CS226 Practice Problem Set 2 (Spring 2016)

Date posted: Feb 4, 2018 **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. How many full- and half-adders are needed to count the total number of ones in a 7-bit wide input using the minimum number of full-adders and half-adders? You are allowed to use only full-adders and half-adders in your solution. Draw a circuit diagram showing the full-adders and half-adders as blocks. Generalize the above for an n-bit input.

2. An array of size $\boldsymbol{m}$ ($= 2^n$) containing m 8-bit numbers is given. Can you design a circuit to find the maximum of the given numbers using atmost $(2^{n+1} - 2)$ 4-bit comparators, $(2^n - 1)$ OR gates, $(2^n - 1)$ AND gates and $(2^{n+3} - 8)$ 2-to-1 multiplexers? Assume that you only have 4-bit comparators, AND gates, OR gates and 2-to-1 multiplexers available for use.

3. This question is about the concept of *functional completeness*. A set of Boolean gates is said to be functionally complete if any Boolean function can be implemented by a circuit consisting of only these gates (and perhaps the constants 0 and 1). For example, the NAND ($out = \overline{in1 \cdot in2}$) and NOR ($out = \overline{in1 + in2}$) gates are functionally complete. This is because any Boolean circuit can be implemented using AND, OR and NOT gates, and NAND (respectively, NOR) gates can be used to implement the functionality of AND, OR and NOT gates. For example, using NAND gates, we can implement a NOT gate as $out = \overline{in \cdot in}$, an OR gate as $out = \overline{\overline{in1 \cdot in1} \cdot \overline{in2 \cdot in2}}$.

   Verify whether $f = A' + BC'$ is functionally complete. Similarly, verify if XOR ($out = in1 \cdot \overline{in2} + in2 \cdot \overline{in2}$ is functionally complete.

4. ***[Adapted from GATE 2017]:*** Figure 1 shows a 4-bit ripple carry adder realized using full-adders and Figure 2 shows the circuit of a full-adder (FA). The propagation delay of the XOR, AND and OR gates in Figure 2 are 20 ns, 15 ns and 10 ns, respectively. Assume all the inputs to the 4-bit adder and Z_0 are initially reset to 0 (and it is kept 0 for a long time).

   Then at time $t = 0$, the inputs to the 4-bit adder are changed to $X_3X_2X_1X_0 = 1100$, $Y_3Y_2Y_1Y_0 = 0110$. Let t_0, t_1, t_2, t_3 and t_4 be the time when outputs S_0, S_1, S_2, S_3, and Z_4 respectively become stable (stop changing). Find the values of t_0 through t_4.

5. ***[Adapted from CS226 endsem exam, Spring 2016:]*** 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 $-$.

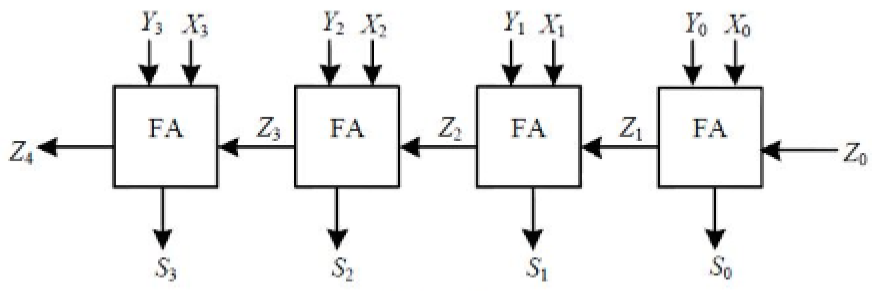

Figure 1: 4-bit adder

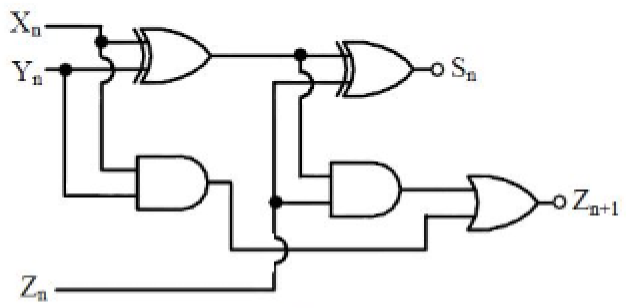

Figure 2: Circuit of full adder used in fig 1.

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

$x_1,x_2\rightarrow$ / $x_3,x_4\downarrow$	**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.

6. ***[From CS226 midsem exam, Spring 2017:]*** We want to implement a data-processing algorithm, written in a C-like language using the datapath shown below (Fig. 3). You may assume that the controller has a Reset output that can be used to asynchronously reset (i.e. immediately set the value to 0) of all registers. You may also assume that aLTb is set to 1 iff the output of register a is less than that of register b.

```
read X and Y; // unsigned integers
```

```
    T = 0;
    while (X >= Y) {
      X = X - Y;
      Y = X - Y;
      T = T + X^2 + Y^2;
    }

    R = X - Y;
    Q = T;
    output Q and R;
  }
```

For the controller, you must give the state transition table in the format given below. You may assume that the values of the inputs (X and Y) do not change until the entire computation is over.

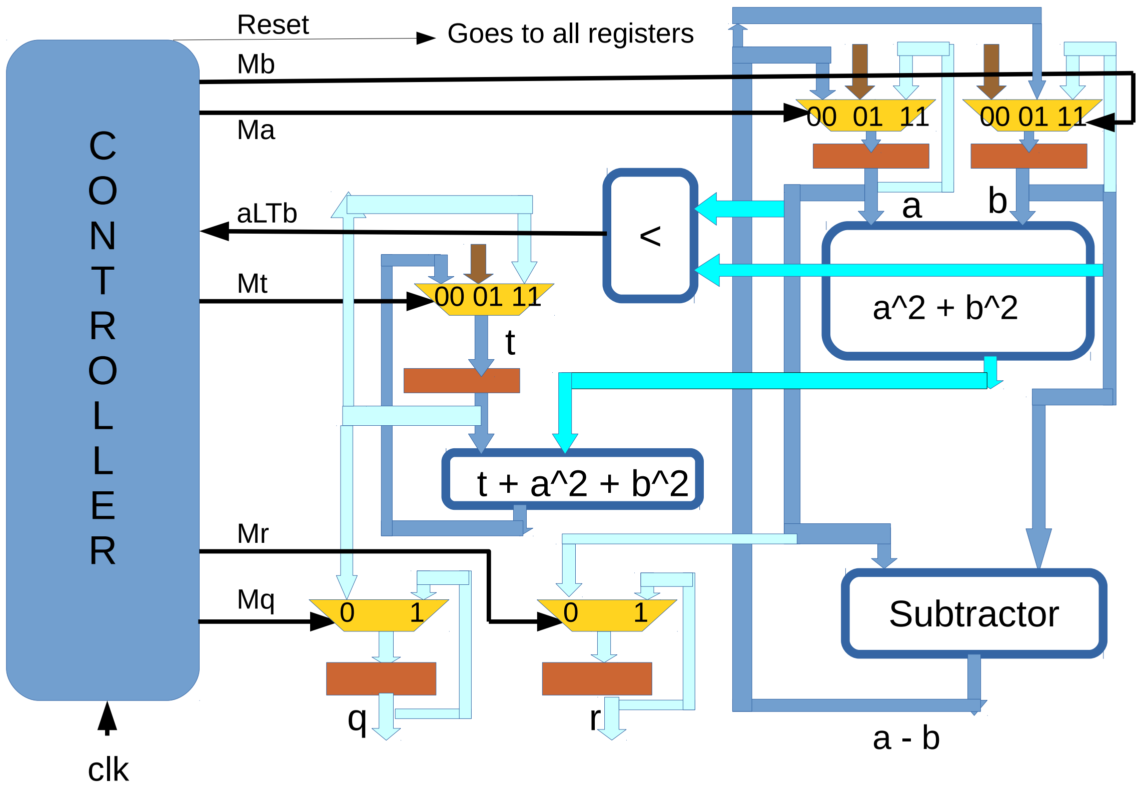

Figure 3: Datapath

You MUST indicate through brief comments what each row of the controller table achieves, e.g. resets registers, or updates T with such and such expression, etc.

CurrState	$aLTB$	NextState	M_a	M_b	M_t	M_q	M_r	Reset	Comment
…	…	…	…	…	…	…	…	…	…
⋮									