Logic Synthesis & Verification, Fall 2014

National Taiwan University

Programming Assignment 1

Due on 2014/10/15 before lecture

1 [Using ABC]

(10%)

(a) Use BLIF manual (`http://www.eecs.berkeley.edu/~alanmi/publications/other/blif.pdf`) to create a BLIF file representing a four-bit adder.

(b) Perform the following steps to practice using ABC (`http://www.eecs.berkeley.edu/~alanmi/abc/`):

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`"; note that `show_bdd` only shows the first PO; command "`cone`" can be applied in combination to show other POs)

Items to turn in:

1. The BLIF file.
2. Screenshot of your ABC execution steps.
3. Results of "`show`" and "`show_bdd`".

Comment 1: For commands "`show`" and "`show_bdd`" to work, please download the binary of software "dot" from GraphViz webpage (`http://www.graphviz.org`) and put it in the same directory as the ABC binary or anywhere else in the path.

Comment 2: Make sure GSview and Ghostscript are installed on your computer. (`http://pages.cs.wisc.edu/~ghost/gsview/`) A proper path of `gsview32.exe` needs to be specified in "`\src\base\abc\abcShow.c`." For Windows 7, the path is likely to be `C:\Program Files (x86)\Ghostgum\gsview\gsview32.exe`.

2 [ABC Boolean Function Representations]

(10%) In ABC there are different ways to represent Boolean functions.

(a) Please compare the following differences.
   1. logic network in AIG (by command "`aig`") vs. structurally hashed AIG (by command "`srtash`")
   2. logic network in BDD (by command "`bdd`") vs. collapsed BDD (by command "`collapse`")

(b) Given a structurally hashed AIG, please find a sequence of ABC commands to covert it to a logic network in SOP.

3 [Programming ABC]

(80%) Write a procedure in ABC environment to perform bit-wise simulation over a given AIG with respect to a given set of input simulation patterns. (Please refer to the code of command "strash" to convert a logic network into an AIG.) Integrate your AIG simulation procedure into ABC, so that running command "`print_aigsim`" would invoke your code, and print the simulation patterns for each output.

The input pattern file, say `C17.pni` for circuit `C17.blif`, is of the following format:

```
.model C17
.patterns 128
.input 1GAT(0)
3DEF63A5B4ACF7E3A4CD4251EC625EC2
.input 2GAT(1)
59C23D0F66A5B484F763AACD4B11EB5D
...
.end
```

The expected output pattern file, say `C17.pno` for circuit `C17.blif`, is of the following format:

```
.model C17
.patterns 128
.output 22GAT(10)
0123456789ABCDEFFEDCBA9876543210
.output 23GAT(9)
...
.end
```

In the files, `.model` is followed by the circuit name, `.patterns` is followed by the number of patterns (in bits), `.input` is followed by an input name and the following line list the patterns in hexadecimal numbers, `.output` is followed by an output name and the following line list the patterns in hexadecimal numbers,

and `.end` signifies the end of the file.

Programming help:

Example of code to iterate over the objects

```
void Abc_NtkCleanCopy( Abc_Ntk_t * pNtk )
{
  Abc_Obj_t * pObj;
  int i;
  Abc_NtkForEachObj( pNtk, pObj, i )
    pObj->pCopy = NULL;
}
```

Example of code to create new command "`print_aigsim`"
Call the new procedure (say, `Abc_AigSim`) from `Abc_CommandAigSim()` in file `\src\base\abci\abc.c`

```
int Abc_CommandAigSim( Abc_Frame_t * pAbc, int argc, char ** argv)
{
  ⋮
  Abc_AigSim( ... );
  ⋮
}
```

Items to turn in:

1. Your code of `Abc_CommandAigSim` and other function calls if there is any.