CS226 Practice Problem Set 1 (Spring 2016)

Date posted: Jan 20, 2017 **Expected Solving Time: 1 hour**

- *Be brief, complete and stick to what has been asked.*
- *Unless asked for explicitly, you may cite results/proofs covered in class without reproducing them.*
- *If you need to make any assumptions, state them clearly.*
- ***These are ungraded practice questions. You are strongly encouraged to solve these on your own to ensure you understand the material being taught in class.***
- ***Mutual discussion is allowed, but copying is not. Please read the guidelines on the course webpage if you don't understand the distinction between the two.***

1. A student claims that if you give her sufficiently many half-adders and no other logic gates or block, she can implement any Boolean function. Either prove the student correct, or give a Boolean function that cannot be designed using only half-adders. In the latter case, you must explain why the function cannot be designed using only half-adders.

2. Given a list of 6 numbers, each of which can be represented using 4 bits, we wish to design a circuit with basic 2-input logic gates (AND, OR, XOR, NAND, NOR, XNOR) and 1-input NOT gates, that gives the maximum of the 6 numbers. Thus, the desired circuit should have 24 binary inputs and generate 4 binary outputs. Assume that each 2-input gate has a delay of 2Δ, and each NOT gate has a delay of Δ. Design your circuit such that the overall delay for computing the maximum is as small as possible. Would your design change if you were trying to minimize the number of gates, instead of the delay?

3. *[Adapted from 2016 endsem]* Usually, when we draw Karnaugh maps (henceforth called K-maps), we fill in the entries with 1, 0 or $-$ (for don't care). In this question, we want to consider symbolic Karnaugh maps, where we can fill in the entries with symbolic variables or even with other Boolean functions, in addition to using 1, 0 and $-$.

   Consider the symbolic K-map shown below, where F and G are Boolean variables, and F' and G' represent their respecctive Boolean complements.

$\frac{x_1,x_2\rightarrow}{x_3,x_4\downarrow}$	**00**	**01**	**11**	**10**
00	F	G	F'	1
01	1	G	1	G'
11	F'	F	G	F'
10	G	F'	0	F

   Clearly, the above K-map gives rise to different Boolean functions of x_1, x_2, x_3 and x_4 for different combinations of Boolean values of F and G. In other words, the K-map represents a Boolean function of x_1, x_2, x_3, x_4, F and G. Let us call this function $\varphi(x_1, x_2, x_3, x_4, F, G)$.

   (a) Suppose F and G are functions of $\{x_1, \ldots x_6\}$. Specifically, suppose $F = (x_1 + x_4) \cdot (x_5 + x_6)$ and $G = (x_1 \oplus x_3 \oplus x_5 \oplus x_6)$. What is the Boolean function φ represented by the symbolic K-map given above?

   (b) Referring to the previous sub-question, how many different pairs of F and G functions (each being a function of $\{x_1, \ldots x_6\}$) can be used to produce the same Boolean function φ as in the previous sub-question? Give reasons for your answer.

4. Give as "simple" sum-of-product (SOP) and product-of-sum (POS) Boolean functions as you can from the following K-maps.

   An example of a Boolean function in sum-of-products form is $f = a.b.c + b'.d.e' + e'.a'$. An example of one in product-of-sums form is $g = (a+b'+c).(a'+c').(b'+d'+e)$. For purposes of this question, we will measure the "simplicity" of a Boolean function in POS or SOP by the number of two-input AND/OR gates and number of inverters needed to implement the function. For exapmle, to implement the example f in SOP form given above, we need two 2-input OR gates, five 2-input AND gates and three inverters. Hence the "simplicity" metric of the SOP form is $2 + 5 + 3 = 10$. Similarly, the "simplicity" metric of g in POS form given above is two 2-input AND gates, five 2-input OR gates and four inverters, i.e. $2+5+4 = 11$. You must choose the "simplest" (i.e. lowest simplicity score) of the SOP and POS forms for each each function represented by the K-maps below, where "-" denotes don't-care.

   (a)

X→ / Y↓	**0**	**1**
0	1	1
1	0	1

   (b)

X,Y→ / Z↓	**00**	**01**	**11**	**10**
0	1	1	0	1
1	1	0	0	1

   (c)

X,Y→ / Z,W↓	**00**	**01**	**11**	**10**
00	1	1	1	1
01	1	1	1	1
11	1	1	0	1
10	1	1	1	1

   (d)

X,Y→ / Z,W↓	**00**	**01**	**11**	**10**
00	1	1	1	1
01	0	-	-	0
11	0	1	-	1
10	0	1	1	1

5. *[Adapted from Practice Problem Set 2 of 2016]* A long message (stream of bits) is being sent from a transmitter to a receiver over a channel. The length of the message is not known a priori. The received message is said to be *valid* if the number of 1's in it is one more than a multiple of 3, *and* if the message ends with a sequence of five consecutive 1's. Thus, a valid message is 11010110011111 (has ten 1's and ends with five 1's), while 110101000011111 (has nine 1's and ends with five 1's) and 110101010100 (doesn't end with five 1's) are not valid messages. You are required to design an end-of-valid-message (EOVM) indicator circuit to be used by the receiver. The EOVM indicator has a 1-bit input In through which the bit-stream comes in, one bit per clock cyclle. It produces a 1-bit output Out in every clock cycle. The output is required to become 1 as soon as the end of a valid message is detected; otherwise it stays at 0.

   (a) Write a C-style pseudocode for solving the above problem, choose a datapath containing the arithmetic units studied in class, and design a circuit for implementing the EOVM indicator. For this subquestion, you may assume that only a single valid message is being transmitted. So, once the EOVM indicator outputs 1, you can choose to keep it at 1 until the circuit is reset.

   (b) Now modify the above controller to take care of the situation where multiple messages are being sent, one after another. In this case, we'd like the EOVM indicator to output 1 when the end of the first valid message is detected. Subsequently, it should output 1 again when the end of the second valid message is detected and so on. For example, the following table shows asequence of bits received on In and the corresponding desired sequence of bits on Out.

ClkCycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
In	1	0	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1
Out	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1

   Note that the first valid message starts in cycle 1 and ends in cycle 9; the second valid message start in cycle 10 and ends in cycle 19.

   Re-write your pseudo-code from the previous sub-question, choose a datapath containing elements studied in class, and re-design the controller. In both the sub-problems above, indicate what are the reset

states of registers/flip-flops in your design.