Logic circuit simulator

Kamil Sobolewski

CONTENTS

ANALYSIS

DESIGN

TECHNICAL SOLUTION

TESTING

EVALUATION

## ANALYSIS

## Background and Problem Definition Many A-level and GCSE students find making and understanding logic circuits quite hard. Simplifying Boolean expressions can be very difficult sometimes. Most people learn the best when they get a hands-on experience wait a concept. Hence developing an application in which you can build your own logic circuit can help many students understand the concept of logic circuits and can also help teachers to demonstrate how a logic circuit behaves under certain inputs e.g how a D-type flip flop works. This application will provide an easy to use interface so it can be accessible and easily used by the vast majority of students, teachers and even people who want to learn and experiment with logic circuits.

## Description of Current System (Research into Problem)

## Logic circuits consist of logic gates and ‘wires’/connections. There are 7 main logic gates that are widely used. There is also an eight one called the buffer however since this program is mostly focused on A-level/GCSE students studying computer science I will not be including the buffer as it is not covered by A-level and GCSE specifications and there is not much logic to the gate as the output is always the same as the input. So that leaves me with 7 different gates.

## 

## The AND, NAND, OR, NOR, NOT, XOR(EOR) and XNOR(ENOR) are the most common types of gates used in most computers and logic circuits. All the gates seen above have different logic meaning that every gate will have a different truth table associated with them. A truth table is a table that shows you the output of the gate based on its input. Most gates have 2 inputs however the special gate within these is the NOT gate as it only has one input. Here are the truth tables for all these gates:

## A picture containing text, clock Description automatically generatedTable Description automatically generated as you can see, I’ve put the NOT gate truth tables separate to all the others. This is because the NOT gate as mentioned before only has 1 input hence it will only have 2 different input combinations either 1 (On) or 0 (Off) whereas the other gates have 2 inputs, so they have 4 different combinations of inputs (A and B being the inputs). On the right side of the table, you see that all the 0,1 patters in every column for each gate are different so the gates have different logic to them one type of gate will not always have the same output as a different type of gate. Meaning that each gate has its own unique use.

1. Analysis of Similar Systems  
   The first system that I’ve found online was logic.ly it is a website that allows you create your own logic circuit. Here is a screenshot of the website:

Diagram

Description automatically generated

This application is quite like what I’m trying to make as there is not much variation to a logic circuit application all of these types of apps are all going to have the same base features. I’ve found the GUI of the application quite hard to use with a lot of precise right clicks required to delete an object. Drawing the wire must be perfect if you are off slightly off, the wire will not be connected. Making the GUI quite annoying to use however, this application can be good to take inspiration from even though it is not written in python which is my programming language of choice for this project as most A-level students use python and if students wanted to study how the code works, they could with no issues. This application (logic.ly) gives me a good basis to see how everything could look.

After finding this I tried to find another similar application which was written in python however, I could not find an example, so my application can be helpful to many students who use python and could provide inspiration to students in the future who want to take on a similar project in python. Now that I got the basic idea of how the program should look, I started looking for python libraries which will allow me to make a nice looking simple to use GUI. The library I’ve settled on was Tkinter. This library is quite hold hence there is a lot of information and resources about it online making it quite easy to learn. I’ve also used Tkinter before for one of my personal programming projects, so it was the best option for me.

Good and Bad (Screenshots of other Applications)  
Reflection of what system will need after analysis of each similar system (Research into Requirements)

1. Identification of End User Needs  
   To get more information about the program im trying to make I asked some of my friends who study computer science a few questions about this application.

‘Would you use a logic gate builder app in your revision or checking your answer to boolean algebra practise questions?’

Most of my friends were interested in the idea and concluded that this app would actually be very useful in some scenarios mainly revision and learning the idea of logic circuits for the first time. Many of my friends also said that they struggle to check their answers to logic circuit questions because there is no real way of checking logic circuit answers (they mainly meant questions on homeworks or online resources with no mark scheme) besides using software or showing it to someone who understands the topic like a teacher for example. So the app just from one question seemed like a good idea with a good target audience.

‘What are the main features of the GUI you would like to see’

From this question I’ve found that a simple,easy to use and clean looking interface was sawed after. They said that the program would be very niche in general but useful to A-level students and that the application should mainly only focus on the basic idea of building a logic circuit and making it as easy and quick as possible to be able to quickly check their answer to a question or quickly simulate what a logic circuit would do.

From this I’ve concluded that I want all the assets which will be used in the program to be simple shapes and add as little of them as possible. Making the program as easy and quick to use without any unnecessary features. Potentially a program that has more built-in features and a different target audience could be developed in the future with this application being the basis.

‘Would you like to see Tkinter used for this application?’

As all of my friends who study computer science have to do their own NEA project which a good portion of them will include a GUI they said that seeing Tkinter used for this project could help them to develop their own NEA project and poetically future programming projects as they could see the approach that I take for my GUI and they would get a good understanding of the main keywords used in Tkinter. Hence they also mentioned that the source code should be publicly available to whoever downloads the application as it would help them.

I concluded that I would make 2 ‘different’ applications. One compiled .exe application which can be run on basically any laptop/computer with a operating system which would be suited to all A-levels students who would not like to see the source code and only care about the main features of the application. And then a second application which would just be the source code .py application which can be run on any laptop/computer with a python interpreter. This would allow A-levels students to see,copy and edit my code for learning purposes. I also told this idea of making 2 sperate apps available and they said it would be the best option for them so it should also be a good option for my target audience.

‘What are some of features you would like to see?’

My friends suggested a lot of idea. But the main ones were that they would like to see a ‘save’ and ‘load’ option which would allow them to save their circuits and send them to another friend for example and allowing them to load it and see it.

Another main feature suggested was to make a ‘puzzle’ mode for the program which would work by giving the user access to only the AND and NOT gates and the final goal being to create all of the other gates. When the user successfully uses the AND and NOT gates for example to create the OR gate the OR gate would be unlocked. I thought this was a very cool idea as all logic gates can be built from only AND and NOT gates however I knew it would be hard to implement however I would take it into consideration.  
(Interview Questions and Answers, Analysis of Response and Key Objectives)  
Getting relevant ideas, Link each question response to the objectives of the system

1. Acceptable Limitations  
   The previously mentioned idea of the ‘puzzle’ mode would most likely be scrapped as this is a grate fun feature for the program it is quite unnecessary for the main target audience and it would be very complex to implement which most likely be out of the scope of the A-level course. However, I would make the program in such a way where extra features can be easily added in the future so students can add their own or I could develop it further to include this mode.

I thought I would include the save and load features however when I was finishing the main code the complexity of this feature was greatly underestimated as many Tkinter objects will be used, a lot of OOP and the state of the program is largely dependent on the user. For example, the user could place down n number of gates down. Connect them in n number of ways etc. So to create a save file a separate algorithm would have to be created (which is very comple and would likely be a separate project by its self) or a way of saving the exact state of the opcode running the code at that instance using a python library (I’ve tried this approach however the loading the placing of the Tkinter objects was very difficult and I would not get this to work as the 2 libraries were not very compatible with each other).  
(Time, Programming Skills, Software on Machine)

## Proposed Solution and **Detailed Objectives** The application will have these main features. Allow the user to create inputs, outputs, place any gate they would like anywhere (within bounds obviously) , change the inputs to see what the circuit will output, visually change the output, visually see the signal going through each wire (1 or 0 / On or Off) and connect the gates.

1. Letting the user place inputs and outputs in a desired position:

* The user should be able to place inputs anywhere on a left input bar and place outputs anywhere on the output bar on the right. This will allow the input and outputs to stay in one unique place making the program easier to understand.
* The signal of the inputs should be able to be changed by the user by binding a keyboard key or mouse button to a function.
* Wires will need to be drawn from the inputs to the gate’s inputs. Meaning that a keyboard key or a mouse button will need to be bound to a function as well and the program will need to track the position of the mouse as the user is drawing the wire.
* The program should also show the preview of the wire before it is drawn when the user releases their keyboard key or mouse button.
* The wire will need to have a separate validate line position function so that the wire will not be drawn if it is outside the window,not connected to a gate, too short to see etc.
* The input should have a fixed x coordinate so that all the inputs are in one line and easy to see and find at anytime making the program easy to understand and use
* The input should also have its own separate check valid position function. This function will only draw the input if the input is not overlapping with another input.
* The input and output bars should have 2 sperate colours green for input and red for output to make the code easy to understand. The inputs and outputs should also be represented by a circle to make the program look nice and simple. Ideally the inputs and outputs should look a bit like this:
* Image above is only a rough idea in the actual code the bulbs(circles) will have no fill and only a solid outline so that the input and output bulbs will have a empty fill (grey) when they are ‘off’ and the fill of the bulbs will change to red when they are on. Same for the wires a red wire will represent the wire is carrying a 1 or a ‘On’ signal and grey for a 0 or a ‘off’ signal. This standard should be throughout the program to make it easy to understand and learn.
* The user should be able to place as many inputs and as many outputs on the input and output lines. This will keep the circuit organised and easy to understand.
* A keyboard key or a mouse button will need to be bound to a function which will allow the user to place a input in the position of their mouse in the moment they click the bound key. The check valid position function should also be run.
* The colour of the wire which is being dragged out of the input bulb will be dependant on the fill colour of the input bulb so when the user draws a line out of the input bulb there will needs to be function which checks the colour of the input bulb and selects the appropriate colour for the wire.
* The code for the input and output bulbs should be created in a class because their behaviour will be similar and there will be an unknown number of them so this feature can’t be hardcoded easily.
* The user will be able to delete any input or output they want if they don’t want to have it anymore.
* By deleting an input or output the connected wires to it should also be deleted to make the program quicker to use.

1. Allowing the user to easily place gates in the desired position:

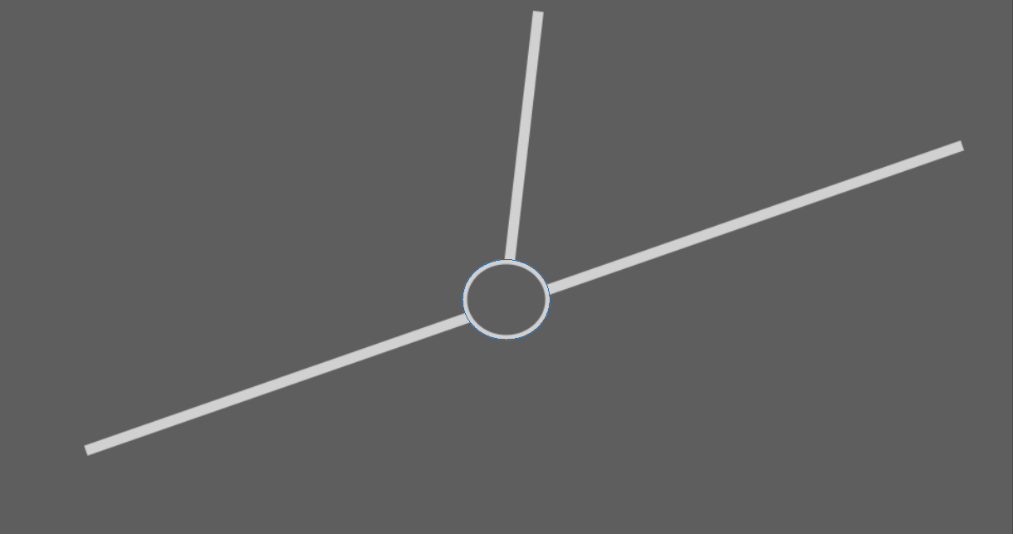
* The application will include a bar at the bottom of the screen which will show all of the available logic gates.
* The user will be able to drag and drop the gates wherever they want within the valid conditions
* The gates will be bound to the motion of the mouse as well as the left click of the mouse. So when the user click and holds the gate they will be able to move and place it.
* There will be a function to draw a new gate in the place of the dragged gate from the bar at the bottom of the screen. So that the user will be able to place down as many gates as they want. As they will self ‘replenish’.
* The check valid position function for the gates will include the check of x and y coordinates of the gate when the user releases their left mouse button. If the x and y are not going to be in range the gate will not be placed and automatically deleted this is to prevent extra bugs and problems which might occur with the idea of a gate being out of bounds like not being able to connect the input or not being able to drag out the output wire etc.
* The gates will also have their own class to keep track of each individual attributes as there will be an unknown number of them.
* The user will be able to move already placed gates with connected wires. The wires will also move in relation with the gates so that all the wires stay connected to the gate.

1. Allowing the user to draw wires from valid places and connect, gates, inputs and outputs together:

* The function to create a new instance of a ‘connection’ / wire class will need to be bound to different object namely the input bulb, the output bulb and the output of individual gates. So that wires can be dragged out of all of these objects allowing for a pleasant and easy user experience.
* The left mouse button presses and drag will be bound to draw the wire as it is the most standard bind across all other applications which include drag and dropping.
* As mentioned before in point nr 1 above the wire will need to have a function to check if its in a valid position when the user finishes drawing it
* The wires will be straight lines. Curves would be quite hard to implement, and it makes some of the math functions in the program over complicated. I think that straight lines are suitable for the target audience and it doesn’t change the functionality of the program
* The wire will have a preview as the user is dragging it for a preview so that the user knows how it is going to look when they let go of the bound key
* The wires should have a ‘snap’ function so that when the user gets close to a valid position with the end of their wire the wire will automatically snap to that position and stay there under the user moves out of the range with their mouse. This can be achieved by using Pythagoras theorem.
* The snap function should also interact with the preview function giving the user a visual queue that they are in the range and that the wire will go to where they want it to go.
* The user will be able to delete any wire they want.
* The user should not be able to: connect 2 or more wires to the same input, connect 2 or more wires to the same output, connect the output of the gate to the input of the same gate, place a wire with its end outside the screen, connect a wire from the final output bulb (the final circle on the red bar on the right) to a input of any gate.
* The use should also be able to draw a wire from the final output bulb to any output bulb of a logic gate.

1. Allowing the user to create intersection points on the wires so they can branch out the wires:

* The intersection will work like a splitter in a wire it will look something like this:



New wire

Wire 1

Intersection

Wire 1

* As you can see the intersection will allow the users to create basically any circuit, they want it will make the circuits look cleaner and potentially easier to understand with this feature.
* The new wire and wire 1 will always carry the same signal.
* The intersection will have an appropriate fill red or grey depending on the signal in wire 1
* There will also need to be a bind on each for the creation of the intersection where the user’s mouse pointer is located when they click.
* The class for the input and output can be reused for the intersection as the intersection will have similar methods and attributes as the input and output bulb. However, polymorphism will have to be used.
* It will need to have a check valid position function as well.
* It will need to have the same colour as the wire. Meaning that if the wire is ‘on’ the intersection will also need to be ‘on’.
* The intersection will need to update appropriately as the signal in the wire changes.
* When the user deletes a line with an intersection on it any lines connected to the intersection should also be deleted as well as the intersection and the wire that was originally pressed on to delete it.

1. DFD, Flowcharts of new system
2. Potential layouts

Sketch and Annotate

# DESIGN

1. System Overview  
     
   (to start section)
2. Modular System Design  
     
   Example below

Diagram

Description automatically generated

## System Flowcharts

## Interface (Screen) Design and Rationale (Annotations)

1. Description (line by line) of algorithms and Coded Algorithms  
     
   Shows you know how the algorithms work  
   Talk about core parts not just login and database  
   Show screenshot of form/page connected to the algorithm.
2. Data Structure and Example Data  
     
   E.g. Examples of questions to be used in Quiz
3. Class Definitions – Diagrams  
     
   Example below

Table

Description automatically generated

Table

Description automatically generated

|  |  |
| --- | --- |
| Class | General function summary |
| Gate | Used to keep track of all of the logic gates the user places, gate input(s) ,output ,output lines and input lines. Used to draw all the circles (Tkinter oval objects) and uses the appropriate image asset to display the gate |
| Gate\_output\_node | Used to draw and keep track of the outputs of the logic gate instance |
| Gate\_input\_node | Uses to draw and keep track of the input(s) of the logic gate instance |
| Main\_bulb | Used to draw and keep track of all the input and output bulbs on the right and left sides of the screen. Binds the bulbs/circles to their appropriate bind. E.g right click to change the input of the input bulb or q on any Main\_bulb to delete it. This class is also re-used to make the intersections in wires. |
| Connection | Used to keep track of each line the user draws. Binds each line to right mouse click to delete it. Used with the Main\_bulb class to create intersections in the connections. |

|  |  |  |
| --- | --- | --- |
| Class | Variable/attribute/properties | Data type/function |
| Gate | Type\_of\_gate | Stores a capitalised string of letters identifying the type of the gate e.g ‘AND’ or ‘XNOR’ etc. |
| Output\_lines | Stores all the instances of the ‘Connection’ class this class is responsible for managing wires and connections. The Output\_lines variable is used to keep track of all the output wires so that they can be modified accordingly in certain functions |
| Root | Stores the canvas Tkinter object. This canvas is exclusive to the main program so only given objects can be drawn on it such as gates,wires,inputs etc. |
| Image | Stores the image of the given gate as a ‘PIL.Image.PhotoImage’ object used when creating the window object |
| Label | Stores the Tkinter label object used to draw the image onto the ‘window’ variable. This variable also uses the image variable to load the image. |
|  | Window | Stores the Tkinter window object this variable is used with the label variable as when the window object is being instantiated the label variable is used to put the label object in its own box |
|  | Selected | Boolean value it stores the state of the logic gate if true the gate is currently being dragged if false it is stationary |
|  | X | Float value storing the current x coordinate of the Tkinter window object (the logic gate) in relation to the Tkinter canvas |
|  | y | Float value storing the current y coordinate of the Tkinter window object (the logic gate) in relation to the Tkinter canvas |
|  | Gate\_num | Stores a integer value used to determine if the gate is the original gate in the bar at the bottom of the screen or a new gate it is used to drag the gate out of the bar and create a new one in its place. |
|  | Input 1 and input 2  (or only input 1 is the type of gate is ‘NOT’ as it only has one input) | Both store a different instance of the Gate\_input\_node class which is in the main Gate class. This variable is used in the code to keep track of the state of the inputs (1 or 0) as well as the wires going into in the input node of the gate |
|  | output | Stores the instance of the Gate\_output\_node class which is in the main Gate class. It is used to keep track of the output as well as keep track of all wires coming out of the output and updating them appropriately |
|  | Parent | Stores the instance of the Gate class |
| Gate\_input\_node  This class is nested within the Gate class | Input | Stores an integer value used to figure out if this instance of the class is the 1st input of the gate or the 2nd this is used to figure out the y offset |
|  | Offset\_y | Stores a float value of the offset required to place the Tkinter oval object in the right place. For the not gate the offset would be 0 as the output is on the same y level as the output. However, for the other gates the offsets are 19.5 or -19.5 as the inputs are 19.5 units up and below the output. The ‘Input’ variable is used to decide whether the offset is 19.5 or -19.5 the higher up input is input nr1 and the bottom input is nr2 |
|  | Offset\_x | Stores an integer value. This value is used to place the Tkinter oval object in the right place in relation to the size of the gate (the image size) |
|  | x | Stores a float value. It represents the centre x coordinate of the Tkinter oval object which represents the input of the gate. This is calculated by using the current x coordinate of the gate as well as the offset\_x variable |
|  | y | Stores a float value. It represents the centre y coordinate of the Tkinter oval object which represents the input of the gate. This is calculated by using the current y coordinate of the gate as well as the offset\_y variable |
|  | root | Stores the Tkinter canvas object. This canvas the same as with the root variable in the ‘Gate’ class is used to place the oval objects in relation to the canvas x and y. |
|  | radius | Stores the radius of the circle (the Tkinter oval object) used to determine if ovals intersect and when instantiating the tkinter oval object |
|  | Input\_gate\_node | Stores the Tkinter oval object. This variable is dependant on the root,x,y and radius variables |
|  | parent | Stores the instance of the ‘Gate’ class |
| Gate\_output\_node  This class is nested within the Gate class | Offset\_x | Stores an integer value. This value is used to place the Tkinter oval object in the right place in relation to the size of the gate (the image size) |
|  | x | Stores a float value of the current x coordinate of the Tkinter oval object. The event coordinates (these are built into Tkinter they show you where the user’s mouse is when a bind is executed for example when the user clicks on the gate) as well as the offset\_x variable are used to calculate the x coordinate |
|  | y | Stores a float value of the y event coordinate (doesn’t need offset\_y as the output of the gate is inline with the centre of the gate) |
|  | radius | Stores the radius of the circle (Tkinter oval object) used to determine if ovals intersect and when instantiating the Tkinter oval object |
|  | root | Stores the Tkinter canvas object used when placing the Tkinter oval object |
|  | line | Stores the instance of the ‘Connection’ class of the line which is currently being drawn this variable is overwritten every time a new line is drawn out of the Output of the gate. When the user stops drawing the line the instance of the ‘Connection’ class is appended to the Output\_lines variable in the ’Gate’ class |
|  |  |  |
| Main\_bulb |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# TECHNICAL SOLUTION

## Program Listing

Use Online Program to layout professionally <https://ndzri.blogspot.com/2016/09/syntax-highlight-code-in-word-documents.html>

(Comments are provided at the start of each subroutine and then in appropriate places in the code to help provide explanation)

# TESTING

## Full System Tests

## Objectives Tests

comprehensive and covers both objectives and full system tests.

video evidence of the testing even better

Robust testing (large quantities of data)

Example Testing Form Below

A screenshot of a computer

Description automatically generated with low confidence

Table

Description automatically generated

# EVALUATION

## General Appraisal (Reflection of the whole system)

1. Meeting Objectives (Reflect upon these)  
     
   Use table example below

Table

Description automatically generated

## End User Feedback

## Analysis of Feedback

1. Suggested Improvements   
     
   (Detailed and includes reflection)