EECS 478 Fall 2014 Project No. 3

The University of Michigan, EECS Department
EECS 478: Logic Synthesis and Optimization, Fall 2014
John Hayes

Project 3: Functional Simulation

Distribution Date: Friday November 21st 2014
Due Date: Wednesday December 10th 2014 at 23:55
Submit Online on CTools → Assignments → Project 3

INSTRUCTIONS:

- You must abide by the Engineering Honor Code. All work submitted should be work done by you and by you alone. If you are unsure about the level of collaboration, please consult John or Shaobo.
- When you are finished, collect the following final deliverables:
  - All `.cpp` files and `.h` files that you use, including any extra files that you added.
  - `Makefile` - your program **must** compile.
  - `report_{uniquename}.pdf` or `report_{uniquename}.doc`, where `uniquename` is your uniquename.
- Create a tar ball with all your deliverables. To do this:
  - Create an empty directory called `EECS478P3_{uniquename}`.
  - Copy all your deliverables into this directory.
  - Type `tar -cvf EECS478P3_{uniquename}.tar EECS478P3_{uniquename}`.
  - Type `gzip EECS478P3_{uniquename}.tar`.

---

1 Overview

This assignment illustrates functional simulation of digital circuits. In particular, using an infrastructure similar to Project 2, you will do the following:

- Find a topological ordering of a circuit.
- Simulate a circuit and find the corresponding output for a given input vector.

2 Provided Files

The provided tar ball has the following files:

- `truthTable.h` and `truthTable.cpp`
- `node.h`
- `circuit.h` and `circuit.cpp`
- `main.cpp`
- `Makefile`

Feel free to modify any or all of these files to suit your needs. Please make sure that your final binary is called `project3`.

2.1 truthTable.h and truthTable.cpp

These two files store the underlying infrastructure for representing a truth table. Each entry stored represents a cover for the logic function.

2.2 node.h

This file defines a logical element (node) or a circuit. It includes a truth table, a list of fanin nodes, and a type (PRIMARY_INPUT, PRIMARY_OUTPUT, INTERNAL, ZERO_NODE, ONE_NODE).

2.3 circuit.h and circuit.cpp

These two files define a circuit, which is defined as a collection of nodes.

2.4 main.cpp

This file contains the general wrapper on how to use the executable. There is a `-help` option built in. To see the usage, type `./project3 -help`.

2.5 Makefile

This file compiles and creates the final binary for this project. You are allowed to modify this file and add any optimization flags of your choice. To compile your program, type `make`.

3 Getting Started

Download the tar ball `eecs478p3.tar.gz` from CTools. To decompress the tar ball, type

```
tar -xvf eecs478p3.tar.gz
```

into a directory of your choice.

This will create a directory called `eecs478p3`, and will include the necessary files for your program. In that directory, compile the program by typing `make`. To run the program, type `./project3`. This will display the usage options.

You are allowed to do your development on any platform. However, your code will be tested and evaluated strictly on the CAEN Linux machines. You are allowed and encouraged to reuse any code and blif files from Project 2.

Note that this project is more open-ended than Project 2. You should feel comfortable with adding member functions to classes, implementing algorithms and using the STL. For more information on the STL, you can go to `http://www.sgi.com/tech/stl/` or `http://www.cppreference.com/wiki/`. If you want to use other libraries or advanced data structures that you did not write, consult with John or Shaobo first.

4 Tasks (100 points)

This section details the different tasks you must complete. Be sure and test your code *thoroughly* before submitting. The more testing you do, the more confident that your code is correct. If you plan on making substantial changes to the code base, such as the underlying infrastructure, please consult with John or Shaobo first.

4.1 Topological Sorting (40 points)

For a given blif file, generate a topological ordering of the gates. At the end of the function, print the inputs, gates, and the outputs in topological order. For the gates, use their output as their name.

For example, your output should look like the following:

```
*** Topological order:
i[1]  i[0]  a[0]  i[2]  a[1]  z
```

When outputting your topological ordering, separate each node name with a single space (' '). Note that your topological ordering algorithm must support an arbitrary gates, where each gate can have an arbitrary number of inputs.

4.2 Functional Simulation (40 points)

For a given blif file and starting logic values for each of the inputs, propagate the values to the output(s) of the circuit. For instance, given a 2-input AND gate, and the inputs **0** and **1**, then the output should be **0**. Note that you will be implementing the same functionality as `simulator`, which was given to you in Project 2.

To perform functional simulation, you will need to set starting values for each of the primary inputs. These inputs will be given in the following format:

```
<input name> <logic value>
```

For instance, for a circuit with primary inputs `i[0]` and `i[1]`, the input file could look like:

```
i[0] 0
i[1] 1
```

The output of the simulation function should have the following format.

```
*** Outputs:
z[0] = 1, z[1] = 0, z[2] = 1
```

Once you have your function working, modify `main.cpp` accordingly in order to be able to run this from the command line. For more information about the usage, run `./project3 -help`.

When implementing functional simulation, you will need to parse this file and store the relevant information accordingly. You will not need to worry about improperly formatted files or do any type of error checking. When simulating, you may assume that all gates are either AND, OR or NOT gates. However, they may have an arbitrary number of inputs (except for the NOT gate, where there is only one input). You may also assume that the truth-table rows of the blif nodes corresponding to the gates are completely enumerated and no "don't care" values are used. For example, a blif node corresponding to a 2-input OR gate has the following format:

```
.names x y z
01 1
10 1
11 1
```

instead of the usual compact form:

```
.names x y z
-1 1
1- 1
```

4.3 Report (15 points)

You will need to write a one-page report using a 12-point font in a .pdf or .doc format. At the top of the front page, state your name and uniquename. Include everything in the report that the previous tasks have asked. Also, include anything of interest that you think might be helpful when evaluating your submission.

4.4 Code Readability (5 points)

Your code does not need to be perfectly clean, but should be well-structured and clear. The following are some helpful guidelines to writing readable code.

- Use proper indentation when declaring while/if/for/functions.
- Avoid going over 80 characters per line.
- Write a 1-2 line comment for explaining non-standard coding syntax.
- Write a short comment for explaining what each function (or sub-function) does.