EE480 Spring 2018 Exam 0

Your name is: **`Sample solution`**

Your account name (as used for assignments) is:

The question you skipped is:

There are 11 questions. Each of the questions is worth 10 points; **you must answer any 10 and indicate above which one you skipped** (no extra credit). For each question with boxes, ☐, in front of each potential answer, read the question carefully to see if you should check a single answer or all answers that apply. The fifth answer is no, but you still need to explain why. The short answer questions should get **short** answers; excessively long answers may be considered incorrect.

No reference materials of any kind are permitted, nor are any electronics (e.g., no calculators nor cell phones).

1. Write a one-line AIK specification that will recognize and encode the KySMet **`add`** instruction such that **`add $7,$8,$9`** would be assembled into a 16-bit word with the value **`0x1897`** and **`add $12,$11,$10`** would result in **`0x1bac`**.

```
add $a , $b , $c := 1:4 b:4 c:4 a:4
```

2. In multi-pass resolution of **forward references**, we talked about two types of passes. We cleverly named them "pass 1" and "pass 2"... despite the fact that there might be multiple "pass 1". Briefly explain what the difference is between a "pass 1" and a "pass 2" — what does each do?

Pass 1: Make symbol table entries recording label values

Pass 2: Generate code using symbol table label values

There may be multiple "Pass 1" before making a "Pass 2."

3. *For this question, mark all answers that apply.* Which of the following Verilog expressions results in a value of 1?

- ■ `4'b1`
- ■ `a` *where* `or(a, 1, x);`
- ■ `a[2]` *where* `reg [2:0] a = 6;`
- ■ `b` *where* `reg a, b; initial {b,a}=2;`
- ■ `(1'bx === 1'bx) - (1'bx == 1'bx)`

4. *For this question, mark all answers that apply.* Which of the following statements about Verilog are true?

- ☐ Given `reg a[15:0];`, `a[7:0]=0` is a valid assignment
- ☐ The `<=` operator can be used to assign a value to a `wire`
- ☐ An `always` construct specifying `posedge` is not synthesizable
- ☐ `initial c=a&b;` means exactly the same thing as `and d(c, a, b);`
- ☐ A clock signal could be defined as `reg clk = 0; assign clk = ~clk;`

5. We've talked about both **line** and **toggle coverage**. If you have 100% toggle coverage, does that *imply* the tests also have 100% line coverage? Briefly explain.

100% toggle coverage does NOT imply 100% line coverage.

All bits may be toggled without having evaluated some of the contructs that could change those bits. Similarly, 100% line coverage does NOT imply 100% toggle coverage.

6. The KySMet instruction set is a little strange because it is designed to support SIMD execution. Briefly explain what the instruction **`jumpf $u0,lab`** does when executed.

Well, with just 1 PE it essentially acts like jump false. However, it actually disables the current PE if $u0 is false and then jumps to lab iff no PE is enabled.

7. The Quine-McCluskey algorithm and the Espresso tool both do very similar things, but not quite the same thing. Briefly explain the difference between the kinds of logic design problems these two tools solve. What kind of logic design problem is Espresso likely to find a cheaper circuit implementation for than would be found using Quine-McClusky?

Both tools find cheap SOP circuits to implement a given logic function.

Espresso uses heuristics to try to share circuitry between multiple output values being computed, while Quine-McCluskey designs an independent circuit for each output value.

8. What does Prof. Dietz advocate using **`` `define ``** for? Why?

`` `define is best used for things that are constants throughout your entire design. For example, the idea that the ADD opcode is 12 or that a WORD has bits [31:0]. ``

These uses simplify your design and ensure these aspects of the design will be globally consistent across modules.

9. What is **`$dumpvars`** used for?

Generating a trace of values changing.

Such traces can be manually examined for debugging, but also can be used by tools like covered for test analysis.

10. Question 5 mentioned *imply* — well, implies is a logic function that returns true unless the first operand is true and the second operand is false. Write a synthesizable combinatorial **`module implies(r, a, b)`** that implements the "implies" logic function where **`r`**, **`a`**, and **`b`** are all single wires such that r=(a implies b).

```
module implies(r, a, b);
output r;
input a, b;
assign r = (~a) | b;
endmodule
```

11. Write a **`module testbench`** to exhaustively test **`module magic(out8, in8)`** against **`module oracle(out8, in8)`**. You should assume you have been given the definitions of these two purely combinatorial modules and that **`out8`** and **`in8`** are repectively the 8-bit values output from and input to those modules.

```
module testbench;
reg [7:0] mo, oo, mi;
integer wrong = 0;

magic uut(mo, mi);
oracle ref(oo, mi);

initial begin
  mi = 0;
  repeat (256) begin #1
    if (mo != oo) begin
      wrong = wrong + 1;
      $display("Expected %d, got %d for %d", oo, mo, mi);
    end
    mi = mi + 1;
  end
  $display("%d wrong", wrong);
end
endmodule
```