LSU EE 4755 **Homework 2** **Due: 15 Oct 2025**

Formatted 14:54, 10 October 2025

For instructions visit `https://www.ece.lsu.edu/koppel/v/proc.html`. *For the complete Verilog for this assignment without visiting the lab follow* `https://www.ece.lsu.edu/koppel/v/2025/hw02.v.html`.

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 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**.

Problem 0: Following instructions at `https://www.ece.lsu.edu/koppel/v/proc.html`, set up your class account (if necessary), copy the assignment, and run the Verilog simulator on the unmodified homework file, `hw02.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Homework Overview

The modules in this assignment have an array input, `a`, and three outputs, `trend`, `sidx`, and `eidx`, such as `is_sorted_to_proc`:

```
module is_sorted_to_proc
  #( 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] );
```

The array input carries an `n`-element array of `w`-bit unsigned integers. The modules are to determine how much of the array is sorted, and whether the direction (trend) is increasing, decreasing, or if all elements are identical. Two-bit output `trend` indicates the direction of the sorted part: `2'b10` for increasing, `2'b01` for decreasing, and `2'b00` when all elements are equal. Output `sidx` indicates the array index of the last sorted element. That is, elements at index $0, 1, \ldots,$ `sidx` should be sorted in direction `trend` and the element at `sidx+1` should ruin the trend. If the entire array is

sorted `sidx=n-1`. The smallest possible value is `sidx=1`. The smallest value is 1 and not 0 because any two values will either form an increasing sequence, a decreasing sequence, or be equal. Output `eidx` indicates the index of the last repeated value (starting from index 0). The smallest value is `eidx=0` (the first and second elements differ) and the largest value is `eidx=n-1`. For examples of what `trend`, `sidx`, and `eidx` should be see the description of the testbench output.

There are three modules that compute these outputs, `is_sorted_proc`, `is_sorted_iter`, and `is_sorted_tree`. Module `is_sorted_proc` is complete and works. It is there for your reference. The other two modules are to be solved. In both cases the comparisons in those modules are to be made by instantiation(s) of `ist_compare`. Module `is_sorted_iter` is to be completed for Problem 1, using iterative generate statements. Module `is_sorted_proc` is to be completed for Problem 2, using recursive instantiations.

Remember that `is_sorted_to_proc` is correct, and it can be used to help with the other modules:

```
localparam logic [1:0] incr = 2, decr = 1;

module is_sorted_to_proc
  #( 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] );

   always_comb begin
      sidx = n-1;
      eidx = 0;
      trend = 0;
      for ( int i=1; i<n; i++ ) begin
         if ( a[i] > a[i-1] ) begin
            if ( trend == decr  ) begin sidx = i - 1; break; end
            trend = incr;
         end
         if ( a[i] < a[i-1] ) begin
            if ( trend == incr  ) begin sidx = i - 1; break; end
            trend = decr;
         end
         if ( trend == 0 ) eidx = i;
      end
   end
endmodule
```

The modules for both problems should use module `ist_compare` to perform the comparisons:

```
module ist_compare
  #( int w = 15 )
   ( output uwire [1:0] gl, input uwire [w-1:0] a, b );
   assign gl = { a < b, a > b };
endmodule
```

The goal of Problem 1 is to exercise understanding of the difference between Verilog structural descriptions and behavioral descriptions, and to work with generate statements. Its solution is straightforward. (If it's not straightforward to you please ask for help.) The goal of Problem 2 is to exercise skill in writing recursive descriptions of tree-structured hardware.

Testbench

To compile your code and run the testbench press F9 in an Emacs buffer in **a properly set up account**. The testbench will instantiate modules `is_sorted_proc`, `is_sorted_iter`, and `is_sorted_tree` for several different array sizes (`n`) and widths (`w`). Each module will be tested with 10,000 arrays (as of this writing). If there are errors the first few errors will be reported in detail. To help with your understanding two outputs of each trend for each module is shown in detail. The testbench finishes up by showing a tally of the total number of errors.

Appearing below is are excerpts from the testbench output for a correctly solved solution.

```
Compilation started at Tue Oct  7 13:11:36

xrun -sv -batch -exit hw02.v
TOOL: xrun(64) 25.03-s001: Started on Oct 07, 2025 at 13:11:37 CDT
[snip]

Starting tests for is_sorted_to_proc n=4, w=8.
Output below is correct, it shows trend 01 -- decreasing.
  0   1e  2   3s
 198 198 138 101
Output below is correct, it shows trend 10 -- increasing.
  0e  1s  2   3
  16  58  57 211
```

A sample output starts with the text `Output below`. That's followed by two lines, the first line shows the indices, numbered 0 to `n-1`, and second line shows the corresponding elements of `a`. The correct value of `eidx` is indicated by an index followed by the letter `e`, such as `1e` and `0e` in the two samples above. The correct value of output `sidx` is indicated by an index followed by the letter `s`, such as `3s` and `1s` in the samples above.

Some additional sample outputs are shown below:

```
Starting tests for is_sorted_to_tree n=10, w=9.
Output below is correct, it shows trend 10 -- increasing.
  0   1   2   3   4e  5   6   7s  8   9
 203 203 203 203 203 273 273 273 272 493
Output below is correct, it shows trend 01 -- decreasing.
  0   1e  2   3   4   5   6   7   8s  9
 483 483 472 472 303 282 282 199  38  39
Output below is correct, it shows trend 01 -- decreasing.
  0   1   2e  3   4   5   6   7   8s  9
 466 466 466 113  56  52  52  52  52  53
Output below is correct, it shows trend 10 -- increasing.
  0   1e  2   3   4   5s  6   7   8   9
  36  36 439 439 439 439 438 476 445  30
Output below is correct, it shows trend 00 -- equal.
  0   1   2   3   4   5   6   7   8   9es
 443 443 443 443 443 443 443 443 443 443
Output below is correct, it shows trend 00 -- equal.
  0   1   2   3   4   5   6   7   8   9es
 246 246 246 246 246 246 246 246 246 246
[snip]
```

The testbench finishes by showing a tally of errors:

```
Total is_sorted_to_proc n= 4: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_proc n= 5: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_proc n= 6: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_proc n= 7: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_proc n= 8: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_proc n= 9: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_proc n=10: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_iter n= 4: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_iter n= 5: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_iter n= 6: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_iter n= 7: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_iter n= 8: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_iter n= 9: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_iter n=10: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_tree n= 4: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_tree n= 5: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_tree n= 6: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_tree n= 7: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_tree n= 8: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_tree n= 9: Errors: 0 trend, 0 sidx, 0 eidx.
Total is_sorted_to_tree n=10: Errors: 0 trend, 0 sidx, 0 eidx.
Grand Total Errors:  0 trend, 0 sidx, 0 eidx.
```

The testbench excerpts above are for a correct solution. When there are errors the testbench will show the incorrect and correct value for each incorrect output, followed by a sample. For example, on an unmodified assignment the testbench (after startup spew) output will start with:

```
Starting tests for is_sorted_to_iter n=4, w=7.
Error trend xx != 01 (correct)
Error sidx x != 3 (correct)
Error eidx x != 1 (correct)
  0   1e  2   3s
  99  99  69  50
Error trend xx != 10 (correct)
Error sidx x != 1 (correct)
Error eidx x != 0 (correct)
  0e  1s  2   3
   8  29  28 105
[snip]
```

In the unmodified assignment none of the outputs of `is_sorted_to_iter` are connected and so the output values are `x`, that's shown above. Appearing below is sample output of a partially solved but still incorrect `is_sorted_to_tree`. The output shows correct and incorrect outputs:

```
Starting tests for is_sorted_to_tree n=4, w=7.
Output below is correct, it shows trend 01 -- decreasing.
  0   1e  2   3s
  99  99  69  50
Error sidx 0 != 1 (correct)
  0e  1s  2   3
   8  29  28 105
Output below is correct, it shows trend 10 -- increasing.
```

```
  0   1e  2   3s
  49  49  70  70
Output below is correct, it shows trend 01 -- decreasing.
  0   1e  2   3s
 109 109  75  14
Error sidx 0 != 1 (correct)
  0e  1s  2   3
  23   3   4 118
[snip]
Did 10000 tests of tree n=4, w=7. Errors 0 trend, 6740 sidx, 0 eidx.
[snip]
```

To help you debug it is okay to modify the testbench so that it applies tests that help you discover flaws. The best thing to do is to add additional tests, or move a test so that it is first. If you eliminate tests, be sure to put them back. Note that the TA-bot will use a fresh testbench, not the version in your file.

References and Helpful Examples

The modules in this assignment must be recursively defined, so that they describe a tree-like structure. See the clz module from 2019 Homework 2. The assignment and its solution are part of the 2025 assignment directory, look for the file names starting `2019`.

A more complex recursive module was the subject of 2024 Homework 3, in which the goal was to count and manipulate bracketing characters.

Problem 1: Complete module `is_sorted_to_iter` so that it categorizes its input array as described in the overview above, and does so using instantiations of module `ist_compare` for array element comparisons. The module should make use of a generate loop to instantiate `ist_compare`, but it can use procedural code to compute `trend`, `sidx`, and `eidx` in terms of `ist_compare` outputs. The module must pass the testbench and be synthesizable. Synthesis can be run with the command `genus -files syn.tcl`.

A good place to start is by looking at module `is_sorted_to_proc`. It computes the correct outputs, but it does not use `ist_compare`.

See the check boxes in the assignment code for additional requirements and tips.

Problem 2: Complete module `is_sorted_to_tree` so that it categorizes its input array as described in the overview above, the module must be recursive and describe tree-shaped hardware, it must use instantiations of module `ist_compare` for array element comparisons, and so that recursive instantiation sizes simplify arithmetic expressions like `nlo + valhi` (see below). The module must pass the testbench and be synthesizable.

Completing this module is more difficult than other modules describing tree-shaped hardware such as `simple_tree` and `min_t` from the lectures. Make sure you understand those before attempting this problem. One difficulty is that there are three outputs to combine, `trend`, `sidx`, `eidx`, not just one as in the `min_t` and `clz` modules. Another difficulty is how to compare the last element in the "lo" module and the first element in the "hi" module. Be creative. Note that the place where `ist_compare` is instantiated depends upon the way in which the problem is solved.

Choose recursive instance sizes to optimize addition of constants to output values. For example, if elaboration-time constant `nlo` is related to the size of one of the recursive instances and `valhi` is some output of the other instance, and you need to compute `nlo + valhi`, choose `nlo` so that it is a power of 2. Unlike 2019 Homework 2, there is no need to rewrite expressions like `nlo + valhi` as `{ 2'b1 + valhi[um], valhi[um-1:0] }`.

In a correct solution the time to compute the outputs will be closer to $O(\log n)$ then $O(n)$. (The time for `is_sorted_to_iter` is closer to $O(n)$.)

The synthesis script just tries out two sizes, so it's difficult to see the trend.