LSU EE 4755 **Homework 3** **Due: 24 October 2025**

Formatted 16:22, 20 October 2025

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 the students' responsibility 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 ChatGPT 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**.

Problems start on next page.

Problem 1: Appearing on the next page is the solution to Homework 2 Problem 2.

(*a*) Show the inferred hardware as described below.

- ☐ Show hardware that will be inferred for an arbitrary $n > 3$.
- ☐ Show the recursive instantiations as boxes (don't show what's inside, other than a label).
- ☐ Be sure to distinguish between synthesized hardware and elaboration time computation!
- ☐ Show each trend object (`trend`, `trlo`, `trhi`) as two wires rather than a two-bit wire.
- ☐ Use a $\boxed{+}$ box for the adder and a $\boxed{\neq}$ box for the inequality `sidxlo != nlo`.
- ☐ Use basic gates (AND, OR, NAND, NOR, XOR, XNOR, NOT) and multiplexors for other logic, especially the `else if` condition.

See 2024 Homework 4 for an example of showing inferred hardware for a recursively described module.

(*b*) Compute the cost and critical path of the hardware *at one level* in terms of n and w, assuming $n > 3$. For cost ignore the cost of the recursive instantiations themselves. Assume (unrealistically) that the arrival time of the outputs of the recursive instantiations, `sidxlo`, `eilo`, `trlo`, `sidxhi`, `eihi`, and `trhi` all arrive at $t = 0$.

- ☐ In terms of n and w compute the cost using the simple model ☐ accounting for constants.
- ☐ In terms of n and w compute the delay for each output, `sidx`, `eidx`, and `trend` ☐ assuming that the outputs of the recursive instantiations all arrive at $t = 0$ and ☐ accounting for constants.
- ☐ Don't forget to distinguish between synthesized hardware and elaboration-time computation.

See the 2024 midterm exam for a similar problem.

```
module is_sorted_to_tree
  #( int n = 12, int w = 32, int wi = $clog2(n), bit pot = 1 )
   ( output logic [wi-1:0] sidx, eidx,
     output logic [1:0] trend,
     input uwire [w-1:0] a[n-1:0] );

   if ( n == 1 ) begin   // Base Case: One element.

      assign sidx = 0;   // "Entire" sequence sorted.
      assign eidx = 0;   // "Entire" sequence equal.
      assign trend = 0;  // All elements (just one) equal.

   end else if ( n == 2 ) begin

      // Base Case: Two elements. Sorted (either ascending or descending).
      assign sidx = 1;                // Entire sequence sorted.
      assign eidx = trend ? 0 : 1;
      ist_compare #(w) lti( trend, a[0], a[1] );

   end else begin

      localparam int nlg = $clog2(n-1) - 1;
      localparam int nlo = pot ? 1 << nlg : n / 2;
      localparam int nhi = n - nlo;
      localparam int wilo = $clog2(nlo+1), wihi = $clog2(nhi);

      uwire [wilo-1:0] sidxlo, eilo;
      uwire [wihi-1:0] sidxhi, eihi;
      uwire [1:0] trlo, trhi;

      is_sorted_to_tree #(nlo+1,w,wilo,pot) islo(sidxlo, eilo, trlo, a[nlo:0] );
      is_sorted_to_tree #(nhi,w,wihi,pot) ishi(sidxhi, eihi, trhi, a[n-1:nlo] );

      always_comb begin
         if ( sidxlo != nlo ) begin
            eidx = eilo;
            sidx = sidxlo;
            trend = trlo;
         end else if ( ( trlo | trhi ) == 2'b11 ) begin
            eidx = eilo;
            sidx = nlo + eihi;
            trend = trlo;
         end else begin
            eidx = trlo ? eilo : nlo + eihi;
            sidx = nlo + sidxhi;
            trend = trlo | trhi;
         end
      end
   end
endmodule
```

Problem 2: Appearing below is a solution to Problem 1.

```
module is_sorted_to_iter #( int n = 12, int w = 32, int wi = $clog2(n) )
   ( output logic [wi-1:0] sidx, eidx, output logic [1:0] trend,
     input uwire [w-1:0] a[n-1:0] );

   // Instantiate ist_compare.
   uwire [1:0] cmpa[n];
   for ( genvar i=1; i<n; i++ )
     ist_compare #(w) lti( cmpa[i], a[i-1], a[i] );

   always_comb begin

      eidx = 0;
      sidx = n-1;
      trend = 0;
      for ( int i=1; i<n; i++ ) begin

         // Check whether there was a non-zero trend and if it is now broken.
         if ( trend && cmpa[i] && cmpa[i] != trend )
           begin
              sidx = i - 1;  // The length of the sorted prefix has been found, set sidx ..
              break;         // .. and exit the loop.
           end

         trend |= cmpa[i];
         if ( trend == 0 ) eidx = i;
      end
   end

endmodule
```

(*a*) Show inferred hardware without optimization as described below.

☐ Show hardware that will be inferred for the hardware above for `n=4` ☐ but only the hardware inferred for the first iteration of the `i` loop.

☐ Note that `trend` and `comp[1]` each have two bits. Show each as two wires and handle the bits individually.

☐ Don't optimize in this part.

(*b*) Show the hardware for the `i=1` iteration after optimizing, especially for constants, including initial values of `trend`, `eidx`, and `sidx`.

☐ Show the hardware for the `i=1` iterration after optimizing.

☐ Be sure to optimize for constants, including initial values of `trend`, `eidx`, and `sidx`.