Digital Design using HDLs
EE 4755
Midterm Examination
Monday, 10 November 2014  11:30–12:20 CST

Problem 1  __________  (20 pts)
Problem 2  __________  (20 pts)
Problem 3  __________  (10 pts)
Problem 4  __________  (15 pts)
Problem 5  __________  (13 pts)
Problem 6  __________  (22 pts)

Alias  ____________________________
Exam Total  __________  (100 pts)

Good Luck!
Problem 1: [20 pts] Write a Verilog description of the hardware shown below.

- Write a Verilog module corresponding to the hardware above.
- Be sure to declare module ports and any wires and vars (logic) used inside.
- Pay attention to the differences between lo and hi and the differences between sa and sb.
Problem 2: [20 pts] Appearing below is the multiply circuit from the solution to Homework 3, in Verilog (slightly simplified) and as a diagram showing what hardware a synthesis program might infer.

```verilog
module mult_seq_csa_m #( int wid = 16, int pp_per_cycle = 2 )
  ( output logic [2*wid-1:0] prod,
    input logic [wid-1:0] plier, input logic [wid-1:0] cand, input clk);

  localparam int iterations = ( wid + pp_per_cycle - 1 ) / pp_per_cycle;
  localparam int iter_lg = $clog2(iterations);
  localparam int wid_lg = $clog2(wid);

  logic [iter_lg:0] iter;
  wire [2*wid-1:0] accum_sum_a[0:pp_per_cycle], accum_sum_b[0:pp_per_cycle];
  logic [2*wid-1:0] accum_sum_a_reg, accum_sum_b_reg;

  assign accum_sum_a[0] = accum_sum_a_reg;
  assign accum_sum_b[0] = accum_sum_b_reg;

  for ( genvar i=0; i<pp_per_cycle; i++ ) begin
    wire [wid_lg:1] pos = iter * pp_per_cycle + i;
    CW_csa #(2*wid) csa
      (.sum(accum_sum_a[i+1]), .carry(accum_sum_b[i+1]), .a(accum_sum_a[i]), .b(accum_sum_b[i]), .c(pp));
  end

  always @( posedge clk )
    if ( iter == iterations ) begin
      prod <= accum_sum_a_reg + accum_sum_b_reg;
      accum_sum_a_reg <= 0;
      accum_sum_b_reg <= 0;
      iter <= 0;
    end else begin
      prod <= prod;
      accum_sum_a_reg <= accum_sum_a[pp_per_cycle];
      accum_sum_b_reg <= accum_sum_b[pp_per_cycle];
      iter <= iter + 1;
    end

endmodule

USE NEXT PAGE FOR SOLUTION

![Diagram of the multiply circuit from the solution to Homework 3, in Verilog (slightly simplified) and as a diagram showing what hardware a synthesis program might infer.](image)

USE NEXT PAGE FOR SOLUTION
(a) Show optimizations that might be performed that exploit the value \( m = 2 \) (that is, \( \text{pp\_per\_iteration}=2 \)).

(b) Show the optimizations that might be performed assuming that \( \text{wid} \) is odd, and assuming that \( \text{wid} \) is even, both for \( m = 2 \).

Modify diagram to show optimizations for \( \text{pp\_per\_iteration} = m = 2 \) and arbitrary \( \text{wid} \).

Modify diagram to show optimizations for \( \text{pp\_per\_iteration} = m = 2 \) and odd \( \text{wid} \).

Modify diagram to show optimizations for \( \text{pp\_per\_iteration} = m = 2 \) and even \( \text{wid} \).
Problem 2, continued:

(c) The cost of the shifters with input plier in the design on the previous pages is significant. Explain how these shifters can be eliminated by adding a register. Quickly sketch the hardware to illustrate your answer.

☐ Show how a register can be used to eliminate the costly shifters.

(d) Explain how the streamlined multiplier described in class eliminated the plier shifter without having to add a register.

☐ Show how the streamlined multiplier does not need an extra register to eliminate the shifter.
Problem 3: [10 pts] The module below computes the prefix sum of a sequence of integers at its input.

```vhls
module prefix_sum #( int len=8, int wid = 8)
    (output logic [wid:1] psum [len], input [wid:1] elts[len]);
    always @* begin
        psum[0] = elts[0];
        for ( int i=1; i<len; i++ ) psum[i] = psum[i-1] + elts[i];
    end
endmodule
```

(a) Show the hardware that would be synthesized for the module before optimization, elaborated with parameters len=4 and wid=8. Label the input ports elts[0], elts[1], elts[2], and elts[3]; and label the output ports psum[0], psum[1], psum[2], and psum[3].

☐ Show synthesized hardware.

(b) Estimate the delay for the synthesized hardware before optimization. Use $w$ for the value of wid and $L$ for len. Assume that a $w$-bit adder has delay $w$.

☐ Delay in terms of $w$ and $L$: 
Problem 4: [15 pts] Answer the following questions about the Verilog module below.

```verilog
module timing();
logic [7:0] a, e, f2, g, g1, g2;  logic clk; wire [7:0] e1, f, f1;
initial begin
    clk = 0;
a = 11;
    #1;
a = 1;
a <= 22;
a <= #5 a + 1;

    #9;
a = 7;
e = 10;
f2 = 30;
g = 40;
g1 = 50;
g2 = 60;
    #10;
    // B0
    a <= 700;
    clk = 1;
    #1;
    // POINT X (See subproblem.)
end
always @(posedge clk ) e = a; // B1
always @* e1 = a; // B2
always @* f = e + 1; // B3
always @* f1 = e1 + 1; // B4
always @(posedge clk ) f2 <= e + 1; // B5
always @(posedge clk ) begin // B6
    g = f; g1 = f1; g2 = f2;
end
endmodule
```

(a) Show values for a versus time in the table below. For this part, only a. The table already shows that a has value 11 from time 0 to time 1. Extend the table as long as necessary, and be sure to show values for both t and a. Note: The original exam did not provide the table. Also, in the original exam there were differences in how a was assigned.

Complete the table.

<table>
<thead>
<tr>
<th>t</th>
<th>0</th>
<th>1</th>
<th>11</th>
<th></th>
</tr>
</thead>
</table>

(b) Show the values that will be present on g, g1, g2 when execution reaches the POINT X comment in the module above. For partial credit also show intermediate values for other signals used to compute the g's. (Look at next part before solving this one.)

At POINT X g=__________, g1=__________, g2=__________.

(c) Recall that the event queue used for Verilog simulation has active, inactive, and NBA regions, among others. Just before B1 starts execution in module timing above the active region might contain B1, B5, and B6 (see the comments on the right). (What the other regions contain is part of this problem.) Show the contents of the three regions when B5 starts. Assume that events in a region are scheduled in order.

When B5 starts: Active = {__________}. Inactive = {__________}. NBA = {__________}.
Problem 5: Answer each question below.

(a) [5 pts] Module add3 is supposed to compute the sum of its three inputs using instances of our_adder, but it won’t work. Fix the problem. The fixed module should still use our_adder.

Fix add3.

```
module add3(output [15:0] sum, input [15:0] a,b,c);

    our_adder a1( sum , a , b );
    our_adder a2( sum , sum , c );
endmodule
```

(b) [8 pts] The output of the module below is like the input except the bit positions are reversed (after enough clock cycles). Re-write the module so that it synthesizes to combinational logic (the clk input will no longer be needed). Add a parameter to indicate the input and output bit width.

```
module bitrev(output logic [7:0] x, input [7:0] a, input clk);
    logic [2:0] pos;
    initial pos = 0;
    always @( posedge clk ) begin
        x[pos] = a[7-pos];
        pos++;
    end
endmodule
```

Re-write so that it is combinational.

Include a parameter wid to specify the size.
Problem 6: Answer each question below.

(a) [5 pts] A Verilog module computes a result in one clock cycle. In our design we need that result in 3 ns, which can easily be achieved. The right way to achieve that in Cadence Encounter is to use the `define_clock` command to set the target clock period to 3 ns. Suppose instead we used `define_clock` to set the period to 1 ps, an impossible goal. Note: The original exam did not have the “can easily be achieved” phrase.

Would the synthesized design meet our 3 ns performance goal?

Considering typical design goals, what would be the disadvantage of setting the period to 1 ps for our design even though we needed 3 ns?

(b) [10 pts] In the module below, `translate` directives are used to prevent the synthesis program from reading the line with `initial`.

```verilog
module mult_seq( output logic [31:0] prod, input logic [15:0] plier, cand, input clk);
    logic [3:0] pos; logic [31:0] accum;
    // cadence translate_off <-- Translate synthesizer directive.
    initial pos = 0;
    // cadence translate_on <-- Translate synthesizer directive.
    always @( posedge clk ) begin
        if ( pos == 0 ) begin prod = accum; accum = 0; end
        if ( cand[pos] == 1 ) accum += plier << pos;
        pos++;
    end
endmodule
```

Why shouldn’t the synthesis program see the line with `initial`?

What would happen if the synthesis program saw the `initial` line?

What would happen if the simulation program didn’t see the line with `initial`?
(c) [7 pts] All four variables below have a size of 32 bits, but there are differences between them.

logic [31:0] a;
logic b [31:0];
logic [0:31] c;
int e;

□ Difference between a and b?

□ Difference between a and c?

□ Difference between a and e?