University of Oxford

Logic Circuit Workbench

User Interface

52162

3rd Year

Undergraduate Project

ABSTRACT

Development of an interactive simulator for digital logic. The project being split into two halves, the User Interface and the Simulator with the former being detailed here and the latter in a separate project report. Some material which is common to both projects is included in the Collaborative Work section.

A user of the software may construct a logic circuit by selecting components from a library and then connect them together in allowable ways whilst enforcing logical and aesthetical design rules. These circuits can then be stored to disk for later modification or simulated and logged over time to produce a graphical representation of the state of the circuit at user chosen locations.

The application can be easily extended with custom component libraries, the visual descriptions of such components can be specified either explicitly with component images or created dynamically from the logical description of each component.

Saved circuits can be imported and used as atomic entities in the creation of new circuits. The behaviour of these sub-circuits can then be monitored in the same way as for standard circuits.

Standard HCI features are implemented including user feedback, a command history and common accessibility features.

TABLE OF CONTENTS

INTRODUCTION

[A description of the objectives of the project, and what makes them worthwhile; an overview of the achievements; a road map of the rest of the report.]

BACKGROUND

[How much background to include?] [Logic and Boolean algebra is fairly elementary!]

COLLABORATIVE WORK

The design of the application has followed the Separation of Concerns paradigm as far as possible; not only has this ensured that each half of the project could progress somewhat independently of the other, but additionally that each half is more extensible, maintainable, stable and examinable.

However, both the user interface and the simulator must communicate heavily and hence mutually agreeable interfaces were fixed. It is those interfaces and communication channels between the two halves which are described here.

A major feature of the application is the support for *netlist library modules*, whereby the user can create custom components for use in their circuits. Each netlist library that is loaded upon start-up contains a mapping from a human-readable taxonomic name to a class describing the logical behaviour of the component. Where a visual representation of this logical component exists (either predefined, dynamically created or user-specified [Reference here]), a mapping also exists in the netlist library from the same key name to a class describing the visual properties of the component. Upon component creation in the user interface, links are created from the visual component to the logical component through use of the netlist, thus insuring complete isolation and customisation of the components.

[Pins]

Ideally one would expect to be able to simulate many circuits concurrently from within one application. For this reason it was decided that each visual circuit that was opened should be contain its own simulator and as such the user interface part of the project is responsible for creating and communicating with the simulators for each circuit.

[Value Listeners: Steve]

[Simulation State Listeners: Steve]

REQUIREMENTS

[FROM WIKI, TO REWRITE]

"This is an interactive programming project with some logic simulation behind it.

The idea is to produce a simulator for a traditional logic breadboard. The user should be able to construct a logic circuit by drawing components from a library of standard gates and latches and so on, and connecting them together in allowable ways. It should then be possible to simulate the behaviour of the circuit, to control its inputs in various ways, and to display its outputs and the values of internal signals in various ways.

The simulator should be able to enforce design rules (such as those about not connecting standard outputs together, or limiting fan-out) but should also cope with partially completed circuits; it might be able to implement circuits described in terms of replicated sub-circuits; it should also be able to some sort of standard netlist.

It might be that this would make a project for two undergraduates: one implementing the interface, the other implementing a simulator that runs the logic"

Description of the software

- A tool to accurately model a digital logic circuit allowing simulation (step by step and continuous)
- Probing and logging of outputs over time – to help the user debug problems in a circuit.
- Allowing composite and sub-circuits
- Save and load circuits to/from disk
- Allow partially complete circuits
- Clocked components

EXISTING SOLUTIONS

(Glass, 2002)

[FROM WIKI] Several solutions that already exist, and were trialled as initial investigation into the requirements (functional and non-functional). Many are free open source software, although others provide only a time-limited demonstration copy.

Very interesting project from a professor at York University who completed a similar project to ours [1]

DesignWorks (Shareware / Demo) [2]

LogicSim - an educational tool for designing and simulating digital logic circuits[3]

Digital Simulator - Digital Simulator is Electronic Design Automation software without the $50,000 price tag. If you work for an educational institution or are a student, Digital Simulator is absolutely free. [4]

MMLogic - A particularly good free one! [5]

PSpice - Over complicated for what we need, but had a good features (user interface, subcomponents, intuitive wire behaviour)[7]

DESIGN

[Stuff about object-oriented programming principles, Command History, etc…] (Spivey, 2007)

(Sun Microsystems, Inc., 2006)

(Sun Microsystems, Inc., 1995-2008)

[UML Diagram]

COMPONENTS

[Invalid Areas] [Bounding boxes] [Selection States][Translation][Rotation][Pins][Labelling]

GRIDS AND CONNECTION POINTS

So that the user can make an intended connection between a two connectable points, a discrete location is required at which the connection can be created. The action of making a connection will be triggered by a mouse action which has an associated location on the screen. The Cartesian coordinate specified by this event although technically discrete due to the finite integer pixel dimensions of the user's VDU, is too precise to be used as the locator for connections. Pixels, by design, are not individually identifiable to a human user; it is therefore unacceptable to expect the user to select a connection point at pixel precision. Consequently, an approximation to the user's screen coordinates are used, the approach used here is that of a 2D Cartesian co-ordinate plane with units of 1-pixel and connection storing points only at human distinguishable displacements n from each other ($n \geq 5$).

This plane (henceforth referred to as *the grid* (in *world coordinates*)), contains mappings from each of the points (which are divisible by n) in its 1st quadrant co-ordinates to a Connection Point object, which lists the connectable points of objects in a circuit which are connected at that point. Obviously all logical components have one or more pins onto which a connection can be made, wires on the other hand can be connected at any of the grid points that lie along its length; these are produced dynamically from the properties of each wire (See Design→Wires). Keeping with the convention of naming the connectable points of a component as pins, any connectable point is herein described as a *pin*.

The translation of the mouse location at each event from device coordinates into valid grid coordinates requires a snap operation σ defined below:

$$\sigma(x, y) \stackrel{\text{def}}{=} (round(n, x), (round(n, y))$$

Where $round(n, x)$ is the function which rounds x to the nearest multiple of n, and n is the grid size.

$$round(n, x) \stackrel{\text{def}}{=} \begin{cases} n((x \bmod n) + 1), & x + \frac{n}{2} \geq n((x \bmod n) + 1) \\ n(x \bmod n), & otherwise \end{cases}$$

For ease of understanding, the origin of world co-ordinates lies at the top left corner of the drawing area with x and y-directions in the same direction as those of the device.

[Modelling connection pins, making/breaking connections]

WIRES

[Drawing][Modifying] [Tidying up][Removing Loops]

GRAPHICAL USER INTERFACE

[Screenshot with labels]

[Graphics] (Texas Instruments Incorporated, 1996)

(Sufrin, 2008)

[Command History] [Clipboard][Options Panel] [Logger Window + screenshot]

WORKAREA

[Internal Scrollable Desktop Pane] (Sun Microsystems, Inc.) [New windows] http://www.javaworld.com/javaworld/jw-05-2001/jw-0525-mdi.html

FILE IO

[Creation, visitor pattern][File format, XML] [Parsing] [Integration with netlist]

REDRAWING CIRCUITS

[Dirty Areas] [Clip boxes] [Redrawing the logger “live”]

NETLIST

[Classes][Create component tree][Creating dynamic visual components]

HUMAN COMPUTER INTERACTION

MOUSE EVENTS

[Drag and click to start wire] [Drag Selection box] [Dragging Selections][Component Selection states]

FEEDBACK

[Error Handler] [Progress Bar/Busy Icon][Component States][Splash Loader][Dialog boxes, saving, confirmation of deletion][Enable/Disable buttons & menus][Status message]

ACCESSIBILLITY

[Shortcuts] [Mnemonics][Text Menu + Icons]

INTERNATIONALISATION

[Resource Bundle, user locale][Constants]

(Sufrin, 2008)

TESTING

[Profiling]

[Assumption Simulator is correct]

[Values pass correctly (Input, output)]

[Connections made correctly (what are the cases? wire-wire, pin-wire, pin-pin, etc...)]

[Logger output is correct?] [Compare with manual trace from simulator]

CONCLUSIONS

[Went well, went not so well]

[Changes]

[Additional Features]

BIBLIOGRAPHY

Foley, James D., et al. 1993. *Introduction to Computer Graphics.* s.l. : Addison-Wesley, 1993.

Glass, Nicholas. 2002. Java Breadboard Simulator. *Department of Computer Science, The University of York.* [Online] 2002. http://www.cs.york.ac.uk/netpro/bboard/jbreadboard.pdf.

Green, Derek. 2008. The Art of Separation of Concerns. *Ctrl-Shift-B.* [Online] 2008. http://ctrl-shift-b.com/2008/01/art-of-separation-of-concerns.html.

Spivey, Michael. 2007. Object-Oriented Programming I. *Oxford University Computing Laboratory.* [Online] 2007. http://spivey.oriel.ox.ac.uk/mike/oop2007/outline.html.

Sufrin, Bernard. 2008. Object-Oriented Programming II. *Oxford University Computing Laboratory.* [Online] 2008. http://web2.comlab.ox.ac.uk/internal/courses/materials07-08/oopii/foils/.

Sun Microsystems, Inc. 2006. JDK(TM) 6 Documentation. *Sun Developer Network.* [Online] 2006. http://java.sun.com/javase/6/docs/.

—. Scrollable Multiple Document Interface in a JDesktopPane. *Bug Database.* [Online] Sun Microsystems, Inc. http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4129109.

—. 1995-2008. The Java(TM) Tutorials. *Sun Developer Network.* [Online] 1995-2008. http://java.sun.com/docs/books/tutorial/.

Texas Instruments Incorporated. 1996. Overview of IEEE Standard 91-1984. *Texas Instruments.* [Online] 1996. http://focus.ti.com/lit/ml/sdyz001a/sdyz001a.pdf.

Wutka, Mark. Double Buffering to Speed-up Drawing. *Java Expert Solutions.* [Online] http://www.webbasedprogramming.com/Java-Expert-Solutions/.

APPENDICIES

[Code] [Screenshots of test circuits][Sample circuit file + Schema][Logger output]