Project GF2: Software
Second Interim Report
Software Design Team 1

George Ayris	James Glanville	**Andrew Holt**
gdwa2	jg597	**ah635**
Emmanuel	Emmanuel	**Emmanuel**
Parser & Scanner	Scanner & Parser	**GUI**

06 June 2013

Contents

1 System Overview

1.1 Logic Simulator

A logic simulation package has been developed, allowing a user to define a logic circuit using a specialised, custom designed definition language and to run this circuit in a simulation, observing the output at various points in the circuit. The circuit may be run for a number of cycles, selected by the user. The simulation may then be run for some further cycles, or restarted from scratch. There is also an option to run the circuit continually, observing the output in a scrolling fashion.

The user may interactively set the value of any switches defined in the specification value and observe the effect of this change in the circuit as it continues to run.

If the user decides they would like to observe the signal at a point previously un-monitored, they may add a new monitor to the circuit. To avoid too many monitors being displayed together and causing difficulty in reading the traces, a monitor may also be removed from the circuit.

The definition file is selected from a file select dialog, allowing multiple circuits to be tested during a single session.

The logic simulator provides helpful text output at the bottom of the screen to warn the user of errors and keep track of how many simulations have been run.

1.2 Software Structure

The software is structured across a number of files in the `src` directory. The main file is `logsim.cc`, which sets up the simulation package and calls the classes in the other files as required.

When a new file is loaded, it is scanned and parsed, using objects derived from the `scanner.cc` and `parser.cc` files respectively. If the parsing is successful (the definition file contains no errors), the circuit is created using classes in the `devices.cc`, `devicetable.cc`, `monitor.cc`, `names.cc` and `network.cc` classes. Each of these files defines a class of the same name to deal with a different part of the logic simulator.

`devices.cc` deals with the creation of different devices in the circuit and the specification of these devices (e.g. the number of inputs), along with setting the values of switches and calculating the output of each device for a clock cycle of the circuit.

`devicetable.cc` defines a data type for associating the devices with more meaningful names, so that the devices may be more easily access.

`monitor.cc` deals with the monitors in the circuit: creation and removal of monitors, as well as giving the signal level of each monitor point at each clock cycle.

`names.cc` translates between the internal representation of each component through an `id` and the more user friendly name given to each device, monitor, switch or clock.

`network.cc` manages the network of devices and components by creating the device outputs as required and defining the connections between device outputs and inputs.

The final file required is `gui.cc`, which operates the user interface. This is created using wxWidgets, a cross platform gui toolkit. The `MyFrame` class creates the gui and handles user interaction events and the drawing of the traces. It can make modifications to the circuit as the user specifies using the graphical options.

The whole project follows the object oriented programming methodology. This means that the code is very modular, and the different classes are not dependant on the other ones. For example, a completely different interface could be developed and use the same back-end software. Equally, a different parser could be implemented and, provided all the features were implemented in some way or another, the whole system would be identical to anyone not looking inside the parser class.

This methodology is ideal for multi-programmer projects such as this as it allows independent development of different classes with no detailed knowledge of how the other programmers are designing and implementing the internals of the other classes.

2 Teamwork Overview

The teamwork was organised such that James would lead development of the scanner and lead testing of the parser. George would lead development of the parser and testing of the scanner. I was responsible

for the gui programming.

This setup worked well for the first stage of the project, allowing each of us to become familiar with a different aspect of the program. A downside of this was that it was difficult to talk through design decisions with each other since our experience and growing expertise for the project lay in different areas. It is often very helpful to talk through the suggested design before implementing it as it both clarifies in your own mind as you explain it and often the other person will think about some unconsidered problem. However, since I know almost nothing about how the parser operates, and similarly George knows little about the GUI code, it is very hard to properly discuss what we are doing.

When the maintenance task was released, each of us took on one of the new tasks. These tasks did not take too long, but there were several changes required to various parts of the code after the week 3 demonstration.

3 Software Development

4 Test Procedures

5 Conclusions and Further Work

A Software Listings

B Example Circuits

C Definition File Specification

D System User Guide

E File Description