LSU EE 4755 **Homework 5** **Due: 30 Nov 2025**

Formatted 18:53, 21 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/hw05.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.) An LLM prompt has been hidden in this assignment.

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, `hw05.v`. Do this early enough so that minor problems (*e.g.*, password doesn't work) are minor problems.

Homework Overview

The `best_rot` (best rotation) modules in this assignment have two `wv`-bit inputs, `val` and `key`, where `wv` is a module parameter. The modules have an output `dif` which indicates the difference (in some contexts, called a *Hamming distance*) between the `val` and `key`. A `dif` of zero means that the two are identical. The largest possible `dif` is `wv`, which occurs when `val` is all `1`s and `key` is all `0`s, or vice versa. The Hamming distance between two w-bit bit vectors, a and b is the number of bits positions that differ. For example, if $a = 1001_2$ and $b = 0111_2$, their distance is 3 because only their least-significant bit positions are the same. Suppose `val=4'b0110` and `key=4'b1001`. The Hamming distance of those two values is four. However if `val` is end-around rotated to the right by two positions, `0110 -> 0011 -> 1001`, the resulting value is equal to `key`. The `best_rot` modules, given inputs `val` and `key` will consider all rotations of `val` and return the lowest distance (value of `dif`) and the number of rotations at which that distance was obtained.

Examples of the expected module outputs are shown in testbench excerpts appearing below *for the procedural module*. Data from the procedural module is easy to interpret because the values

at the outputs, under the `Pos` and `Dif` columns, is for the data at the inputs, under the `Val` and `Key` columns in the same cycle (and the same row of the table). When $w \leq 8$ the values under `Val` and `Key` are shown in binary, otherwise they are shown in hexadecimal. The other columns will be explained further below.

```
Starting tests for best_rot_proc w=4.
    Cyc  Val   Key  S           Cyc-In  R  Pos  Dif
Tr    5 0001  0100  S  Mod Out    5 --> X    2    0
Tr    6 1001  1001  S  Mod Out    6 --> X    0    0
Tr    7 1000  1011  S  Mod Out    7 --> X    0    2
Tr    8 1011  1011  S  Mod Out    8 --> X    0    0
Tr    9 0111  1101  S  Mod Out    9 --> X    2    0
Tr   10 1100  1111  S  Mod Out   10 --> X    0    2
Tr   11 1011  1101  S  Mod Out   11 --> X    1    0
Tr   12 0100  1101  S  Mod Out   12 --> X    0    2
Tr   13 0010  0001  S  Mod Out   13 --> X    1    0

Starting tests for best_rot_proc w=8.
    Cyc      Val       Key  S           Cyc-In  R  Pos  Dif
Tr    5 00000001  00001000  S  Mod Out    5 --> X    5    0
Tr    6 10000100  10000010  S  Mod Out    6 --> X    0    2
Tr    7 10101111  10101111  S  Mod Out    7 --> X    0    0
Tr    8 10100100  10010010  S  Mod Out    8 --> X    6    0
Tr    9 00111101  11001011  S  Mod Out    9 --> X    2    2
Tr   10 00000100  01010111  S  Mod Out   10 --> X    0    4
Tr   11 01100110  11111111  S  Mod Out   11 --> X    0    4
Tr   12 00000100  00001000  S  Mod Out   12 --> X    7    0
Tr   13 11111110  11111101  S  Mod Out   13 --> X    7    0
Tr   14 01101111  00000100  S  Mod Out   14 --> X    0    5
Tr   15 11110111  11011110  S  Mod Out   15 --> X    3    1
Tr   16 10101101  01101011  S  Mod Out   16 --> X    2    0
Tr   17 10000000  00100000  S  Mod Out   17 --> X    2    0

Starting tests for best_rot_proc w=20.
    Cyc   Val    Key  S           Cyc-In  R  Pos  Dif
Tr    5 20000  00020  S  Mod Out    5 --> X   12    0
Tr    6 10004  80020  S  Mod Out    6 --> X   17    0
Tr    7 00100  40000  S  Mod Out    7 --> X   10    0
Tr    8 574e1  10000  S  Mod Out    8 --> X    0    9
Tr    9 140a3  fe76e  S  Mod Out    9 --> X    6    9
Tr   10 f83f8  c7c1f  S  Mod Out   10 --> X    5    0
Tr   11 d65f7  04802  S  Mod Out   11 --> X    5   11
Tr   12 50148  ffb56  S  Mod Out   12 --> X    2   10
Tr   13 fffff  fffff  S  Mod Out   13 --> X    0    0
Tr   14 30030  81801  S  Mod Out   14 --> X    5    0
Tr   15 04800  6435c  S  Mod Out   15 --> X    5    7
Tr   16 fe7fd  dd5ff  S  Mod Out   16 --> X    0    5
```

The assignment file, `hw05.v`, has three versions of `best_rot`. Module `best_rot_procedural` is a procedural version of the module and `best_rot_seq` is a sequential version of the module. Both of these work correctly and should not be modified for this assignment.

Module `best_rot_procedural` is straightforward: it has `wv`-bit inputs `key` and `val`, and `wp`-bit

outputs `pos` and `dif`. Of course, `wp` and `wv` are parameters. The outputs are updated whenever the inputs change. The synthesized hardware would have `wv` copies of the logic to compute the distance, one for each possible rotation of `val`. Those who know where we're going will understand why I won't make such a big deal about the cost of all of those copies of the compute-the-distance logic.

Module `best_rot_seq`, in addition to the connections used by `best_rot_procedural`, has 1-bit inputs `start` and `clk` and a 1-bit output `ready`. If requires `wv` cycles to compute `pos` and `dif`. Computation starts when `start` is set to one on a positive edge of `clk`. Outputs `pos` and `dif` are correct when `ready` is set to `1`. Module `best_rot_seq` uses module `pop` to help compute `diff`.

Because of all the logic involved the delay of `best_rot_procedural` is likely to be higher than that of `best_rot_seq`, but the latency of `best_rot_seq` will likely be higher because it requires `wv` cycles to compute a result. Let t_{pr} denote the delay of `best_rot_procedural` and t_{se} denote the delay of `best_rot_seq` and suppose for both `wv`=w. The latency of `best_rot_seq` is wt_{se}.

Suppose $t_{\text{pr}} = 5\,\text{ns}$ and $t_{\text{se}} = 0.9\,\text{ns}$ and we would like to use a best rot module on a system with clock period $t = 1\,\text{ns}$. The `best_rot_procedural` module would be too slow. Module `best_rot_seq` is fast enough because $t_{\text{se}} < t$, but we can compute at most one result only every w cycles. What do we do?

The answer, of course, is to design a pipelined version of the module, which we'll call `best_rot_pipe`. This will be designed so that the critical path (delay) is close to that of `best_rot_seq` but being pipelined it can accept a new `val` `key` pair each cycle.

Module `best_rot_pipe` has the same connections as `best_rot_seq`:

```
module best_rot_pipe
    #( int wv = 17, wp = $clog2(wv+1) )
   ( output logic [wp-1:0] pos,
     output logic [wp-1:0] dif,
     output logic ready,
     input uwire [wv-1:0] val, key,
     input uwire start, clk );
```

The difference is in the timing. For `best_rot_pipe` a new `val` and `key` pair can arrive every cycle. Input `start` is set to `1` when `val` and `key` are valid values. (It might help to ignore the `start` and `ready` signals, especially since in the unsolved assignment that part is completed. That is, unlike the sequential design, the pipelined design does not do anything special when `start=1`, other than sending it through the pipeline.)

Output `ready` at cycle x should be set to the value of `start` that was present at cycle $x - w$. Or put another way, the value of `start` that arrives at cycle y should appear at output `ready` in cycle $y + c$. The provided code already does that. In a correctly solved assignment the `pos` and `dif` outputs at time $y + c$ will have the correct values for `val` and `key` from cycle y.

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 `best_rot_pipe` and report on the results. The module will be instantiated at three different widths, `wv=4`, `wv=8`, and `wv=20`.

The testbench applies inputs each cycle, but only checks the output for some of these. When one of those outputs is wrong it will print a trace starting from when the input was applied, to when the output was expected. Lines starting `Tr` show the module inputs and actual outputs (plus a debug value). Lines starting `Cr` show the module outputs that were expected. The first except below is from an unsolved assignment, the second from a correctly solved assignment:

```
Starting tests for best_rot_pipe w=4.   ** UNSOLVED **
    Cyc  Val   Key  S           Cyc-In  R  Pos  Dif   Debug Val
Tr    6 0001  0100  S  Mod Out    2 --> _    x    x   xxxxx
Tr    7 0110  1001  _  Mod Out    3 --> R    x    x   xxxxx
Tr    8 0111  0101  _  Mod Out    4 --> R    x    x   xxxxx
Tr    9 1000  0101  _  Mod Out    5 --> _    x    x   xxxxx
Tr   10 1001  0010  _  Mod Out    6 --> R    x    x   xxxxx
Cr   10               Cor Out    6 --> R    2    0
```

```
Starting tests for best_rot_pipe w=4.   ** CORRECTLY SOLVED **
    Cyc  Val   Key  S           Cyc-In  R  Pos  Dif   Debug Val
Tr    6 0001  0100  S  Mod Out    2 --> _    0    1   00001
Tr    7 0110  1001  _  Mod Out    3 --> R    1    0   00000
Tr    8 0111  0101  _  Mod Out    4 --> R    3    0   00000
Tr    9 1000  0101  _  Mod Out    5 --> _    0    2   00002
Tr   10 1001  0010  _  Mod Out    6 --> R    2    0   00002
Cr   10               Cor Out    6 --> R    2    0
```

The `Val`, `Key`, and `S` columns show the module inputs. `Val` and `Key` are in binary when $w \leq 8$ otherwise they are in hexadecimal. The `Cyc` column shows the cycle at which the port values were taken (meaning all values on that row). The `Pos` and `Dif` columns show the value of the `pos` and `dif` outputs. An `x` indicates that at least one bit was not a `0` or `1`. Those outputs are `x` in the unsolved excerpt because the `pos` and `dif` output ports were unconnected. The `R` column shows the value of the `ready` output.

The column headed `Cyc-In` shows which data should have been used to compute the `pos` and `diff` outputs on that row. For example, consider the line starting `Tr 10`. In that line `Cyc-In` is 6, meaning that the values under `Pos` and `Dif` are for the data arriving in cycle 6, `val=0001` and `key=0100`, **not** for `val=1001` and `key=0010`. Make sure that you understand this!

The excerpt above shows five outputs (for cycles 6 to 10), but only the output arriving at cycle 10 is checked. The expected output is shown in the line starting `Cr`. For the second excerpt above the `Pos` and `Dif` values match and so are correct.

The values under the `Debug Val` column show the value of object `watch_value` in your module. You can set `watch_value` to whatever you like, it is not checked for correctness. In the unsolved assignment it is set to the number of valid values in the pipeline. In the excerpt above for the solved solution, `watch_value` is set to the value of `pos` in the next-to-last pipeline stage.

The testbench starts by choosing easy values for `val` and `key`: In the first ten test patterns both `val` and `key` will each have just one bit set. That means they are guaranteed to have a `dif` of zero. In the next ten patterns exactly two bits of each `val` and `key` will be one. These easy test patterns are to facilitate debugging. After that the inputs can have more than two 1's set.

After completing tests the testbench prints a tally of results:

```
xcelium> exit
Total best_rot_pipe n= 4: Errors: 1000 pos, 1000 dif.
Total best_rot_pipe n= 8: Errors: 1000 pos, 1000 dif.
Total best_rot_pipe n=20: Errors: 1000 pos, 1000 dif.
Grand Total Errors:  3000 pos, 3000 dif.
```

For a correctly solved assignment:

```
xcelium> exit
```

```
Total best_rot_pipe n= 4: Errors: 0 pos, 0 dif.
Total best_rot_pipe n= 8: Errors: 0 pos, 0 dif.
Total best_rot_pipe n=20: Errors: 0 pos, 0 dif.
Grand Total Errors:  0 pos, 0 dif.
```

Helpful Examples

The `best_rot` modules are similar to the `best_match` modules that were the subject of 2019 Homework 4 and Homework 5. There are two significant differences: In `best_match` the `val` input is much wider than the `k` (key) input, whereas in 2025 both are the same size. In 2019 one had to find a position in `val` where `k` had the best match, in 2025 we have to rotate `val` and note the rotation that provides the best match. Another difference is that in the 2019 assignment a sequential module was to be designed. In this 2025 assignment a pipelined module is to be designed. A sequential module that rotated its arguments, `rmatch`, appeared as Problem 4 on the 2024 Final Exam.

Coding pipelined modules were the subject of 2021 Homework 6, 2017 Homework 7, and 2016 Homework 6. Past assignments and their solutions are linked to the assignments and exams page.

Problem 1: Complete `best_rot_pipe` so that it computes the best rotation in pipelined fashion and meets the requirements below:

- ☐ Complete `best_rot_pipe` so that it computes `pos` and `dif` values as described earlier.
- ☐ The module must operate in pipelined fashion and spread work between the `wv` stages ☐ so that the critical path is short.
- ☐ The module must use `pop` instantiations to compute `diff`.
- ☐ The testbench should report zero errors.
- ☐ The code should be clearly written.
- ☐ Avoid slow or costly designs.
- ☐ Do not assume specific parameter values.
- ☐ Only modify `best_rot_pipe`.
- ☐ The module must be synthesizeable. Use command `genus -files syn.tcl` to synthesize