18-240: Structure and Design of Digital Systems

HW1 [12 problems, 64 points]

Covers lectures L2 – L3 *Due: 11 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 hw1
```

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 **`HW1_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.

For drawing and labeling K-maps, follow the *A Guide to K-Map Formats* document on Canvas.

Drill Problems [32 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.

0. [0 points] If you have not done so already, follow all of the directions in the *Setting Up the Homework Handin Script* document on Canvas to set up the script.

1. [3 points, Lecture 2] Draw the schematic diagram of this circuit. Do not simplify the circuit, just draw it as described below.

```
module hw1prob1
  (input  logic a, b, c, d,
   output logic f, g);

  logic h, i, j, k, l;
  not #3 n0(i, b);
  not    n1(h, d);
  and #2 a0(j, a, b, h);
  or  #1 o0(k, b, c, i);
  xor #2 x0(l, i, b);
  nor #2 n3(f, i, j, k);
  and #4 a1(g, d, c, l);

endmodule : hw1prob1
```

2. [6 points, Lecture 3] Write the canonical POS and SOP form for **`f`** and **`g`** from problem 1. Then, use a K-map to simplify **`f`** and **`g`**. Comment on the differences between the three methods of describing the same functionality (i.e. SystemVerilog, POS/SOP equation, Karnaugh Map).

3. [3 points, Lecture 3] For the truth table shown, draw the K-map for functions F, G and H. Use the K-maps to write the simplified Boolean expression for each function.

A	B	C	F	G	H
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	0	1
1	1	1	0	1	0

4. [3 points, Lecture 2] Write a SystemVerilog module, using structural style, to model functions F, G and H from the previous problem.

   Here is your module header:

```
module hw1prob4
  (input  logic A, B, C,
   output logic F, G, H);
```

   ▷ **Submit your code in a file named `hw1prob4.sv`.**

5. [4 points, Lecture 3] Draw the K-map for the function:

$$F(A,B,C,D,E) = \Pi M(0,4,5,6,7,10,17,19,20,21,22,24,31) \cdot D(1,23,26,27,29)$$

   Use the K-map to write the minimized POS Boolean expression.

6. [6 points, Lecture 3] Draw the K-map for the function:

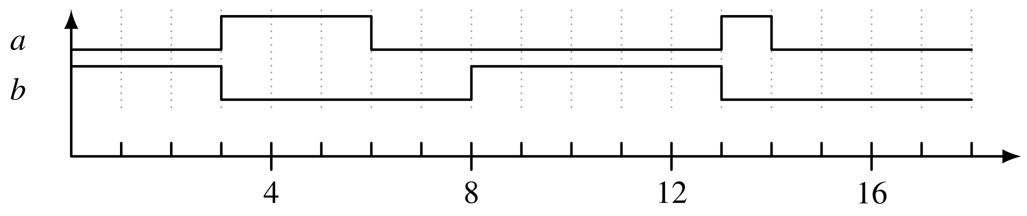
$$\begin{aligned} F(A,B,C,D,E,F) \quad = \quad & \Sigma m(0,2,3,5,9,13,14,19,21,22,23,26, \\ & 29,32,34,35,40,41,43,45,48,49,51,53,56, \\ & 58,59,61) + d(16,37,42,50,57) \end{aligned}$$

   Use the K-map to write the simplified SOP Boolean expression.

7. [2 points, Lecture 3] For the function F in problem 6, how many gates are needed to implement the minterm canonical form (don't count inverters)? How many to implement the reduced form? How many literals are there in each of the implementations?

8. [5 points, Lecture 2] Write an **`initial`** block that will generate the waveform below. Do not include a **`$monitor`** statement. Put this **`initial`** block in a module (with outputs **`a`** and **`b`**) named **`hw1prob8`**.

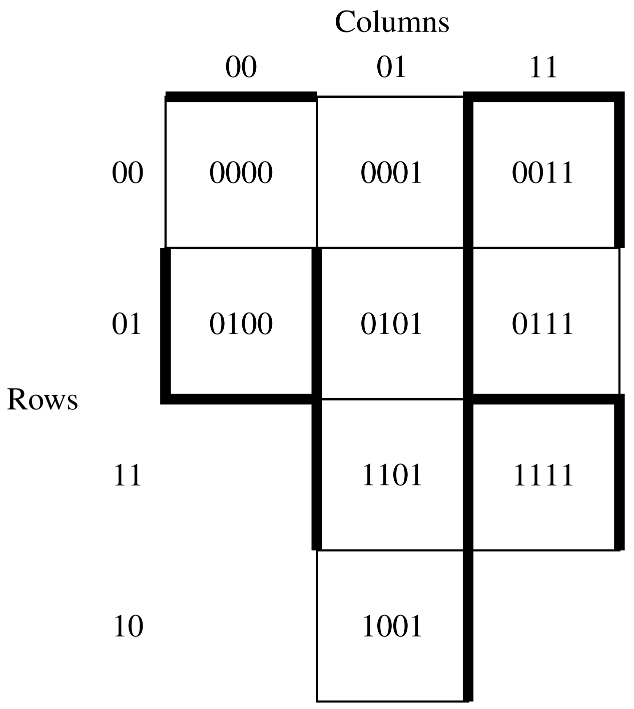
   ▷ **Submit your work as a file named `hw1prob8.sv`.**

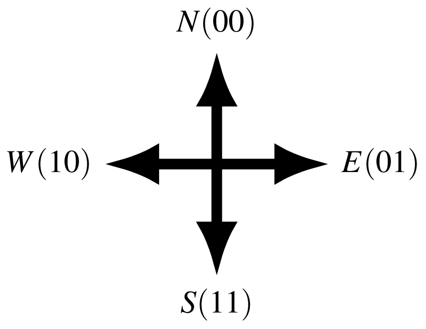


Non-Drill Problems [32 points]

9. [12 points, Lecture 3] **Good times at the Hamster Training Academy**

   You just can't get good hamsters for psychology experiments anymore, so CMU-H has been setup to educate the best — they're even building a new building for it! Assume a hamster can be placed in any one of the squares in the maze below, labeled by the row input **R** (2 bits) followed by the column input **C** (2 bits). (Darker lines are walls that the hamster can't go through.) We need a logic design that will take the four bits **RC** as input and output the direction the hamster should face to either step out of the maze, or point him/her (what, you thought we could tell the difference?) to the nearest exit. The direction to point will be encoded as **AB** = **00**, **11**, **01**, **10** (north, south, east, or west respectively). Write the canonical SOP form of both functions `A(R1, R0, C1, C0)` and `B(R1, R0, C1, C0)` and reduce in K-maps. Write the minimized SOP equations from the K-maps.





10. [2 points, Lecture 3] What is the minimized equation for the function described in this K-map?

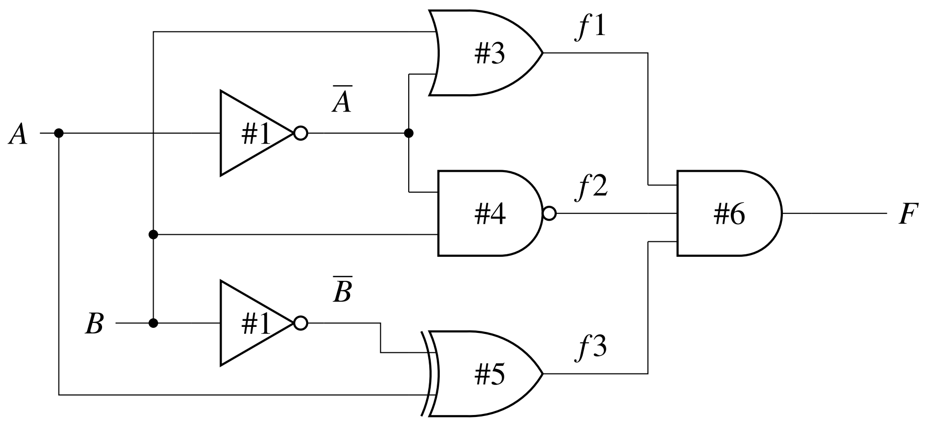
AB \ CD	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

11. [6 points, Lecture 2] In HW0 you proved that AND does distribute over XOR (i.e. $A \cdot (B \oplus C) = (AB) \oplus (AC)$) . Write a SystemVerilog testbench to show this fact. Your testbench should iterate over all possible values for `A`, `B` and `C`, with a `$monitor` statement showing `A`, `B` and `C` as well as both sides of the expression. Further, your testbench should use an if statement to test whether the left side actually does equal the right side and `$display`ing an error in the case where they aren't actually equal (yes, I realize this statement won't ever get printed, as you have proved the fact). You should, when developing your testbench, purposely insert an error in the equations to ensure the `$display`ed statement would work if there was an error.

    Your module should be named `hw1prob11`. As with any SystemVerilog code you submit, you should simulate it to ensure it works properly. Compilation errors in your code will result in zero points being awarded for the problem.

    ▷ **Submit your code in a file named `hw1prob11.sv`.**

12. [12 points, Lecture 2] Given the following logic diagram and initial block that will drive its inputs, list every event on the simulator's "next event" list (i.e. the to-do list) from the beginning of time until the end of the simulation using a table such as below. Each simulation cycle should have a separate row. Refer to the last few slides of Lecture 02 for guidance about what the simulator is doing.



```
initial begin
   A = 1;
   B = 0;
   #13 A = 0;
   #13 B = 1;
   #13 $finish;
 end
```

Time	Next Event List	Values of Variables							
		A	B	$\overline{A}$	$\overline{B}$	$f1$	$f2$	$f3$	F
start	initial@0	X	X	X	X	X	X	X	X