

**ECE 464-001 / 520-001 -601**  
Midterm, 2010

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**

The number of new ASIC designs being started each year goes down year by year. Which statement best describes the reason why?

- A. Society has need for fewer different types of high performance digital parts than it has in past years.
- B. The cost of designing a part and getting the first production run has increased to the point where fewer markets can produce enough revenue to justify the investment.
- C. The performance and cost gap between FPGAs and standard ASIC has closed so much, that the latter are rarely justified.
- D. The performance and cost gap between gate arrays and standard ASIC has closed so much, that the latter are rarely justified.

### Question 2

You are designing a digital chip intended to be used in a system with a total production run of approximately 100 units. The main architecture of the system is a 64-bit wide pipeline. There are several complex stages in the pipeline but the throughput requirement is modest – around 100 Mwords per second. There is a reasonable chance that some functionality would have to be updated during the production run. Which ASIC implementation style would you use?

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

### Question 3

Which statement is most true about timing calculations to determine the clock period and prevent hold violations?

- A. Flip-flop based design is preferred because set-up constraints are easier to satisfy than in a latch based design.
- B. Latch based designs will always operate at a 50% faster clock rate than flip-flop based designs (assuming a 50% duty cycle).
- C. It is easier to prevent hold violations in a latch based design than in a flip-flop based design.
- 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, 7

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 from any input to any output.

Consider the following code (implemented exactly as described):

```
reg A, B, C, D, E;
always@(posedge clock)
begin
    A <= D ? B : 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. 4 ns
- C. 10 ns
- D. 14 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

If this design was converted to a latch based design and “cycle stealing” was enabled, what would the clock period be (assuming a 50% duty cycle and rounded to the nearest integer)?

- A. 5 ns
- B. 10 ns
- C. 9 ns
- D. 14 ns
- E. None of the above are correct.

#### Question 7

If you intended a latch based design how would the trigger statement be changed?

- A. always@(posedge clock or negedge clock)
- B. always@(negedge clock)
- C. always@(clock)
- D. always@(clock or A or B or C or D or E)
- E. None of the above

#### Question 8

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

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

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 1.0 -clock clock all_inputs() - clock
set_output_delay 2.0 -clock clock all_outputs() - clock

```

Flip flop tck-Q delay is 1 ns, setup time is 1 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

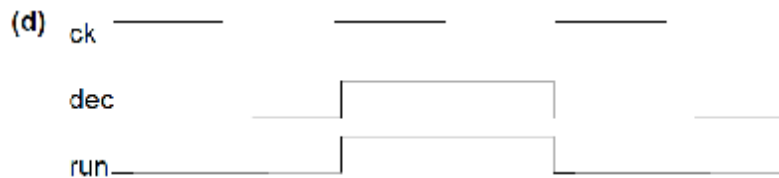
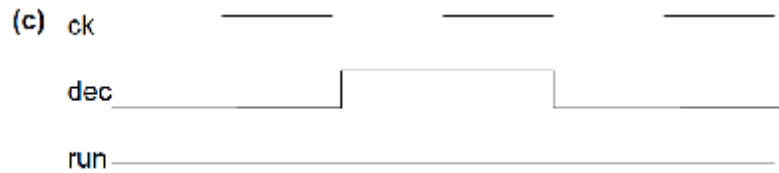
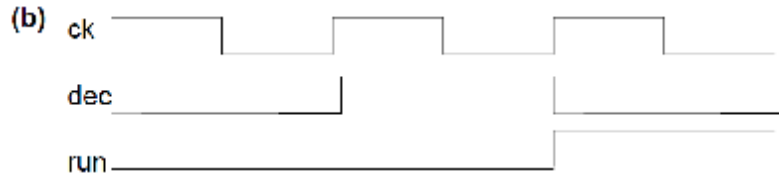
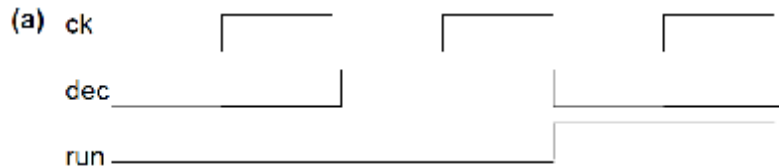
**Question 9**

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

```

initial
    dec = 0; run = 0; ck = 1;
    #30 $finish;
end
always #5 ck = ~ck;
always #10 dec=~dec;
always #10 run=run ^ dec;

```



- E. None of the above

### Question 10

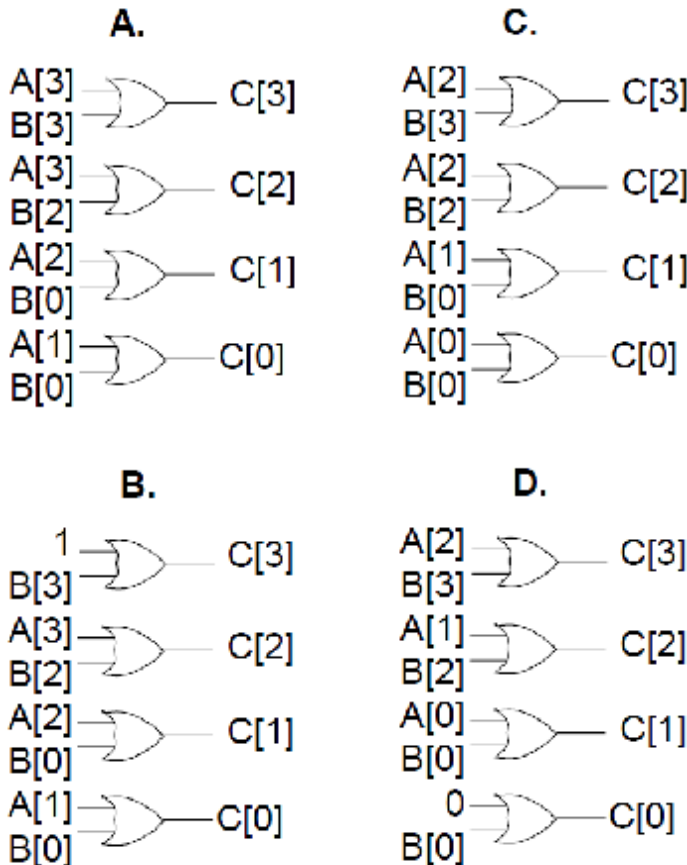
In the class synthesis script, the command “translate” is used several times. Within the context of this script the purpose of translate can best be described as?

- A. Translating from this cell library in a slow process to the same cell library in a fast process, and vice-versa. The actual cell designs do not change – the transistor level designs are identical.
- B. Translating from a fast cell library to a slow cell library and vice-versa. The actual cell designs do change to introduce faster (or slower) transistor level designs.
- C. Translate the design from a fast clock implementation to a slow clock implementation and vice-versa.
- D. Translate the design from performing set-up checks to one performing hold checks.
- E. None of the above.

### Question 11

Which alternative will the following code fragment synthesize to?

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



**Question 12**

Is the following code fragment an example of correct use of a for loop in synthesizable code? `reg [8:0] A;`

```
always@(*)
begin
    B=0;
    for (i=0; i<=8; i=i+1)
        if (A[i]) B = B+1;
end
```

- A. True
- B. False

**Question 13**

What type of bug might NOT be caught if “`default: out=x`” is not included in case statements?

- A. An unintentional latch.
- B. A timing arc.
- C. Unintentional wired-of logic.
- D. A bug in the reset logic.
- E. None of the above.

**Question 14**

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 = 6'hF;
end
```

- A. 11 1111;
- B. 00 1111;
- C. xx 1111;
- D. 1111;
- E. None of the above.

**Question 15**

Consider the following code block. Note the using of BLOCKING assignment.

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

Which of the following non-blocking assignment code blocks will produce the same results?

A.

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

B

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

C

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

D

None of the above