Stony Brook University

Project 1: Warm-up and 5-Value Logic Simulation

Issued: 1/31/16, Due: 2/18/16 2:30PM

1. Introduction

During this semester, we will complete projects in C++ using a library that lets you manipulate digital circuits at the Boolean gate level. The goal of *this project* is to acquaint you with the basic structures of the library and its API, and to demonstrate this by adding five-valued logic simulation capability into the software.

For this project, you will use the computers in the Graduate CAD Lab (Light Engineering room 183). You can either work in the lab, or connect to these machines from home using SSH. (Please see the Lab Computer Overview for information on this.)

In principle it is possible for you to work with this code on your personal computer, but you will need to:

1. Figure out how to make the system and the required libraries work on your machine. The project relies on a parser I wrote using the open source tools Bison and Lex. It *should* be possible to make everything work under Windows, but it may take some effort on your part to make it work.

2. You also are responsible for making sure that everything compiles and works correctly on the lab Linux computers, because **this is where I will be evaluating and grading your code**. If you do decide to use your personal computer, make sure you save time at the end to test and debug on the lab machines.

For these two reasons, I strongly encourage you to use the lab computers[1]. If you decide to use any other computer, then you do so at your own risk.

You will be working alone on this project. Please do not discuss details with each other or look at each other's code.

We will be using Piazza for discussion related to this project. I am providing you with a lot of information, but almost certainly there will be some useful things I have forgotten. Part of the goal of this is for you to learn how my code works (since we will continue to use the same basic code base for other projects), so please feel free to ask lots of questions, and check Piazza to see if others have had similar queries.

2. Getting Started

I am providing you with the following things:

- This project description (on Blackboard)

- Lab Computer Overview: this is a short overview for using the lab computers (on Blackboard)

[1] That said, if your personal computer is a Mac or Linux machine, then it's reasonably easy to make it work, and I can give you some basic help with that. If you are using Windows, I cannot help you.

- Baseline software. This software will take care of all file input/output, and will setup all the data structures for you. You can find this on the lab computers at `/home/home4/pmilder/ese549/proj1` (directions to copy it to your home directory are below).

- API documentation for the Circuit and Gate classes. You can view this documentation as a PDF (apidoc.pdf), or as a nice interactive webpage (which you can download as apidochtml.zip, unzip, and then browse, starting from index.html). Both are available on Blackboard. (This documentation is automatically generated from the source using a tool called Doxyegen.)

I recommend you start by first setting up a private work directory (using the directions in the Lab Computer Overview), and copy the project code into it. Type the following:

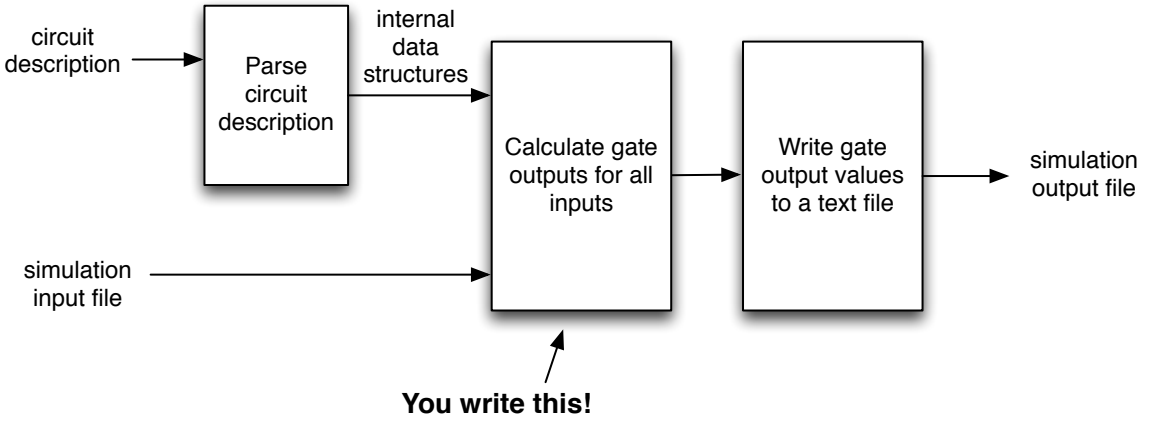
```
cd
mkdir ese549work
chmod 700 ese549work
cd ese549work
cp -r /home/home4/pmilder/ese549/proj1 .
```

3. Overview

Your system will take as input:

- a description of a circuit at the Boolean gate level (see Section 5 for a description of the file format)
- a text file with a list of desired inputs to simulate (see Section 6)

It will simulate the circuit using five-valued logic (see Section 4). As output it will produce a text file giving the circuit's outputs for the provided inputs (see Section 6).



You are given code to parse in the circuit, setup the data structures, and read in the desired inputs. Then, you will provide the code to compute all gate output values for the given inputs. Then, you will use code provided to you to store those outputs to a text file.

4. Five-Valued Logic

We will be using "5 valued logic." This means that each signal in the circuit can have one of five values: 0, 1, X, D, or D'.

- 0 and 1 are normal logic values.

- The X symbol represents an "unknown." If the input to a gate is X, the output may or may not be X. For example, AND(1,X) = X, but AND(0,X) = 0.

- Lastly, we can define D and D' as logical values that distinguish between "correct" and "faulty" behavior. Logic value D means "logic 1 in a good circuit, but logic 0 if the circuit is defective." D' is the complement: "logic 0 in a good circuit, but logic 1 if the circuit is defective." For example, NAND(1, D) = D', and NOT(D') = D.

In this project, you will not be simulating faults within the circuit's logic, but by making your logical simulation capable of correctly operating on D and D' values, we will be able to build on this for future fault simulation.

One last note: inside of our program, we will use the character B to represent D'. This way, we can represent each logical value with a single character: 0, 1, X, D, or B.

5. Bench File Format

For test circuits, we will be using designs from the ISCAS-85/89 benchmark library. These designs are provided in "bench" file format. For example (tests/simple.bench):

```
INPUT(A)
INPUT(B)
INPUT(C)
INPUT(D)
INPUT(E)
OUTPUT(F)

G=AND(A, B)
H=AND(C, D)
I=AND(G, H)
F=AND(I, E)
```

This describes a circuit with five inputs (A, B, C, D, E), one output (F), and four AND gates. Legal gates are NAND, NOR, AND, OR, XOR, XNOR, BUFF, and NOT. BUFF and NOT gates take one input. XOR and XNOR gates take two inputs. NAND, NOR, AND, and OR gates can take in any number of inputs.

The provided codebase contains a parser that is able to read in a bench file and construct a data structure to represent the circuit.[2]

The tests/ directory contains circuits you can use to test. It contains several very small .bench files for you to use for getting started; for these you can write your own input files

[2] If you are curious about the parser, take a look at parse_bench.y and parse_bench.l, and read documentation for Lex and Bison. This is just for your information if you are curious; you will not be modifying or building parsers like this in this class.

and check the result by hand. Once you are comfortable that your system works on these very small inputs, I have provided four other circuits (c17, c432, c499, and c5315) with larger sets of test inputs (.refinput files) and the desired output files (.refout files).

6. Simulation Inputs and Outputs

We will be using a very simple file format for simulation inputs and outputs. For example, simulation inputs for the circuit described above (simple.bench) would look like:

```
0D101
10110
XD11X
111B1
```

Each line represents a full set of inputs for one moment in time. (Thus there should be five characters per line in this example because the example circuit above has five inputs.) Since this example contains four lines, we are asking the system to simulate four different vectors and simulate the response to each. Recall that valid logical values are 0, 1, X, D, and B (and that B represents D'—see Section 4.)

The simulation output will be stored in the same format: each line contains the system's output for one input, and each line will have one character for each output of the circuit. The simple.bench circuit described above has one output, so if I simulate it with this input file, I should get the following output:

```
0
0
X
B
```

Also note, if you see the character "U" in the output, this represents an *unset* logic value. (The system uses U so it can distinguish between signals with unknown value "X" and signals whose values may be possible to determine, but has not been determined yet "U".) When the system begins simulating, it sets everything to U. ***If you are seeing U in the output after you write your simulation code, it means that you are not correctly setting some gate values.***

The code provided with this assignment will handle the file I/O operations.

7. Data structures and APIs

The baseline code contains two C++ classes to represent a digital circuit (at the Boolean gate level) and the gates themselves. The classes are called Circuit and Gate. The best way to understand how these work is to read the accompanying API reference documentation, which explains the structure of these classes and how you can work with them, and of course, read through the code: ClassCircuit.cc, ClassCircuit.h, ClassGate.cc, and ClassGate.h.

Please do not change or edit these files at all. If you think anything useful is missing from these classes, please let me know and potentially we can add it for everyone.

8. Main function

The main() function in main.cc contains all of the code for input/output and setting up the data structures. Please look at this function carefully. You will see a blank space in the middle labeled "Your code here" for you to write your function. Write your code in this place. If you choose to use any extra functions (which is probably a good idea), write the prototypes in the section labeled at the top of the file, and put the functions themselves in the section at the bottom of the file. ***Please keep your code in the requested regions, because it will make it much easier for me to read and understand what you did.***

I have included a couple of ideas in a comment in main.cc to help you think about some possibilities for getting started.

9. Running and testing your code

We are using a Makefile to make it very easy to compile your code. To compile, simply type:

```
make
```

This will produce an executable called logicsim. This executable takes in three parameters at the command line, specifying the circuit (bench file), the simulation input file, and the file in which to save your simulation output. For example, you can run:

```
./logicsim tests/c17.bench tests/c17.refinput myc17.out
```

This will simulate the circuit in tests/c17.bench with the inputs in tests/c17.refinput. It will store the output values in myc17.out. I have provided a (correct) reference output for c17. After running the tool, you can check that your myc17.out file matches the reference output by typing:

```
diff myc17.out tests/c17.refout
```

This command will show you any differences between the two files. (If it reports nothing, the files are identical.)

The tests/ subdirectory contains a few circuits to work with, but you can easily construct your own in the proper format (this is especially important when getting started). I also provided four circuits with reference inputs and outputs for you to use and compare with.

10. Report

Write a very short (1 page) report describing:

- How your solution works
- How you verified your solution
- Any major problems you had in completing the assignment

You will submit this report as a PDF on Blackboard (see below). Please do not submit other file formats—if you are not able to make a PDF, then print it out and submit in class.

11. Code and Report Submission

Following the directions above, your code should be entirely contained in the main.cc file. Please submit only this file—do not include the other source files from the project. Please make sure you include comments that explain what you did. Also place a comment at the top of the file with your name.

You can submit your code and the PDF of your report on Blackboard. On Blackboard go to Assignments → Project 1.