LSU EE 4755 **Homework 4** **Due: 18 Nov 2025**

Formatted 18:24, 11 November 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/hw04.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 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 answer questions and 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 somehow necessary at this point), copy the assignment, and run the Verilog simulator on the unmodified homework file, `hw04.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Homework Overview

The module in this assignment is something like a sequential version of the sorted-sequence trend from Homework 2, except the goal here is to detect when the sequence switches from ascending to descending and vice versa.

The input to module `trend2` consists of a sequence of *samples*. Samples are totally ordered and module `ist_compare` determines their ordering. In this discussion the samples are shown as unsigned integers, though your modules should never treat them as such (instead just have `ist_compare` determine ordering).

The *index* of a sample is the position within the sequence. A sequence starts in the cycle the `reset` input is asserted, and the sample has index zero. The sample arriving in the next cycle has index one, and so on.

The samples will at one point be ascending (not necessarily incrementing by one each time), will switch to descending, and switch again. The module is to find *ascending segments* and *descending segments* and report their start index and length.

An ascending segment consists of a sequence of increasing or equal (non-decreasing) samples, such as $1, 3, 7, 12, 12, 20$. Every sample is in exactly one segment, ascending or descending. The first sample of a descending segment is strictly larger than the last sample of the ascending segment that precedes it. The first sample of an ascending segment is strictly smaller than the last sample of the descending segment that precedes it. Put another way, the sample at the start index (beginning) of an ascending segment is always smaller than both its neighbors (preceding and succeeding), and the sample at the start index (beginning) of an descending segment is always larger than its neighbors.

Consider the sequence $7_{100}, 3_{101}, 3_{102}, 5_{103}, 8_{104}, 10_{105}, 10_{106}, 9_{107}, 6_{108}, 11_{109}, \ldots$, where the subscripts indicate the index of the sample. For example, sample 3 is at index 101. The sub-sequence $3, 3, 5, 8$ is an ascending segment, and $10, 10, 9$ is a descending segment. Notice that the 10s are at the beginning of the descending segment, they **are not** at the end of the ascending segment. The ascending segment has start index 101 and length four. The descending segment has start index 105 and length 3.

Module `trend2` has two parameters `wd` and `wi`. Parameter `wd` indicates the number of bits in a sample and `wi` indicates the number of bits in an index or in a length. There are three inputs, `wd`-bit input `samp`, and one bit inputs `reset` and `clk`. At each cycle `samp` holds a new sample. If `reset` is 1, the sample arriving at that cycle is at index 0.

The module has five outputs, and each must be the outputs of registers. Their values at clock cycle t should be based on the state of the inputs and history at clock cycle $t-1$. At cycle t output `last_2_trend` is the ordering (output of `ist_compare`) of the samples that arrived at cycles $t-2$ and $t-1$, or `tr_equal` if `reset` was `1` at cycle $t-1$.

Outputs `prev_trend` and `prev_length` are the trend and length of the most recent completed segment, or `0` if no segment had been completed since the earliest reset. As with `last_2_trend`, `prev_trend` is based on the output of `ist_compare`, and follow the same rules as the comparisons used in Homework 2.

Output `curr_idx_start` is the start index of the most recently detected segment, and `curr_trend` is its trend. After a `reset` output `curr_idx_start` should be set to `0` and `curr_trend` should be set to `tr_equal`.

Note that it is only possible to detect the start of a segment at least one cycle **after** the first sample of the segment arrived. For example, a descending segment can only be detected when the current sample is smaller than the previous sample, but the previous sample and perhaps equal-valued samples before it are part of the new descending segment. See the description of testbench output, below, for examples of sequences and the outputs that are expected.

Testbench

To compile your code and run the testbench press F9 in an Emacs buffer in a properly set up account. The testbench will apply inputs to module `trend2` and report on the results. Unlike other assignments, only one module instantiation will be tested.

The testbench will show a trace for at least the first 10 cycles, with additional trace output shown before errors. (To change the number of trace lines edit `trace_lines_n_wanted`, located near the top of the testbench.) The trace is intended to both help you debug your module and also to understand what outputs are expected. The trace lines from a correctly solved assignment appear below, though more than 10 trace lines are shown.

```
     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr     3 R    0   662  00-Eql  00-Eql     0   00-Eql    0 <- Correct      0
Tr     4      1   666  10-Inc  10-Inc     0   00-Eql    0 <- Correct      1
Tr     5      2   696  10-Inc  10-Inc     0   00-Eql    0 <- Correct      2
Tr     6      3   709  10-Inc  10-Inc     0   00-Eql    0 <- Correct      3
Tr     7      4   723  10-Inc  10-Inc     0   00-Eql    0 <- Correct      4
```

```
Tr     8      5   730  10-Inc  10-Inc     0   00-Eql    0 <- Correct      5
Tr     9      6   730  00-Eql  10-Inc     0   00-Eql    0 <- Correct      6
Tr    10      7   730  00-Eql  10-Inc     0   00-Eql    0 <- Correct      7
Tr    11      8   730  00-Eql  10-Inc     0   00-Eql    0 <- Correct      8
Tr    12      9   718  01-Dec  01-Dec     5   10-Inc    9 <- Correct      9
Tr    13     10   727  10-Inc  10-Inc     9   01-Dec    5 <- Correct     10
Tr    14     11   727  00-Eql  10-Inc     9   01-Dec    5 <- Correct     11
Tr    15     12   737  10-Inc  10-Inc     9   01-Dec    5 <- Correct     12

Errors last-2-trend: 0
Errors current trend, idx start: 0, 0
Errors previ trend, length: 0, 0
Done. Summary: 10 resets, 6706 changes, errs: (pt,pl,ct,is,lt) 0,0,0,0,0
```

The last four lines shown above, those starting with `Errors` and `Done`, show the number of errors detected by the testbench. The output above is correct, and so has zero errors.

The lines starting with `Tr` are trace lines, which show the state of the sequence, expected module outputs, and the value of the `debug_val` object in module `trend2`. Column headings for the trace lines are shown about every sixteen lines. If there had been errors then an `Er` line would appear after the trace line.

The `Cyc` column shows the current cycle. Unlike the index (`Idx`), cycle numbers never repeat. The value under column `c` shows an `R` when the `reset` input is set to `1`. In the sample above this occurs in cycle 3. The `Idx` column shows the index of the sequence. Recall that it is zero when `reset` is `1`, and then is incremented in subsequent cycles. The `Samp` column shows the value of the `samp` input to the module (which is the sequence sample).

The `TrLst2` column shows the correct value of the `last_2_trend` output, which should be set to the trend of the last two samples. As with all trends, the value is shown in binary and using descriptive text. On `Er` lines the actual module output is shown, but only if it is incorrect.

For trace lines the `TrCurr` column shows the correct value of the `curr_trend` output, and `IdxSt` shows the correct value of the `curr_idx_start` output. For error lines those columns show actual module output (but only when it is incorrect).

Column `TrPrev` and `LenP` show the correct value of the `prev_trend` and `prev_Len` outputs. For error lines those columns show actual module output (but only when it is incorrect).

The `Debug` line shows the actual value of `trend2.debug_val`. This value is shown but not checked. It is intended for debugging and can be set to whatever you like. In the output above `debug_val` is set to the module's value of `idx`.

The except below are the trace lines from an unmodified assignment:

```
     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr     3 R    0   662  00-Eql  00-Eql     0   00-Eql    0 <- Correct    662
Er                     xx-Unk  xx-Unk     X   xx-Unk    X <- Mod Errors
Tr     4      1   666  10-Inc  10-Inc     0   00-Eql    0 <- Correct    666
Er                     xx-Unk  xx-Unk     X   xx-Unk    X <- Mod Errors
Tr     5      2   696  10-Inc  10-Inc     0   00-Eql    0 <- Correct    696
Er                     xx-Unk  xx-Unk     X   xx-Unk    X <- Mod Errors
```

In an unmodified assignments the module outputs are not connected, and so their values are `x` (because they are `var` objects, had they been `net` objects their values would have been `z`). The `debug_val` object is set to the `samp` input.

The output below is excerpted from a correct assignment. The sequences show what outputs are expected. To start, consider the outputs following a reset (which will always occur at the start,

and at other times during testing):

```
     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr     3 R    0   662  00-Eql  00-Eql     0   00-Eql    0 <- Correct      0
Tr     4      1   666  10-Inc  10-Inc     0   00-Eql    0 <- Correct      1
Tr     5      2   696  10-Inc  10-Inc     0   00-Eql    0 <- Correct      2
Tr     6      3   709  10-Inc  10-Inc     0   00-Eql    0 <- Correct      3
Tr     7      4   723  10-Inc  10-Inc     0   00-Eql    0 <- Correct      4
Tr     8      5   730  10-Inc  10-Inc     0   00-Eql    0 <- Correct      5
Tr     9      6   730  00-Eql  10-Inc     0   00-Eql    0 <- Correct      6
Tr    10      7   730  00-Eql  10-Inc     0   00-Eql    0 <- Correct      7
Tr    11      8   730  00-Eql  10-Inc     0   00-Eql    0 <- Correct      8
Tr    12      9   718  01-Dec  01-Dec     5   10-Inc    5 <- Correct      9
Tr    13     10   727  10-Inc  10-Inc     9   01-Dec    4 <- Correct     10
Tr    14     11   727  00-Eql  10-Inc     9   01-Dec    4 <- Correct     11
Tr    15     12   737  10-Inc  10-Inc     9   01-Dec    4 <- Correct     12
Tr    16     13   759  10-Inc  10-Inc     9   01-Dec    4 <- Correct     13
Tr    17     14   772  10-Inc  10-Inc     9   01-Dec    4 <- Correct     14
```

After a reset the last-2 trend, previous trend, and length outputs are all set to zero. In cycle 4 the last-2 output indicates an increasing sequence, so the current trend is set to increasing (a value of 10_2) and the start index is set to zero (which it was initialized at anyway).

Up until cycle 12 the samples had been increasing, but at cycle 12 the sample is smaller than the sample at cycle 11. By the definition of a descending segment, the first sample must be larger than the one before it, so the descending segment detected at cycle 12 (index 9) actually started on cycle 8 (index 5), and so the start index is 5. That's the value shown under `IdxSt`. This descending sequence ends quickly, in cycle 13 by the arrival of sample 727. The new ascending sequence starts at cycle 12 (index 9), and continues through cycle 17.

The sequence of samples above, from index 0 to index 14, consists of three segments. The first segment, from index 0 to 4, is ascending. The second segment, from index 5 to 8 is descending. The third segment, starting at index 9 is ascending (and continues until index 13, though that's not shown above).

Notice that when a new segment is detected, the information on the current one is moved to the `TrPrev` and `LenP` columns.

Additional output appears below:

```
     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr    16     13   759  10-Inc  10-Inc     9   01-Dec    4 <- Correct     13
Tr    17     14   772  10-Inc  10-Inc     9   01-Dec    4 <- Correct     14
Tr    18     15   769  01-Dec  01-Dec    14   10-Inc    5 <- Correct     15
Tr    19     16   762  01-Dec  01-Dec    14   10-Inc    5 <- Correct     16
Tr    20     17   762  00-Eql  01-Dec    14   10-Inc    5 <- Correct     17
Tr    21     18   764  10-Inc  10-Inc    16   01-Dec    2 <- Correct     18
Tr    22     19   764  00-Eql  10-Inc    16   01-Dec    2 <- Correct     19
Tr    23     20   764  00-Eql  10-Inc    16   01-Dec    2 <- Correct     20
Tr    24     21   776  10-Inc  10-Inc    16   01-Dec    2 <- Correct     21
Tr    25     22   776  00-Eql  10-Inc    16   01-Dec    2 <- Correct     22
Tr    26     23   787  10-Inc  10-Inc    16   01-Dec    2 <- Correct     23
Tr    27     24   787  00-Eql  10-Inc    16   01-Dec    2 <- Correct     24
Tr    28     25   771  01-Dec  01-Dec    23   10-Inc    7 <- Correct     25
```

```
     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr    29     26   769  01-Dec  01-Dec    23   10-Inc    7 <- Correct     26
Tr    30     27   762  01-Dec  01-Dec    23   10-Inc    7 <- Correct     27
Tr    31     28   771  10-Inc  10-Inc    27   01-Dec    4 <- Correct     28
Tr    32     29   762  01-Dec  01-Dec    28   10-Inc    1 <- Correct     29
Tr    33     30   738  01-Dec  01-Dec    28   10-Inc    1 <- Correct     30
Tr    34     31   738  00-Eql  01-Dec    28   10-Inc    1 <- Correct     31
[snip]
     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr    55     52   615  10-Inc  10-Inc    48   01-Dec   20 <- Correct     52
Tr    56     53   622  10-Inc  10-Inc    48   01-Dec   20 <- Correct     53
Tr    57     54   611  01-Dec  01-Dec    53   10-Inc    5 <- Correct     54
Tr    58     55   618  10-Inc  10-Inc    54   01-Dec    1 <- Correct     55
Tr    59     56   638  10-Inc  10-Inc    54   01-Dec    1 <- Correct     56
Tr    60     57   633  01-Dec  01-Dec    56   10-Inc    2 <- Correct     57
Tr    61     58   612  01-Dec  01-Dec    56   10-Inc    2 <- Correct     58
Tr    62     59   598  01-Dec  01-Dec    56   10-Inc    2 <- Correct     59
Tr    63     60   589  01-Dec  01-Dec    56   10-Inc    2 <- Correct     60
Tr    64     61   609  10-Inc  10-Inc    60   01-Dec    4 <- Correct     61
Tr    65     62   597  01-Dec  01-Dec    61   10-Inc    1 <- Correct     62
Tr    66     63   607  10-Inc  10-Inc    62   01-Dec    1 <- Correct     63
Tr    67     64   599  01-Dec  01-Dec    63   10-Inc    1 <- Correct     64

     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr    68     65   599  00-Eql  01-Dec    63   10-Inc    1 <- Correct     65
Tr    69     66   606  10-Inc  10-Inc    64   01-Dec    1 <- Correct     66
Tr    70     67   609  10-Inc  10-Inc    64   01-Dec    1 <- Correct     67
Tr    71     68   611  10-Inc  10-Inc    64   01-Dec    1 <- Correct     68
Tr    72     69   625  10-Inc  10-Inc    64   01-Dec    1 <- Correct     69
Tr    73     70   631  10-Inc  10-Inc    64   01-Dec    1 <- Correct     70
Tr    74     71   618  01-Dec  01-Dec    70   10-Inc    6 <- Correct     71
Tr    75     72   629  10-Inc  10-Inc    71   01-Dec    1 <- Correct     72
Tr    76     73   629  00-Eql  10-Inc    71   01-Dec    1 <- Correct     73
Tr    77     74   613  01-Dec  01-Dec    72   10-Inc    1 <- Correct     74
Tr    78     75   620  10-Inc  10-Inc    74   01-Dec    2 <- Correct     75
Tr    79     76   627  10-Inc  10-Inc    74   01-Dec    2 <- Correct     76
Tr    80     77   629  10-Inc  10-Inc    74   01-Dec    2 <- Correct     77

     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr    81     78   614  01-Dec  01-Dec    77   10-Inc    3 <- Correct     78
Tr    82     79   609  01-Dec  01-Dec    77   10-Inc    3 <- Correct     79
Tr    83     80   597  01-Dec  01-Dec    77   10-Inc    3 <- Correct     80
Tr    84     81   609  10-Inc  10-Inc    80   01-Dec    3 <- Correct     81
Tr    85     82   609  00-Eql  10-Inc    80   01-Dec    3 <- Correct     82
Tr    86     83   617  10-Inc  10-Inc    80   01-Dec    3 <- Correct     83
Tr    87     84   611  01-Dec  01-Dec    83   10-Inc    3 <- Correct     84
Tr    88     85   601  01-Dec  01-Dec    83   10-Inc    3 <- Correct     85
Tr    89     86   586  01-Dec  01-Dec    83   10-Inc    3 <- Correct     86
```

```
Tr    90     87   575  01-Dec  01-Dec    83   10-Inc    3 <- Correct     87
Tr    91     88   557  01-Dec  01-Dec    83   10-Inc    3 <- Correct     88
Tr    92     89   549  01-Dec  01-Dec    83   10-Inc    3 <- Correct     89
Tr    93     90   561  10-Inc  10-Inc    89   01-Dec    6 <- Correct     90

     Cyc c  Idx  Samp  TrLst2  TrCurr IdxSt   TrPrev LenP              Debug
Tr    94     91   556  01-Dec  01-Dec    90   10-Inc    1 <- Correct     91
Tr    95     92   552  01-Dec  01-Dec    90   10-Inc    1 <- Correct     92
Tr    96     93   560  10-Inc  10-Inc    92   01-Dec    2 <- Correct     93
Tr    97     94   555  01-Dec  01-Dec    93   10-Inc    1 <- Correct     94
Tr    98     95   555  00-Eql  01-Dec    93   10-Inc    1 <- Correct     95
Tr    99     96   530  01-Dec  01-Dec    93   10-Inc    1 <- Correct     96
```

Helpful Examples

Past assignments and their solutions are linked to the assignments and exams page.

Problem 1: In the unsolved assignment `trend2` does not set any of its outputs.

- [ ] Complete `trend2` so that it identifies the current and previous sequence segments as defined above.
- [ ] Use instantiation(s) of `ist_compare` to determine the order of samples.
- [ ] The module must be synthesizable. Use `genus -files syn.tcl` to synthesize your module.
- [ ] Do not assume specific parameter values.
- [ ] Code clearly.
- [ ] Avoid overly costly or slow solutions.