Logic Synthesis & Verification, Fall 2023

National Taiwan University

Programming Assignment 1

Exercises 1-3 are due on 9/17 23:59; Exercise 4 is due on 10/1 23:59.
Submit Exercises 1 and 4 on GitHub, and Exercises 2 and 3 on NTU Cool.

Submission Guidelines. For Exercises 2 and 3, please put the required items under `lsv/pa1/`, i.e., this folder. Compress the `lsv/pa1/` folder as a single `.tgz` file and submit it on NTU cool. For Exercise 4, please develop your code under `src/ext-lsv`. You are asked to submit your assignments by creating pull requests to your own branch. To avoid plagiarism, please push files and create pull requests at the last moment before the deadline. Please see the GitHub page (https://github.com/NTU-ALComLab/LSV-PA) for more details.

1 [Getting Familiar with GitHub] (0%)

(a) Open the GitHub page and check whether there is your own branch named by your student ID number. If you cannot find your own branch or your branch is inconsistent with the `master` branch, please contact TA.
(b) Fork the repository to your personal GitHub account. You will need to develop your programs on your forked repository.
(c) Edit `participants-id.csv` under `lsv/admin/` to register your student ID, name, and GitHub account. Send a pull request to the `master` branch after you finish it. Note that this is the only time you send a pull request to the `master` branch. In the following, you are asked to send pull requests to your own branch.

2 [Using ABC] (10%)

(a) Create a BLIF file named "`mul.blif`" to represent a two-bit unsigned multiplier Y = A * B, where A = (a1 a0) and B = (b1 b0) are two-bit unsigned integers, and Y = (y3 y2 y1 y0) is a four-bit unsigned integer. You can find the BLIF manual in `http://www.eecs.berkeley.edu/~alanmi/publications/other/blif.pdf`.
(b) Perform the following steps to practice using ABC with the two-bit unsigned multiplier example. Screenshot the results after running the commands and put them in your report.
   1. read the BLIF file into ABC (command "`read`")
   2. check statistics (command "`print_stats`")
   3. visualize the network structure (command "`show`")
   4. convert to AIG (command "`strash`")
   5. visualize the AIG (command "`show`")

6. convert to BDD (command "`collapse`")
7. visualize the BDD (command "`show_bdd -g`"; note that "`show_bdd`" only shows the first PO; option "`-g`" can be applied to show all POs)

3 [ABC Boolean Function Representations] (10%)

In ABC, there are different ways to represent Boolean functions.

(a) Compare the following differences with the two-bit unsigned multiplier example. Screenshot the results and briefly describe your findings in your report.
   1. logic network in AIG (by command "`aig`") vs.
      structurally hashed AIG (by command "`strash`")
   2. logic network in BDD (by command "`bdd`") vs.
      collapsed BDD (by command "`collapse`")

(b) Given a structurally hashed AIG, find a sequence of ABC commands to convert it to a logic network with node function expressed in sum-of-products (SOP). Use the two-bit unsigned multiplier example to test your command sequence, screenshot the results, and put them in your report.

4 [Programming ABC] (80%)

In this problem, you are asked to write your own procedures and integrate them into ABC, so that your self-defined commands can be executed within ABC. You may need to trace some source codes in ABC to understand the data structures and function usages.

Hint 1: You may refer to `src/base/abc/abc.h` to find definitions of some important variables, functions, iterators, etc.
Hint 2: You may use the command "`grep -R <keyword>`" to find whether a certain keyword appears in the source codes.
Hint 3: You may refer to the CUDD package website to find some useful BDD operations.

4.1 Function simulation with BDD

Write a procedure in ABC to do simulations for a given BDD and an input pattern. Integrate this procedure into ABC (under `src/ext-lsv/`), so that after reading in a circuit (by command "`read`") and transforming it into BDD (by command "`collapse`"), running the command "`lsv_sim_bdd`" would invoke your code. The command should have the following format.

```
lsv_sim_bdd <input_pattern>
```

The output should have the following format.

```
<po_name_1>: <value>
<po_name_2>: <value>
...
```

For example, suppose the BDD represents the function $y = a + b + c$. The command and the output should look like:

```
abc 01> lsv_sim_bdd 000
y: 0
abc 01> lsv_sim_bdd 010
y: 1
```

To achieve this function, you can use the BDD *cofactor* operator. Let f be a Boolean function and a_i be an input variable. The $cofactor(f, a_i)$ operation substitutes all appearances of a_i in f by 1, and the $cofactor(f, a_i')$ operation substitutes all appearances of a_i in f by 0. To decide the function output of a given input pattern, you can do the $cofactor$ operation against all input variables and compare the result to constant 1.

4.2 Parallel function simulation with AIG

Write a procedure in ABC to do 32-bit parallel simulations for a given AIG and some input patterns. Integrate this procedure into ABC (under `src/ext-lsv/`), so that after reading in a circuit (by command "`read`") and transforming it into AIG (by command "`strash`"), running the command "`lsv_sim_aig`" would invoke your code.

The patterns under simulation are written in a file, in which each line represents an input pattern, and each bit in a line represents the value of an input variable. The variable order is the same as defined in the BLIF file. The command should have the following format.

```
lsv_sim_aig <input_pattern_file>
```

The output should have the following format.

```
<po_name_1>: <value>
<po_name_2>: <value>
...
```

For example, suppose the AIG represents the function $y = a + b + c$, and the content in `demo.in` is as follows.

```
000
001
110
111
```

Then the command and the output should look like:

```
abc 01> lsv_sim_aig demo.in
y: 0111
```

Notice. For problems 4.1 and 4.2, you may see some built-in functions to perform functional simulations on BDD or AIG. You are allowed to refer to them, but you have to write your own procedures. Directly calling or copying from the built-in functions will be viewed as plagiarism.

Files to Submit

1. The BLIF file in problem 2(a).
2. The PDF report named `report.pdf`.
3. The source codes in problem 4.