ECEN 449 - Microprocessor System Design
Test #1

Instructor: Sunil P Khatri

Spring 2014

NAME:

Problem	maximum points	your points
Problem 1	20	
Problem 2	20	
Problem 3	25	
Problem 4	35	
Total	100	

For all your answers which involve Verilog code or C code, you should provide comments in your code for full credit.

For other answers, you must provide comments about *how* you obtained the answer for full credit.

If you give me more than one answer for the same question, I will grade all your answers, and score your points for the answer which obtained the least points.

For a question with N points, I suggest that you budget 1.2 $\times$ N minutes for it.

1. Assume you have a clock signal `CLK`. I want a circuit which takes `CLK` and `mode` as input, and produces the output `CLKOUT`, which is the `CLK` signal divided by 13 or 7. When the input `mode` is 1, `CLK` is divided by 13, and when `mode` is 0, `CLK` is divided by 7.

   (a) Write a Verilog module to implement the above circuit. You may use either structural, dataflow or behavioral Verilog, as you prefer.

   (b) Write down the truth table for the `notif1(out, in, control)` built-in gate using 4-valued logic, in the template below:

control

in	0	1	X	Z
0				
1				
X				
Z				

   (c) A gate is said to be `symmetric` if the output of the gate is unchanged when the inputs are swapped. For example, an `AND` gate is symmetric, since `AND(a,b)` yields the same output as `AND(b,a)`, for all values of `a` and `b`. Is the `notif1` gate of part (b) above symmetric? Why?

This page intentionally left blank

This page intentionally left blank

2. (a) Recall that the target variable of an `assign` statement in Verilog cannot be a register. State why this is reasonable based on the semantics of the `assign` construct.

   (b) Recall that the target variable in an `always` block in Verilog must be a register. State why this is reasonable based on the semantics of the `always` construct.

   (c) Assume that you are given a Verilog module for a full adder. The module for the full adder is `FA(sum, cy-out, a, b, cy-in)`, where the `a, b, cy-in` are the two operands being added, and the input carry respectively. The outputs are `sum, cy-out`, which are the sum and carry outputs respectively.

   Write a structural Verilog module to generate a 128-bit ripple carry adder, using the `FA` as a building block. More compact Verilog code will be given more credit.

This page intentionally left blank

This page intentionally left blank

3. (a) Suppose you are asked to implement a 32-input OR function (`F`). You need to realize this function using 4-input OR gates (you have an infinite supply of 4-input OR gates at your disposal). Each 4-input OR gate has a delay of 10ns.

   i. Assume that all 32 inputs of your function are available at time 0ns. Draw a circuit (using 4-input OR gates) that realizes the function `F` with the minimum delay. What is the delay of your circuit?

   ii. Now assume that 31 of the 32 inputs of your function arrive at time 0, and the 32^{nd} input arrives at time 30ns. Suppose you used the circuit you drew for part (i) above. What is the delay of your circuit?

   iii. Again assume that 31 of the 32 inputs of your function arrive at time 0, and the 32^{nd} input arrives at time 30ns. Draw a circuit (using 4-input OR gates) that realizes the function `F` with the minimum delay. What is the delay of your circuit?

(b) Suppose I have 4 user tasks that are running on my multitasking operating system (OS). The hardware on which my OS is running is a 1 GHz, single core processor. At time `t = 0`, these 4 tasks (let's call them `T1, T2, T3 and T4`) are 3, 3, 5 and 2 milliseconds away from completion respectively. All tasks have the same priority, and task `Ti` is said to have task ID `i`. The scheduling task uses round-robin scheduling, and takes 1000 clock cycles to run. A context switch takes 500 clock cycles. The scheduler runs any task for 1 millisecond before it switches to another task. The scheduler task is run each time the currently running task is retired, to determine which task to run next.

   i. At what time will `all` the 4 tasks finally complete execution?

   ii. What fraction of the total execution time is devoted to housekeeping (scheduling and context switches) tasks?

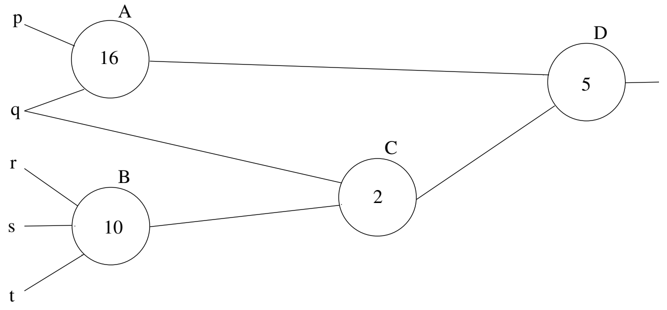
   iii. Starting at time `0`, write down the sequence in which the 4 tasks get scheduled, until the time that all the tasks finally complete execution. Assume that the round robin scheduler schedules tasks which have the lowest task ID first.

This page intentionally left blank

This page intentionally left blank

4. In this problem, we will write a C program that will be used by gourmet chefs to figure out how long it will take them to cook any recipe. The data that is needed for any recipe is expressed as a graph, with each graph node expressing i) the time it takes (in minutes) to perform the particular step in the preparation of the final recipe and ii) the other steps that must be completed before this step is performed. The gourmet chef has a large number of trainee chefs who are always eager and ready to help the gourmet chef by performing the steps needed for the recipe.

   An example of the graph is shown below:



   In this figure, I have labeled the graph nodes to make the following discussion easier. The ingredients required for the final recipe are `p, q, r, s` and `t`. Node A in the graph requires ingredients `p` and `q`, and it takes 16 minutes to complete the step of combining these ingredients. Similarly, node C in the graph needs ingredient `q`, and the output of the step represented by node B in the graph. The final "output" of the recipe is obtained after the step in node D has been completed.

   The struct for any node is:

```
struct node {
int duration;
node* child1;
node* child2;
node* child3;
node* child4;
int extra;
};
```

   Note that any node can have up to 4 children. If a node has `k` (less than 4) children, then the last `4-k` child pointers are null. In the example, node A's `child3` and `child4` pointers would be null. The ingredients in the recipe (`p, q, r, s` and `t` in our example) have null child pointers for all 4 children. Assume that all ingredients are available when the chef starts preparing the recipe.

   (a) What is the value of the `duration` entry for the nodes that correspond to the ingredients of the recipe?

   (b) Write a **recursive** C program that finds the time it would take to cook any given recipe. Your program is given a pointer to the `struct` corresponding to the final (output) node of the recipe. Your code should work for any recipe, not just the one in the example above.

   (c) Write a **recursive** C program that prints all the ingredients needed for a given recipe. Your program is given a pointer to the `struct` corresponding to the final (output) node of the recipe. Your code should work for any recipe, not just the one in the example above.

The ingredients should be printed in one line, with a space character between ingredients. The same ingredient should not appear more than once in the output.

(d) Show the call stack (i.e. the sequence of recursive calls) when your program in part (b) is run on the example graph shown above. What is the total time for preparing the recipe that your program returns?

(e) How would you make your code more efficient (i.e. make it run faster) by utilizing the `extra` field of the `node` structure? Do not rewrite the C program for this part. Instead, just state in words how you would use the `extra` field to improve code efficiency.

This page intentionally left blank

This page intentionally left blank