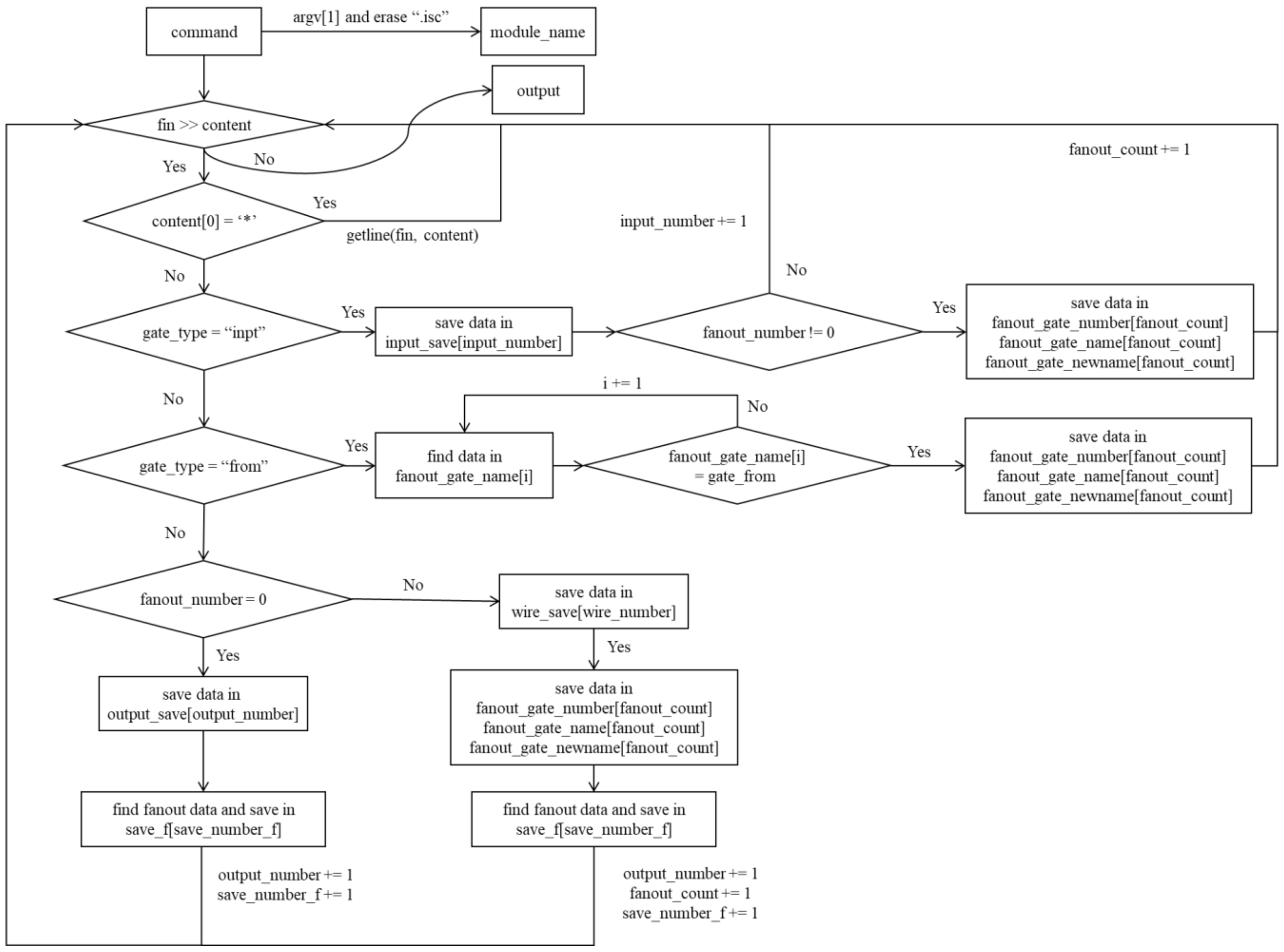
EE6094 CAD for VLSI Design
Programming Assignment 1: Benchmark Translator

Student ID: 107501540
Student name: 林宜霆

A. Readme

I. Flow chart


command
argv[1] and erase ".isc"
module_name
output
fin >> content
No
Yes
content[0] = '*'
Yes
getline(fin, content)
No
gate_type = "inpt"
Yes
save data in input_save[input_number]
fanout_number != 0
input_number += 1
No
Yes
save data in fanout_gate_number[fanout_count] fanout_gate_name[fanout_count] fanout_gate_newname[fanout_count]
fanout_count += 1
gate_type = "from"
Yes
find data in fanout_gate_name[i]
fanout_gate_name[i] = gate_from
i += 1
No
Yes
save data in fanout_gate_number[fanout_count] fanout_gate_name[fanout_count] fanout_gate_newname[fanout_count]
No
fanout_number = 0
No
save data in wire_save[wire_number]
Yes
save data in fanout_gate_number[fanout_count] fanout_gate_name[fanout_count] fanout_gate_newname[fanout_count]
Yes
save data in output_save[output_number]
find fanout data and save in save_f[save_number_f]
find fanout data and save in save_f[save_number_f]
output_number += 1
save_number_f += 1
output_number += 1
fanout_count += 1
save_number_f += 1

II. Execute and test

You need to add the sources to the workstation first:

```
source /usr/cad/cadence/CIC/incisiv.cshrc
source /usr/cad/synopsys/CIC/verdi.cshrc
```

The program's name is "107501540_PA1.cpp", you can compile the program by:

```
g++ -std=c++11 107501540_PA1.cpp -o 107501540_PA1.exe
```

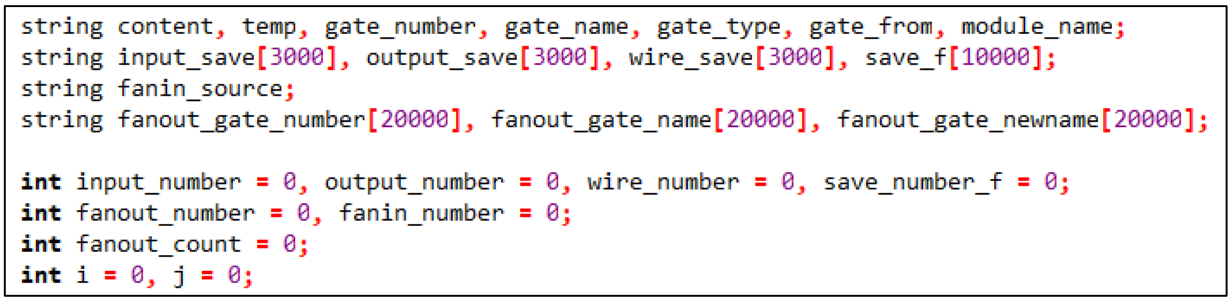
The executable file is called “107501540_PA1.exe”, then execute it:

```
107501540_PA1.exe c17.isc c17.v
107501540_PA1.exe c880.isc c880.v
107501540_PA1.exe c6288.isc c6288.v
```

After generate the verilog files, you can test whether they are correct or not by:

```
ncverilog +access+r c17.v c17_testbench.v
ncverilog +access+r c880.v c880_testbench.v
ncverilog +access+r c6288.v c6288_testbench.v
```

III. Variables



```
string content, temp, gate_number, gate_name, gate_type, gate_from, module_name;
string input_save[3000], output_save[3000], wire_save[3000], save_f[10000];
string fanin_source;
string fanout_gate_number[20000], fanout_gate_name[20000], fanout_gate_newname[20000];

int input_number = 0, output_number = 0, wire_number = 0, save_number_f = 0;
int fanout_number = 0, fanin_number = 0;
int fanout_count = 0;
int i = 0, j = 0;
```

(1) string

content, temp, gate_number, gate_name, gate_type, gate_from, module_name

➔ Save the information as the name of the *string*.

input_save[3000], output_save[3000], wire_save[3000], save_f[10000]

➔ Save the input, output, wire, and the all gates data for the output needs.

fanin_source

➔ Save fan-in to check which gate we need for the gates.

fanout_gate_number[20000], fanout_gate_name[20000], fanout_gate_newname[20000]

➔ Save the information for the fan-out needs.

(2) int

input_number, output_number, wire_number, save_number_f

➔ The number counted for those *strings*.

fanout_number, fanin_number

➔ Hold the fanout and fanin numbers from input.

fanout_count

➔ The number counted for *strings* such as fanout_gate_number, fanout_gate_name, and fanout_gate_newname.

i, j

➔ The integers use for the *for* loop.

IV. Codes

There are three main parts of the program: module name, input, and output.

(1) Module name

```
int main (int argc, char *argv[]) {
    ifstream fin;
    ofstream fout;

    fin.open(argv[1]); if(fin.fail()) {cout << "ERROR" << endl;}
    fout.open(argv[2]); if(fout.fail()) {cout << "ERROR" << endl;}
```

We use “argc” and “argv” to get the information from the command.

```
//  ---------------------------------- module_name ----------------------------------
    module_name = argv[1];
    module_name = module_name.erase(module_name.find("."), 4);
```

We get the module name by erasing “.isc”.

(2) Input

```
//  ---------------------------------- input ----------------------------------
    while (fin >> content) {
```

We get the input in *while* loop, and it will stop when it doesn’t have data anymore.

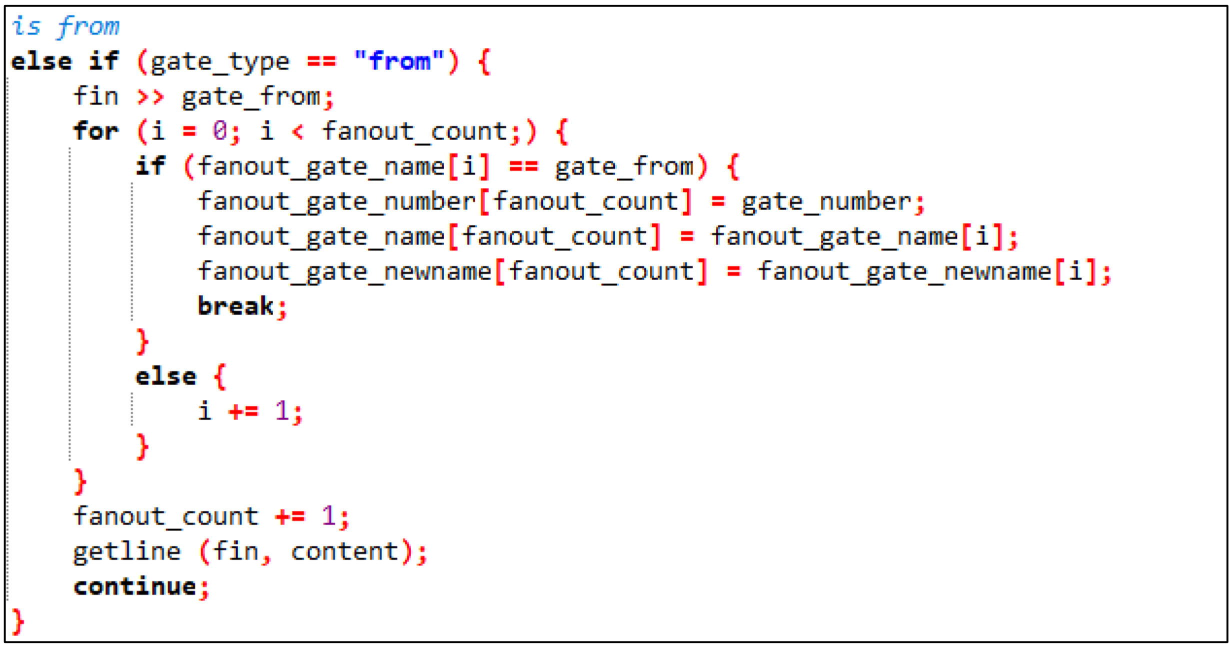
```
if (content[0] == '*') {
    getline (fin, content);
    continue;
}
```

The line start by “*” can be ignored, so we use “getline” to get the rest of the line.

If the input is not start from “*”, which can’t be ignored, we will enter the part below.
We separate the gate types as “input”, “from”, and others.

```
is input
if (gate_type == "inpt") {
    fin >> fanout_number >> fanin_number;
    input_save[input_number] = "gat" + gate_number;
    if (fanout_number != 0) {
        fanout_gate_number[fanout_count] = gate_number;
        fanout_gate_name[fanout_count] = gate_name;
        fanout_gate_newname[fanout_count] = input_save[input_number];
    }
    input_number += 1;
    fanout_count += 1;
    getline (fin, content);
    continue;
}
```

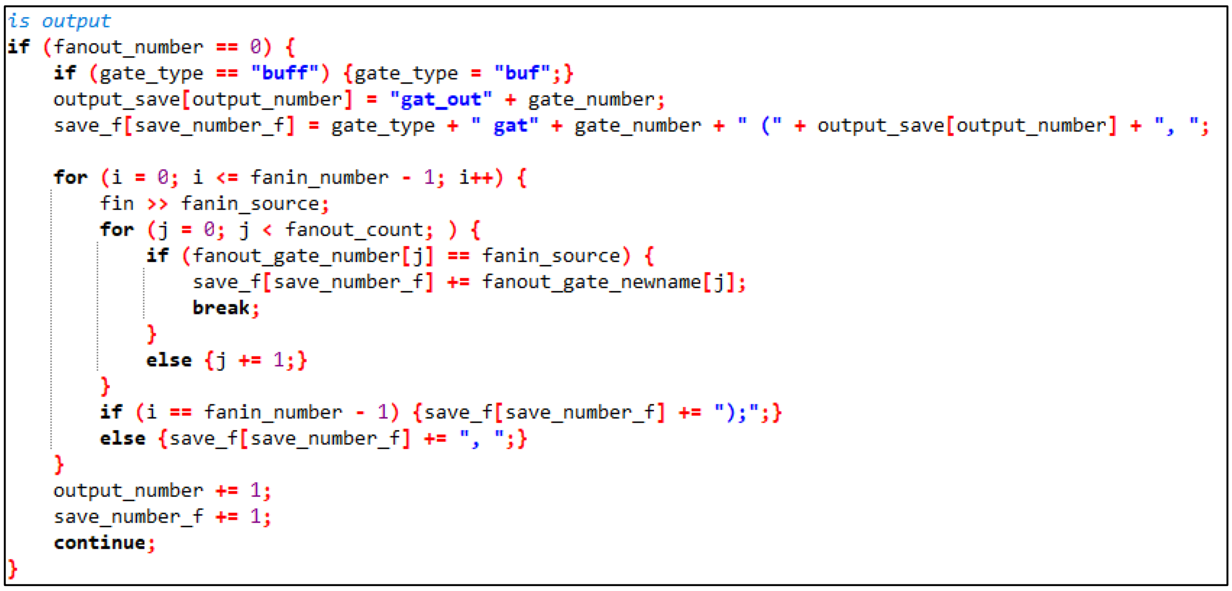
If the input gate is also the fan-out gate to other gates, we will save it in the *strings* in order to get the data when we need it.



```
is from
else if (gate_type == "from") {
    fin >> gate_from;
    for (i = 0; i < fanout_count;) {
        if (fanout_gate_name[i] == gate_from) {
            fanout_gate_number[fanout_count] = gate_number;
            fanout_gate_name[fanout_count] = fanout_gate_name[i];
            fanout_gate_newname[fanout_count] = fanout_gate_newname[i];
            break;
        }
        else {
            i += 1;
        }
    }
    fanout_count += 1;
    getline (fin, content);
    continue;
}
```

Search which gate we need, and save the current gate number with the information of the gate we need in the *strings*.

If the gate type is not input or from, we separate the rest gates as "is output" or "not output".



```
is output
if (fanout_number == 0) {
    if (gate_type == "buff") {gate_type = "buf";}
    output_save[output_number] = "gat_out" + gate_number;
    save_f[save_number_f] = gate_type + " gat" + gate_number + " (" + output_save[output_number] + ", ";

    for (i = 0; i <= fanin_number - 1; i++) {
        fin >> fanin_source;
        for (j = 0; j < fanout_count; ) {
            if (fanout_gate_number[j] == fanin_source) {
                save_f[save_number_f] += fanout_gate_newname[j];
                break;
            }
            else {j += 1;}
        }
        if (i == fanin_number - 1) {save_f[save_number_f] += ");";}
        else {save_f[save_number_f] += ", ";}
    }
    output_number += 1;
    save_number_f += 1;
    continue;
}
```

If we need the fan-in of the previous gate, we will get the data from the *strings* of fan-out, and save the data in the *strings*.

```
not output
else {
    wire_save[wire_number] = "gat_out" + gate_number;
    fanout_gate_number[fanout_count] = gate_number;
    fanout_gate_name[fanout_count] = gate_name;
    fanout_gate_newname[fanout_count] = wire_save[wire_number];
    save_f[save_number_f] = gate_type + " gat" + gate_number + " (" + wire_save[wire_number] + ", ";

    for (i = 0; i <= fanin_number - 1; i++) {
        fin >> fanin_source;
        for (j = 0; j < fanout_count; ) {
            if (fanout_gate_number[j] == fanin_source) {
                save_f[save_number_f] += fanout_gate_newname[j];
                break;
            }
            else {j += 1;}
        }
        if (i == fanin_number - 1) {save_f[save_number_f] += ");";}
        else {save_f[save_number_f] += ", ";}
    }
    wire_number += 1;
    fanout_count += 1;
    save_number_f += 1;
    continue;
}
```

If we need the fan-in of the previous gate, we will get the data from the *strings* of fan-out, and save the data in the *strings*.

(3) Output

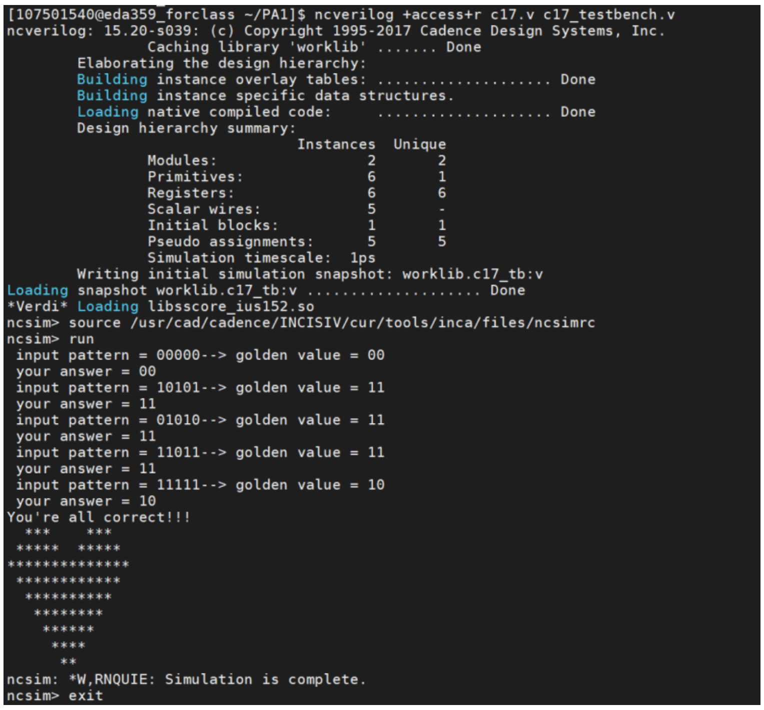
```
//  -------------------------------- output --------------------------------

    fout << "`timescale 1ns/1ps" << endl;

    fout << "module " << module_name << " (" << input_save[0];
    for (i = 1; i < input_number; i++) {fout << ", " << input_save[i];}
    for (i = 0; i < output_number; i++) {fout << ", " << output_save[i];}
    fout << ");" << endl;

    fout << "input " << input_save[0];
    for (i = 1; i < input_number; i++) {fout << ", " << input_save[i];}
    fout << ";" << endl;

    fout << "output " << output_save[0];
    for (i = 1; i < output_number; i++) {fout << ", " << output_save[i];}
    fout << ";" << endl;

    fout << "wire " << wire_save[0];
    for (i = 1; i < wire_number; i++) {fout << ", " << wire_save[i];}
    fout << ";" << endl;

    for (i = 0; i < save_number_f; i++) {fout << save_f[i] << endl;}

    fout << "endmodule";
```

Output format is as usual Verilog code.

B. Completion

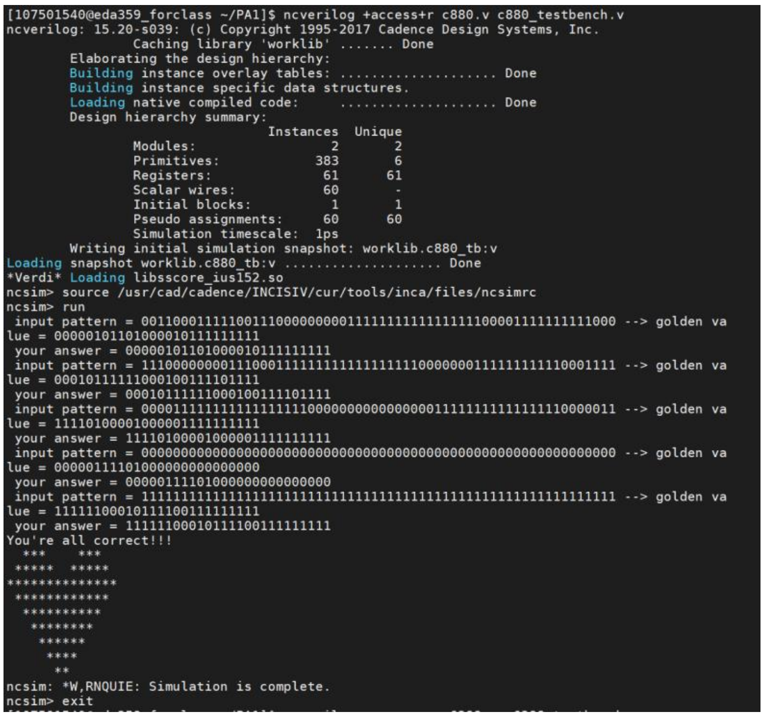
The completion of three files are as follow. Each of them has occurred the “heart” as we execute the program.

I. c17



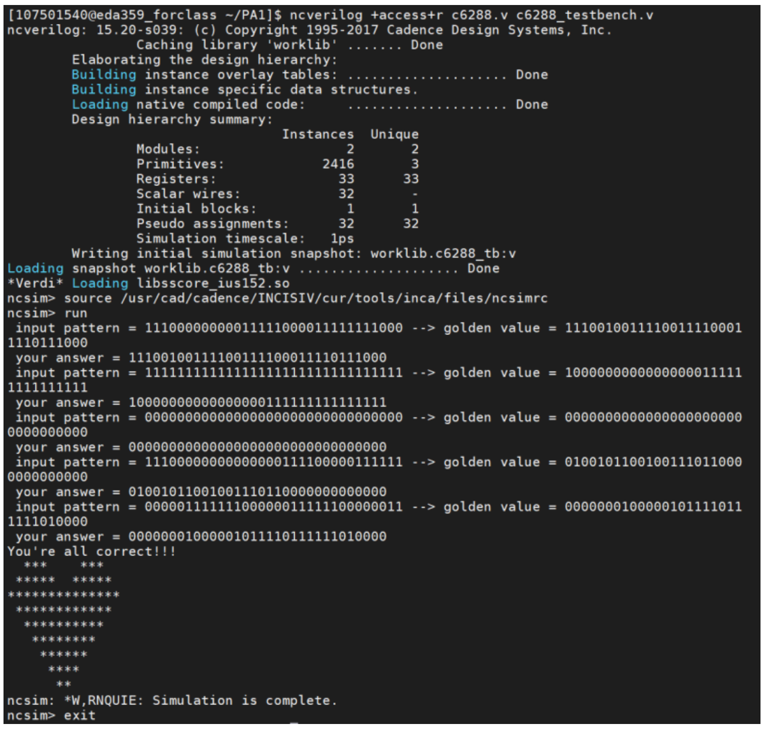
```
[107501540@eda359_forclass ~/PA1]$ ncverilog +access+r c17.v c17_testbench.v
ncverilog: 15.20-s039: (c) Copyright 1995-2017 Cadence Design Systems, Inc.
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Building instance overlay tables: .................... Done
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                                 Instances  Unique
                Modules:                 2       2
                Primitives:              6       1
                Registers:               6       6
                Scalar wires:            5       -
                Initial blocks:          1       1
                Pseudo assignments:      5       5
                Simulation timescale:  1ps
        Writing initial simulation snapshot: worklib.c17_tb:v
Loading snapshot worklib.c17_tb:v .................... Done
*Verdi* Loading libsscore_ius152.so
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
 input pattern = 00000--> golden value = 00
 your answer = 00
 input pattern = 10101--> golden value = 11
 your answer = 11
 input pattern = 01010--> golden value = 11
 your answer = 11
 input pattern = 11011--> golden value = 11
 your answer = 11
 input pattern = 11111--> golden value = 10
 your answer = 10
You're all correct!!!
  ***    ***
 *****  *****
**************
 ************
  **********
   ********
    ******
     ****
      **
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
```

II. c880



```
[107501540@eda359_forclass ~/PA1]$ ncverilog +access+r c880.v c880_testbench.v
ncverilog: 15.20-s039: (c) Copyright 1995-2017 Cadence Design Systems, Inc.
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Building instance overlay tables: .................... Done
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                                 Instances  Unique
                Modules:                 2       2
                Primitives:            383       6
                Registers:              61      61
                Scalar wires:           60       -
                Initial blocks:          1       1
                Pseudo assignments:     60      60
                Simulation timescale:  1ps
        Writing initial simulation snapshot: worklib.c880_tb:v
Loading snapshot worklib.c880_tb:v .................... Done
*Verdi* Loading libsscore_ius152.so
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
 input pattern = 001100011111001110000000000111111111111111100001111111111000 --> golden va
lue = 00000101101000010111111111
 your answer = 00000101101000010111111111
 input pattern = 111000000001110001111111111111111110000000011111111110001111 --> golden va
lue = 00010111111000100111101111
 your answer = 00010111111000100111101111
 input pattern = 000011111111111111111000000000000000001111111111111110000011 --> golden va
lue = 11110100001000001111111111
 your answer = 11110100001000001111111111
 input pattern = 000000000000000000000000000000000000000000000000000000000000 --> golden va
lue = 00000111101000000000000000
 your answer = 00000111101000000000000000
 input pattern = 111111111111111111111111111111111111111111111111111111111111 --> golden va
lue = 11111100010111100111111111
 your answer = 11111100010111100111111111
You're all correct!!!
  ***    ***
 *****  *****
**************
 ************
  **********
   ********
    ******
     ****
      **
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
```

III. c6288



```
[107501540@eda359_forclass ~/PA1]$ ncverilog +access+r c6288.v c6288_testbench.v
ncverilog: 15.20-s039: (c) Copyright 1995-2017 Cadence Design Systems, Inc.
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Building instance overlay tables: .................... Done
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                                 Instances  Unique
                Modules:                 2       2
                Primitives:           2416       3
                Registers:              33      33
                Scalar wires:           32       -
                Initial blocks:          1       1
                Pseudo assignments:     32      32
                Simulation timescale:   1ps
        Writing initial simulation snapshot: worklib.c6288_tb:v
Loading snapshot worklib.c6288_tb:v .................... Done
*Verdi* Loading libsscore_ius152.so
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
 input pattern = 11100000000011111000011111111000 --> golden value = 1110010011110011110001
1110111000
 your answer = 11100100111100111100011110111000
 input pattern = 11111111111111111111111111111111 --> golden value = 1000000000000000011111
1111111111
 your answer = 10000000000000000111111111111111
 input pattern = 00000000000000000000000000000000 --> golden value = 0000000000000000000000
0000000000
 your answer = 00000000000000000000000000000000
 input pattern = 11100000000000000111100000111111 --> golden value = 0100101100100111011000
0000000000
 your answer = 01001011001001110110000000000000
 input pattern = 00000111111100000011111100000011 --> golden value = 0000000100000101111011
1111010000
 your answer = 00000001000001011110111111010000
You're all correct!!!
  ***    ***
 *****  *****
**************
 ************
  **********
   ********
    ******
     ****
      **
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
```

C. Hardness

At the beginning, I was confused how to separate the input information. The methods I was thinking about are "string", "vector", and "linked-list". Since I started to code late, I chose "string", which it's more familiar to me for saving data. I also hesitated about the input format, and decided to use the format like "cin" instead of "getline". But I still use "getline" when I need to collect the rest of the line.

I detected all of the gate type (like "and", "nand", and so on) originally, yet found out that we just need to separate "input", "from", and others. So the codes could be merged since the input detection was reduced, and the runtime decreased a lot.

Though I was stick out at the beginning, I figured out the solutions and completed the work in the end.

D. Suggestions

I think the PA document is clear enough, and the support materials are useful. Thanks for whom dedicated to this PA.