LSU EE 3755 Homework 4 Solution Due: 27 October 2013

Problem 0: **Complete this problem as soon as possible.** Follow the instructions for *Account Setup* and *Verilog Homework Workflow* on `http://www.ece.lsu.edu/ee3755/proc.html`.

When the account is set up copy the assignment into your class account using the following commands:

```
[ee37551@frost ~]$ cd ~
[ee37551@frost ~]$ cp -r /home/faculty/koppel/pub/ee3755/hw/2013f/hw04 .
[ee37551@frost ~]$ cd hw04
[ee37551@frost hw04]$ ls
Makefile  hdl.var  hw04.v  syn-prob1.cmd  syn-prob2.cmd
```

The first two commands above copy the files, the last two verify that they have been copied by showing a directory listing.

Start Emacs and load the assignment, `hw04.v`, into an Emacs buffer. If the instructions were followed correctly the Verilog comments should appear in red and the text "LSU EE 3755..." should appear **red bold**. There should also be an "EE 3755" entry on the menu bar.

When a Verilog file is loaded the EE 3755 menu should contain one entry for simulating and two for synthesizing, one for each problem. Selecting "Simulate" from the menu should run the Verilog simulator, output will appear in a pane (called a *window* in Emacs parlance) below the window in which the Verilog code appears. For the unmodified assignment the following lines should appear near the end of the output:

```
Error in Problem 2 module for input 0xb000:  x !=          12 (correct)
Finished with  1000 tests,  1000 Prob-1 and  1000 Prob-2 errors found.
```

The `Finished with` line above indicates that the Verilog modules where tested with 1000 different inputs and that the module for problem 1 had 1000 errors (failed every test), and the same for problem 2. When the problems are solved correctly the line will read:

```
Finished with  1000 tests,     0 Prob-1 and     0 Prob-2 errors found.
```

The command to simulate runs tests on both modules (see problems below). For synthesis, there is a separate command for each problem (see the menu). Try out a synthesis command by switching to the buffer visiting `hw04.v` and selecting Synthesizing Problem 1 from the EE 3755 menu. The output of the synthesis program appears in two places, in the window below the Verilog code and in a file named `rc.logX`, where `X` is either not present or a number.

The synthesis output will start with:

```
-*- mode: compilation; default-directory: "~/teach/co13f/hw/hw04/" -*-
Compilation started at Mon Oct 21 12:09:37

make prob1
rc -E -nogui -files syn-prob1.cmd
Checking out license 'RTL_Compiler_Physical'... (0 seconds elapsed)
License RTL_Compiler_Physical checkout failed.
Checking out license 'RTL_Compiler_Ultra'... (0 seconds elapsed)

                      Cadence Encounter(R) RTL Compiler
        Version RC10.1.306 - v10.10-s357_1 (64-bit), built Apr 10 2012
```

For the unmodified assignment the synthesis output will end with:

```
No paths found.

Normal exit.
```

The synthesis program output contains three important things for this assignment: synthesis errors (if any), an area report, and a timing report. The sample outputs shown below will be based on the completed assignment.

Almost anything that is an error for the Verilog simulator is also an error for the synthesis program. But, as discussed in class, there are many things valid in Verilog that are not valid for the synthesis program. The following output indicates that there were no synthesis errors:

```
  Elaborating top-level block 'czc16_prob1' from file 'hw04.v'.
  Done elaborating 'czc16_prob1'.
  Synthesis succeeded.
```

Here is an example with errors:

```
  Elaborating top-level block 'czc16_prob2' from file 'hw04.v'.
Error   : Unsupported 'for' statement initialization assignment. [CDFG-453] [elaborate]
        : The unsupported initialization assignment is in file 'hw04.v' on line 126.
        : The initialization expression in a 'for' statement must evaluate to a constant when synthesiz-
ing the design.
  Module 'czc16_prob2' contains errors and cannot be elaborated.
Encountered problems processing file: syn-prob2.cmd
```

If synthesis is successful there will be an area report, it appears under the text “Area Report Below.” Here is a sample:

```
 *** Area Report Below **

============================================================
  Generated by:          Encounter(R) RTL Compiler RC10.1.306 - v10.10-s357_1
  Generated on:          Oct 21 2013  01:10:46 pm
  Module:                czc16_prob1
  Technology library:    osu035_stdcells
  Operating conditions:  typical (balanced_tree)
  Wireload mode:         enclosed
  Area mode:             timing library
============================================================

  Instance   Cells  Cell Area  Net Area   Wireload
-------------------------------------------------------
czc16_prob1     50       5240         0     <none> (D)     <---- AREA OF ENTIRE DESIGN
  c3             8        832         0     <none> (D)
  c2             8        832         0     <none> (D)
  c1             8        832         0     <none> (D)
  c0             8        832         0     <none> (D)
```

The total area is the first line listed on the table. For the example above, 5240 includes the area of the four `czc4` modules instantiated in `czc16_prob1`.

The timing appears under “Timing Report Below,” here is a sample:

```
 *** Timing Report Below **

Warning : Possible timing problems have been detected in this design. [TIM-11]
        : The design is 'czc16_prob1'.
        : Use 'report timing -lint' for more information.
============================================================
  Generated by:          Encounter(R) RTL Compiler RC10.1.306 - v10.10-s357_1
  Generated on:          Oct 21 2013  01:21:24 pm
  Module:                czc16_prob1
  Technology library:    osu035_stdcells
  Operating conditions:  typical (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

 Pin       Type     Fanout Load Slew Delay Arrival
                           (fF) (ps)  (ps)   (ps)
-----------------------------------------------------
a[7]      in port        2 48.3    0    +0       0 R
c1/a[3]
  g86/A                                 +0       0
  g86/Y   NOR2X1         2 40.5  138  +125     125 F
  g2/B                                  +0     125
  g2/Y    AND2X1         2 39.9  115  +211     336 F
c1/lz[2]
g256/A                                  +0     336
g256/Y    NAND2X1        3 72.7  253  +214     550 R
g252/B                                  +0     550
g252/Y    NOR2X1         3 81.2  241  +235     785 F
g244/B                                  +0     785
g244/Y    AOI22X1        1 18.0  138  +130     915 R
g242/B                                  +0     915
g242/Y    NAND2X1        1  0.0   54   +28     943 F
lz[0]     out port                      +0     943 F   <--- DELAY OF ENTIRE MODULE
-----------------------------------------------------
Timing slack :  UNCONSTRAINED
Start-point  : a[7]
End-point    : lz[0]
```

The table above shows timing along the critical path, the delay for the circuit is the *last* entry in the table, 943 ps. Ignore the warning about timing problems, it does not apply to isolated combinational logic.

To make it easier to move between windows please take time to familiarize yourself with some basic Emacs commands:

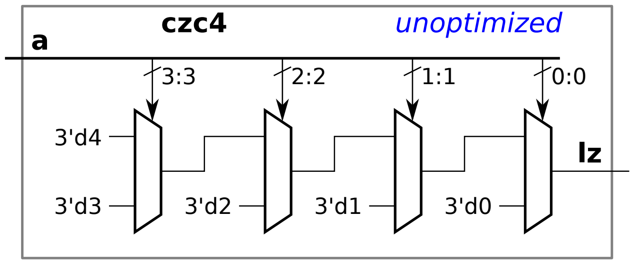
- Switching between buffers. (Use the Buffers menu.)
- Removing a *Window* (what Emacs refers to as a window, other programs refer to as a pane). (From the File menu select Remove Other Windows.)

Problem 1: The *consecutive zero count*, *CZC*, or zero count for short, of an n-bit integer is the number of consecutive zeros in the number's base 2 representation, counting starting at the least significant bit. For example the CZC of 1011000_2 is 3, the CZC of 1011_2 is 0, and the CZC of 8-bit integer 0_2 is 8.

Module `czc4`, in `hw04.v`, computes the zero count of a four-bit integer. Input `a` is the four-bit integer, output `lz` is the zero count.

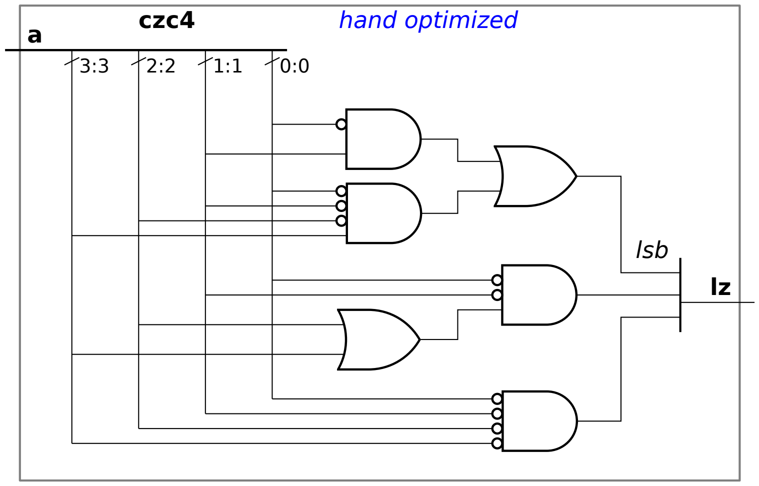
(a) Draw the hardware that might be synthesized for this module when no optimization is done. Do this yourself, *do not* just look in the file `hw04-prob1-syn-inferred.v` which will show what the Cadence Encounter program really did infer.

Solution appears below. The logic below would be the first step in the synthesis process, we expect that the synthesis program would optimize this logic to get something less expensive, which is the point of part (b).



(b) Draw a schematic of hand optimized code. Again, don't look at the synthesis program output. *Hint: Consider each bit of* `lz` *separately. For example, the LSB of* `lz` *will be* `1` *when* `a[0]` *is* `0` *and* `a[1]` *is* `1` *or when* `a[0]` *to* `a[2]` *are zero and* `a[3]` *is* `1`*. Draw the logic using this reasoning.*

Solution appears below. The logic was obtained using the reasoning in the hint above. We should expect the synthesis program to do at least as good as this logic. That is, if the synthesis program, after optimization, used four multiplexors then there would be something wrong with the synthesis program or with the way we wrote the Verilog.



(c) Notice that module `czc16_prob1` is empty. That is, the ports are declared but they are not connected to anything. As the name and comments suggest, the output of the module, `lz`, should be the zero count of its 16-bit input, `a`. Using four `czc4` modules and additional synthesizable implicit or explicit structural code complete `czc16_prob1` so that it computes the zero count.

The module can be tested for correct Verilog selecting Simulate from the EE 3755 menu. Assuming the Verilog compiled and ran, near the end a line will start with "Finished with 1000 tests". The number to the left of "Prob-1" will be zero if the module works correctly. If it's non-zero the first five incorrect results will be displayed further above.

Module `czc16_prob1` has convenient check boxes listing things that need to be done or taken into account. Please review them after finishing.

The solution appears below. Notice that the module splits the output bits of the `czc4` instances into a most-significant part, such as `z0` and the lower 2, such as `l0`. This simplifies the hardware for detecting, for example, whether the output of the module is less than 4 (just check if `z0` is zero).

```
// SOLUTION
module czc16_prob1(lz, a);
   input wire [15:0] a;   output wire [4:0] lz;

   wire   z0, z1, z2;
   wire [1:0] L0, L1, L2;
   wire [2:0] L3;

   czc4 c0( {z0,L0}, a[3:0]   );
   czc4 c1( {z1,L1}, a[7:4]   );
   czc4 c2( {z2,L2}, a[11:8]  );
   czc4 c3( L3,      a[15:12] );

   assign lz = !z0 ? L0 : !z1 ? 4 + L1 : !z2 ? 8 + L2 : 12 + L3;
endmodule
```

Problem 2: Module `czc16_prob2` is also supposed to compute the zero count of an integer and is also empty. Complete the module using synthesizable *behavioral* code.

As with the first problem, make sure that the code produces the correct results and synthesizes. Use the checkboxes to make sure that nothing is omitted.

The solution appears below.

Some solutions included an `if/else` chain that tested for every possible value of `a`. Points were deducted for this because such code is not just tedious to write, it is also highly prone to errors.

```
// SOLUTION
module czc16_prob2(lz, a);
   input wire [15:0] a;  output reg [4:0] lz;

   integer i;
   always @ ( a ) begin
      lz = 16;
      for ( i=0; i<16; i=i+1 )
        if ( a[i] == 1 && lz == 16 )
          lz = i;
   end
endmodule
```

Problem 3: In this problem your delay expectations for the modules from the two previous problems will be compared with the timing provided by the synthesis program.

(*a*) Show the cost and delay (timing) of each design provided by the synthesis program.

For Problem 1, the cost is 5240 and the delay is 943 ps. For Problem 2, the cost is 4148 and the delay is 970 ps.

Of course, the costs and delays depend on the solutions to problems 1 and 2 and so these aren't the one-and-only correct answers to this problem.

(*b*) Which of the two designs do you think should be faster? Why?

In the solution to Problem 1 the 16 bits are divided into 4 sets of 4, and these 4 values are later combined. The critical path will go through a module that looks at 4 bits (one of the `czc4` instances) and the assign statement that examines just 4 bits.

For Problem 2, the Verilog tests each bit of `a` sequentially. This would initially be inferred as sixteen sets of multiplexors, with a path length of 16. If there was no optimization the solution for Problem 2 would be much slower.

The synthesis program has to figure out that the 16 stages can be transformed into logic that can test bits of `a` in parallel. A reasonable guess is that the synthesis program would be able to figure this out and so the delay of the two would be about the same.

The answer above would be similar for solutions to Problem 2 that used a long `if/else` chain instead of a loop.

(*c*) If your answer above does not agree with the synthesis program provide a possible answer (or maybe "fix" your Verilog).

The delay reported for Problem 2 is longer than that of Problem 1, suggesting that the synthesis program was not able to fully optimize the logic.