18-240: Structure and Design of Digital Systems

Electrical & Computer ENGINEERING

HW3 Solutions [12 problems, 64 points]

Covers lectures L06 – L07 *Due: 25 September 2023*

Drill Problems [32 points]

1. [3 points, Lecture 6] Pick any six decimal numbers between -127 and 127 (inclusive). Convert them to 8-bit signed magnitude and 8-bit two's complement format. Show your work.

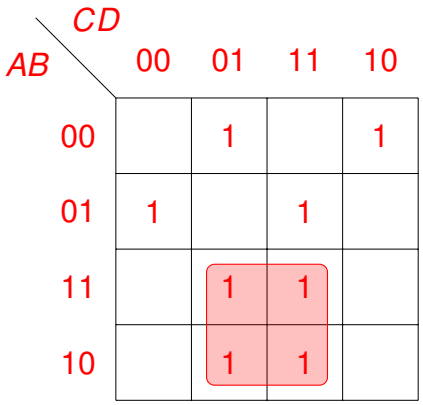
   Details depend on the numbers you picked, obviously. If you chose zero, then you should have two signed magnitude versions (`1000_0000` and `0000_0000`).

2. [3 points, Lecture 6] Pick any six 8-bit binary numbers (other than those in Problem 1). Convert each into decimal three times; once for an unsigned binary format, a second time for a signed magnitude format and a third time for two's-complement format. Show your work.

   Details depend on the numbers you picked, again. Again, zero is interesting.

3. [5 points, Lecture 7] The 4-input LUT on an FPGA has been configured such that TT0-TT15 = {0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1}. What will be the minimized function of the output for inputs {**A**, **B**, **C**, **D**} where **A** is the most significant?

   Those bits given correspond to the output column on the truth table for the function. Were we to throw it into a Karnaugh map, we would see something like that to the right.
   The stuff on the top half is clearly an XOR checkerboard, which will only occur when **A** is zero. The rest can be covered with a single product term.
   $F(A,B,C,D) = \overline{A}(B \oplus C \oplus D) + A \cdot D$



4. [3 points, Lecture 7] What is wrong with this code for a 2-4 decoder with enable? Explain.

   The enable is active low. Therefore, the last line of the **`always_comb`** block should not have the $\sim$ on the **`en_L`** signal.

   The lack of a $\sim$ on an active low signal indicates *not active* so this last line should be read as:

   If enable is not active, set the outputs all to zero.

   but with the $\sim$, it reads:

   If enable is active, set the outputs all to zero.

5. [6 points, Lecture 6] Write a SystemVerilog module that calculates the sum of two BCD numbers and a **`carryIn`**.

   `hw3prob5.sv` is posted on Canvas.

6. [6 points, Lecture 6] In Lecture 7, you learned that a carry-lookahead adder relies upon the recurrence equation $C_{i+1} = G_i + C_i P_i$.

   Expand this equation to get an expression for C_4 in terms of Ps and Gs and C_0.

   Use that equation to provide a count of how many gates of each type (and with each number of inputs) is required to build the entire carry-lookahead adder (for instance, C_2 requires one 3-input OR gate, one 3-input AND gate, ...).

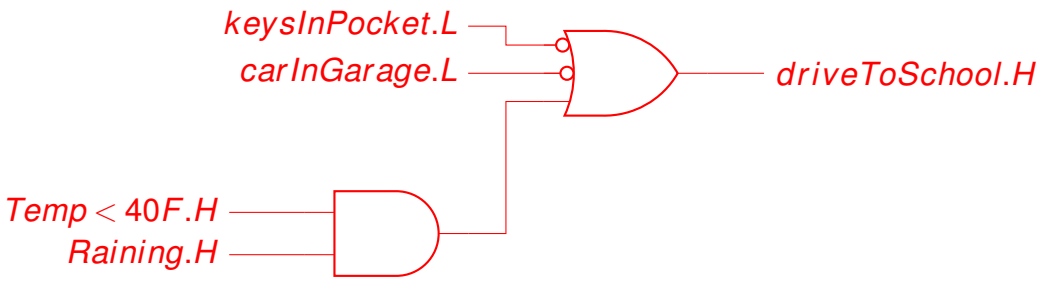
$$C_4 = G_4 + C_3 \cdot P_4$$
$$C_4 = G_4 + (G_3 + C_2 \cdot P_3) \cdot P_4$$
$$C_4 = G_4 + (G_3 + (G_2 + C_1 \cdot P_2) \cdot P_3) \cdot P_4$$
$$C_4 = G_4 + (G_3 + (G_2 + (G_1 + C_0 \cdot P_1) \cdot P_2) \cdot P_3) \cdot P_4$$
$$C_4 = G_4 + G_3 \cdot P_4 + G_2 \cdot P_3 \cdot P_4 + G_1 \cdot P_2 \cdot P_3 \cdot P_4 + C_0 \cdot P_1 \cdot P_2 \cdot P_3 \cdot P_4$$

   This looks like one 5-input OR gate, one 2-input AND gate, one 3-input AND gate, one 4-input AND gate and one 5-input AND gate.

   However, you also need to create all those P and Gs. Each P is a 2-input XOR gate (quantity 4 – you can reuse the P4's XOR gate for each instance of P_4 in the equation). Each G is a 2-input AND gate (quantity 4).

7. [6 points, Lecture 7] Lecture 7, slide 14 showed a circuit that I could use to determine if I could drive to school each day by generating the signal **`driveToSchool.L`**. It relied upon input signals **`keysInPocket.H`**, **`carInGarage.H`**, **`Temp<40F.L`**, and **`Raining.L`**.

   If I push the bubble from the output through the NAND gate, it becomes an OR gate with bubbles on the inputs. Two of those bubbles match up with the now-active-low input signals **`keysInPocket.L`** and **`carInGarage.L`**. The other bubble can get pushed through that OR-gate-with-bubbles-on-the-inputs to make a simple AND gate that takes the now-active-high input signals.



Non-Drill Problems [32 points]

8. [4 points, Lecture 7] If you attach a multi-meter to the output of a tri-state driver, what voltage will it measure when the driver is not enabled? Explain.

   Difficult to tell, as this is exactly the same scenario as touching a multi-meter to a disconnected wire sitting on a lab bench. The wire isn't connected to anything, but may have a voltage associated with it. Similarly, the output of the tri-state driver isn't connected to anything – it is *electrically disconnected*.

9. [4 points, Lecture 7] If you attach a multi-meter to the the output of a tri-state driver on a a circuit board with lots and lots of chips and connections, what voltage will it measure if the driver is not enabled? Assume the circuit board is powered and functioning normally. Explain. (Note: this question is very similar to the one above. The fact that I'm asking it again must mean something is different about this scenario.)

   It may measure the voltage associated with a logic 0 or logic 1 (which might be, but isn't necessarily 0V and 5V). This case is different from problem 7 in that the same wire that you are measuring has other tri-state drivers connected to it. It is possible, perhaps likely (depending on the mission of the wire), that one of those other tri-state drivers is currently driving a logic zero or one on the wire.

10. [6 points, Lecture 6] The old mechanical cash registers were more complicated than you might think. They did their arithmetic in decimal. For the most part, you just added items (the costs of the items). But if you had to subtract off a discount, it used tens complement arithmetic to do the subtraction. Given a cash register whose highest sum total is $999.99, and I wanted to subtract off a discount totaling $23.37 from the current sum, what value would the machine add to the sum? i.e., what is -23.37? Just writing down the answer isn't good enough here. You must answer in terms of "complement arithmetic." e.g., use analogies to 2s complement arithmetic to explain how you calculated the answer.

    976.63. The point of this question is that when doing fixed width arithmetic, all arithmetic operations are modulo (in this case) 1000.00. Thus, we never see a value of 1000.00 or greater. The radix complement of an n-digit number is obtained by subtracting it from 10^n. In this case, $1000.00 - 23.37 = 976.63$. Also note, $976.63 + 23.37 = 0$. It follows that 976.63 can be thought of as -23.37 because when you add them together you get 0 (mod 1000.00). This is called 10s complement arithmetic.

11. [10 points, Lecture 6] Suppose we are working with 6-bit two's-complement numbers. Consider the two arithmetic operations below:

	1	1	0	1	0	0		0	1	1	0	1	0
+	a_5	a_4	a_3	a_2	a_1	a_0	-	b_5	b_4	b_3	b_2	b_1	b_0

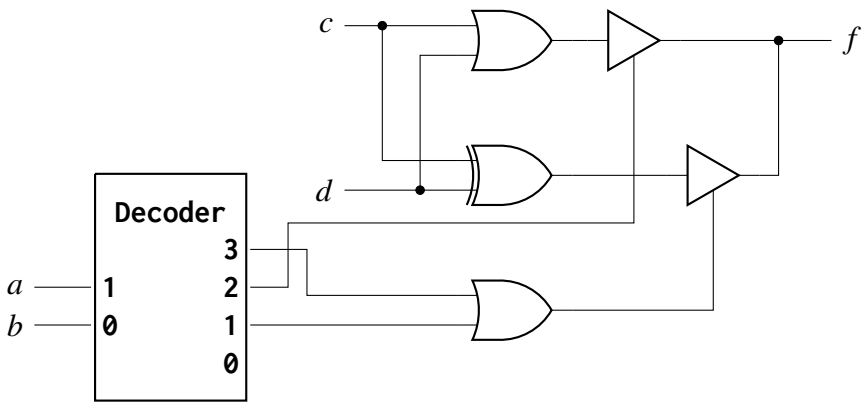
    The largest value for $a_5a_4a_3a_2a_1a_0$ is: **011111** (31_{10}). Since the first operand is negative, we can safely add the most positive number and be guaranteed not to have overflow.

    The smallest value for $a_5a_4a_3a_2a_1a_0$ is: **101100** (-20_{10}). The lowest sum we can have is -32 so the smallest value is $-32 + -12 = -20$.

    The largest value for $b_5b_4b_3b_2b_1b_0$ is: **011111** (31_{10}). Since the first operand is positive, we can safely subtract the most positive number.

    The smallest value for $b_5b_4b_3b_2b_1b_0$ is: **111011** (-5_{10}). The great difference we can have is 31 so the smallest value is $31 - 26 = -5$.

12. [8 points, Lecture 7] What is the truth table for $F(A,B,C,D)$ from the circuit below?



The two tri-state drivers both are connected to **f**, which works because they are never both enabled at the same time. This guarantee is ensured via the use of the decoder.

The decoder converts the binary number **{a, b}** into a one-hot code. When **{a, b} = 2'b00**, neither driver is enabled, so **f** will be a **z**. When **{a, b}** is a one or three, the lower driver will be enabled and **f** will be connected to the output of the XOR gate. Likewise, when **{a, b}** is two, the OR gate will drive **f**.

Thus, this truth table:

a	b	c	d	f
0	0	x	x	z
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0