CS226 Mid-Semester Examination (Spring 2017)

Max marks: 45 **Time: 120 mins**

- ***The exam is open book and notes brought to the exam hall.***
- *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.*
- ***Please start writing your answer to each sub-question on a fresh page. DO NOT write answers to multiple sub-questions on the same page.***
- ***The use of internet enabled devices is strictly prohibited. You will be debarred from taking the examination if you are found accessing the internet during the examination. All IIT Bombay rules apply in this respect.***
- ***Please do not engage in unfair or dishonest practices during the examination. Anybody found indulging in such practices will be referred to the D-ADAC.***

1. As the chief digital logic designer of DontCares Inc., you have been asked to design two circuits that implement the partially specified functions $F(x_1, x_2, x_3, x_4, x_5, x_6)$ and $G(x_1, x_2, x_3, x_4, x_5, x_6)$ shown in the symbolic K-maps below, where "-" denotes don't care. Note carefully that the K-maps **do not** use the same variables for labeling of rows and columns.

F:

$x_1,x_2\rightarrow$ / $x_3,x_4\downarrow$	**00**	**01**	**11**	**10**
00	$x_5 + x_6$	$\overline{x_6}$	-	-
01	0	$\overline{x_5}$	x_6	1
11	-	-	0	$x_5 + x_6$
10	1	1	-	1

G:

$x_3,x_4\rightarrow$ / $x_5,x_6\downarrow$	**00**	**01**	**11**	**10**
00	$x_1.x_2$	-	1	-
01	0	$\overline{x_2}$	-	1
11	-	-	-	$x_1 + x_2$
10	1	x_1	x_2	1

(a) *[5 + 5 marks]* Implement F and G in sum-of-products form, using the don't cares to simplify your circuit as much as possible.

(b) *[7.5 + 7.5 marks]* Your client now wants to have implementations of F and G as given by the above K-maps, except that they also require that your design must satisfy the following constraint: *Except for input combinations where the values of both F and G are specified by the respective K-maps, the two circuits must never generate* 0 *for the same input combination.*

Let us illustrate what the above constraint means. For $x_1x_2x_3x_4x_5x_6 = 000000$, the K-maps for F and G specify that F and G must both evaluate to 0. Hence this is fine. However, for $x_1x_2x_3x_4x_5x_6 = 111111$, the K-map for F specifies that F must evaluate to 0, while the K-map for G doesn't specify what G must evaluate to. The above constraint however forces G to evaluate to 1 for this input combination. Similarly, for $x_1x_2x_3x_4x_5x_6 = 111000$, neither K-map specifies the value of F or that of G. We are therefore free to have F and G evaluate to whatever values we want for this input combination, as long as both of them are not 0.

Is it possible to obtain implementations of F and G in sum-of-products form, such that the above constraint is satisfied. If so, give such an implementation that uses the don't cares specified in the K-maps as much as possible, subject to the constraint given above. If not, give clear reasons why no implementation of F and G satisfying the constraints exists.

2. Consider the Boolean function $F(u, v, x, y, z) = (x + y + \overline{z}).(\overline{x} + \overline{y}).(z + u + v)$.

   (a) *[5 marks]* Using the variable order $u < v < x < y < z$, construct an ROBDD with complement edges for F. In your diagram, only 0-edges or dotted-edges can have bubbles on them, and you must use only the 1-leaf (i.e no 0-leaf). Of course, if needed, you can have an edge with a bubble coming into the root node of the diagram. Your diagram **must be reduced**, i.e. it **must not** have any two nodes that are functionally equivalent.

   (b) *[5 marks]* We have seen in class that an ROBDD (without complement edges) for a function F yields a circuit implementing F that uses only multiplexers. This is not surprising, since each node of an ROBDD represents the function $\mathsf{ite}(c, i_1, i_0)$, which is also what a multiplexer implements.

   We now want to ask if we can implement any arbitrary Boolean function using only **inverted multiplexers** and nothing else, where an inverted multiplexer implements the function $\mathsf{ite}(c, \overline{i_1}, \overline{i_0}$. For purposes of this question, you are not allowed to invert the outputs of inverted multiplexers to get back original multiplexers.

   Clearly, our goal would be easily achieved if we could represent any Boolean function as a decision diagram where the nodes represent the function $\mathsf{ite}(c, \overline{i_1}, \overline{i_0}$, instead of the usual $\mathsf{ite}(c, i_1, i_2)$ in an ROBDD. Let us call such a decision diagram as I-ROBDD. Thus, an I-ROBDD is a decision diagram that is like an ROBDD in all respects, except for the fact that each of its internal nodes represents the function $\mathsf{ite}(c, \overline{i_1}, \overline{i_0}$ instead of $\mathsf{ite}(c, i_1, i_0)$, as in an ROBDD.

   Construct an I-ROBDD for the function F given in this question. You can represent nodes of an I-ROBDD by squares labeled with the decision variable. The 0-edges and 1-edges can be represented using dooted and solid arrows, as with ROBDDs, and you can use both 0- and 1-leaves.

3. *[10 marks]* We want to implement a data-processing algorithm, written in a C-like language using the datapath shown below (Fig. 1). Note that this is **NOT** the same datapath as used in Quiz 1. 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.

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
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
				$\vdots$					

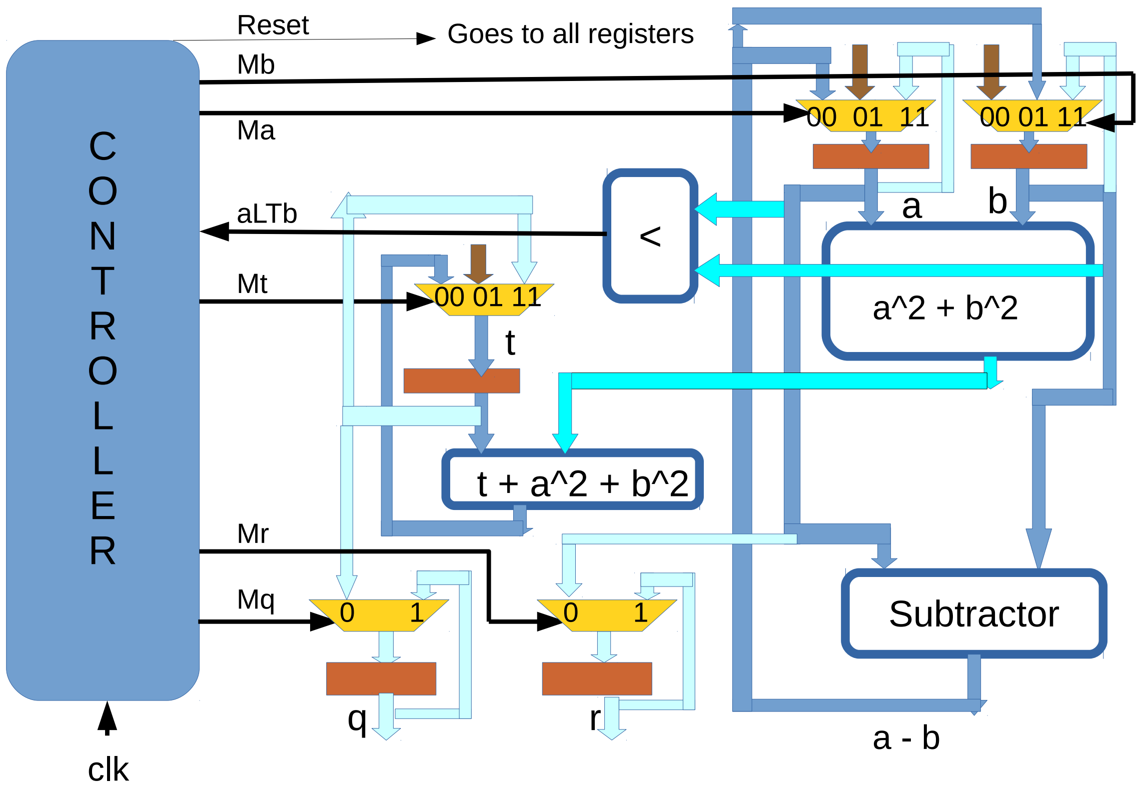

Figure 1: Datapath