ECE/CS 5740/6740: Logic Synthesis & Optimization

Spring 2019, Homework # 1
Due Date: Friday, 1/25/2019
Upload on Canvas, or better yet, give me a hard copy.

1) (25 points) Let f be a completely specified Boolean function. Assume that x is a support variable of f, *i.e.* f depends upon x. State whether the following statements are TRUE or FALSE. If TRUE, prove it. Otherwise, show a counter-example. No points for just stating TRUE or FALSE.
   a) The Boolean function f can be written as $f = f_{\overline{x}} \oplus x(f_x \oplus f_{\overline{x}})$. *Hint:* As always, think of the Shannon's expansion. Shannon's expansion has OR's. But the expression above has XORs!
   b) The Boolean function f is given as: $f = (x + f_{\overline{x}}) \cdot (\overline{x} + f_x)$.
   c) If f is **positive unate** in x, then the **complement** (inverse) of f can be represented as:

$$\overline{f} = \overline{f_x} + \overline{x} \cdot \overline{f_{\overline{x}}} \tag{1}$$

   Here $\overline{f_x} = \overline{(f_x)}$ is the complement of the cofactor.
   d) If f is **positive unate** in x, then the Boolean Difference of f w.r.t. x is given by:

$$\frac{\delta f}{\delta x} = f_x \cdot \overline{f_{\overline{x}}} \tag{2}$$

   e) The function f is TAUTOLOGY **if and only if** both its cofactors w.r.t. any variable x are TAUTOLOGY. In other words, $f \equiv 1 \iff (f_x \equiv 1) \wedge (f_{x'} \equiv 1)$.

2) (5 points) Let f be an arbitrary Boolean function and let c be a cube such that c is also an implicant of f. Prove that f_c is TAUTOLOGY, i.e. $f_c \equiv 1$. (Note that this result is valid even if we relax the condition from c being a cube to c being a *set of cubes*. For example, let f and g be two non-trivial Boolean functions such that $f \supset g$. Then it can be proved that $f_g \equiv 1$. Also note that the expression f_g is called the *generalized co-factor* of f w.r.t. g. It is also called the *restriction* of f w.r.t. g. We will consider generalized cofactors in Boolean function decomposition and don't care analysis.)

3) (15 points) Consider the Boolean function $f = ab + a'c + b'c'$.
   a) Identify the function g that is the smallest function, larger than f, that contains f, and does not contain b in its support.
   b) Identify the function h that is the largest function, smaller than f, that is contained in f, and does not contain b in its support.
   c) On a 3-D cube, depict the cubes of f, g, h and demonstrate the containment relationships.

Please draw a separate cube for each of the above.

4) (10 points) For the circuit shown in Fig. 1, identify the set of assignments to the input variables that allow to propagate the changes on signal a to output Z.

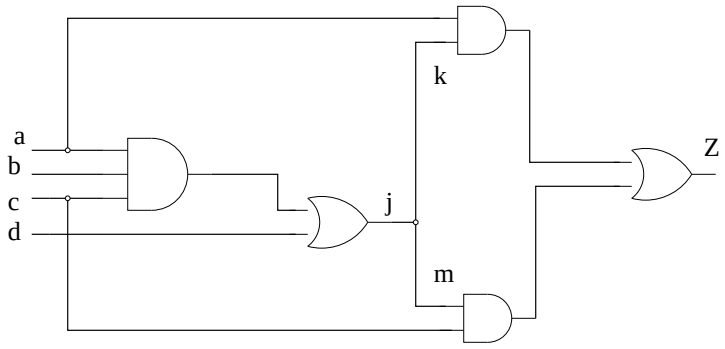

Fig. 1. Is Z sensitive to changes in a?

5) (15 points) Let C be a circuit that represents a Boolean function f. Let x be a primary input net of C. In class, I showed you that test generation for the fault x-stuck-at-0 ($x/0$) can be computed as $x \cdot \frac{\delta f}{\delta x}$. Here x is the fault excitation condition, and $\frac{\delta f}{\delta x}$ is the fault propagation condition. [Analogously, the test for $x/1$: $x' \cdot \frac{\delta f}{\delta x}$].

In this question, you are asked to derive the result that $T_{x/0} = x \cdot \frac{\delta f}{\delta x}$.

How to proceed: Consider Fig. 2. $T_{x/0}$ corresponds to the set of all inputs that **differentiate** between the faulty and fault-free output of the circuit.

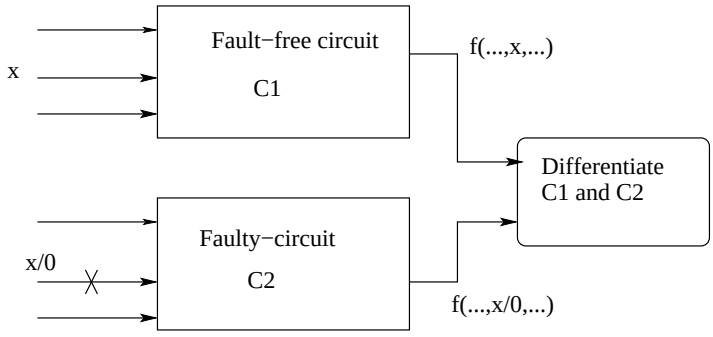

Fig. 2. Prove: $T_{x/0} = x \cdot \frac{\delta f}{\delta x}$

6) (20 points) Provide an algorithm (or pseudocode) that, given a Reduced Ordered Binary Decision Diagram (ROBDD) for f, returns a ROBDD for f_x. Note that variable x may correspond to any level in the ROBDD f, not necessarily the top variable. Demonstrate your approach to construct the ROBDD for f_b, given the ROBDD for $f = ab + ac + bc$, with variable order $a > b > c$.

7) (10 points) The objective of the following exercise is to get you used to some logic synthesis tools and procedures. You will download the tools, and some support files from the class website and run some experiments. (In fact, I encourage you to play with these tools by running your own set of experiments on some small circuits and see how the circuit changes). I have compiled these tools on Linux machines, so they should work on lab1-machines in the CADE lab. Their manuals are also available on the class website for reference.

   The tools that you will download and use are the following:

   - Espresso: Two-level logic minimizer. Takes a .kiss file (or a .pla file) as input. Take a look at the file 'majority.kiss' (also uploaded on the class website, the link for which can be found near Espresso-related stuff). Run 'espresso majority.kiss' and observe the output.
   - Stamina: State minimization tool. Takes a finite-state-machine (FSM) and minimizes the states. The state-machine is also given in .kiss format. See file 'fsm.kiss' also linked on the class website along with stamina. Run 'stamina -h' for help, and 'stamina fsm.kiss' to minimize the fsm.
   - Nova: State-Assignment tool. Nova can also take fsm.kiss and encode it. Run 'nova -h' for help.
   - Finally, download the SIS tool. Note that although espresso, stamina and nova are stand-alone tools, SIS also calls them for corresponding logic synthesis steps. So it would be nice to download all these tools in one directory (if you are a unix expert, you can download them anywhere and set-up your paths in your .rc files). SIS can operate on two-level (read_kiss), multi-level (read_blif), and fsm (read_kiss) file formats. SIS also needs a technology library for tech-mapping; so download the 'lib2.genlib' and 'lib2_latch.genlib' files (or you can create your own technology library in the genlib format - which is given in the SIS manual). Also, the 'script.rugged' is a script of SIS commands that are used for area-optimization and 'script.delay' for delay-optimization.

   Now come the assignments:

   - **Two-Level Optimization** Let f be a Boolean function given by minterms $f = \sum m(0, 4, 8, 10, 11, 12, 13, 15)$. Minimize this function using a Karnaugh map. Then enter this function in a kiss file and use espresso to minimize it. Compare your answer with that of Espresso. Attach your input .kiss file, as well as the output of Espresso, to your HW submission.
   - **FSM Synthesis**: Consider the finite state machine given in Table I.

TABLE I
STATE TRANSITION TABLE OF MACHINE M1

Present State	Next State, Output	
	$x = 0$	$x = 1$
A	E, 0	D, 1
B	F, 0	D, 0
C	E, 0	B, 1
D	F, 0	B, 0
E	C, 0	F, 1
F	B, 0	C, 0

   Enter this machine in the kiss format and synthesize this into a circuit using SIS. First you have to

minimize the machine using 'stamina', assign codes to the states using nova, optimize the logic using 'script.rugged' and subsequently optimize for delay using 'script.delay'. You can print the synthesis stats after every optimization step. Compare the results of area-optimization versus delay-optimization and report the results.

Given below is a sequence of steps you can follow in SIS. If you want to see how the circuit is changing after every optimization step, you can use the 'write_blif' command.

```
prompt>> ./sis
UC Berkeley, SIS 1.3 (compiled 2004-04-04 18:58:04)
sis> read_kiss filename.kiss
sis> state_minimize
sis> state_assign nova -e ih
sis> write_blif
sis> print_stats
sis> source script.rugged
sis> print_stats
sis> write_blif
sis> read_library lib2.genlib
sis> read_library -a lib2_latch.genlib
sis> map
sis> print_map_stats
sis> source script.delay
sis> print_map_stats
sis> write_blif -n file.n.blif
sis> quit
```

Note that the command 'state_assign nova -e ih' finds a minimum-length encoding for the states. You can use one-hot encoding using the command 'state_assign nova -e h'. Re-run the above FSM synthesis with a one-hot code and compare and report the results.