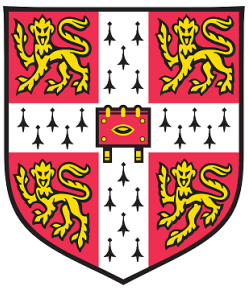
CAMBRIDGE UNIVERSITY ENGINEERING DEPARTMENT

Part IIA - GF2 Software
FINAL REPORT

Logic Simulator



Name: Jonty Page
College: Pembroke
Date: 04/06/2019

Abstract
A client has contracted a software development company to create a logic simulation program. This program should be able to pass, simulate and display the output of logic circuits and meet a detailed set of requirements given by the client. The program is divided into eight functional modules, four of which are the responsibility of the sub-team to develop, integrate and test. After three weeks of development, a change to the specification was introduced by the client which also have to be implemented by the sub-team. Overall, an out-of-the-box solution was developed which met all of the clients requirements, was easy and intuitive to use and robust to errors in execution and input.

1 Introduction

A software company was contracted by a client to produce a logic simulation program to enable the operation of both combinatorial and clocked logic circuits to be studied by a computer prior to their implementation in hardware. The client provided a detailed specifications document listing the requirements of the software and this can be found in appendix A of the CUED GF2 project main handout [1]. The general function of the software should be:

1. The program reads in a text file (".txt") which fully defines the logic elements, the connections between them, the generators for inputs and the signals to be monitored.
2. The user is then able to control the initial state of switches, add/remove points which are monitored and set the number of clock cycles for which the simulation should be run at will.
3. The network is executed and the output at each user-defined monitoring point is displayed.

There are two methods in which the user can interact with the software, either through a command-line user interface (UI) (which was already fully developed and functional) or through a graphical user interface (GUI) (which was the responsibility of the sub-team to develop). In initial implementation, six logic elements and two generators (a clock and a switch) were to be supported. The circuit should be read into the software through a circuit definition file written in a logic description language created by the developers. The logic description language was to be described formally by EBNF notation and adhere to the specifications laid out in part A3 and A6 of the client requirements document. The GUI was to have the same functionality as the UI and full error-checking was to be applied to user command inputs in-order to maintain robustness. Finally, there was to be no limit to the size of the circuit, number of monitoring points or devices except that implied by the memory of the computer.

2 Software Structure

2.1 Overview

The software is split into eight functional modules, four of which were already developed and four of which were developed by the sub-team. These modules are described in Table 1 below:

Module Name	Description
Names	Maps variable names and string names to unique integers
Scanner	Translates sequence of characters in definition file into sequence of symbols.
Parse	Analyses syntactic and semantic correctness of scanned definition file.
Devices*	Contains routines for creating, configuring and querying devices.
Network*	Manages connections between devices and execution of the logic circuit.
Monitors*	Records signal levels at designated monitor points for each simulation cycle.
Userint*	Handles the text-based user interface.
Gui	Handles the graphical user interface

Table 1: Overview of eight functional modules within the Logic Simulator software. **indicates modules already fully developed.*

A more detailed description of the functionality of each module is described in the subsequent sub sections. These modules are wrapped togther using a top level python program *logsim.py*

which is the program which should be run to start the software. Also, a ninth module was added by the development team, *errors.py*, in order to manage error reporting through an object which is populated when an error occurs.

The general workflow of the backend of the software is first four objects are created - Names, Devices, Network, Monitors. These objects store the circuit variables and contain all the information required to obtain the output signal for each defined monitoring point required by the GUI. The circuit definition file selected is then ran through the Scanner, which populates the Names object with unique IDs for each variable/string names. This Scanner object is then ran through the parser, alongside the Names, Devices, Network and Monitors objects, the file is checked for sytactic and semantic errors and if none arise, the circuit variables are loaded into each of the four objects as required. When the GUI or UI is asked to execute the circuit, or modify one of the circuit variables such as the state of a switch or the monitor points set, the interfaces make calls to each of the objects and communicate with the variables as needed. The figure below (Figure 1) shows graphically communications between each module required to run the software.

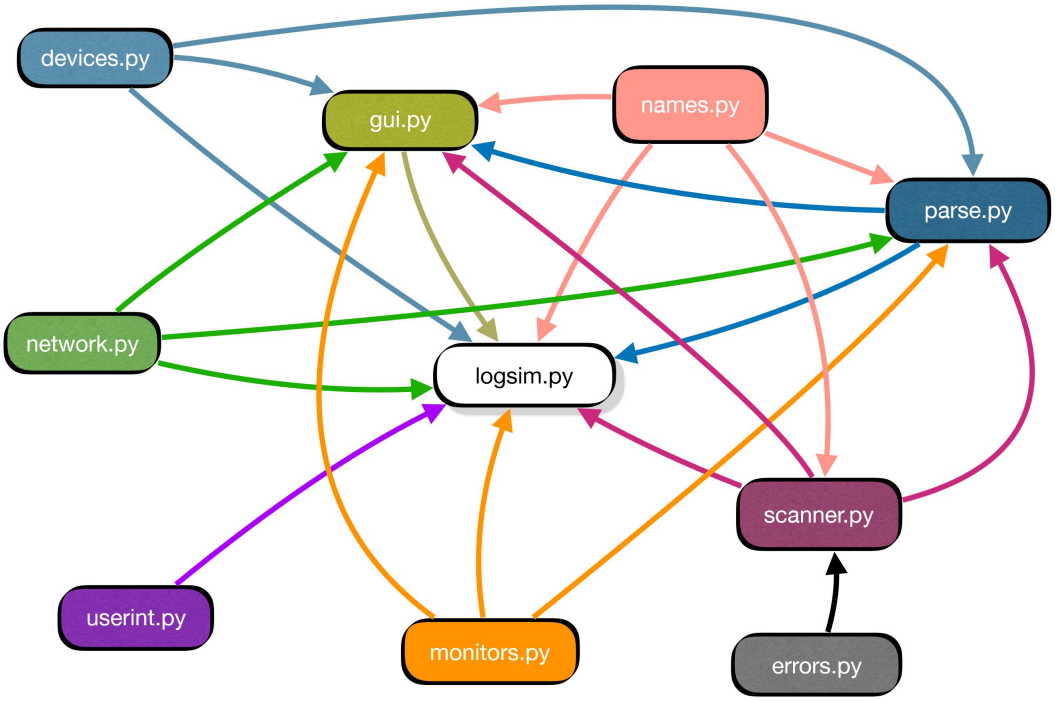


Figure 1: Graph showing the layout of modules in the Logic Simulator

2.2 logsim.py

This is the top-file for the Logic Simulator. It parses the options and arguments specified by the user on the command line and runs the graphical user interface (GUI) or command-line interface accordingly. The user will run a command of the form *python logsim.py [-c] [path_to_def_file]* to start the program. The *[-c]* flag indicates that the following definition file should be loaded and the command-line interface run. If the user does not use the *[-c]* flag then this indicates that

they wish to run the GUI and in this case, the path to a suitable definition file is an optional argument. If a file is given, then it will be parsed and if it is without error, the GUI will be pre-loaded with the contents of the definition file accordingly. If there is no file path argument given, then the GUI will be run without a file pre-loaded and the user is able to select a definition file from the interface.

2.3 gui.py

This module contains the the entire contents of the graphical user interface (GUI) object which is shown by logsim.py. The module consists of three classes: Gui, MyGLCanvas and My3DGLCanvas. The latter two classes handle the set-up and drawing operations in 2D and 3D respectively. The signals are drawn onto a canvas using PyOpenGL. The Gui class details the entire set up of the GUI and all event handlers in order to serve user inputs to the interface. The GUI is implemented in wxpython and enables the user to interact with the circuit.

2.4 userint.py

This module contains the code required to run the command-line interface. It has the exact same functionality as the GUI however the user interacts with the circuit through text inputs into the terminal. The commands allow the user to run or continue the simulation for a number of cycles, set switches, add or zap monitors, show help, or quit the program. The output signals are displayed as ASCII Art graphs in the terminal window.

2.5 scanner.py

Scanner reads the characters in the definition file and translates them into symbols that are usable by the parser. This populates the Names object which is passed as an argument upon creation of the scanner. It contains various methods to interact with the definition file which are later used when the Parser object is created and creates an Errors object with blank parameters to be stored within the Scanner object and accessed by the UI.

2.6 parse.py

This module analyses the syntactic and semantic correctness of the symbols received from the scanner and then builds the logic network by populating the objects monitors, network and devices and returning true. If the definition file is not successfully parsed then the syntactic and semantic errors are loaded into the Errors object to be displayed to the user and the parser returns false.

2.7 names.py

This module contains the class Names which maps variable names and string names to unique integers. It also contains a number of methods to query name strings, lookup unique ids and add new strings/variables.

2.8 monitors.py

This module contains functions for recording and displaying the signal state of outputs specified by their device and port IDs. This is populated by the parsing of the network and the gui and

userint interact with it in order to find the signals they are to display/add and remove monitor points.

2.9 devices.py

The class Device stores the device properties for each device in the networks. Device properties include but are not limited to id, type, inputs, outputs, period, etc. The class Devices contains many functions for making devices and ports. It stores all the devices in a list which can be queried and device properties extracted.

2.10 network.py

The class Network contains many functions required for connecting devices together in the network, getting information about connections, and executing all the devices in the network. The GUI and command-line interface interact with this object when the user wishes to execute the network. All variables in the network update simulating the network undergoing a single clock cycle.

2.11 errors.py

An error object is initialised bby the Scanner with blank parameters but each is reassigned in the scanner and/or parser. They are populated at various points in the simulation, either by the parser in the case of syntactic/semantic errors, or by the network in the case of run-time errors. They can then be recalled by the GUI when attempting to display helpful error messages to the user.

Appendix A Example Circuit Definition Files, Diagrams and Results

Appendix B EBNF Descripton of the Logic Description Language

Appendix C Software User Guide

Appendix D Brief Description of File Contents

References

[1] Andrew Gee and Mojisola Agboola. *CUED GF2 Software: Main Handout*. 2018. URL: https://www.vle.cam.ac.uk/pluginfile.php/13279532/mod_resource/content/18/handout_19.pdf.