ECE/CS 5745/6745: Testing & Verification of Digital Circuits

Prepared by *Priyank Kalla*

Fall 2023, Homework # 2

Due Date: Wednesday Sept. 27, 11:59pm. Upload on Canvas.

I. Some notes regarding the assignment

Note: Questions 1 and 3 are required of all students. Question 2 is required of ECE/CS 6745 (grad) students, but ECE/CS 5745 (undergrad) students can solve it for extra credit.

This Homework includes a programming assignment with BDDs using the CUDD package. Along with this assignment, I am also providing a C-program main file "`main.c`" that you may use as a "skeleton code" to get started with your assignment. I am also providing you with a BLIF file that describes a simple circuit. BLIF is a file format to describe logic circuits as Boolean networks. BLIF stands for the Berkeley Logic Intermediate Format. A couple of example BLIF files are uploaded on the class website along with this HW2. You may experiment with these files to construct BDDs. Please go to the class website http://www.ece.utah.edu/~kalla/index_6745.html, scroll down to the HW section, and you'll find all the HW2 related files there.

Also note that the CUDD package is of course designed for Unix systems. You can use the CADE lab machines (e.g. lab1-7.eng.utah.edu). However, Ritaja believes that these should compile on Windows too if you know what you are doing, e.g. using Visual Studio. She has also compiled the package on MAC OS, and it works.

Finally, don't delay working on this assignment. It's not super hard, but it may be a bit tedious compiling the tool. Also, if you need clarification, feel free to ask me or the TA. I may briefly explain the assignment in class too.

II. The Assignment Questions

1) **Some Programming with the BDD package (60 points):** In this assignment, you are asked to write some simple programs using Prof. Fabio Somenzi's CUDD package. First of all, on the class website, I've provided a link to download the CUDD package. Also, there is a link to online documentation that will be helpful. For the setup, do the following:
   - Log in to one of CADE machines, which supports x86-64 bit instruction set. Try compiling and running the CUDD package. Follow the instructions below.

- Download and untar the package in a specific directory. Go through the README file. You will also find directories such as, cudd, nanotrav, sis, st, util, etc.
- Enter the ./cudd directory and just glance through the cudd.h and cuddInt.h files. These are header files that declare the data structures: DdNode, DdManager, DdChildren. Glance through these data-structures. You will extensively use pointers to DdManager and DdNode structures. You can also go through the #define macros declared.
- Now just list all the *.c files in the cudd directory: '`ls -al *.c`'. **NOTE**: the CUDD package implements BDDs, ADDs (called Algebraic Decision Diagrams) as well as ZDDs (called Zero-Suppressed BDDs). You should just ignore ADD and ZDD related files and routines; just concentrate on BDD related files. The file names are self explanatory. For example, `cuddCof.c` contains routines for performing cofactor computations. You don't need to study the routines - just get introduced to the package.
- Now go through the `nanotrav` directory. Go through the README file. Also, glance through the "main.c" file. You may not understand much, but don't worry. Later on, we will overwrite this main.c file with (y)our own somewhat simpler "main.c" file.
- Come back to the top directory. Go through the Makefile again. Just type 'make' and the package will be compiled: libraries will be created and linked and an executable, also named 'nanotrav', will be created in the ./nanotrav directory. Just run a test program as given in the ./nanotrav/README file. For example, type 'nanotrav -p 1 -cover C17.blif > C17.out' just to check that the program compiled properly and is indeed running fine. You can go through the C17.out file to see the ROBDD stats corresponding to the circuit.
- The circuit C17.blif is a very small circuit. Re-run the above experiment with the circuit named `MastrovitoF_q16.blif`. This BLIF file is a modulo arithmetic circuit used in cryptography. ROBDDs can be built for this circuit, but it may take around 5 mins. How many BDD nodes are created in this circuit's BDD? Don't be surprised if it goes beyond a million.
- You may repeat the same experiment with C6288.blif, which is a 16-bit multiplier. ROBDDs are infeasible for multipliers, and the memory on your computer will explode. You can run 'nanotrav' on this file, and check how the memory is growing (using the 'top' command in unix), but please kill the program (CNTRL-c) within a couple of minutes so you don't crash the computer.
- Now the ./nanotrav/main.c file is too complicated. I've written a simple program that: i) initializes a BDD manager; ii) creates variables; iii) performs some simple ITE compu-

tation; iv) prints out the resulting BDD, by declaring variables appropriately and calling some BDD routines. This file, again called 'main.c' is also uploaded on the class website - within the 'Homeworks' section. Download this file, go through it properly. Replace ./nanotrav/main.c by this file 'main.c' and compile and just run the program. Relate the output to the code in this file. **Note**: the CUDD documentation (online) or the .ps file in the cudd-l.m.n/cudd/doc/doc.ps will tell you all about these routines. In the .ps file, the index from page 43 onwards gives a list of all the BDD manipulation routines that you can use.

- **Now here are the assignments**: In this main.c file, you will write code that does the following:
  - Create ROBDD for $f = ab + ac + bc$. Use any variable order.
  - Write code that tests whether the function is positive unate or negative unate in variable b. [Recall: How does one check for unate functions?]
  - Repeat the above test for $f = a \oplus b \oplus c$, three input xor function.
  - Print the outputs properly. Try to use the Cudd_DumpDot() routine and try to use the 'dot' package to print the BDD that you build or want to show. The 'dot' package allows you to draw graphs. If you have never used this package before, you can get it from http://www.graphviz.org.

2) **[(20 points) This question is for ECE/CS 6745 students. 5745 students can solve it for extra credits!]** In class we studied the concept of Shannon's expansion: $f = xf_x + x'f_{x'}$, where f is a function and x is a variable. The concept of Shannon's expansion can be generalized to expansion of f with respect to another function g. That is, given Boolean functions f, g, we have

$$f = gf_g + g'f_{g'} \tag{1}$$

where $f_g, f_{g'}$ are called the *generalized cofactors*. Generalized cofactors are **not unique**, but they satisfy the following bounds:

$$f \cdot g \subseteq f_g \subseteq f + g' \tag{2}$$

You are asked to <u>prove</u> the bounds given in Eqn. (2) above. [**Hint**: Generalized Shannon's expansion of f w.r.t. g, use of Venn diagrams to visualize the intersection of $f \cdot g$, etc. can help deriving the above bounds.]

3) **(20 points)** You can also use the BDD package to perform the *generalized co-factor* computations. The routines are given in file 'cuddGenCof.c'. Take a look at the *description* of the

following subroutines: `i) bddRestrict, ii) bddConstrain.` Using any of these routines, you are asked to do the following:

- Let $f = w'x'z' + wx'z + w'yz + wyz'$ and let $g = w \oplus z$. Construct BDDs for f, g. Then using the above routines, compute f_g and $f_{g'}$, and print the cofactor functions (Cudd_PrintMinterm()).
- As mentioned before, since the generalized co-factors are not unique, but they agree with the bounds: $f \cdot g \subseteq f_g \subseteq f + g'$, *using a suitable formulation*, test whether the bounds are satisfied for the above functions f, g, f_g. [Recall: Containment check is formulated using tautology check!]

4) **Turn in** your code and the program output. Preserve your code. I may test it myself. Happy programming and good luck!