Kozell YON-BROOKS

5727

51319

boolean logic circuit simulator

AQA Computer Science NEA

[Date]

Contents

[Analysis 2](#_Toc94782673)

[The Problem 2](#_Toc94782674)

[Overview of Logic Gates 2](#_Toc94782675)

[Features Needed for GCSE Students 3](#_Toc94782676)

[Design 6](#_Toc94782677)

[Input, 6](#_Toc94782678)

# Analysis

## The Problem

GCSE Computer science is where a student’s introduction to logic gates may happen, this is a part of the curriculum but also helps to further a student’s logical thinking which can help in other areas of the subject, such as programming. A way to help me better understand logic gates were logic gate simulators, these simulators allowed me to create simple circuits using basic AND, OR, XOR, and NOT gates virtually. A simulator would allow the students to create circuits that give automatically give outputs without needing to be traced. However, the current simulators are too complex for what is needed for GCSE computer science and lack the teaching tools that my proposed system will include.

## Overview of Logic Gates

Logic gates are a model of computation that take one or two inputs and returns a single output based on the gate's logical operation / Boolean function, they are the fundamentals of logical circuits and physical logic gates made of diodes and transistors are what allow computers to work. Logic gates can be combined to produce a certain output based on the inputs of the circuit. An AND gate (Figure 1) for example will take two inputs and return a True output if both inputs are True, and a False output otherwise. Inputs and outputs can be True or False as they are Boolean, this is usually represented as a 1 and 0 for True and False, respectively. Certain gates such as the Not gate will only need one input. There are other parts to the logic circuits besides the gates; switches, constant inputs, and clocks can provide initial inputs. Output can be handled by a simple ‘bulb’ that is on/lit for True and off for False. A more complex output such as a 4-bit digit would produce an integer output based on a binary sequence from 4 Boolean inputs.



Figure 1: AND Logic Gate

## Features Needed for GCSE Students

The truth table below (Figure 2) displays all the possible inputs and outputs in tabular form for the Logic operations AND, OR, XOR, and NOT. These are the only gates that are needed for the AQA course as per the specification (Figure 3).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** |  | **Output** |  |  |  |
| A | B | AND | OR | XOR | NOT (Input only A) |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

Figure 2: Truth Table

Table

Description automatically generated

Figure 3: AQA GCSE Specification for Boolean Algebra 3.4.2

As these are the only gates needed for the AQA course, they will be the only gates included, simplifying the program for the students. My solution would be more focused on teaching, specifically what is needed for the AQA GCSE specification. I had a short conversation with Mr Flynn about what features would make my program relevant to GCSE students. These included:

* Abstracting logic gates that are not needed.
* Including a checklist to determine if the user has used each of the gates at least once.
* A truth table generator to display the results of a circuit as a truth table.
* Converting a written Boolean expression into a truth table.
* Displaying the Boolean expression for the created circuit.
* Allowing individuals to save and load circuits to and from their computer’s local storage.

These are all the features I believe are needed for the student to understand and solve GCSE level questions (Example shown in Figure 4).

A picture containing diagram

Description automatically generatedTable

Description automatically generated

Figure 4: Logic Gate question from AQA GCSE Computer Science June 2019 (8020/1)

## Acceptable Limitations

Hardware and software constraints – The program must run on school computers, therefore must not be resource intensive. The school uses Windows 10 as their operating system, so the program does not need run on any other OS.

## Platform, Programming Language, and Modules

The program would run as either a web application or a windows desktop application. These choices are suitable as Highdown uses windows as the operating system on its computers, and all its computers have a browser and constant access to the internet.

A desktop application is a type of software that is directly installed onto the hard drive of the computer. It can be launched whenever, independent of other applications, i.e., it does not need a web browser to run within, like a web app. They also work regardless of internet connection (unless the program itself requires it).

A Web application is a type of software application that is used through the internet via a web browser. The files are stored on a remote server, the backend processing is done remotely, and the application is only accessible via an internet connection and browser.

As a desktop application will provide all the features needed for the program, I have chosen it over the web app as it does not need a server to be stored/run on nor require a constant internet connection to access.

The programming language used will somewhat depend on the platform that the program will be run on. For a web app, the programming languages I would use would be Python, JavaScript, and HTML; I would also need to be familiar with CSS. For a windows application, there are many languages that would be suitable, including python. As python is a language that I am already familiar with, and it is my top choice for programming language. Because the program will be a desktop application only python is needed.

For the GUI, python has an array of frameworks to help build user interfaces. These include: PyQt5, Tkinter, Kivy, wxPython, PySimpleGUI. Kivy is currently my choice as it has many great features an also allows for the possibility of deploying the program on multiple platforms (windows, macOS, iOS, android). <https://kivy.org/#home>

## General Objectives

The general objective is to create a program that will allow a user to create logical gate circuits using draggable components that will evaluate and return an output based on the circuit. The components will be able to be linked and added/removed. The program will also show a truth table of an expression either taken from a component or input manually. The program should be responsive, easy to use, compatible with the school computers, and have the gates that are on the AQA GCSE computer science specification.

## Specific Objectives

1. The program must create and evaluate logic gate circuits.
   1. Can have gates, switches, and/or outputs (the 3 referred to as components).
      1. Each component will have a state of either true or false.
      2. The gates will be AND, OR, NOT, and XOR (from GCSE spec).
      3. The components can connect to each other.
         1. The components will have input and output nodes that point to their connected gates.
      4. The components can return their state as an output.
      5. The gates can calculate their state based on the gate’s Boolean operator and its inputs.
      6. The switches can be flipped, changing their state from false to true or vice-versa.
   2. There will be a board class containing the components.
      1. The board will create and destroy components.
      2. The board will store the created components in an array and tree structure.
      3. The board will tell the gates to connect and disconnect to and from each other.
      4. The board will traverse its tree of components, making them evaluate their states.
   3. There will be a truth table generator­.
      1. It will produce a truth table for a given expression.
         1. It can use the expression of an output component.
         2. It can use an input Boolean expression string.
      2. It will produce a list of possible input combinations based on the number of inputs.
      3. It will substitute each combination of inputs into the expression to get an output.
      4. It will list the outputs alongside the input combinations in a table.
2. There will be a graphical user interface.
   1. It will allow dynamic placing of the components onto a canvas.
      1. It will use mouse inputs to drag the components and move them about.
   2. It will have a component toolbar which will have buttons that can add components to the canvas.
   3. It will have a tool toolbar which will determine what will happen when the components are interacted with depending on the tool selected.
      1. Connect tool: the program will tell the board to connect these components.
      2. Disconnect tool: the program will disconnect the selected components from each other.
      3. Move tool: the selected component can be moved about the canvas.
      4. Delete tool: the selected components will be deleted.
      5. Clear tool: All components will be deleted.
      6. Truth Table tool: the program will show a truth table based on the expression of the selected component.
         1. Will show a popup with truth table and text input box that the user can enter their own expression into
   4. It will have a menu bar with options:
      1. Save, which saves the current circuit to a json file.
      2. Load, which loads a circuit from a json file.
      3. Quit, which exits the program.
         1. When clicked, a popup will appear asking if you are sure you want to exit.
   5. More…

## Guideline

1 Introduction

1.1 Background to the problem

1.2 Describe the problem

1.3 Justify the features of the problem

1.4 Why is this problem solvable by Computational Thinking?

2 Research

2.1 Description of the current system

2.2 Problems with the current system

2.3 Algorithms used with the current system

3 Stake Holders

3.1 Introduction

3.2 Identification of Prospective Users

3.3 Record of Client Interviews

3.4 Identification of User Need and Acceptable Limitations

4 Background

4.1 Description of Proposed Solution

4.2 Software development platform

4.3 Specify Hardware Requirements

4.4 Justify Hardware Requirements

4.5 Specify Software requirements

4.6 Justify Software requirements

4.7 Development language

5 Objectives

5.1 Introduction

5.2 General Objectives

5.3 Specific Objectives

5.4 Essential Features of the Computational Solution

5.5 Acceptable Limitations of the proposed solution

6 Modelling of the Problem

6.1 Introduction

6.2 Data Sources

6.3 Data Destinations

6.4 Data Volumes

6.5 Data Dictionary

6.6 Database Information

6.7 Database Design of the Current System

6.8 Database Design of the New System

6.9 Object Orientation Plan

# Design

## User Interface

When the program is initially opened, the first thing seen will be the full interactable program, allowing the user to start creating their circuits right away. The user interface should be simple and intuitive to use, no features should be hidden away or

Diagram

Description automatically generated

Figure 5: Possible GUI layout

Diagram

Description automatically generated

Figure 6: Object diagram for board and logic gates