LSU EE 4755 Homework 3 Due: 25 September 2018

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/2018/hw03.v.html`.

Problem 0: Following instructions at `https://www.ece.lsu.edu/koppel/v/proc.html`, set up your class account, copy the assignment, and run the Verilog simulator and synthesis program on the unmodified homework file, `hw03.v`.

Homework Overview

The sorting networks from Homework assignments 1 and 2 sorted keys only, and they only sorted unsigned integer keys. In this assignment sorter inputs will consist of keys and data, and those keys can be signed integers or floating-point values. The only module to be modified for this assignment is `sort2`.

Module `sort2` has two inputs, `a0` and `a1`, and parameters `w`, `k`, `exp`, and `sig`. Parameter `w` is the total size of each input, in bits, `k` is the size of the key, `exp` is size of the exponent (for FP keys) and `sig` is the size of the significand (for FP keys). Each input consists of data, in bit positions `w-1:k+1`, a key type, in bit position `k`, and a key, in bit positions `k-1:0`. If bit `k` is zero the key is a **signed** integer in 2's complement representation. If bit `k` is one the key is a FP value in a format similar to IEEE 754: Bit `k-1` is the sign, bits `k-2:sig` are the exponent, and bits `sig-1:0` are the significand. For a description of these fields see the floating-point modules in the ChipWare documentation (linked to the course references page, `https://www.ece.lsu.edu/koppel/v/ref.html`, and also linked to the HTMLized assignment code, `https://www.ece.lsu.edu/koppel/v/2018/hw03.v.html`). Also see the `fp_to_val` function in the testbench code, this function converts this floating-point representation to a value.

In the unmodified file the `sort2` module compares the inputs as unsigned integers. This is wrong because the high bits of each input are data, not the keys. The `mux` connections are correct because each input should be sent to the appropriate output unmodified. The solution to the problems below involve setting `c` (the mux select signal) to the correct value.

Testbench Code

The testbench for this assignment, which can be run when visiting the file in Emacs in a properly set-up account by pressing `F9`, tests module `sort2` at two different sizes and using a mix of input types. It first tries integer-only keys (labeled `ii` in the output), then floating-point only keys (labeled `ff`), and finally integer/FP keys (labeled `if`). It reports the first five errors of each type, and for each module size reports a tally by type.

Here is a transcript showing the start of the testbench (after the compiler's own messages):

```
Starting testbench for w=32, k=16, exp=6  sig width=9...
Test ii    3, error (x0,x1): (462cf78c,7cfcf78b) != (7cfcf78b,462cf78c) correct.
           a0: data 462c, key  -2164.00000 = INT 'hf78c
           a1: data 7cfc, key  -2165.00000 = INT 'hf78b
           To re-run paste:  tests.push_back('h462cf78c); tests.push_back('h7cfcf78b);
Test ii    4, error (x0,x1): (72aed2ac,d512d2aa) != (d512d2aa,72aed2ac) correct.
           a0: data d512, key -11606.00000 = INT 'hd2aa
           a1: data 72ae, key -11604.00000 = INT 'hd2ac
           To re-run paste:  tests.push_back('hd512d2aa); tests.push_back('h72aed2ac);
```

The transcript above shows two errors, both for integer key pairs. The first line shows the actual output followed by the correct output (labeled `correct`). The number before `error` is a

test number, these start at zero and go up to `num_tests-1` (see the testbench code). The next two lines show the input values broken into data and key, including the value and representation details. The last line of each error report has text that can be put into the testbench code so that particular test can be re-run as one of the first tests.

The testbench tests the `sort2` module at two sizes. At the end of the code for each is a tally of the number of errors:

```
Done with 3000000 tests for k=16, exp=6:   499679 ff errs,   499666 if errs,   499400
ii errs,
```

In the sample above there are many errors for each type of test. Here is the output when all tests pass:

```
Starting testbench for w=32, k=16, exp=6  sig width=9...
Done with 3000000 tests for k=16, exp=6:   0 ff errs,   0 if errs,   0 ii errs,
Starting testbench for w=24, k=14, exp=5  sig width=8...
Done with 3000000 tests for k=14, exp=5:   0 ff errs,   0 if errs,   0 ii errs,

All done.
```

Debugging

To debug your code identify an error that looks easy to figure out and copy the text to the right of `paste:` into the `testbench_size` module near the comment `Add tests below`. Also change the value of `num_tests` to a small number, say 3. (Don't forget to change it back!) Verify that the code fails on test 0 (or some other small number). Next, run SimVision: `irun -gui hw03.v`. Locate your module (it will be under `t1` or `t2`) and copy symbols from `s2` to the waveform viewer. See the SimVision instructions on the `https://www.ece.lsu.edu/koppel/v/proc.html` page.

Synthesis

The synthesis script, `syn.tcl`, will synthesize `sort2` with two delay targets, an easy 10 ns and a unachievable 0.1 ns. If the module doesn't synthesize $-.001$ s is shown for the delay. The script is run using the shell command `genus -files syn.tcl`, which invokes Cadence Genus. In past semesters Cadence RTL Compiler (`rc`) was used, which would be invoked using `rc -files syn.tcl`, **but that won't work on the 2018 homework assignments**.

The synthesis script shows area (cost), delay, and the delay target in a neat table. Additional output of the synthesis program is written to file `spew.log`. Sample synthesis script output appears below:

Problem 1: Complete module `sort2` so that it correctly sorts inputs with **signed** integer keys. Avoid unnecessarily costly or slow designs.

Problem 2: Complete module `sort2` so that it also correctly sorts inputs with floating-point keys. Instantiate at least one ChipWare module, it's okay to use more. When adding ChipWare modules be sure to put in an `include` directive at the end of the file. Avoid unnecessarily costly or slow designs.

Problem 3: Complete module `sort2` so that it also correctly sorts inputs when one key is a signed integer and the other is floating point. Avoid unnecessarily costly or slow designs. Try to avoid solutions that use a larger significand than is specified by the parameters or other brute-force approaches.