

CSE370 Winter 2010 – Exam 2 (24 February 2010)

Please read through the entire examination first! This exam was designed to be completed in 50 minutes and, hopefully, this estimate will be reasonable.

There are 3 problems for a total of 110 points. The point value of each problem is indicated in the table below. Each problem and sub-problem is on a separate sheet of paper. Write your answer neatly in the space provided. If you need more space (you shouldn't), you can write on the back of the sheet where the question is posed, but please make sure that you indicate clearly the problem to which the comments apply. Do NOT use any other paper to hand in your answers. If you have difficulty with part of a problem, move on to the next one. They are mostly independent of each other.

The exam is CLOSED book and CLOSED notes. Please do not ask or provide anything to anyone else in the class during the exam. Make sure to ask clarification questions early so that both you and the others may benefit as much as possible from the answers.

Name:

ID#:

Problem	Max Score	Score
1	45	
2	20	
3	45	
TOTAL	110	

1. Adders (45 points)

(a – 15 pts) The planet Polydactylus-12 uses a base-12 number system. They are just making their way into the digital age and are in need of a basic BCP (binary-coded polydactyl) adder. A BCP adder adds to BCP digits (between 0 and 11) and asserts a carry out if the sum is greater than 11 as well as the sum BCP digit (also between 0 and 11).

To start, write a Verilog description of the BCP module. Below is a module definition for you to fill in. Assume that your inputs A and B will already be legal BCP digits (i.e., 0 to 11). Note: you should not require more than a few lines of Verilog, if you find yourself writing more than that, STOP, and re-consider your approach.

```
module BCP_adder (Cout, S, A, B, Cin)
```

```
    output Cout;
    output S[3:0];
    input  A[3:0], B[3:0];
    input  Cin;
```

```
    reg C[4:0];
```

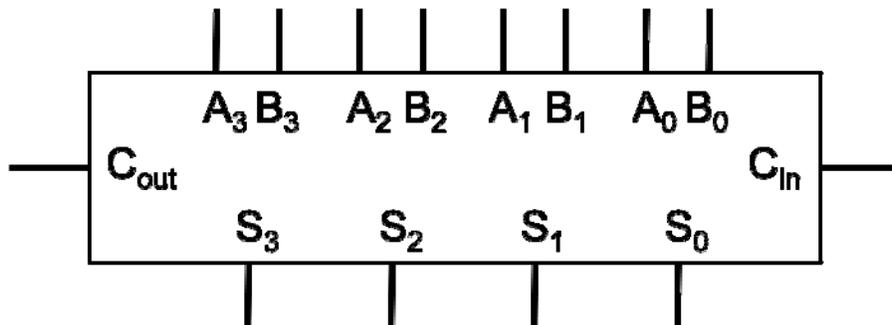
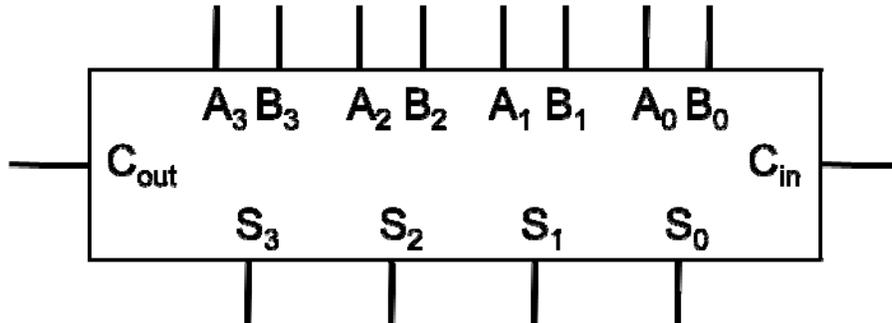
```
    //    YOUR CODE GOES HERE
```

```
    //    BETWEEN COMMENTS
```

```
    assign Cout = C[4];
    assign S[3:0] = C[3:0];
```

```
endmodule
```

(b – 20 pts) The next step is to develop an implementation of the BCP adder quickly. You are in luck since Earth has a surplus of 4-bit binary adders. Construct a BCP single-digit adder using no more than two 4-bit binary adders (two have been drawn below so that you can use them, if your solution requires them) and as few additional gates as possible (none shown below). Don't forget to consider C_{in} and C_{out} of your module so that BCP adders can be combined easily for multiple digits.



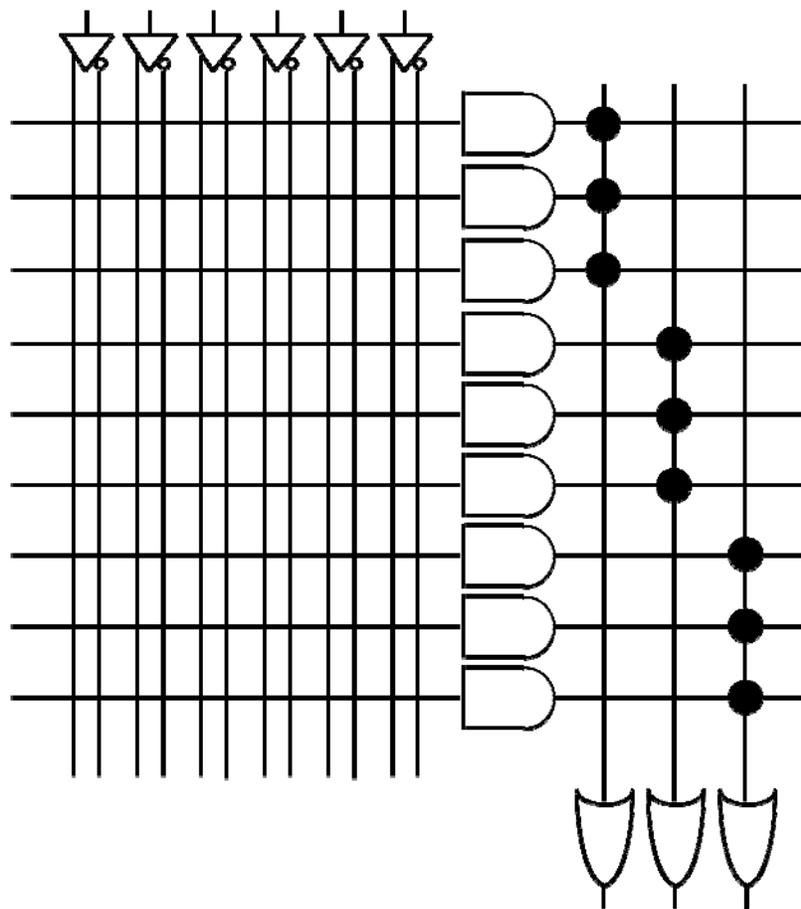
(c – 10 pts) If the delay of 4-bit adder of part (b) is 4 time units for the S bits and 3 time units for C_{out} , what is the delay of your implementation of the BCP adder (assume that all gates have a delay of 1). Make sure to explain how you derived your final delay for both C_{out} and the S outputs. Please highlight your result.

2. Programmable Logic (20 points)

Given the following K-maps for two logic functions, X and Y, respectively, show how you would program the PAL on the next page (6 inputs, 3 outputs, 9 product terms overall) to implement them. Clearly label all you inputs, outputs, and product terms on the PAL.

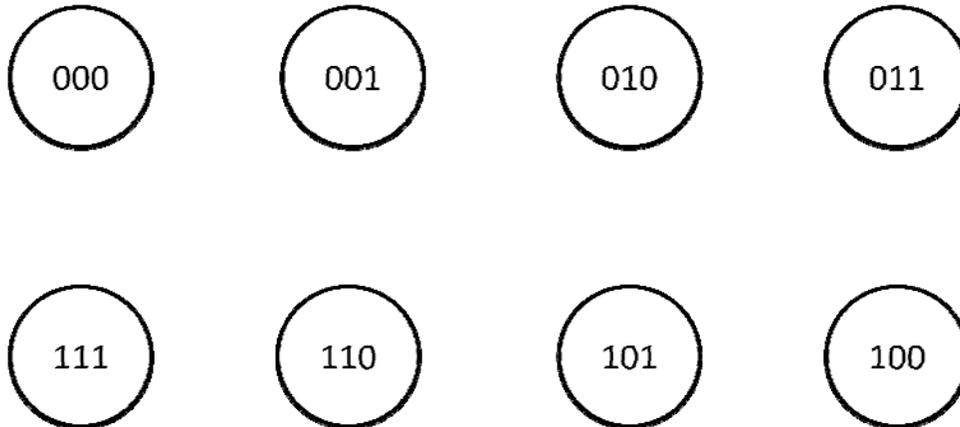
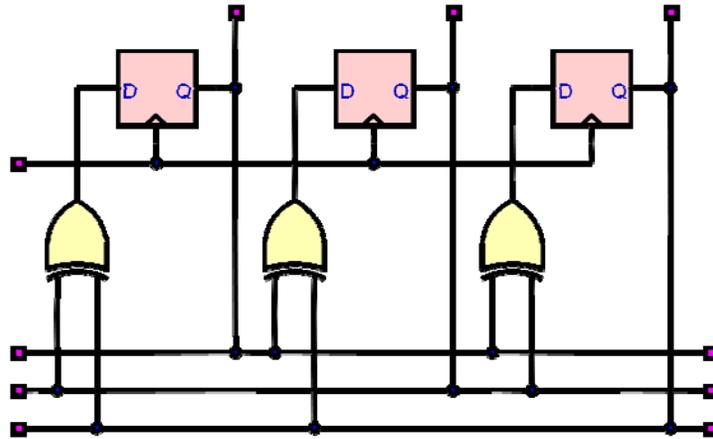
		A				
		<i>0</i>	<i>0</i>	<i>1</i>	<i>0</i>	
		<i>0</i>	<i>0</i>	<i>0</i>	<i>1</i>	
		<i>1</i>	<i>0</i>	<i>0</i>	<i>0</i>	D
C		<i>0</i>	<i>1</i>	<i>0</i>	<i>0</i>	
		B				

		A				
		<i>0</i>	<i>0</i>	<i>1</i>	<i>1</i>	
		<i>0</i>	<i>0</i>	<i>0</i>	<i>1</i>	
		<i>1</i>	<i>0</i>	<i>0</i>	<i>0</i>	D
C		<i>1</i>	<i>1</i>	<i>0</i>	<i>0</i>	
		B				



3. Registers/Counters (45 points)

(a – 15pts) Given the following circuit, derive the state diagram by filling in the arcs below (the state codes in the circles correspond to that states of the FFs in the same order). Is this a counter? If so, why? Is it a self-starting counter? If it isn't a counter, why not?



(b – 5 pts) Given the Verilog description below, synthesize the corresponding circuit.

```
module problem_3_b (clk, A, B, C)

    output A, B, C;
    input  clk;

    reg    A, B, C;

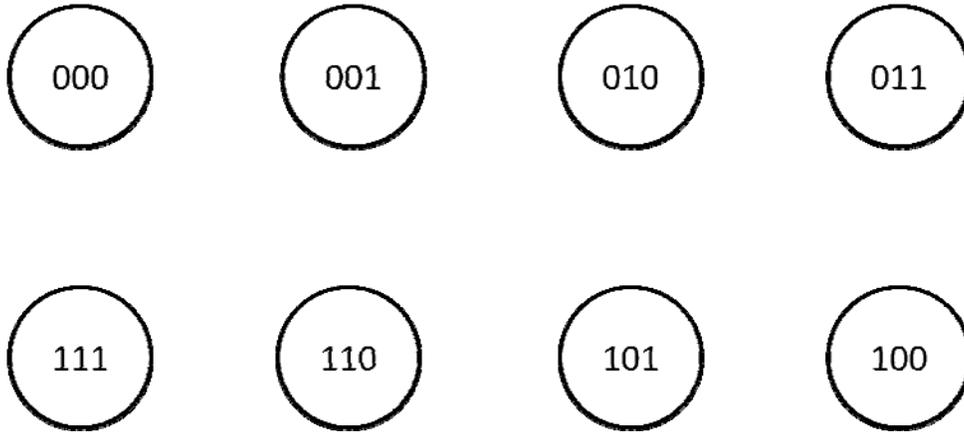
    always @(posedge clk) B <= A;

    always @(posedge clk) C <= B;

    always @(posedge clk) A <= A ^ (B ^ C);

endmodule
```

(c – 10 pts) Determine the state diagram for the circuit you synthesized in part (b) by filling in the arcs below (the state codes in the circles correspond to the FFs A, B, C). Is this a counter? If so, why? Is it a self-starting counter? If it isn't a counter, why not?



(d – 15 pts) The circuit below describes a counter that is NOT self-starting, use no more than a single 2-input OR gate to make the counter self-starting when a “RESET” input is asserted. Note that RESET is just another input, the FFs are not resettable or settable. The state diagram skeleton below is provided simply for your convenience, the state codes in the circles correspond to that states of the FFs in the same order.

