University of Cambridge

Engineering Tripos Part IIA

Girton College

GF2 Software: Logic Simulator
Final Report

Clinton N. Igwegbu (cni22) Team 14 June 04, 2019

Introduction

This project consisted of the design and implementation of a logic simulator. This involved formulating a logic description language to describe digital circuits and writing a program that allowed a user to monitor the response of various ports in the circuit to different input waveforms. A graphical interface allowed the user to specify various properties of the logic circuit and in particular select the ports to be monitored. The simulator displayed simulated signals on a canvas. The project proceeded in two stages, a first implementation and a maintenance phase. After the first implementation, errors and possible improvements were noted. These improvements and additional customer requests were incorporated in the maintenance phase. Testing in this phase produced satisfactory performance with little to no errors noted.

1 Function of Logic Simulator

The function of the logic simulator will be decomposed primarily into the functions performed by the scanner, parser and GUI (Graphical User Interface) as well as the definition file which describes the circuit to be simulated.

1.1 Definition File

The definition file describes a logic circuit using a set of keywords (vocabulary) and a prescribed syntax (grammar). The allowable vocabulary and grammar used to create a definition file were described using the Extended Backus-Naur Form (EBNF) in the first interim report. The definition file is made up of: a device list, which lists all the devices in the circuit with their names and essential properties; a connection list which forms the connections between devices by stating which device output ports and input ports are linked; and a monitors list which lists the ports in the circuit whose activity will be monitored in the simulation. The simulation displays the activity at each port in the monitors list as a discrete binary waveform.

1.2 Scanner

As stated in the section above, the definition file includes keywords (vocabulary) and syntax (grammar). It is the scanner's job to search for keywords within the definition file. Each keyword picked up the scanner is translated into a symbol which is passed onto the parser. A symbol is the basic unit which the parser can act upon.

1.3 Parser

The parser checks the syntax of the definition file by comparing successive symbols received from the scanner, and then it processes the symbols into something meaningful. In other words, it constructs a 'sentence', checks that it makes sense, and then carries out the instruction issued by the sentence.

The parser uses logic level 1 (LL1). This means that it can only process one symbol at a time. In other words, once it has received at least one symbol, it must be able to check if correct syntax of the logic description language has been obeyed after receiving only one more symbol. If the syntax has been disobeyed it raises a syntax error. If the syntax has been obeyed but the constructed 'sentence' has no meaning it must raise a semantic error.

Once the parser has checked that the sentence has meaning it builds the logic circuit.

1.4 GUI

The GUI is the user's portal to the functionality of the simulator. By the time the GUI starts up the logic circuit has already been built. The GUI allows the user to execute the coer simulator commands: run for N cycles, continue for N cycles, create monitor, zap monitor, set switch, and quit the simulation. Additionally it allows the user to interact with the simulated waveform by panning, zooming and viewing it in 3D among other features. Refer to the user guide in appendix C for an overview of the GUI's features.

2 Structure of Simulator Software

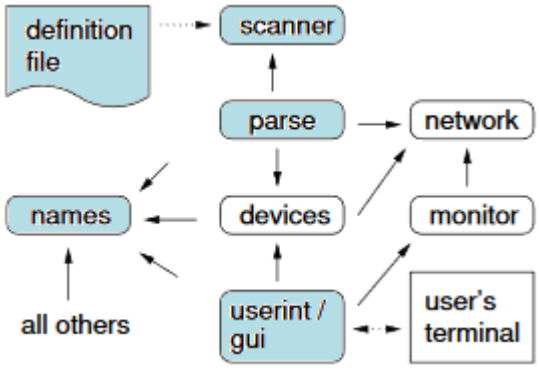

Figure 1: Top-level software structure

The main/parent file in the logic simulator is logsim.py. It is the first file run upon start-up and all functionality is directed through this file. The software is very modular such that much of the functionality described in section 1.1 takes place in the form of class methods and parameters which are stored in other files:

userint.py

Allowed the logic simulator to be run from the command-line though it was replaced by gui.py in the student implementation.

scanner.py

Executes all scanning functions as decribed in section 1.2.

parser.py

Parses the definition file then builds the circuit via network.py, as described in section 1.3.

gui.py

Constructs the GUI as described in section 1.4. As I wrote this file, I have described it in more detail in section 4.

devices.py

Manages the Device and Devices classes. The Devices class maintains a dictionary of all devices present in the circuit such as logic gates, clocks, signal generators and switches. The Device class manages the properties of each device such as switch state, output ports and input ports. The functionality for managing these properties comes in the form of class methods.

network.py

Forms connections between all devices and contains methods to manage the signals at each port.

monitors.py

Maintains a dictionary of all the signals being monitored. Each port in the monitors dictionary can be referenced by a device id, and also an output id if the corresponding device has more than one output port. The signal recoreded at each port is stored as a value in the monitors dictionary and is used to plot the waveforms in the simulation.

names.py

Houses the Names class which stores a list of keywords in the logic description language.

3 Evaluation of Teamwork Approach

3.1 Leading up to Second Interim Report

It was decided to divide the work according to our interests and the skills we each wanted to learn. Admittedly this led to an uneven division of the workload though the team members agrees that we would try to help out where we could. As a result Steve Zhang implemented the names and scanner files, David de Oliveira the parser, and I the GUI. The uneven workload division meant that some tasks had been rushed in the first implementation.

It was anticipated that there would not be much time to carry out this project so a series of intermediate deadlines were set as shown in the Gantt chart in appendix A of the first interim report. Debugging of the scanner took longer than expected and so that little time was left to test the fully integrated system. Consequently there were some uncorrected bugs and obvious improvements that could have been made to the system as detailed in section 4.1.

3.2 Maintenance Phase

It was decided that David should make the changes to the parser necessary to incorporate the signal generator since he implemented the parser in the first stage. I implemented the 3D signal feature, as the designer of the GUI. Steve was reallocated to the implementation of the Chinese translation feature. The workload was more evenly divided in the maintenance phase, an improvement on the approach followed in first implementation. This allowed enough time to test the system more thoroughly and therefore rectify the bugs found in the first implementation as well as new bugs that came along during maintenance.

3.3 Possible Improvements on Teamwork Approach

A more even division of the workload should have been sought after in the first implementation, particularly as the deadline approached while the simulation program was still at a sub-optimal stage of its development. Some members of the team did still help out where they could on parts not assigned to them, as it was clear we each had our own strengths and weaknesses.

4 Description of gui.py

4.1 First Implementation

This file is split into a number of classes. The Gui class acts as a parent of all other classes in the file. All components of the user interface are integrated into this class. The tabbed layout in figure 2 describes the structure of the GUI concisely, with class names in gui.py closely matching corresponding features in the GUI. The interface was designed using widgets from the wxpython package. The simulation tab, definition tab and the menu bar form the top level of the application. In the first software release the definition tab had a text box to display the definition file and a file directory tree to select a new file. The plan was to give the user option to run a different definition file by selecting it in the file tree. However there was not enough time to implement this optional extra so the file manager was removed in the maintenance phase. The lines of

code that created the file manager have been commented out in gui.py so that this task can be completed later should there be interest to do so. The simulation tab contains the main features of Logic Simulator. This includes a panel of controls, a signal panel which holds the canvas that displays the monitored signals and a status bar which displays messages and instructions for the user. These signals were drawn with the OpenGL library. In the first implementation phase, signals were displayed in 2D. The menu bar, consisted of File, Edit and View menus. The File menu gave the user the option to exit the application, read about the application or open a new definition file. The Open feature simply opened the Definition File tab which had the file manager on display with the already open definition text file. The Edit menu contained basic copy, paste, delete and select functions for editing the text file. The View menu gave the user the option to show/hide the control panel or the file manager.

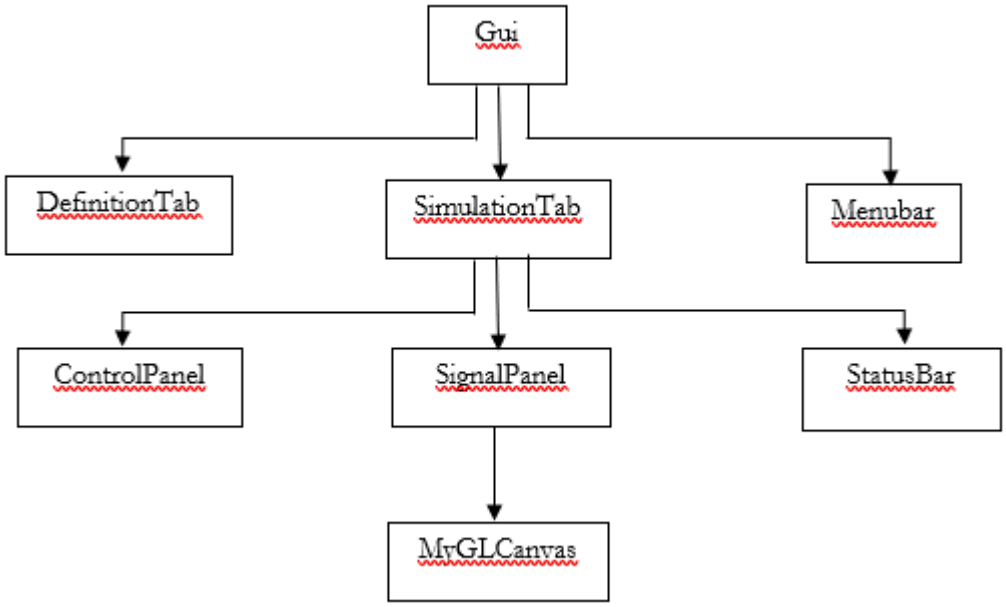

Figure 2: gui.py structure

4.1.1 Bugs

- During testing it was noted that creating a new monitor and then selecting the Continue feature led to poorly plotted signals; the plots from two different monitored signals had a tendency to merge. This bug went unnoticed by the project demonstrators

4.1.2 Recommended Improvements

- The project demonstrators noted that there ought to be numbered axes though these were not included.
- The user was given plenty flexibility to pan the signals offscreen. It was suggested that panning should be restricted so that the signals do not go out of view.
- Bring signals back in view upon pressing the run button.
- Include a reset button to return the simulation to default values and reset the monitors and switches to startup values.
- Change the default number of simulation cycles; 0 is not ideal.
- gui.py was noted to have very little documentation and to be very PEP8 incompliant.

4.2 Maintenance Phase (Final Implementation)

This phase consisted of adding the new features that were requested by the client to the GUI as well as fixing bugs and making improvements to the logic simulator.

4.2.1 Improvements and Bug Fixes

- 'Continue' feature bug corrected.
- In 2D view mode user can now double-click left mouse button to reset signal view i.e. restore signal to bottom-left corner, and reset zoom.
- Numbered axes included.
- Docstrings and commentary added to gui.py and made more comprehensive.
- Reset button included as suggested.
- Default number of run cycles was set to 10.
- Panning has been restricted so that user cannot move signals completely offscreen.
- Signals automatically brought within view upon resize of simulator window.
- As there was little time to deal with file management, this optional extra was removed. The file tree in the Definition File tab was removed though the final code still contains commented-out artefacts. These were left in anticipation of future additions to the GUI.

4.2.2 Other Changes

- 3D view feature added with a button to toggle between the two view modes.
- Due to the greater degree of freedom in 3D view mode it was decided to allow the unrestricted panning though user can double-click left mouse button to bring the signals within view.
- Included option to translate GUI into Chinese.

5 Description of test procedures adopted

5.1 Parser and Scanner

- Pytest files test_parse.py, test_network.py and test_devices.py were modified during maintenance to account for inclusion of the signal generator. All 60 tests passed.

5.2 GUI / Integrated simulator

The GUI and integrated simulator were tested by checking all buttons and controls under different scenarios as described below:

- Tested simulator with four different definition files such that all possible devices were tested. The file johnsonRing_defiFile.txt was modified during maintenance to incorporate the new signal generator device.
- The GUI did not run when first tested with johnsonRing_defiFile.txt. It was noted that the presence of single output devices in the definition file caused errors. The problem has been rectified.
- Tests with fullAdder3Bit_defiFile.txt showed that switch states were not always properly set on startup. The problem has been rectified.
- All basic GUI controls were tested for expected functionality: run, continue, toggle view, creating and zapping monitors, setting switches, setting monitors, etc. No issues found.
- checked for error messages in terminal:
- print to terminal whether monitor make/zap has been successful
- The maintenance phase included an option to start the logic simulator in Chinese. All GUI buttons and status bar messages were checked to ensure they had been translated correctly. It was noted that the status bar run message had not been translated. The problem has been rectified.

6 Conclusions and Recommendations for Improvements

At the end of the maintenance phase, the team is satisfied with the performance of the logic simulator.

The simulator could be improved by giving users the option to edit, save and open new definition files within the GUI. The simulator could also be improved by allowing users to build logic circuit using drag and drop graphical interface, instead of logic description language.

7 Appendices

```
GRAMMAR         = DEVICELIST , CONNECTIONLIST , MONITORLIST ;

DEVICELIST      = "DEVICE_LIST" , ":" , DEVICE , ";" , { DEVICE , ";" } , "END" ;
DEVICE          = DEVICETYPE , NAME , [ PROPERTY ] ;

DEVICETYPE      = GENERATOR | MULTIINPUTLE | OTHERLE ;
GENERATOR       = "CLOCK" | "SWITCH" | "SIGGEN" ;
MULTIINPUTLE    = "AND" | "NAND" | "OR" | "NOR" ;
OTHERLE         = "XOR" | "DTYPE" ;

NAME            = letter , { letter | number } ;

PROPERTY        = NUMREPETITIONS | INISTATE | NUMINPUT | WAVEFORM ;
NUMREPETITIONS  = number ;
INISTATE        = "OFF" | "ON" ;
NUMINPUT        = number ;
WAVEFORM        = ( "0" | "1" ) , { "0" | "1" } ;

CONNECTIONLIST  = "CONNECTION_LIST" , ":" , CONNECTION , ";" , { CONNECTION , ";" } , "E
CONNECTION      = OUTPUT , "->" , INPUT ;

INPUT           = NAME , "." , ( GATEINPUT | DTYPEINPUT ) ;
GATEINPUT       = "I" , number ;
DTYPEINPUT      = "DATA" | "CLK" | "SET" | "CLEAR" ;
OUTPUT          = NAME , [ "." , ( "Q" | "QBAR" ) ] ;

MONITORLIST     = "MONITOR_LIST" , ":" , OUTPUT , ";" , { MONITOR , ";" } , "END" ;
```

```
number          = digit , { digit } ;
letter          = "A" | .... | "z" ;
digit           = "0" | .... | "9" ;
```

1. Test definition files and corresponding circuit diagrams:
   (a) comparator_defiFile.txt
   (b) comparator.png
   (c) mod5Counter_defiFile.txt
   (d) mod5Counter.png
   (e) fullAdder3Bit_defiFile.txt
   (f) fullAdder3Bit.png
   (g) johnsonRing_defiFile.txt
   (h) johnsonRing.png
2. Localisation files for translation into Chinese:
   (a) locale/zh/zh_CN.mo
   (b) locale/zh/zh_CN.po
3. Pytests for all the modules:
   (a) test_scanner.py
   (b) test_parse.py
   (c) test_network.py
   (d) test_names.py
   (e) test_monitors.py
   (f) testFiles/ – This folder contains all the files required by test_parse and test_scanner
4. *ebnfSpec.txt* – The specification for the newly defined logic circuit language.
5. *logsim.py* – The main program; creates instances of *Names*, *Scanner*, *Parser* and *gui*, and launches the application.
6. *names.py* – Module which maps indices to position in a list of strings, which are either added to the list by one of the other modules, or by the user definition file.
7. *scanner.py* – Contains the Symbol class and Scanner class. Reads in a file and creates different ymbols depending on what is read.
8. *parse.py* – Module which parses the definition file, checking for syntactic and semantic errors. Builds the internal circuit representation by calling methods from other classes.
9. *devices.py* – Contains the Device and Devices classes. Contains methods to make a Device, and holds a list of all the devices made.

10. *network.py* – Contains methods which map device outputs to device inputs, and also to run the simulation for a specified number of cycles.
11. *monitors.py* – Contains methods which set outputs as monitor points, and records the signals.
12. *gui.py* – Contains the gui class which creates the window object for the application. Within the GUI, there is an OpenGL canvas, onto which the monitored signals are drawn.
13. *userint.py* – Command line interface application for running logsim.