B. xxxx xxxx xxxx xxxx xxxx xxxx xxxx;
- C. 000x;
- D. xxxx;
- E. None of the above

**Question 15**

You asked for this question many times, so I am giving it to you, and it's a tough one. The following combinational logic uses non-blocking assignments. Which logic sketch will produce the closest behavior to the code when simulated? Note, I am NOT asking what the code will synthesize too. This shows why using non-blocking in CL is dumb.

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

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



E. None of the above. DRAW YOUR SOLUTION.