Name _____ *Formatted For 2-Sided Printing*

Staple Here

Digital Design using HDLs
LSU EE 4755
Final Examination

Friday, 13 December 2019 10:00-12:00 CST

Problem 1	_____	(30 pts)
Problem 2	_____	(25 pts)
Problem 3	_____	(20 pts)
Problem 4	_____	(25 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Staple Here

Problem 1: [30 pts] Appearing below is the solution to Homework 6, the accumulation module. The next page shows the pipelined adder and `st_occ`, which is some of the inferred hardware. Show the rest of the inferred hardware after some optimization. Leave the pipelined adder as a box.

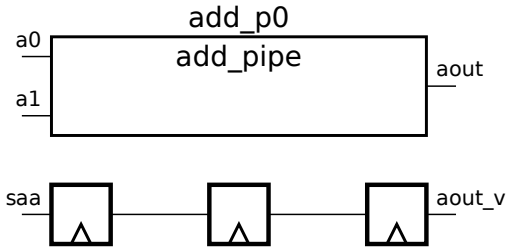
```
module add_accum #( int w = 21, n_stages = 3 )
   ( output logic [w-1:0] sum,   output logic sum_valid,
     input uwire [w-1:0] ai,     input uwire ai_v, reset, clk );

   logic [n_stages-1:0] st_occ;
   assign sum_valid = !st_occ;
   uwire aout_v = st_occ[n_stages-1];

   uwire [w-1:0] aout;
   uwire [w-1:0] a0 = ai_v   ? ai   : sum;
   uwire [w-1:0] a1 = aout_v ? aout : sum;

   add_pipe #(w,n_stages) add_p0( aout, a0, a1, clk );

   logic sum_occupied;
   uwire [1:0] n_values = ai_v + sum_occupied + aout_v;
   uwire saa = n_values >= 2;  // Start an addition.
   uwire write_sum = !sum_occupied && n_values == 1;

   always_ff @( posedge clk ) if ( reset ) begin
      sum <= 0;
      sum_occupied <= 0;
      st_occ <= 0;
   end else begin
      if ( write_sum ) sum <= aout_v ? aout : ai;
      sum_occupied <= n_values[0];
      st_occ <= { st_occ[n_stages-1:0], saa };
   end
endmodule
```

☐ Show inferred hardware after some optimization, but ☐ leave `add_pipe` as a box.

☐ Show logic associated with `n_values` as basic gates and a single BFA, **do not show adders and do not show comparison units**.

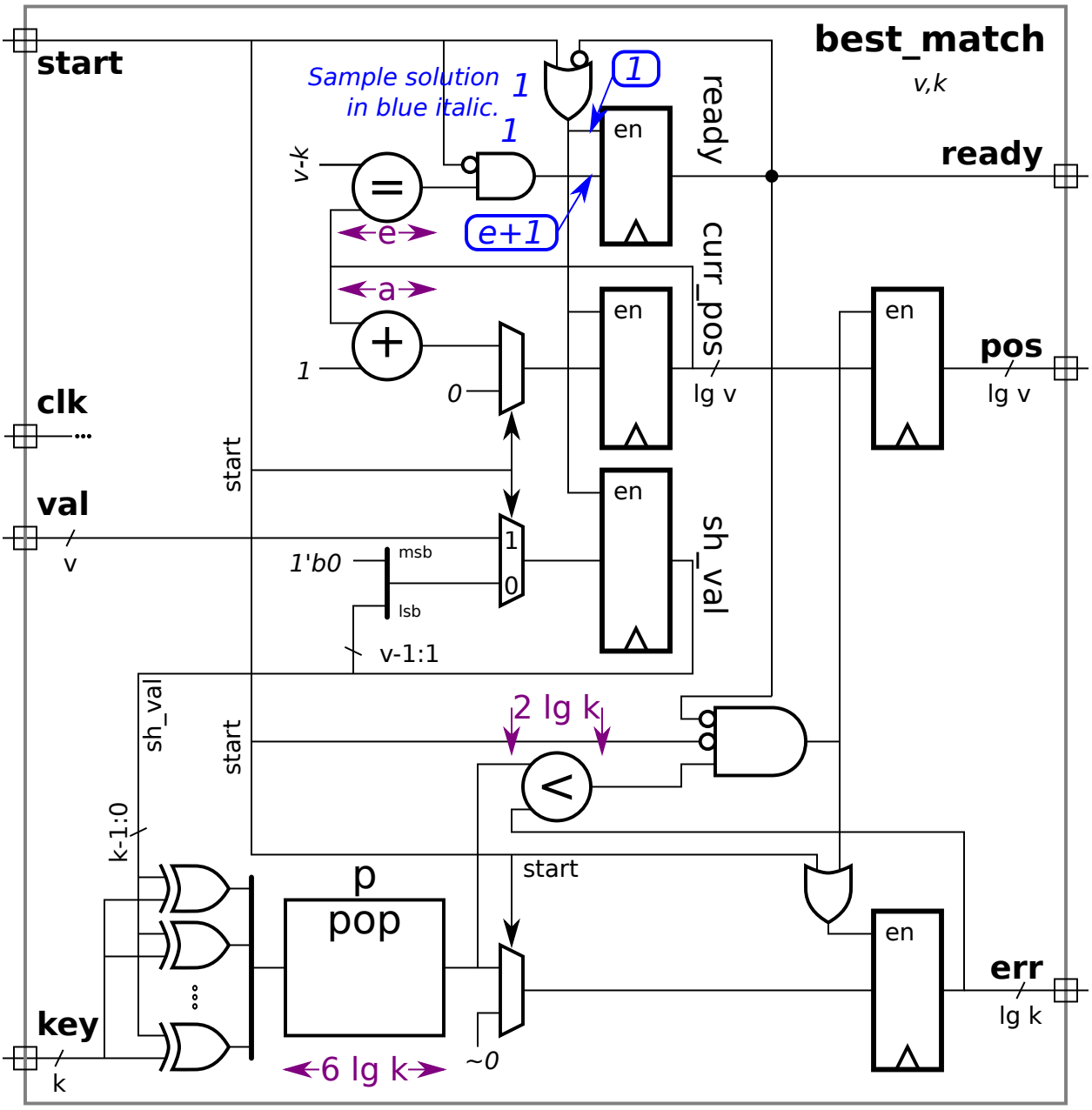
☐ Clearly show all input and output ports, do not confuse parameters with ports.

☐ Avoid effortlessly optimized hardware, such as gates with constant inputs.



Problem 2: [25 pts] Appearing below is hardware from the solution to Homework 5, Problem 2. The parameter names have been shortened, such as changing `wv` to `v` and using `lg v` for `wvb`. The diagram shows the delay through some of the modules, including the `pop` module. Treat `e` and `a` (delays for [=] and [+]) as given constants for the first part.

(*a*) Based on the provided delays and using the simple model for others, compute the arrival time (delay) of signals at each register input. That's two inputs for each of five registers. The solution for `ready` is shown in blue, so only four registers remain. Also, highlight the/a critical path to the `err` register.



☐ Show the arrival time of the enable and data signal at each register input and ☐ Highlight a critical path to `err` with a squiggly line.

☐ Take into account constant inputs when computing delays.

Staple Here

Staple Here

Staple Here

(*b*) The equality module is shown with a delay of `e`. Show the hardware for that module and compute the cost and delay using the simple model. Take into account the width of the inputs and the fact that one input is a constant.

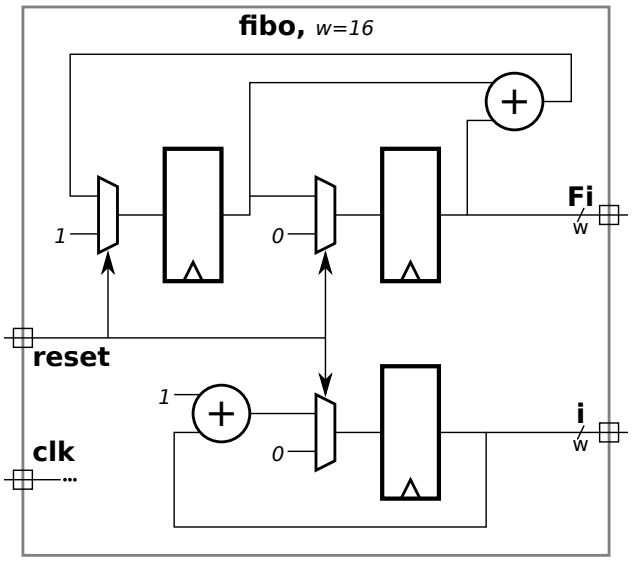
☐ Sketch hardware for equality module for $\lg v = 8$ and $v - k = 1011\,0001_2$, and ☐ taking into account the constant input.

☐ Show the cost of the hardware for the equality module above based on the simple model in terms of $\lg v$.
☐ Don't forget to take the constant input into account.

☐ Show the delay of the hardware based on the simple model in terms of $\lg v$. ☐ Don't forget to take the constant input into account.

Staple Here

Problem 3: [20 pts] The hardware illustrated to the right emits a famous integer sequence. Write a synthesizable Verilog description of the hardware.



☐ Complete the module, ☐ be sure that it is synthesizable.

☐ Use non-blocking assignments carefully.

☐ Be sure to include all ☐ input and output ports and ☐ parameters.

☐ Make sure that all objects have the appropriate widths.

```
module fibo
```

```
endmodule
```

Staple Here

Problem 4: [25 pts] Answer each question below.

(*a*) Appearing below are synthesis script results for the pipelined integer adder from Homework 6. That adder computes a w-bit integer sum using an n-stage pipeline in which each stage computes $\lceil w/n \rceil$ bits of the sum, starting with the $\lceil w/n \rceil$ least-significant bits in the first stage.

All syntheses are of a $w = 24$-bit adder, versions with $n = 1$, 2, 3, 4, and 6 stages are synthesized. The delay target is set to an easy 90 ns.

Module Name	Area	Delay Actual	Delay Target
add_pipe_w24_n_stages1	29928	10.174	90.000 ns
add_pipe_w24_n_stages2	47043	5.428	90.000 ns
add_pipe_w24_n_stages3	64159	3.701	90.000 ns
add_pipe_w24_n_stages4	81275	2.837	90.000 ns
add_pipe_w24_n_stages6	115506	1.973	90.000 ns

☐ Based on this data provide the ☐ latency and ☐ throughput for the three-stage adder. Be sure to ☐ use appropriate units for the throughput.

☐ Note that the area (cost) increases with the number of stages. Based on the description above what is the main contributor to the increase in cost?

Staple Here

(*b*) The two modules below appear to be similar.

```
module plan_I(output logic [7:0] e, input logic [7:0] a,b);
   logic [7:0] c;
   always_comb begin
      c = a + b;
      e = c + a;
   end
endmodule

module plan_II(output logic [7:0] e, input logic [7:0] a,b);
   logic [7:0] c;
   always_comb e = c + a;
   always_comb c = a + b;
endmodule
```

☐ For which module will the simulator perform unnecessary addition? ☐ Explain.

☐ Is the result computed by the two modules different or the same? ☐ Explain.

(*c*) What value will y have at the end of the initial block?

```
module s;
   logic [15:0] a,b,y;
   initial begin
      a = 1;
      b = 100;
      b <= 10;
      y = 0;
      y <= a + b;
      y = 999;
      #1;
      a = 2;
      b <= 20;
      #200;
      // Show value of y at this point in execution.
   end
endmodule
```

☐ Value of y at end of block is:

Staple Here

(*d*) Consider the declarations below.

```
module types;
   int en;
   logic [31:0] lo;
   bit [31:0] b;
   uwire [31:0] u = 33;
   localparam int p = 22;
endmodule
```

☐ Object `u` has the same data type as one of the other objects. Which is it?

☐ What is the difference between `lo` and `b` (`logic` and `bit`)?

☐ Notice that `u` is assigned a value. What is it about object `lo` that makes it illegal to assign a value in its declaration?

☐ Add correct code to assign value 44 to `lo`.

Staple Here