LSU EE 4755 Homework 5 Due: 7 November 2016

Problem 0: This first problem provides background on the module used in this assignment. Please read the background and then solve the problems further below. The Verilog source can be found in directory `hw05`, however for this assignment there is no need to do anything with it.

Module `ortho` has one input, `v`, a three-element vector of signed integers, and one output, `u`, also a three-element vector of signed integers. The output is computed so that `u` is orthogonal to `v` in the geometric sense. For those who are rusty on linear algebra, non-zero vectors u and v are orthogonal if $u \cdot v = 0$ or $u_x v_x + u_y v_y + u_z v_z = 0$. Using Verilog notation, `u` is computed so that `u[0]*v[0]+u[1]*v[1]+u[2]*v[2]=0` and at least one element of `u` is not zero. It does so by finding the smallest element of `v`, setting the corresponding element in `u` to zero, swapping the to remaining two elements, and negating one of the two. For example, if $v = (4, 7, 55)$ then the module would set $u = (0, 55, -7)$.

```
module ortho #( int alternative = 1, int w = 32 )
   ( output logic signed [w-1:0] u [3],   input wire signed [w-1:0] v [3] );

   logic [1:0] idx_min, idx_a, idx_b;

   always_comb begin

      idx_min = 0;
      for ( int i=1; i<3; i++ )
        if ( $abs(v[i]) < $abs(v[idx_min]) ) idx_min = i;

      idx_a = ( idx_min + 1 ) % 3;
      idx_b = ( idx_min + 2 ) % 3;

      if ( alternative == 1 ) begin

         // The loop below is needed as a hint to the synthesis program
         // Cadence Encounter 14.28.
         for ( int i=0; i<3; i++ ) u[i] = 0;

         u[idx_min] = 0;
         u[idx_a] = v[idx_b];
         u[idx_b] = -v[idx_a];

      end else if ( alternative == 2 ) begin

         for ( int i=0; i<3; i++ )
           u[i] = idx_min == i ? 0 : idx_a == i ? v[idx_b] : -v[idx_a];

      end else $fatal(1);

   end

endmodule
```

Important: For all problems below in which hardware is shown:

- Clearly show inputs and outputs of `ortho`.
- Try to draw diagrams showing all hardware for `ortho` and refer to parts of the diagram in your answers below.

Problem 1: Consider the following part of the module:

```
idx_min = 0;
for ( int i=1; i<3; i++ )
  if ( $abs(v[i]) < $abs(v[idx_min]) ) idx_min = i;
```

(*a*) Show the hardware that will be synthesized for this fragment. (Please refer to the entire module when determining what will be synthesized.) Make reasonable optimizations. (See the next subproblem.) In this subpart show `abs` as a box.

(*b*) The synthesis program synthesizes hardware that contains four absolute value units for this code, even with effort set to high. Explain why four is too many, perhaps by referring your own version that uses fewer absolute value units.

Problem 2: Consider the part of the module below: Show the hardware that will be synthesized for this code, taking into consideration that `idx_min` is two bits. *Hint: This is easy. Just consider all possible values of* `idx_min`.

```
idx_a = ( idx_min + 1 ) % 3;
idx_b = ( idx_min + 2 ) % 3;
```

Problem 3: Show the hardware that will be synthesized for the alternative 2 code, below, after optimization. As with the other problems, take into account the rest of the module. Look for opportunities to optimize `-v[idx_a]` taking advantage of hardware for `abs`.

```
for ( int i=0; i<3; i++ )
  u[i] = idx_min == i ? 0 : idx_a == i ? v[idx_b] : -v[idx_a];
```

Problem 4: As directed below, estimate the critical path in `ortho` for a w-bit instantiation. Do so using ripple-adder like implementations for absolute value, comparison, and negation. Use the performance model in which n-input AND and OR gates have delay $\lceil \lg n \rceil$ units.

(*a*) Find the critical path using the assumption that in hardware for an expression like $a + b < c$ the delay through the adder must be added to the delay through the comparison unit. The answer should be a function of w.

(*b*) Find the critical path accounting for the fact that in ripple-like hardware for an expression like $a + b < c$ the low bits of the comparison can start as soon as the low bits of the sum are available. The answer should be a function of w.

(*c*) Show a sketch of the hardware with an arrow tracing the critical path through the hardware, from input to output. Annotating that arrow with intermediate delays will help in assigning partial credit.