LSU EE 4755 Homework 2 Due: 25 September 2017

For instructions visit `https://www.ece.lsu.edu/koppel/v/proc.html`*. For the complete Verilog for this assignment without visiting the lab visit* `https://www.ece.lsu.edu/koppel/v/2017/hw02.v.html`.

Problem 1: Suppose point x_1 has value a_1, and suppose point x_2 has value a_2. Let $a(x)$ be the value at point x determined by linearly interpolating the values at x_1 and x_2. For example, $a(x_1) = a_1$ and $a(\frac{1}{2}(x_1 + x_2)) = a_1 + \frac{1}{2}(a_1 + a_2)$. In general, $a(x) = a_1 + (x - x_1)\frac{a_2 - a_1}{x_2 - x_1}$ for real values x_1, x_2, a_1, a_2, and x. Define a version of linear interpolation, $a_i(j)$, in which the point to interpolate, j, is relative to x_1 and in which the interpolated value is the floor of the actual value. That is, $a_i(j) = \lfloor a_1 + j\frac{a_2 - a_1}{x_2 - x_1} \rfloor$. For example, $a_i(0) = \lfloor a_1 \rfloor$ and $a_i(x_2 - x_1) = \lfloor a_2 \rfloor$.

Module `interp_behav` in `hw02.v` performs such a linear interpolation. (You might want to examine this module to double-check your understanding of what $a_i(j)$ does.) Alas, it is not synthesizable. Modify module `interp` in `hw02.v` (partially shown below) so that it correctly performs the linear interpolation, is synthesizable, and uses the floating-point modules from the ChipWare library. (Don't try to do everything using integer arithmetic.) (Additional information is provided after the subproblems.)

(*a*) Complete module `interp` below so that it sets output `valid` to `1` if $\lfloor x_1 \rfloor + j \leq \lfloor x_2 \rfloor$.

(*b*) Complete module `interp` below so that it sets output `aj` to $a_i(j)$ based on the values at its input ports, with port names approximately matching the symbols used above.

```
module interp
  #( int jw = 12, int amax = 255 )
   ( output uwire valid,
     output uwire [7:0] aj,
     input uwire [31:0] x1, a1, x2, a2,
     input uwire [jw-1:0] j );

endmodule
```

Module `interp_behav` is not synthesizable because it uses operators to perform floating-point arithmetic. In module `interp` instantiate ChipWare modules to perform floating-point operations. Module `interp` already instantiates an adder and a float-to-int converter. Find additional modules in the ChipWare documentation, which can be found on the course references page. When using a ChipWare module **put in an include directive at the end of the file**. See the end of `hw02.v` for examples.

The testbench will test module `interp`, it should initially report lots of errors. Of course, when you are done there should be zero errors.

Follow the synthesis steps on the course procedures page to determine if `interp` is synthesizable. If the elaborate step is successful then the module is synthesizable.

Problem 2: Floating-point hardware is relatively costly. Compare the cost of FP and integer arithmetic units by synthesizing equivalent FP and integer adders and dividers. Wrap the ChipWare modules in your own modules, (such as `fp_add` in `hw02.v`) and set parameters so the FP and integer units are comparable. Then modify the synthesis script, `syn.tcl`, so that it will synthesize these modules. The modules should be added to the list assigned on the `set modules` line.

Based on the data collected above, indicate how much less you think the cost would be of an `interp` module that used integer arithmetic.