LSU EE 4755 **Homework 3** **Due: 23 October 2026**

Solution *Formatted 11:13, 28 October 2026*

Student Expectations

To solve this assignment students are expected to avail themselves of references provided in class and on the Web site, such as for Verilog programming and synthesis examples, and to seek out any additional help and resources that might be needed. (Of course this doesn't mean asking someone else to solve it for you.) It is each student's duty to himself or herself to resolve frustrations and roadblocks quickly. (If you get stuck *just ask for help!*)

This assignment cannot be solved by blindly pasting together parts of past assignments. Solving the assignment is a multi-step learning process that takes effort, but one that also provides the satisfaction of progress and of developing skills and understanding.

Collaboration Rules

Each student is expected to complete his or her own assignment. It is okay to work with other students and to ask questions in order to get ideas on how to solve the problems or how to overcome some obstacle (be it a question of Verilog syntax, interpreting error messages, how a part of the problem might be solved, etc.) It is also acceptable to seek out digital design resources for help on Verilog, digital design, etc. It is okay to make use of AI LLM tools such as Claude and Copilot to generate sample Verilog code. (Do not assume LLM output is correct. Treat LLM output the same way one might treat legal advice given by a lawyer character in a movie: it may sound impressive, but it can range from sage advice to utter nonsense.)

After availing oneself to these resources **each student is expected to be able to complete the assignment alone**. Test questions will be based on homework questions and **the assumed time needed to complete the question will be for a student who had solved the homework assignment on which it was based**.

Resources

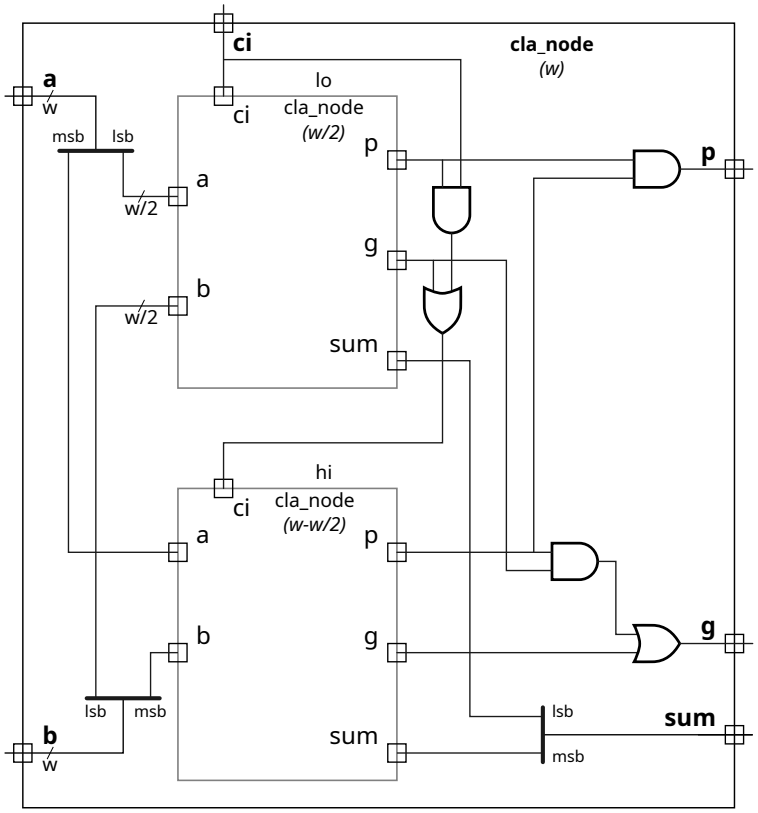
In addition to the lecture slides and demo code, look at prior semesters' homework and exam questions to see examples of finding inferred hardware and of using the simple model to compute cost, arrival times, and critical path.

Problems start on next page.

Problem 1: Below is the Verilog description of a CLA used in class, followed by the inferred hardware for the $w > 1$ case. The SVG source for the diagram is at `https://www.ece.lsu.edu/koppel/v/2026/ill-cla.svg`, and a collection of gates is at `https://www.ece.lsu.edu/koppel/v/2026/c.svg`, they can be edited using Inkscape or any other SVG editor, or for that matter a text editor since SVG is plain text.

```
module cla_node #( int w = 16 )
   ( output uwire [w-1:0] sum,    output uwire p, g,
     input uwire cin,             input uwire [w-1:0] a, b );

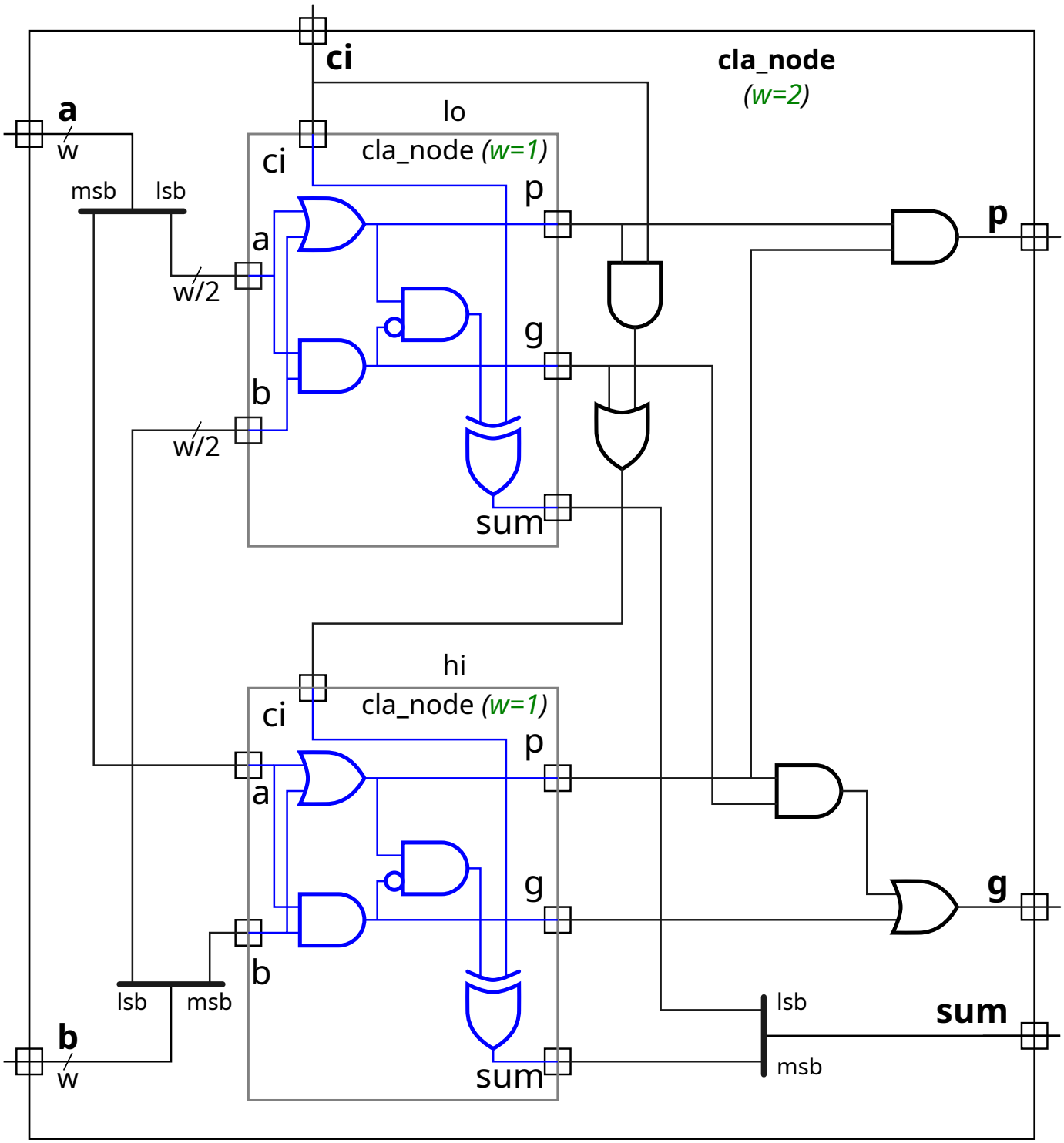
   if ( w == 1 ) begin
      assign g = a && b;
      assign p = a || b;
      assign sum = ( !g && p ) ^ cin;  // Same as a ^ b ^ cin.
   end else begin
      localparam int wlo = w/2;
      localparam int whi = w - wlo;
      uwire plo, glo, phi, ghi;
      cla_node #(wlo) lo(sum[wlo-1:0], plo,glo, cin, a[wlo-1:0],b[wlo-1:0] );
      uwire chi = cin && plo || glo;
      cla_node #(whi) hi(sum[w-1:wlo], phi,ghi, chi, a[w-1:wlo],b[w-1:wlo] );
      assign p = plo && phi;
      assign g = glo && phi || ghi;
   end
endmodule
```



(*a*) Complete the diagram below for a `w=2` instantiation by drawing the inferred hardware for the `w=1` cases inside the hi and lo instantiation boxes.

☑ Show hardware inferred for the `hi` and `lo` instantiations.

Solution appears below in blue.



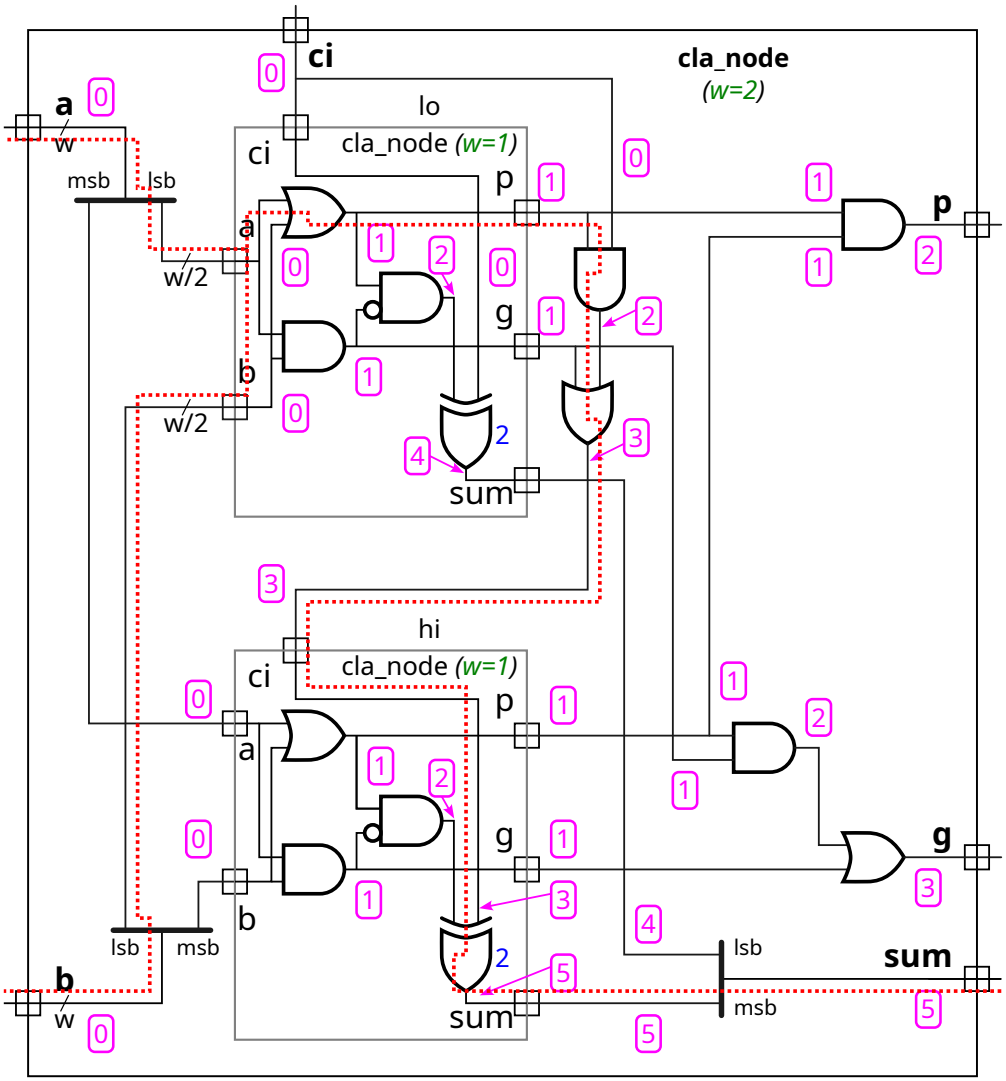
(*b*) On your solution to the previous part show the arrival time and critical path for the $w = 2$ module, include the time needed by the hardware in the `lo` and `hi` recursive instantiations. The diagram shows three signals labeled `ci`. In Verilog notation they are `cla_node.ci`, `cla_node.lo.ci`, and `cla_node.hi.ci`. For the arrival times assume that `a`, `b`, and `cla_node.ci` arrive at $t = 0$ and that `cla_node.ci` is a non-constant input.

☑ Show the arrival times including hardware in `lo` and `hi`.

☑ Show the critical path. ▷ Assume outputs `cla_node.p`, `cla_node.g`, and `cla_node.sum` are all used.

☑ Don't confuse `cla_node.ci` with `cla_node.lo.ci` and especially with `cla_node.hi.ci`.

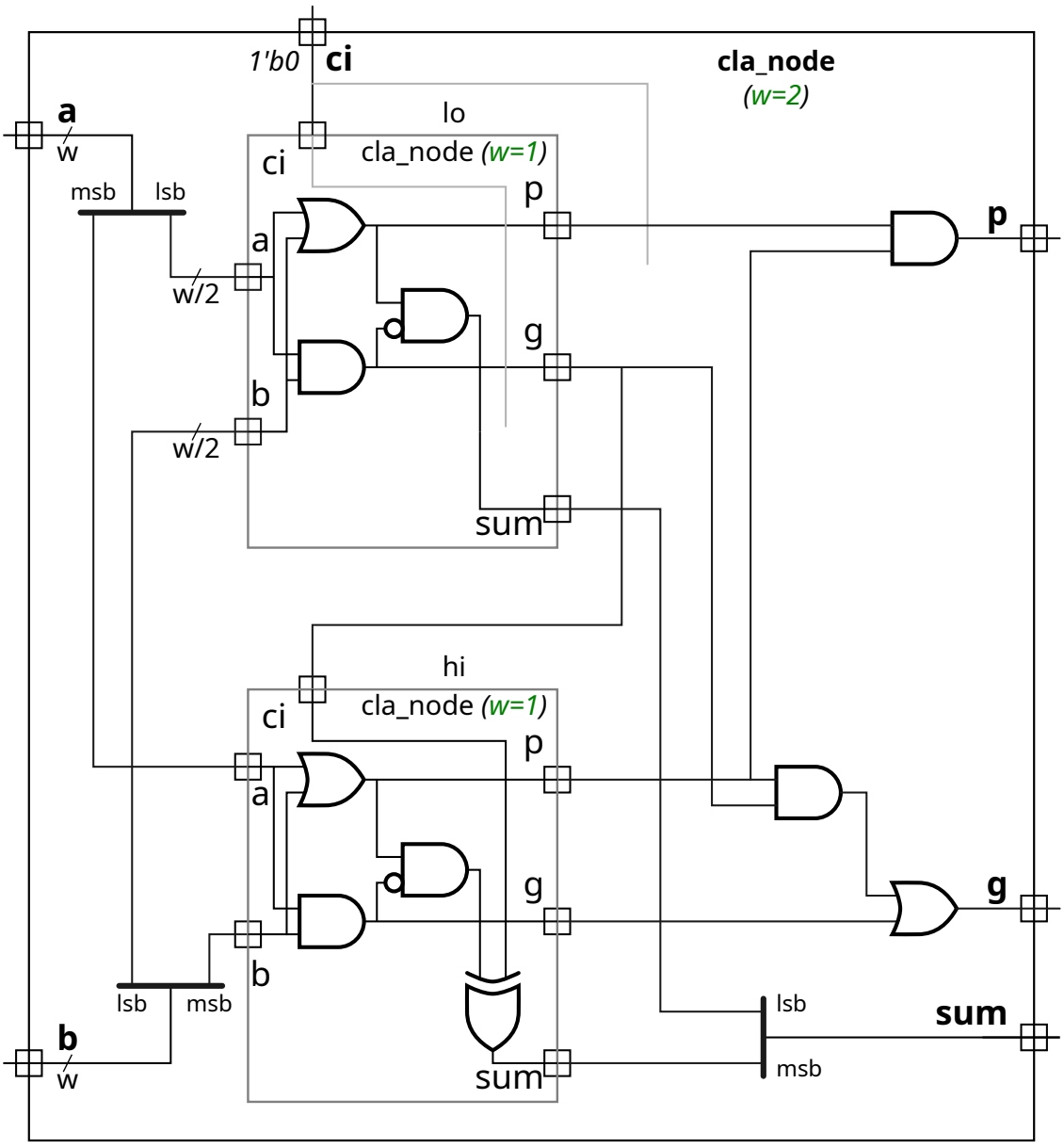
Solution appears below, with the critical paths in red. The critical path is found by starting at the output with the highest arrival time, `sum` here, and then following the path back to the module inputs, at each item with multiple inputs (such as gates) the path proceeds through an input with the highest arrival time. (It would be correct to show the critical path starting at either `a` or `b`, here both are shown.) Notice that even at $w = 2$ the propagate (`p`) and generate (`g`) signals have lower arrival times than the sum and carry signals.



(c) Consider the common case where the carry-in is zero.

☑ Optimize the hardware for `cla_node (w=2)` (and the instantiations) for the case where `cla_node.ci=0`. To save time you can just redraw the parts that have changed.

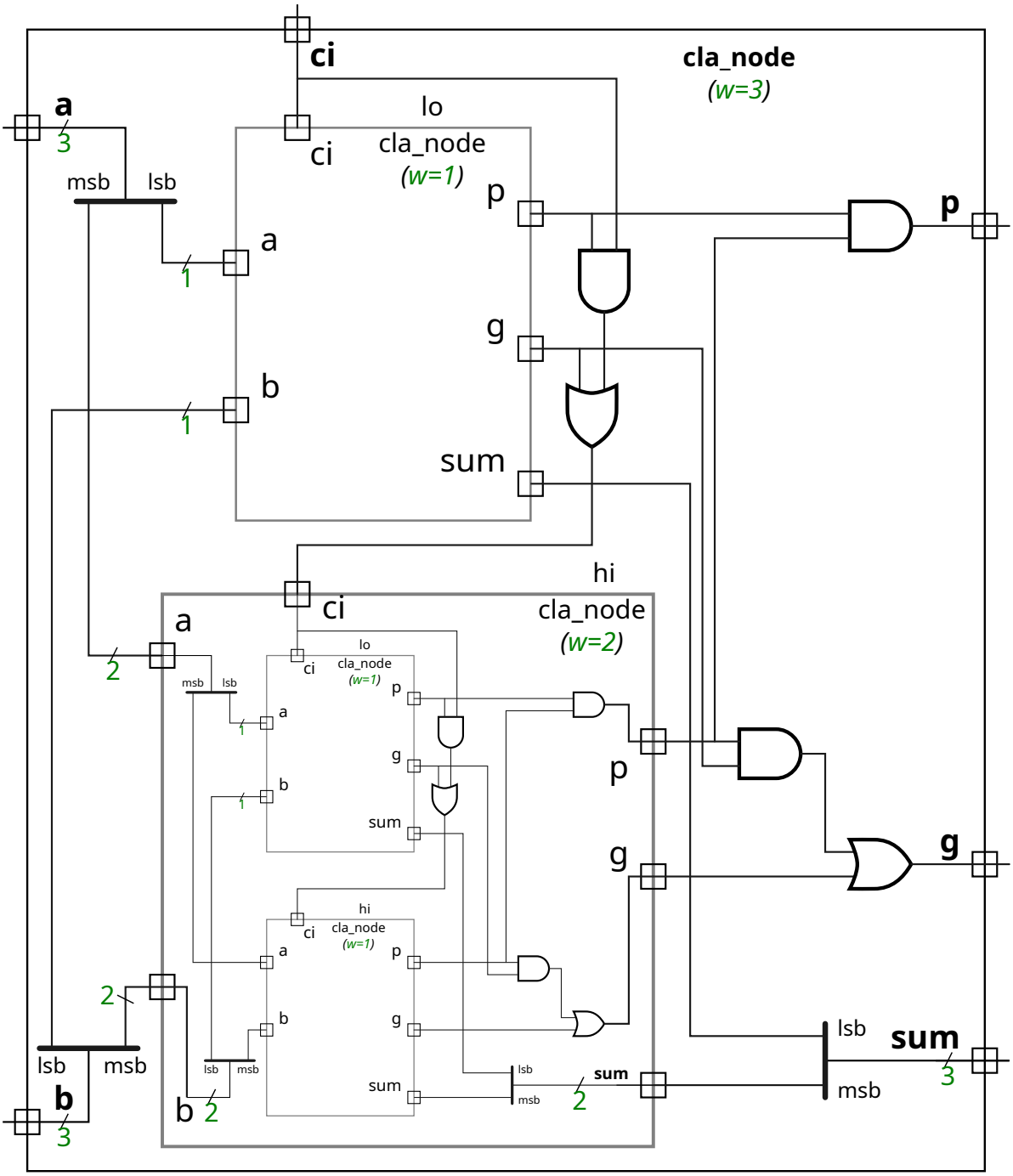
Solution appears below. With the carry-in set to the constant 0 three gates are eliminated, including an XOR, for a cost savings of $5\,\mathrm{u_c}$. An interesting question is what the cost savings would be for some $w > 2$.



(*d*) Show the inferred hardware for $w = 3$.

☑ Show inferred hardware for `w=3`. Only show the hardware in the recursive instantiations where $w > 1$.

Solution appears below.



Problem 2: Appearing below is the solution to Homework 2 Problem 2, the left-justify module.

(*a*) Show the inferred hardware for `w=8` as requested below.

☑ Show the hardware that will be inferred for this module for `w=8`.

☑ Don't forget to show module input and output ports.

☑ Do not confuse elaboration-time computation with hardware performing computation.

☑ Do not show shift modules or operators. Show how the shift is achieved by renumbering bits and using multiplexors.

☑ Do not show equality units, such as `==`, instead use basic gates.

☑ Show the value of `i` for each stage.

☑ Don't be confused by loop iteration order.

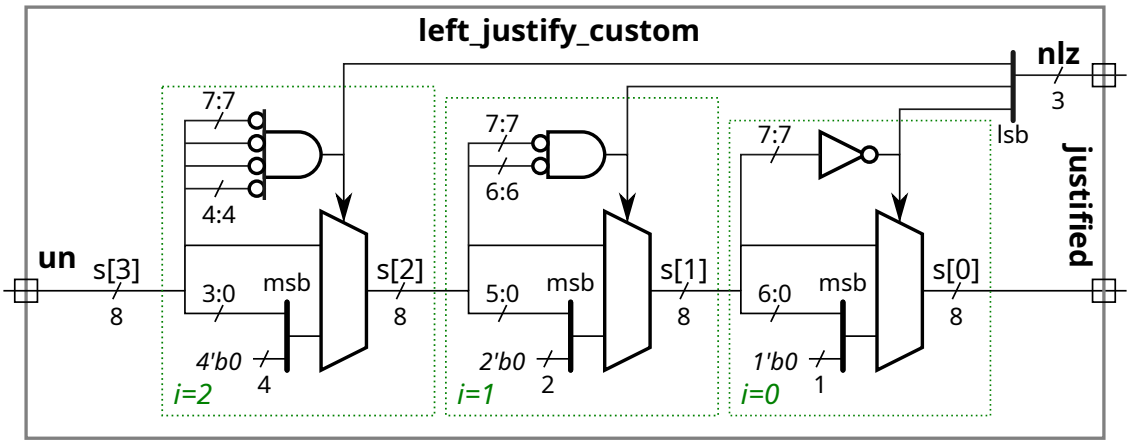
```
module left_justify_custom #( int w = 16, lgw = $clog2(w) )
   ( output uwire [w-1:0] justified, output uwire [lgw-1:0] nlz,
     input uwire [w-1:0] un );

   uwire [w-1:0] s[lgw:-1];  // Array of wires to interconnect mux2 instances.
   assign s[lgw] = un;
   assign justified = s[0];

   for ( genvar i=0; i<lgw; i++ ) begin
      localparam logic [lgw-1:0] these_width = 1 << i;
      uwire these_zero = s[i+1][w-1:w-these_width] == these_width'(1'b0);
      assign nlz[i] = these_zero;
      mux2 #(w) st( s[i],  these_zero,  s[i+1],  s[i+1] << these_width  );
   end

endmodule
```

Solution appears below.



(*b*) Compute the cost of the module. In your cost computation the identity $\sum_{i=0}^{a-1} 2^i = 2^a - 1$ may come in handy.

☑ Compute the simple-model cost for `left_justify_custom` in terms of w, assuming w is a power of 2.

☑ Do not assume a particular value for w.

Ignoring constant inputs, each multiplexor has cost $3w\,\mathrm{u_c}$ and so the total cost for the $\lg w$ iterations is $3w \lg w\,\mathrm{u_c}$.

The cost of the AND gate in iteration i is $[2^i - 1]\,\mathrm{u_c}$, the total cost is $\sum_{i=0}^{\lg(w)-1} 2^i - 1 = 2^{\lg w} - 1 - \lg w = [w - 1 - \lg w]\,\mathrm{u_c}$. So ignoring constant inputs the total cost is $[3w \lg(w) + w - \lg(w) - 1]\,\mathrm{u_c}$.

The number of constant inputs to the multiplexor in iteration i is 2^i bits, so at iteration i we save $2 \times 2^i\,\mathrm{u_c}$ due to constant inputs. (Savings is relative to the "full price" of $3w\,\mathrm{u_c}$ for the mux.) The total savings is $2 \sum_{i=0}^{\lg w} 2^i = 2(2^{\lg w} - 1) = 2(w-1)\,\mathrm{u_c}$. So now the total price is $3w \lg(w) + w - \lg(w) - 1 - 2(w-1) = [3w \lg(w) - w - \lg(w) + 1]\,\mathrm{u_c}$.