18-240: Structure and Design of Digital Systems

Electrical & Computer ENGINEERING

HW2 [9 problems, 64 points]

Covers lectures L4 – L5 *Due: 18 September 2023*

Homework sets are due at 5:00PM on the due date. Upload your answers, as described below, to Gradescope by then. No late homework will be accepted. Remember, we let you drop two homework assignments over the semester.

To hand in your homework, create a PDF either by scanning your paper work, or by generating it in a software tool to start with (e.g., a word processor, LaTeX). Ensure your PDF is readable, your work legible and rotated correctly. Upload your work to Gradescope and follow the Gradescope instructions carefully to ensure your work is graded properly. Points will be deducted if you do not follow directions, for instance, by not assigning problems to pages (a five point penalty).

All other files must be submitted by running our handin script, which generates *a second PDF file* of your source code. If you have never run the handin script for *any* 18-240 assignment before, please read the *Overview of handin240* page on the course Wiki. The setup needs to be done only one time for the semester. Make sure you have a copy of all of the files that you need to submit in the same directory in your AFS space. Once you are ready to submit, run:

```
$ handin240 hw2
```

Note that everything that you type in should be lowercase. This script will do three things: (1) it will check to see if all of the required files are present, (2) it will automatically copy all of the files required for the homework to a handin directory where the TAs can run the code, and (3) it will generate a PDF of all of the required files called **`HW2_code.pdf`**. *After the script finishes, you must upload this generated PDF to Gradescope.*

Remember: **all files need to be generated and uploaded before the deadline**, so leave ample time for submission!

Discussions about homework in small groups are encouraged — think of this as giving hints, not solutions, to each other. However, homework must be written up individually (no copying is allowed). If you discussed your homework solutions with someone else, either as the giver or receiver of information, your write-up must explicitly identify the individuals and the manner information was shared.

If you use an AI assistant (ChatGPT, or others) for help on any of these problems, you must ensure that your answer is completely your work. Do not simply copy-n-paste any part of a ChatGPT conversation into your answer. And you must cite the AI assistant, with a thorough description of what help you received. For example, *"Conversation with ChatGPT 3.5 consisting of approximately 12 prompts asking for a thorough understanding of how to do Boolean proofs."*

You must show details of your work. There is no credit for just writing down an answer.

Drill Problems [28 points]

Drill problems are graded leniently based on your approach and effort; and not entirely on correctness. That means it is possible for you to have a perfect score on a problem whose answer is actually incorrect. Please check your work with the published solutions to verify correctness.

1. [12 points, Lecture 5] Write a SystemVerilog module and testbench for each of the following components.

   You've seen some of these in lectures and the LDVUS readings. Copying from those sources is fine. In each case, name your module and inputs *exactly* as shown in the following drawings. Capitalization follows the coding standard. Name your testbench modules according to the class coding standards. Follow the directions posted on Canvas to have VCS simulate only a single testbench at a time[1].

   (a) 3:8 decoder with enable

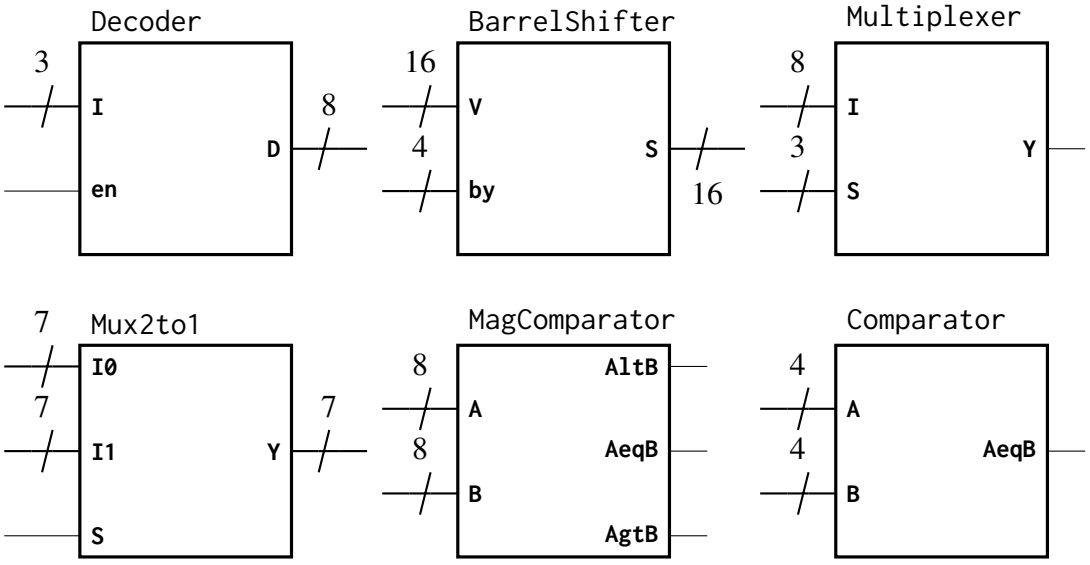
   (b) 16-bit combinational barrel shifter (shifts to the left)

   (c) 8:1 multiplexer

   (d) 2:1 multiplexer (for 7-bit values)

   (e) 8-bit magnitude comparator

   (f) 4-bit comparator (non-magnitude variety)



▷ Submit your modules in a file named `library.sv`. Yes, this is an unusual name for a homework. You will be using this library of components throughout the course.

▷ Submit your testbenches in `library_tests.sv`

[1] Located in Supplemental Handouts → Simulating a Single Module form a File with VCS.

2. [6 points, Lecture 4] Sketch a schematic of the circuit described by the following SystemVerilog code. Do not simplify:

```
module hw2prob2
  (output logic e, f,
   input  logic a, b, c);

  assign e = a & (b | c);
  assign f = ~ (a | b);
```

3. [4 points, Lecture 4] Using *procedural* SystemVerilog, write a module that computes a four-input XOR function. The module has a 4-bit input **a**, and output **y**.

   Use the following module header.

```
module hw2prob3
  (output logic y,
   input  logic [3:0] a);
```

   ▷ **Submit this as a file named `hw2prob3.sv`.**

4. [6 points, Lecture 4] Using *procedural* SystemVerilog, write a module called **`Minority`**. The module's output is a one only if more than half of its inputs are zero.

   Use the following module header.

```
module Minority
  (output logic y,
   input  logic a, b, c, d, e);
```

   ▷ **Submit this as a file named `hw2prob4.sv`.**

Non-Drill Problems [36 points]

5. [8 points, Lecture 5] Use an 8:1 multiplexer module to implement $F(A,B,C,D) = A'B'CD + A'B + ABD + A(B'CD' + C'D)$.

   First, you will have to design a multiplexer module (and write a testbench to check it). Design it to have this module header, using any SystemVerilog techniques you like:

```
module multiplexer8
  (input  logic       I0, I1, I2, I3, I4, I5, I6, I7,
   input  logic [2:0] S,
   output logic       Y);
```

   Then, design the module to use the multiplexer. In this module you should not have any logic, just structural instantiations of the inverter and multiplexer.

   Use the following module header:

```
module hw2prob5
  (input  logic A, B, C, D,
   output logic F);
```

   The testbench for this module (**hw2prob5_test**) should exhaustively test the module and print an error message only if the module is incorrect (i.e. no **$monitor** statement).

   ▷ Submit your code as hw2prob5.sv. This file should include the testbench modules multiplexer8_test and hw2prob5_test).

6. [12 points, Lecture 5] Let's test Shannon's Expansion Theorem. Here is a 6-input function:

$$G(a,b,c,d,e,f) = (a'+c'+e'+f')\cdot(a'+b'+d+e+f')\cdot(c'+d'+e'+f')\cdot(c'+d+e+f)\cdot (a+b'+c'+d'+e)\cdot(a+b+c'+f)\cdot(a'+b+c+d'+e'+f')\cdot (a'+b'+e'+f')\cdot((a\oplus d)'+b+e+f)\cdot(a'+b+e'+f)\cdot (b'+c+d+e)\cdot(a+b'+c+d'+e')\cdot(a+b+d+e')$$

   Find all the cofactors where c and f are factored out.

   Implement this module with all 5 (the original and 4 cofactors) expressions:

```
module hw2prob6
  (input  logic a, b, c, d, e, f,
   output logic g, g00, g01, g10, g11); //g01 is G(c=0,f=1, .*)
```

Finally, create a testbench that exhaustively loops through all input combinations for a-f. For each combination, check that the Expansion Theorem properly can be used to combine the cofactors into an expression that matches G. If not, output a string starting with "Ooops!".

▷ **Turn in `hw2prob6.sv` as well as your minimization work (kmaps, etc).**

7. [4 points, Lecture 4] Write a SystemVerilog module that receives as input a year number, **`year`**, and produces an output, **`leap4`**, that is one if the year is evenly divisible by 4 (i.e., the remainder is 0), and zero otherwise. Assume that valid years are zero through 2040, inclusive.

   Hint: Think what it means to divide by a power of two. Don't use the divide operator.

   Include a testbench that checks all of the years between 2000 and 2021, printing out **`leap4`** for each.

   Use the following module header.

```
module hw2prob7
  (output logic        leap4,
   input  logic [10:0] year);
```

   ▷ **Submit this as a file named `hw2prob7.sv`.**

8. [4 points, Lecture 4] But, wait a minute! If you know about leap years, you probably know that the "divide by four" definition is not quite correct. A year is a leap year if:

   (a) The year is evenly divisible by four

   (b) except in the case where it is evenly divisible by 100 and thus is not a leap year,

   (c) except in the case where it is evenly divisible by 400 and thus is a leap year.

   Your answer to the last problem only followed rule #1. Implement a better version, using the following header, to correctly determine if a year is, in fact, a leap year. You may use your module from the last problem to implement this newer version, but if you do, please include a copy in your file submission.

   Hint: Don't even think about dividing by 100 or 400. In this case, there must be a better way.

   Yes, you should include a testbench that tests some interesting values.

   Use the following module header.

```
module hw2prob8
  (output logic        leap4_100_400,
   input  logic [10:0] year);
```

   ▷ **Submit this as a file named `hw2prob8.sv`.**

9. [8 points, Lecture 4] After all this talk about not dividing in the previous problems, let's go ahead and build a very simple divider.

   Develop a minimized Boolean implementation of a 2-bit divider. The system has 2-bit inputs **AB** (dividend) and **CD** (divisor), and generates 2-bit outputs, **EF** (quotient), and **GH** (remainder).

   (a) Draw the truth table for **E**, **F**, **G**, and **H**.

   (b) Minimize the four output functions.

   (c) Using structural SystemVerilog (hmm, what does *structural* mean?), write a module that describes functions **E**, **F**, **G**, and **H** from part A. Use this module header.

```
module Divider_C
  (output logic e, f, g, h,
   input  logic a, b, c, d);
```

   (d) Using procedural SystemVerilog (hmm, what does *procedural* mean?), write a module that describes functions **E**, **F**, **G**, and **H** from part A. Use this module header.

```
module Divider_D
  (output logic e, f, g, h,
   input  logic a, b, c, d);
```

   ▷ **Submit your code in a file named `hw2prob9.sv`.**