

EE6094 107501567 蔡書儀 PA1

● Readme

1. How to compile my file?

To compile my homework file, type the command below on the workstation:

```
g++ -std=c++11 107501567_PA1.cpp -o 107501567_PA1.exe
```

then we can get an executable output file `107501567_PA1.exe`.

```
[107501567@eda359_forclass ~/HW1]$ g++ -std=c++11 107501567_PA1.cpp -o 107501567_PA1.exe
```

Figure 1

2. How to execute

To execute my homework file, first type the following three commands:

```
./107501567_PA1.exe c17.isc c17.v
```

```
./107501567_PA1.exe c880.isc c880.v
```

```
./107501567_PA1.exe c6288.isc c6288.v
```

to get three output Verilog files `c17.v`, `c880.v`, `c6288.v`, then we can verify the correctness of the three output Verilog files using “ncverilog” commands:

```
ncverilog +access+r c17.v c17_testbench.v
```

```
ncverilog +access+r c880.v c880_testbench.v
```

```
ncverilog +access+r c6288.v c6288_testbench.v
```

if we see a heart on the screen, it means the Verilog files we generate are correct.

The figure of this step will be shown in the following section.

● Completion: All

Generating three Verilog files:

```
[107501567@eda359_forclass ~/HW1]$ ./107501567_PA1.exe c17.isc c17.v
Reading file
Input file name: c17.isc
Writing file
File name: c17.v
Module name: c17
End of writing file

[107501567@eda359_forclass ~/HW1]$ ./107501567_PA1.exe c880.isc c880.v
Reading file
Input file name: c880.isc
Writing file
File name: c880.v
Module name: c880
End of writing file

[107501567@eda359_forclass ~/HW1]$ ./107501567_PA1.exe c6288.isc c6288.v
Reading file
Input file name: c6288.isc
Writing file
File name: c6288.v
Module name: c6288
End of writing file
```

Figure 2 Verilog File Generation

1. `c17.isc` \rightarrow `c17.v`

```

1075051570@ed3350_forclass ~/HW1$ ncverilog -access+ c17.v c17_testbench.v
ncverilog: 15.26-s039: (C) Copyright 1995-2017 Cadence Design Systems, Inc.
Recompiling... reason: file `./c17.v' is newer than expected.
expected: Sat Feb 26 10:36:00 2022
actual:   Sat Feb 26 13:40:00 2022

  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:

      Modules:                Instances  Unique
      Primitives:             2          2
      Registers:               6          6
      Scalar wires:            5          -
      Initial blocks:          1          1
      Pseudo assignments:      5          5
      Simulation timescale:    lps

      Writing initial simulation snapshot: worklib.c17.tbv
Loading snapshot worklib.c17.tbv ..... Done
Verbs: Loading libkernel.tu152.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,RQUTE: Simulation is complete.
ncsim: exit

```

Figure 3 Verifying c17.v

2. c880.isc \rightarrow c880.v

[illegible]

Figure 4 Verifying c880.v

3. c6288.isc \rightarrow c6288.v

```

1071901567@do239: /src/cv2/~/HW11 ncsimlog_ncsimcsrc_v6280h_v6280h_testbench.v
ncsimlog: 15:28:20: Design: © Copyright 1995-2017 Cadence 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     1
Initial blocks:        1     1
Pseudo assignments:   32    32

Simulation timescale: 1ps
Writing initial simulation snapshot: worklib.v6280h.v
Loading snapshot worklib.v6280h.v ..... Done
*Verdi: Loading libscore_usj52.so
ncsim: run /usr/cad/cadence/ENCUSIR/cv/tools/inca/files/ncsimrc
ncsim: run
input pattern = 11100000000001111100001111111000 --> golden value = 11100111110011110001110011001100
your answer = 11100100111111111110001110011000
input pattern = 11111111111111111111111111111111 --> golden value = 100000000000000000000000011111111111111
your answer = 1000000000000000000000000000000000
input pattern = 00000000000000000000000000000000 --> golden value = 0000000000000000000000000000000000
your answer = 0000000000000000000000000000000000
input pattern = 11100000000000000000000000000000 --> golden value = 01001100100111011000000000000000
your answer = 01001110010011101100000000000000
input pattern = 00000111111100000000000000000001 --> golden value = 0000000100000101111011101110110000
your answer = 0000000100000101111011111010000
You're all correct!!!

**
**
****
*****
*****
*****
*****
*****
**
**
ncsim: *WARNING: Simulation is complete.

```

Figure 5 Verifying c6288.v

● Program Structure

Figure 6 is the flow chart of my program, first I open the file connected after “./107501567_PA1.exe” as input file, then read and store the content inside sequentially until the input file is ended. Finally I write the output file after all the process upon at the end of the program.

Since I don’t know the number of nodes in the circuits, I choose to use linked-list to store the node data instead of declaring a large array in order to save memory space. Figure 7 is the structure of the data structure I create:

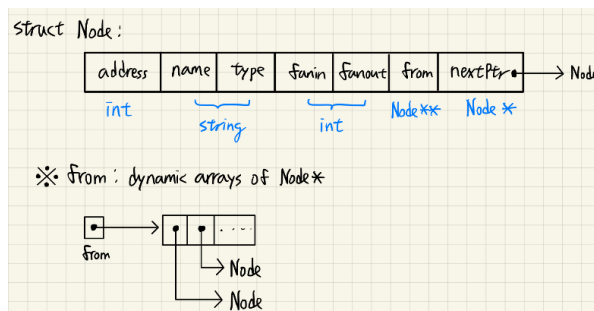


Figure 7 Data Structure

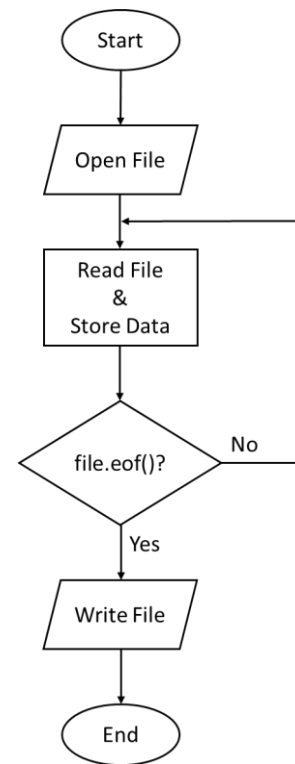


Figure 6 Flow Chart

1. address: integer variable which stores the address of the node
2. name & type: string variables which store the name and type of the node.
3. fanin & fanout: integer variables which stores the number of fanin and fanout of the node.
4. from: pointer points to a dynamic array of pointers of data type **Node**, which store the address of input nodes of current node so we can find where the input comes from easily. Size of array depends on the number of node’s fanin.
5. nextPtr: pointer to data type **Node**, which stores the address of next node in linked-list (NULL if no next node).

● Hardness

Since I haven’t learned data structure and algorithm before selecting this course, I was confused about how to read and fetch useful information from input files efficiently. Although I can declare a very large array for data storage, but it’s a stupid way which cause a lot of memory space waste, so I decided to learn data structure by doing and finally I create a linked-list of nodes for my program.

The largest problem was about the unfamiliarity of pointer, I stuck at this kind of problems for a while. My program can work correctly on my own PC, but it had segmentation fault when I test it on the workstation and I didn't know why and how to solve it. Thanks to my friend's help that I found out the mistake which was about the initialization of pointer.

- **Suggestion**

N/A