CSc 131 Computer Software Engineering

LogiSim

An Android Logic Simulator

Matthew Mendoza
9-19-2019

Table of Contents

Overview....2

1. Project Background and Description....2
2. Project Scope....2
3. High-Level Requirements....2
4. Affected Business Processes or Systems....3
5. Specific Exclusions from Scope....3
6. High-Level Timeline/Schedule....3
7. LogiSim Design....4

Logic Gates....4

Lines/Lanes....4

Moving Logic Gates....5

Circuit Line color and Node Points....5

Change Log....5

1. When....6
2. Who....6
3. What....6
4. Description....6
5. Published Version....6

Summary....6

Change Log Table....7

MATTHEW MENDOZA
LOGISIM DESIGN

9/19/2019 – Version 2.0

OVERVIEW

1. Project Background and Description

Consider students moving from Discrete Mathematics (CSC28) to Computer Organization (CSC137) at Sac State. Such students have some familiarity with evaluating logic expressions, but it would be helpful if they could test out simple circuits in real time to validate their intuition. In CSC137 students will eventually learn to work with a sophisticated logic simulation language called Verilog, but while that software is very capable, it is not particularly intuitive or immediate.

What would be helpful would be a simple logic simulator that can run on Android devices. This logic simulator would allow students to experiment with simple logic circuits in real time and get immediate feedback on their understanding. This simulator would primarily be used during the early part of the course prior to the use of Verilog, hence, it only need simulate the simplest of logic circuits.

2. Project Scope

Project scope defines the boundaries of a project. Think of the scope as an imaginary box that will enclose all the project elements/activities. It not only defines what you are doing (what goes into the box), but it sets limits for what will not be done as part of the project (what doesn't fit in the box). Scope answers questions including what will be done, what won't be done, and what the result will look like.

This project will encompass topics and lessons learned in CSc 131 - Computer Software Engineering. In CSc 131 we will learn the Principles of Software Engineering covering the software development life cycle, including software requirements engineering (elicitation, modeling, analysis and specification), software design, software implementation and testing. Main topics include various software development process models, method and techniques for specifying requirements, architectural and detailed design specification, prototyping, top-down and bottom-up software implementation and testing. Topics also include project management, project documentation and the development of communication skills through written documentation and oral presentation.

3. High-Level Requirements

1. The software must run on an Android tablet or phone

2. The software must use a single screen for all operations

3. The software should have a very clean and simple interface

4. The software must implement AND, OR, and NOT gates

5. The software must implement toggle switches for inputs and LEDs/LAMPs for outputs.

6. The software must allow the user to implement arbitrary small logic circuits. That is, users should be able to add and remove gates as well as wiring them as they see fit.

7. The size of logic circuits does not need to exceed what can fit on a single screen along with the remainder of the UI elements.

8. The software should allow the user to both edit the schematic and run the simulation.

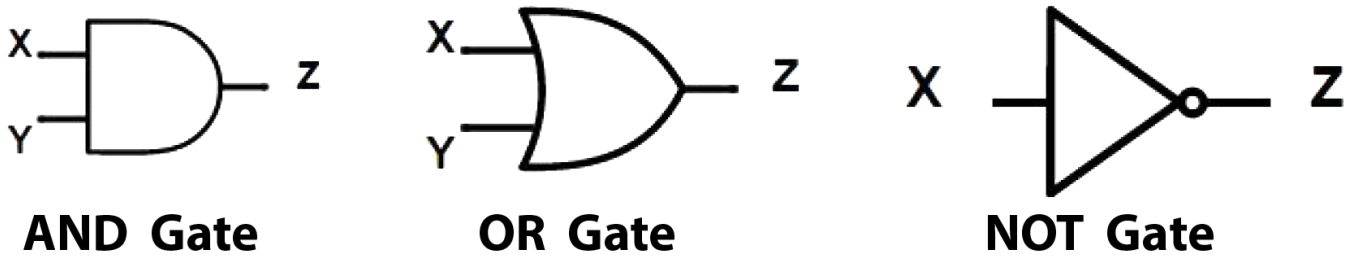
9. When running the user should be able to turn switches on and off and immediately see the result on the output LEDs/Lamps.

10. The software will only be used for basic functional understanding of combinatorial circuits and does not need to consider timing issues such as propagation delay.

11. The software should use standard logic symbols for visual representation of the gates and, additionally, images of switches and lamps that show both their on and off state.

12. The software does not need to be able to save and load different circuits, but the current circuit should be saved and loaded when the application stops and restarts.

Images of logic gates for reference:



4. Affected Business Processes or Systems

Describe any specific components that are excluded from this project.

For the scope of this project and time restrictions, and to reduce the level of complications only AND, OR, and NOT Gates will be used; therefore, this app will not include NAND, NOR, and ect.

5. Specific Exclusions from Scope

Describe how you plan to implement the project. For example, will all parts of the project be rolled out at once or will it be incremental? What will be included in each release?

This project will be an on-going project. Bug fixes and additional features will be released incrementally or just right before Dr. Posnett's assigned due date.

6. High-Level Timeline/Schedule

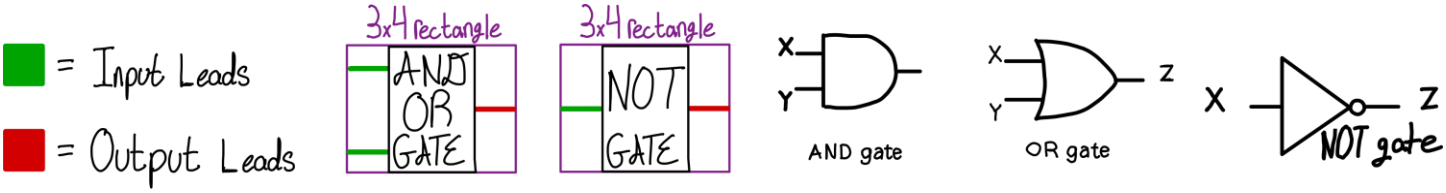
Describe what the high level timeline/schedule will be to plan, design, develop and deploy the project. Generally, by when do you expect this project to be finished?

PROJECT PHASE	DESCRIPTION	DEADLINE
LogiSim - Sprint 1	Get as much of the user interface coded and operational as possible.	September 27th at 5pm
LogiSim - Sprint 2		~ October 11th at 5pm
LogiSim - Sprint 3		~ October 25th at 5pm

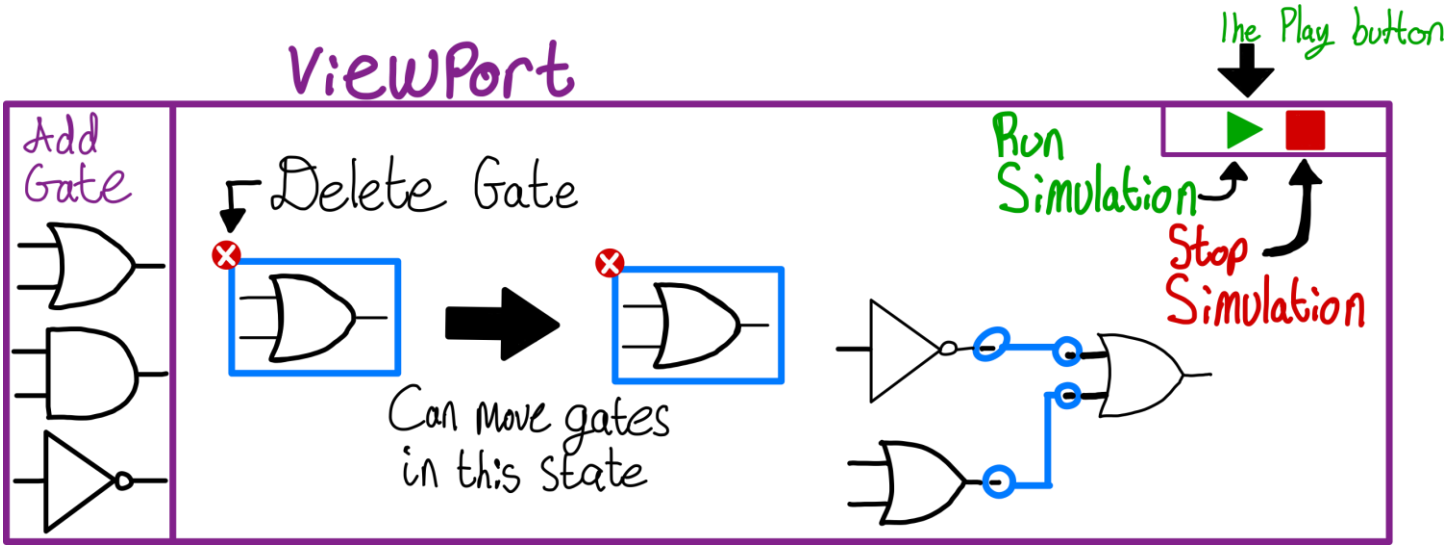
7. LogiSim Design

Logic Gates

Gates should populate a 3x4 rectangles grid area.

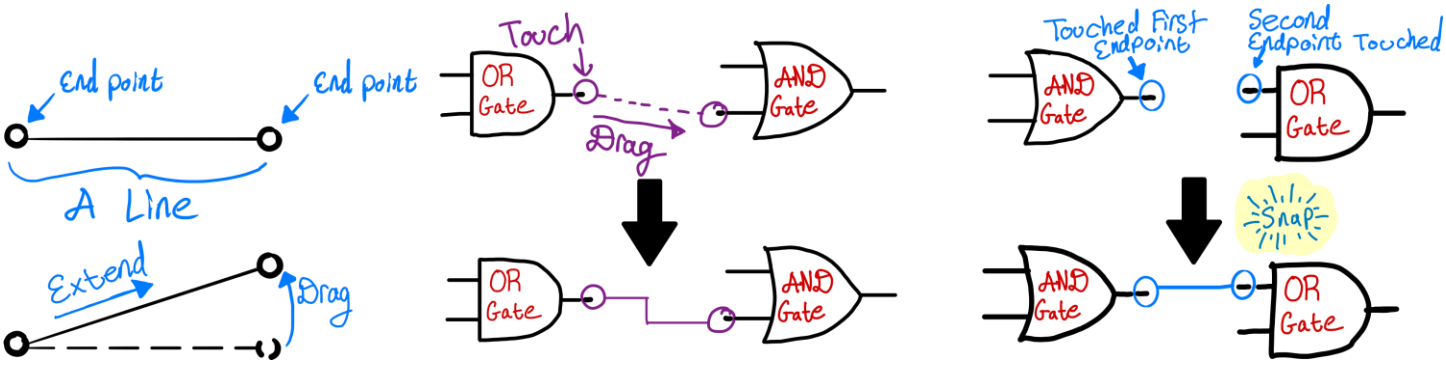


User can add gates from a side panel, remove gates by touching and holding, connect gates by touching an output lead to another input, run the simulation by a play button.

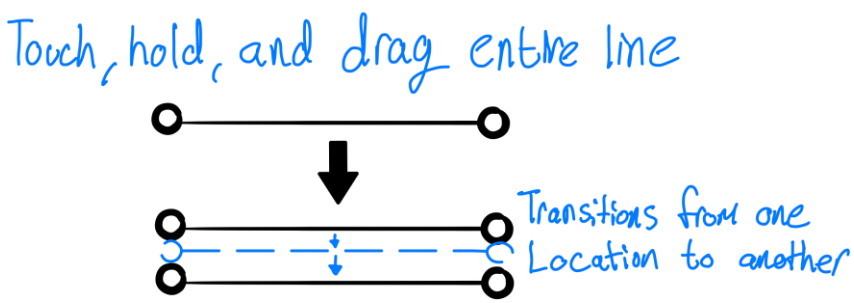


Lines/Lanes

The user can drag a line or touch from an output to another input lead.



The user can touch the center of the line (or touch and hold the center of the line) and can move the entire line (including the endpoints). When touch and held the line will be bordered where one can delete the line.

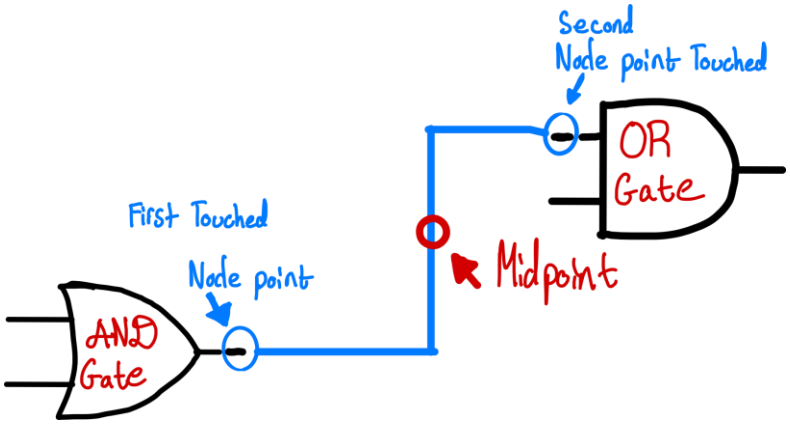


Moving Logic Gates

Logic gates can be moved by touching/holding the respective logic gate and drag from point A to point B.

If circuits are mapped to the gates and is then moved the respective circuits to those gates will extend/shrink to the corresponding location of the logic gate.

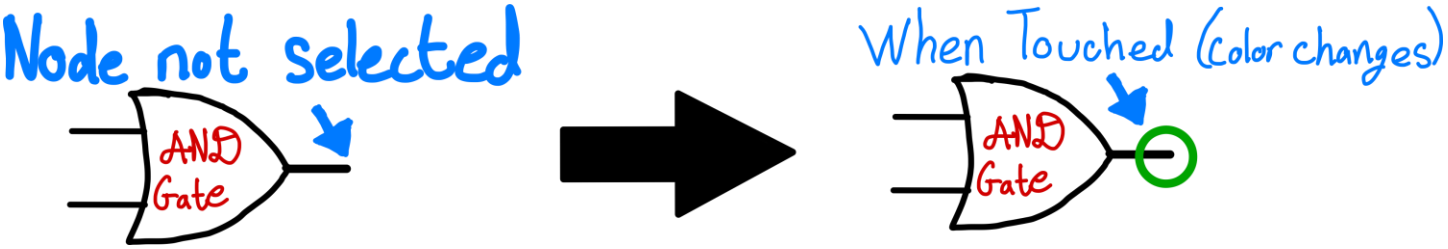
The circuits that are mapped to the logic gates should make a linear horizontal path to the destination; otherwise, they will extend horizontally and at mid-point of the line make a vertical line up or down to the vertical distance from the source and then continue a horizontal path to the destination input of the logic gate.



Circuit Line color and Node Points

The color of the of the circuit path should be white when matching outputs to inputs; however, when the user presses “Play” (make the circuit live) the line will then become green.

Node Points are the head to the respective input or output of the logic gates. By which when selected (touched on) a green circle at the tail end of the input/output lead will appear.



CHANGE LOG

This Change Log will follow the specifications as defined here: https://keepachangelog.com/en/1.0.0/

CHANGE LOG STYLE

1. When

Dates should follow MM/DD/YYYY (Month/Day/Year)

2. Who

Enter your name followed by the first initial of your last name; for example, Matthew M.

3. What

Type of changes were made:

- Added: for new features.
- Changed: for changes in existing functionality.
- Deprecated: For soon-to-be removed features.
- Removed: for now removed features.
- Fixed: for any bug fixes.

4. Description

The purpose of a changelog entry is to document the noteworthy difference, often across multiple commits, to communicate them clearly to end users.

5. Published Version

Follows version number convention detailed here : https://semver.org/

The software version number will be referenced here on change log; for this reason, the Document version number will share the same version number as the source code.

Summary

- Do not copy your Git commits messages into the change log description.
  The description is a high-level view of the changes made.
- Given a version number MAJOR.MINOR.PATCH, increment the:
  - MAJOR version when you make incompatible API changes,
  - MINOR version when you add functionality in a backwards compatible manner
  - PATCH version when you make backwards compatible bug fixes.

CHANGE LOG TABLE

When	Who	What	Description	Published Version