Logic Synthesis & Optimization

Spring 2019, Homework # 5

Due Date: Wednesday, April 24, **midnight**. Upload on Canvas. This is a strict deadline. I will upload solutions the solutions by then, so you have them when you prepare for your exams.

1) **(5 points) – Logic Synthesis using ABC, compared with script.rugged of SIS:** ABC is a AIG-based synthesis and verification tool from Alan Mishchenko, Univ. California Berkeley. Downloading and compiling ABC is rather painless. Go to Alan's ABC website, download the ABC source and compile. I've also uploaded and older version of my personal compiled copy for you (in case you are having problems). *Please also go through the ABC manual and tutorials available on Alan's website*. ABC is integrated with BDDs and SAT. It is possible to read BLIF (read_blif) and EQN files in ABC and also write out CNF files (write_cnf) through ABC, so a SAT solver can be invoked on those CNF files, outside of ABC. In this experiment, you will compare the relative powers of optimization of SIS and ABC (including how much faster ABC runs than SIS).
   a) Take the circuit C7552.blif (uploaded on the class website), read it into SIS, optimize with script.rugged ('source script.rugged'), and print the number of literals (print_stats).
   b) Now we will perform a script.rugged type minimization using ABC. For this purpose, please ensure that you also download the resource file "abc.rc" and keep it in your "current directory" in which you are operating. When ABC runs, it reads this resource file by default, and loads a whole bunch of command aliases.
   c) To keep things simple, let us focus on the commands in the *resyn* and *resyn2* scripts; these are given in the abc.rc file.
   d) In these scripts, the main commands are *balance, drw, refactor* etc., that basically perform AIG rewriting to synthesize the circuit. Type 'drw -h', 'refactor -h', etc. to see what these commands do.
   e) Bi-decomposition can be performed using the command *bidec*.
   f) Just by repeating the scripts *resyn* and *resyn2*, we can achieve better quality results than SIS, and that too in much less execution time.
   g) Invoke ABC, read C7552.blif, and perform the following commands: *aig; bidec; st; resyn; resyn2; write_blif C7552_ABC_opt.blif.*
   h) Now you can read *C7552_ABC_opt.blif* into SIS and print stats to compare the literal cost. Compare the area-delay characteristics of the versions of the circuits synthesized by ABC and

SIS. Which tool seems to be more powerful?

2) **Don't care Computation (15 points):** Solve Problem 7 from Chapter 11 in the textbook, pp 474.

3) **(10 points)**: Consider the multi-output circuit shown in Fig. 1.

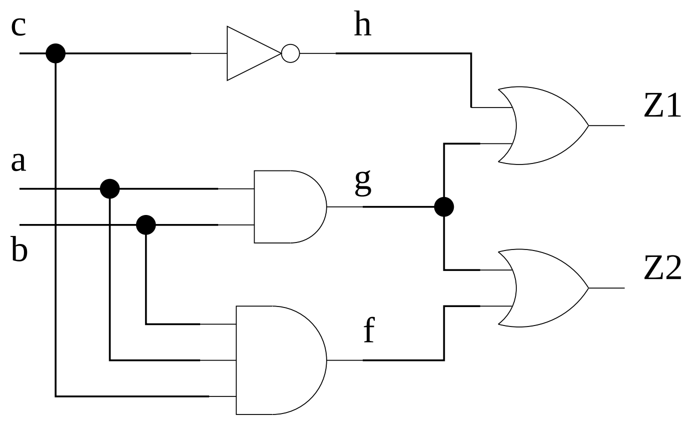

Fig. 1. Simplification of a multi-level, multi-output Boolean network using ODCs

Identify/Compute the *observability don't care* conditions for each of the internal nodes (f, g, h) corresponding to the circuit outputs Z_1, Z_2. Using these don't cares, simplify the network. (Note: No points for collapsing and simplification).

4) **FSM Minimization (30 points):** Minimize the FSM and show the minimized state table for the two machines M1, M2 shown below. Build the merger table, get the compatibles, get the set of maximal compatibles, **stop-think-and-proceed** to draw the compatibility graphs to identify the minimal machine. You can solve these using paper-and-pencil methods (such as the ones given in Kohavi's book, the scanned notes that I emailed to you), you don't need to formulate and solve BCP.

The above machines M1, M2 can be minimized by a program called STAMINA. Enter the above machines in KISS format (an example is given in the textbook, page 361, Pb. 7) and invoke stamina. Compare the result with your answers in the above Question. Use a high verbosity-level to observe what the program is trying to do: `'stamina -s 1 -v 10'`. Stamina is uploaded on the class

TABLE I
STATE TRANSITION TABLE OF MACHINE M1

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

TABLE II
STATE TRANSITION TABLE OF MACHINE M2

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

website.

5) **FSM minimization using BCP (25 points)** For the Machine M2, you will now solve the problem using the Binate covering problem formulation as given in out textbook. Since you already have all the maximal compatibles from the previous question, derive all the **prime compatibles**, then set-up the covering and closure constraints, and derive the constraint matrix for BCP. Finally, solve the BCP – show your steps.

6) **Rectification using Partial Synthesis (15 points)**
   a) Consider the circuit of Fig. 2. Assume that this is a "specification" model. Write the corresponding BLIF file for this circuit: spec.blif.

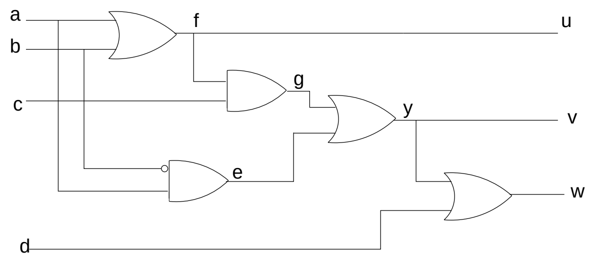

Fig. 2. The specification circuit for the rectification problem

b) Consider the net $e = a \cdot \overline{b}$ in the circuit. Introduce a bug by changing this gate in the circuit to an XOR gate, i.e. introduce the buggy gate: $e = a \oplus b$. Write the corresponding buggy implementation blif file: impl.blif.

c) Perform equivalence checking using ABC: `cec spec.blif impl.blif`. Confirm that the gate change is indeed a bug, i.e. CEC generates a counter-example.

d) This bug should be rectifiable at the same net e, of course. Using the single-fix rectification theorem (Thm 1 in our slides), confirm if the circuit can be rectified at net e.

e) Subsequently, compute the rectification function using both the smallest and the largest interpolants, and validate your result by re-running CEC with the rectified functions.

f) For the bug $e = a \oplus b$, can the circuit be rectified at net g? If so, compute the corresponding rectification function at g for the bug at e. Keep in mind the fanout at net f.