University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

# Level 4 Project Report

**Brodie West**

September 19, 2018

# Abstract

Looking at existing projects aimed at teaching students about circuits, it was found that they were outdated, or unsuitable for beginners. For this project, a circuit simulation program was created to teach computer science students the basics of synchronous digital circuit design, to be used by students and in demonstrations by lecturers. There was a focus on knowledge about common circuits, validity and clocking.

This program was evaluated showing that it works correctly, is usable and can improve users' knowledge of the relevant topics. Users had some difficulties using the circuit building tool and this would be the main focus of any future work on the project.

# Education Use Consent

# Contents

**1 Introduction** **1**
- 1.1 Motivation 1
- 1.2 Goals 1

**2 Background** **3**
- 2.1 Synchronous Digital Circuits 3
- 2.2 Existing Products 3
  - 2.2.1 Overview 3
  - 2.2.2 Logic Gate Simulator 4
  - 2.2.3 CircuitVerse 5
  - 2.2.4 LogicWorks 6

**3 Analysis/Requirements** **7**
- 3.1 Problem Specification 7
- 3.2 Chosen Limitations 8
- 3.3 Changes to the specification 9

**4 Design** **10**
- 4.1 Simulator Logic 10
  - 4.1.1 Signal representation 10
  - 4.1.2 Gate Delays 10
  - 4.1.3 Word Wires 10
  - 4.1.4 Black Box Components 11
  - 4.1.5 Save Files 11
- 4.2 User Interface 11
  - 4.2.1 External consistency 11
  - 4.2.2 Constraints in design 12
- 4.3 Technologies 12
  - 4.3.1 Java 12
  - 4.3.2 Javafx 13
  - 4.3.3 Desktop application 13

**5 Implementation** **14**
- 5.1 User Interface 14
  - 5.1.1 Model View Controller 14
  - 5.1.2 OS native appearance 15
- 5.2 Aesthetics 16
  - 5.2.1 Component aesthetics 16
  - 5.2.2 Wire Aesthetics 16
- 5.3 Features 18
  - 5.3.1 Signal validity 18
  - 5.3.2 Simulation modes 18
  - 5.3.3 Counting path depth 19
  - 5.3.4 Black box components 19
  - 5.3.5 Building circuits 20

5.3.6 Word wires 22
5.3.7 Correctness checking 22
5.3.8 Range of pre-defined circuits 23
5.3.9 Integration with existing systems 25
5.3.10 Descriptions 25
5.4 Deployment 26
5.5 Maintainability 26
5.5.1 Documentation 26
5.5.2 Continuous integration 26

**6 Evaluation 27**
6.1 Use within a lecture 27
6.1.1 Results 27
6.2 Monitored user test 28
6.2.1 Task explanation 28
6.2.2 Monitoring 29
6.2.3 Evaluation questions 29
6.2.4 Results 29
6.2.5 Limitations 30
6.3 Unmonitored user test 31
6.3.1 Task explanation 31
6.3.2 Evaluation questions 32
6.3.3 Results 32
6.3.4 Limitations 33
6.4 Verification of program efficiency 33
6.4.1 Method 33
6.4.2 Results 34
6.4.3 Limitations 34
6.5 Overall 34
6.5.1 Verification for functional requirements 34
6.5.2 Verification for non-functional requirements 35
6.5.3 Meeting program goals 35
6.6 Unit testing 36
6.7 Automated user interface tests 36

**7 Conclusion 38**
7.1 Summary 38
7.2 Future work 38
7.2.1 Circuit Builder Improvements 38
7.2.2 Variable word size 39
7.2.3 User defined descriptions 39
7.2.4 Performance increase 39
7.2.5 Circuit animation 39
7.2.6 Alternative methods of deployment 40
7.3 Reflection 40

**Appendices 41**

**A Appendices 41**
A.1 User task sheet for unmonitored evaluation. 41
A.2 User task sheet for monitored evaluation. 55
A.3 User evaluation form for unmonitored evaluation. 60
A.4 User evaluation form for monitored evaluation. 66

A.5 Results for unmonitored evaluation. 73
A.6 Results for monitored evaluation. 99
A.7 Notes from monitored evaluation. 135

**Bibliography** **136**

# 1 Introduction

## 1.1 Motivation

There are many important reasons for computer scientists to have an understanding of digital logic. Firstly, from an academic viewpoint, those learning about computers should have, and will likely be interested in, knowledge of how computers work at a low level. This will help students to understand both the potential and limitations of modern computers. This knowledge also has practical applications as when looking at algorithms, students can understand why operations such as division and floating point arithmetic are likely to take longer to run than integer addition, subtraction or multiplication. An understanding of gate delays also allows students to understand how to build efficient circuits and why there are physical limitations in hardware. Understanding clock cycles allows students to understand memory in computers and to give numeric values to the time taken to perform a simple action inside a CPU.

An understanding of hardware can also be useful when looking to improve the speed of algorithms, as the students should be able to recognise that some operations are more efficient than others. A computer simulation is an easy, effective way of learning about digital logic and the evaluation at the end of this project evidences this. This project looks at the creation of a synchronous digital logic simulator. The focus was to create a useful learning tool for those with little knowledge of the topic, over a technically accurate model. This allows users to learn important aspects of circuits, without having to spend too much time on unnecessary levels of detail.

## 1.2 Goals

**Improve users' circuit knowledge** The overall goal for the program is to allow computer science students to improve their knowledge of digital circuits. They do not require complex knowledge, such as how analogue circuits or complex systems with multiple clock components function. As a result, it was decided that the program should not be required to simulate asynchronous logic. A simple model was devised where users could add only logic gates, I/O components and delay flip flops. All these flip flops would be attached to a single clock without any requirement for wires to show the clock signal as these clutter a large circuit diagram. Also it was found that, the simulation of asynchronous circuits is very complex, particularly for beginners, and to simulate a realistic CPU it is only necessary to have flip flops as well as logic gates. It was therefore decided that only synchronous circuits would be shown. The exact makeup of flip flops can be very different in different systems, but the important point to get across is a simple model of signals being stored and changing on each clock tick. Users would be warned if the circuit they had created was asynchronous and would not be able to simulate it.

**Simulate synchronous digital logic** The program should be able to simulate any synchronous digital logic circuit, considering some limitations, listed in section 3.2. This is essential to allow the simulation of all common circuits as well as allowing users to design any of their own and see their function and efficiency. Being able to see the resulting values when simulating inputs on a circuit is important to allow users to easily see what the circuit does and how it could be used.

Users should also be able to see how a smaller circuit can act as a component in a larger circuit and be able to look at the components making up these sub-circuits and simulate them.

**Use by students and lecturers** The program should be able to be used by a demonstrator in a lecture on computer hardware. This is important so that a lecturer can discuss the circuit with the group, as well as show students how the circuit works. The main considerations for allowing the program to be used in a lecture are that the circuit should be easy to read and understand and can be seen clearly on a projector screen. The other main user group is first-year students. Students should have the opportunity to understand digital circuits by simulating them at the pace that they choose. To allow this, it is important that the program is easy to use for new users and that the interface is intuitive.

**Users can load and build circuits** Users should be able to load standard circuits from a built-in library, this is because there are many standard circuits that are important to understand when looking at existing digital hardware. Users should also be able to build their own circuits in an onscreen editor. It was considered that they could design circuits only in a text editor. However, an onscreen editor was chosen because defining the positioning of components and wires is important, but very difficult and time consuming in a text file. Positioning could have been automatically chosen, although, this would be difficult to achieve and it would prevent users from choosing their own designs or editing designs to focus on a part of the circuit they consider important.

**Different simulation modes** Users should be able to perform either a single gate delay or a single clock tick on a circuit. This allows users to improve their understanding of circuit validity by seeing how each gate delay affects a circuit. Users should be able to understand that reducing the path depth is important in efficient circuit design by seeing how many gate delays are necessary to make each signal valid. They should also understand that each component takes time to produce an output from inputs.

Users should be able to simulate a full clock cycle on a circuit in a single step to improve their understanding of clocking. More complex circuits that include registers for example, the register transfer machine used in the first year computer systems course, are easier to simulate and understand if users can perform a full clock cycle in one click. Users can understand that values are only loaded into flip flops when a clock tick happens and see that signals must be valid before they can be loaded into flip flops.

# 2 Background

## 2.1 Synchronous Digital Circuits

Synchronous digital circuits provide all of the logical functionality for modern computers. To simulate the operation of any one of these circuits, it is enough to provide just 3 logic gates, AND, OR and NOT. These simply take in a single signal, or many, and produce an output based on that, according to Boolean algebra. This could be a 0, corresponding to a very low voltage, or a 1 corresponding to a value close to a chosen voltage, e.g. 5V or 3.3V. A short delay, called a gate delay, is also required. For simplicity of the model, this will be assumed to be constant for all gates. As the model is designed for academic understanding, the actual delay of hardware components is not required to be modelled. A delay flip flop is also required. Again a simplification is made that this will be the only type of flip flop available. In real-world circuits, other choices may be made based on the given application; however, this program is intended to give a general overview, without reference to specific hardware components.

A valid signal, in the context of this program, is stable and will not change as more gate delays pass. Only when inputs change or a clock tick happens will signals become invalid or change. The actual value of an invalid signal could be a correct 1 or 0 value, or could be incorrect. It could also be unstable i.e. an uncertain value between 1 and 0. However, this is not reflected in the program, as users should understand that they can only rely on valid signals. The path depth of a circuit is the number of gate delays required for every signal to become valid. Clock signals are also simplified to only be a signal to each flip flop that triggers a change in its value. The actual waveform of the clock signal was seen as too complex for beginners to understand.

## 2.2 Existing Products

Similar existing products were researched and evaluated before the start of this project.

### 2.2.1 Overview

There are a range of existing tools that can simulate digital logic. These can roughly be split into two categories, simple learning tools to provide basic understanding, and complex analysis tools that are used by those creating actual circuits. The second group is too complex for beginners, and provides an unnecessarily high level of understanding for the purpose of giving computer science students an overview of hardware. For example, there might be different logic components to show the differences between components made by different companies. The actual position of ports on the real hardware devices could also be modelled. Power lines going into each device could also be modelled to show where they would be on the real circuit. These are clearly unnecessary for beginners and modelling them would add unnecessary levels of complexity.

There are many simple learning tools, aimed at beginners that are available for students to use. However, the quality and effectiveness of these as learning tools varies between the products available. These products are more relevant to this project, and this chapter focuses on an analysis of them. After research it was found that there were far too many of these to usefully evaluate,

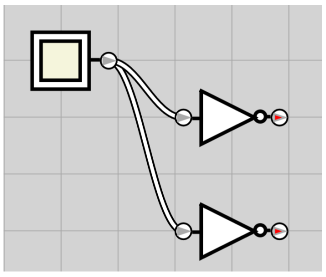


***Figure 2.1:*** *Screenshot showing wires paths from the program Logic Gate Simulator. This was initially considered as a possible solution to the problem of how to route wires for this project.*

likely over one hundred. Many of these were proprietary software. Three existing products that were designed with similar goals to this project, and were freely available were chosen for this evaluation. These products were evaluated to see how well they could meet the goals described for this project. Study of these programs helped to define the requirements and objectives for this project.

### 2.2.2 Logic Gate Simulator

Logic Gate Simulator is an open source project aimed at teaching circuit basics to computer science students. Kollmansberger (2011) The software is said to have been inspired by Logicaly LLC (2019), however, Logicaly is proprietary software and could not be accessed for this evaluation.

#### Advantages

- The editor allows the creation of any circuit that is a combination of logic gates and includes single bit input and output as well as a clock component.
- Wave-forms can be shown these give and an idea of the actual signal on wire when it is in a valid or invalid state. This allows for complex analysis of the effect of gate delays during a clock cycle.
- Adding gates and wires is easy and intuitive. Unlike in most editors, wires are not lines that are always at right angles to each other. They curve from one port to another as shown in figure 2.1. This makes it easy to place wires as all of the wire positioning is performed by the program. It also allows components to be easily moved as the software can easily reposition the wires.
- "Integrated circuits", which are referred to as black box circuits for this report, can easily be created while building a full circuit by dragging over the required components and pressing a button to create the new component. This can then be given a name and added to the circuit as many times as required. This allows users to easily build up complex circuits.
- The "flatten circuits" option allows black boxes to be replaced with their actual components. As a result, users can use black boxes to help build circuits, without them being part of the design of the final circuit.
- Users can also see the internals of a black box by simply double clicking on it. This also shows the circuit animation as values change. This allows users to analyse a complex circuit without it taking up the whole screen.
- The circuit as a whole can be animated to show values changing on gate delays and clock ticks. The speed of this can also be changed. This allows users to have a greater understanding of the fact that it takes time for signals to move across gates.

**Disadvantages**

- Lacks an easy way of displaying to users when a signal is valid. Users can look at a wave form diagram over time to work out this information, however, this is difficult to understand, particularly for beginners.
- When starting the program, users can only build circuits from a selection of basic gates as well as some I/O components. This means that each user would have to build common circuits like a multiplexer or adder from scratch to understand it.
- Flip flops have not been added to the program as primitive components. Instead, users could build these from asynchronous combinational logic and a clock signal. This would be difficult, particularly for beginners who do not understand the function of clock signals in a circuit. It may also give the impression that this is exactly how a gate will function asynchronously, when real hardware is actually more complex.
- It is very easy for users to add multiple clocks to the circuit as well as to use gates on clock lines. While this may be very useful for a small number of users who want to look at very complex or unusual circuits, it could give a false impression to beginners that this is a very common use of these components.
- The unusual technique for wire positioning used in this simulator is easy for beginners, but makes complex readable circuits particularly difficult, or simply impossible to make. While most wires are positioned appropriately, feedback wires travelling from the end of a large circuit to the beginning simply go under all of the other components, making large circuits difficult to read.
- Another limiting factor when using this program to build larger circuits is that each wire can only transfer a single bit of information. Creating a circuit that has signals that are words would be difficult and create a very cluttered diagram.

**Overall** While building combinational circuits is easy, this editor is limited in its ability to allow users to create and simulate large synchronous circuits. It is therefore not ideal to teach students about the importance of the clock in digital circuits. It would also not be ideal for creating circuits to be shown to a group of students in a lecture, or as a demonstration, as the circuits created can be difficult to read and understand.

### 2.2.3 CircuitVerse

CircuitVerse is an online tool used by first years in the University of Glasgow to understand digital circuits. It is an open source program aimed at teaching beginners. Bangalore (2019)

**Advantages**

- Building circuits including a range of logic gates, flip flops, word wires and black box components is easy and intuitive.
- CircuitVerse is available online and could easily be accessed by students. Circuit diagrams can also be embedded into any web page making it easy to share diagrams. As well as having a huge range of circuits available, that were made by users from the large online user base.
- Black box components can easily be created from existing circuits. They also allow ports on the components to be named to make it easier to integrate them into other circuits.
- A good range of components are included, such as multiplexers and complex I/O devices. This makes it easier for users to learn about important circuits without having to build them from scratch.
- Users can also change options on components, for example, to give an AND gate 4 inputs instead of 2 or change the clock speed. This makes circuit building easier and will help beginners learn about important components such as multiplexers.

- Combinational Analysis allows users to define circuits from their truth table. This is both useful and accessible to users who do not have a high level of experience with circuits.

**Disadvantages**

- Synchronous circuits must have clock lines. This means that larger circuit diagrams become cluttered.
- It is not possible to see the internals of built in circuits such as a multiplexer. This would have allowed users to understand that this circuit can be built up from a few logic gates and improve their knowledge of the circuit.
- Word wires look the same as single bit wires. This could potentially confuse users.
- Users must create an account before they can save circuits. This could be annoying for student users who are only using it a small number of times.

**Overall** CircuitVerse is a useful program for beginners learning about digital logic. There are a few possible areas for improvement including allowing synchronous components to have a connection to a clock that is not shown that could improve the program as a learning tool for computer science students.

### 2.2.4 LogicWorks

LogicWorks was used in the University of Glasgow first year computer science course by students. It is a complex program, designed not only for students but also for those who are building actual circuits. For example, it allows the use of VHDL to create circuit specifications. This is a complex language designed for specifying actual circuits. Only the elements likely to be used by beginners will be evaluated here. DesignWorks (2019) This is proprietary software, but a demo version with all of the key features of the original was available for testing.

**Advantages**

- LogicWorks has a wide range of different example circuits and components, including flip flops that users can load and simulate. This is useful for beginners who want to understand how common circuits such as a multiplexer work.
- LogicWorks has helpful information for new users when the program is started.
- Users can specify lots of details when building black box circuits such as specifying the names of ports. This can be useful when adding them into larger circuits. These can also be saved for reuse.
- Diagrams can be made clear and readable with the help of being able to name components, route wires anywhere at right angles, and add wires that carry more than 1 bit.
- Circuits can be simulated to show changes in signals over time.
- Separate modes of operation for building and simulating circuits allow user to focus on these separately.

**Disadvantages**

- While a range of components is useful for beginners, LogicWorks has too many making it confusing. These components are designed to simulate real hardware components which is an unnecessary level of complexity for beginners.
- The user interface is outdated. There are many small icon buttons, most of which would be of no use to a beginner and make it more difficult to work out how to use the software.
- LogicWorks can be used to show how the validity of signals changes over time, however, this is only by looking at a complex waveform diagram. This is too complex for beginners.

**Overall** LogicWorks provides all of the functionality required to meet the goals of this project. However, it is difficult for beginners learning about digital logic for the first time. It is also difficult to use, regardless of the users' circuit knowledge.

# 3 Analysis/Requirements

This project was presented by a University of Glasgow lecturer, John O'Donnell, who required an effective tool to teach university students with little or no knowledge of digital logic the basic concepts. A list of requirements was created in weekly meetings with this lecturer. This was an agile process where the newest version of the program and design documents were presented and discussed each week. Documentation such as the lecture notes for a beginner university course on computer systems was also used to aid program design. This provided detail of the level of understanding required in this course. Further requirements were gathered through two forms of evaluation. Firstly, the program was used in a demonstration in a lecture, and then in user evaluations by students. This allowed iterative improvements to both the program and the specification based on the needs of real world users.

## 3.1 Problem Specification

The main expectations for the final program are listed below. The main aim, as stated in the introduction, was to allow users to improve their understanding of digital hardware circuits. It was hypothesised that this could be achieved by allowing users to both build and simulate logical circuits. As the program allows users to perform actions such as a gate delay on a circuit, they can, in their own time, learn to understand how each component works. This hypothesis is tested in the user evaluation later in the project. The list of functional requirements below describe actions that users should be able to perform with the program. The list of non-functional requirements are requirements of the program itself.

**Functional requirements:**

- Add synchronous digital logic circuit components to the onscreen simulation. These should include primitive components such as logic gates and flip flops and add wires between onscreen components. This allows users to build circuits to be simulated, helping them understand individual components and how they interact with each other.
- View predefined circuits such as multiplexers and load them as black boxes inside a larger circuit. This allows users to understand how common digital components work as well as how they are used.
- Be told where a circuit is invalid (asynchronous). This allows users to understand that having asynchronous parts in a circuit is poor design and could lead to problems such as instability.
- Add input components whose values can be changed and clearly see the output values and intermediate values on wires and inside flip flops and registers. This allows users to understand the circuit simulations. It should allow them to understand the functionality of a synchronous digital circuit and how it works internally.
- Recognise where the value on a wire/ output is invalid. This shows that values inside a circuit only become valid after a certain number of gate delays. Users should understand from this that it takes time for a circuit to produce a value, and that the final result cannot be relied upon until that time has passed.

- Simulate a single gate delay and see the resulting change in values. This makes it easy for users to see which gates become valid on each gate delay and when gates become valid simultaneously. This can encourage good circuit design. For example, users can see that building a 4-bit adder as a tree of 2-bit will create a more efficient circuit than a simple line of 2-bit adders.
- Simulate a single clock tick and see the resulting change in values. This allows users to understand that loading values into a flip flop or register requires a clock tick and that signals must be valid to be loaded correctly into a flip flop.
- Simulate a complete clock cycle in a single step and see values loaded into flip flops. This allows simulation of more complex circuits, such as the RTM circuit without users having to repeatedly press the gate delay button. It also makes the program useful to slightly more advanced users who want to understand more complex circuits.
- Save and load user defined circuits. This makes it easier to share circuits and restore previous work.
- View a description of each component to understand its function. This should provide assistance to users who have very little understanding of digital circuits. It should also allow users to understand how circuits work even if they fail to understand that from looking at the circuit and simulating it.
- View a description of each black box component when deciding to add it. This should allow users to choose the correct component when building a larger circuit.

**Non-functional requirements:**

- Should be able to be used by beginners who have no experience of using it and little knowledge of digital circuits, as this is the main target group of users.
- Should be usable without large amounts of external documentation. Users are less likely to make good use of the program if they have to read a long document to understand how the program works.
- Must run on University of Glasgow first year computer science lab computers, as well as on any modern windows machine. This will allow both students and lecturers to use it without difficulty.
- Must be able to simulate circuits up to the size of the Register Transfer Machine circuit at a reasonable speed. The RTM is the largest circuit that beginners are expected to understand, and any long delays could be annoying and confusing for users.

## 3.2 Chosen Limitations

The digital hardware making up modern computer systems is complex and varies with each system, and even with each component. It is also continuously changing over time as new manufacturing techniques are developed. However, these can be abstracted into a simpler model that has been consistent in most modern hardware. This model can be understood by computer scientists studying digital circuits for the first time, so was chosen for this project. A number of limitations were chosen to simplify that model for first time users as listed below.

- It is assumed that every component is connected to a power source, without these lines being shown and cluttering the diagram.
- Every synchronous component is connected to the same clock, and again, these lines are not shown. This allows a realistic CPU to be simulated, and any more complex use of clocks was decided to be outside the scope of this project. It also represents the most common use of clocking in computers, where it is desirable that all synchronous components get a clock signal at the same time.

- Every gate is assumed to have the same time delay. This gets across the important idea that it takes time for a signal to pass through a gate. It also allows users to focus on building circuits that give the correct result, not on swapping one gate for another to make it more time efficient. It is more important for users to understand that changes making gates work in parallel or adding flip flops between long sections of combinational logic can reduce the path depth, over understanding that some components have a reduced gate delay. In real circuit design, gate for gate changes could be done by a computer program or algorithm, whereas the more complex aspects of design require detailed understanding of good circuit design.
- Wires are assumed to have no delay. This is because, at an abstract level, their delay is almost negligible in comparison to a gate delay. The circuits produced on this program are designed to be human readable and it would not make sense to encourage users to make wires shorter, as this is very different from real circuit design.
- Asynchronous circuits, circuits that have feedback in combinational logic, cannot be simulated. This was decided as these can be very complex to model and can be unpredictable. They are also not relevant to an understanding of computer hardware, with the exception of understanding the makeup of flip flops. Users should instead understand that asynchronous circuits should be avoided for good circuit design.
- Delay flip flops are assumed to take no gate delays and only act on clock ticks. This allows users to instantly recognise the result from the clock tick to improve their understanding of the component.
- Inputs and outputs are greatly simplified. Protocols for connecting to real-world I/O devices were decided to be outside the scope for the project, and it is enough for users to see data stored in computer memory and values on wires that allow data to move outside the system.
- Each wire has only one input so that it will have only one value. Unstable values are too complex and not necessary for beginners.

## 3.3 Changes to the specification

The specification was changed due to evaluation and conversation with the customer as described at the beginning of this chapter, as well as due to time constraints during development. Each possible feature was evaluated based on the time it was likely to take as well as its usefulness. As a result, the on screen circuit builder was planned to only be built if there was time left after the circuit simulation component of the project was complete. This meant that early development focused on loading from a simple file specification that would allow circuit save files to be written by users. This was no longer the focus of development after it was found that there would be enough time to create the on screen circuit builder. It was also found from use of the program that an on-screen circuit builder would make creating example circuits far easier, saving development time.

# 4 Design

## 4.1 Simulator Logic

This section describes the internal logic, data structures and algorithms within the program.

### 4.1.1 Signal representation

The key goal when designing the underlying circuit logic was that logical operations within gates were very efficient. One of the aims of the project is to simulate the register transfer machine circuit, this has a total of 170 components, excluding I/O. On each clock tick, signals should move through all of these without a noticeable delay. It was, therefore, decided that signals should be represented by just 2 booleans. One to store whether or not the signal was valid, and another to store the value 1 or 0. This also allows boolean operations such as AND and NOT to easily be carried out.

### 4.1.2 Gate Delays

The next challenge was to simulate a gate delay on the circuit. This has a potential issue in that when a gate delay happens each component must calculate the value of its outputs based on the value of its inputs at the start of the gate delay. This could mean that if components are processed in the wrong order, the signal could move through multiple components, as the first giving an output changes the input of the second. To avoid this issue, gate delays were performed in two separate stages. Firstly, each component looks at the values that are stored in its input ports and changes the values stored in its output ports based on these. Each input or output port on a component stores a logic value, not the wires themselves, to allow this. Secondly, each wire copies the values from its single input port to its one or more output ports. This ensures that gates are acting on the correct input value. However, this technique has limitations as each gate is processed for each gate delay. This is inefficient considering that only gates at a given path depth, distance from an input or flip flop, need to act on each gate delay. The future work section 7.2 describes an algorithm that achieves this. However, the algorithm used does not have a noticeable effect for users on even the largest circuits simulated as shown in section 6.4 and changing it was not seen as a high priority.

### 4.1.3 Word Wires

Another design consideration was how to represent wires that can carry multiple bit signals. The simple solution was to store an array of the logic objects described above that have two Boolean values for each bit. The program was adapted so that each port had an array of logic objects. To ensure that the size of these never changed, the array was populated with these objects on creation, and then only the values in the objects were allowed to change, not the objects themselves. This also made splitting the word wire into bit wires as vice versa a lot easier. Two new components were created, one with a word wire input and many bit outputs and one with many bit inputs and a single word output. These do not require a gate delay to function as they are only added to make the diagram easier to understand. The values from the list of logic objects in the word port of a splitter can easily be copied to the values in the single bit ports on the output.

### 4.1.4 Black Box Components

Black box components within the program are components that contain multiple internal components. It was decided that these should include both predefined components, such as multiplexers, that are standard when building larger circuits, and also the ability for users to define their own black box circuits. To allow this it was important that the internal structure of black box components should be a combination of primitive components and other black box components. This allows users to build black box components in the same way that they would build any other circuit. It also allowed the reuse of the existing design for save files, with only some extra metadata.

### 4.1.5 Save Files

The program's save files were designed to be simple text files that an advanced user could create or edit. Information such as the rotation and number of ports on any given component were set to default values, if not specified, to make it easier for users to create circuits using this file notation. The notation also allows the use of variables to describe the positioning of a component or wire, as well as having pre-defined variables that allow the positioning of corners on wires to be determined based on the position of ports on components. However, this was not the focus of development as it was concluded that for most users the on-screen circuit builder was going to be the only way that they would build circuits. The file description method was, however, very useful in early development, as it allowed the circuit simulation element to be the focus of early design and implementation, not the circuit simulator.

## 4.2 User Interface

The user interface was carefully designed so that users could quickly learn the basics and start to build and simulate circuits. This section describes some of the techniques used during that design process.

### 4.2.1 External consistency

The user interface was designed to be easy and intuitive to use. In order to achieve this, designs of existing graphics and drawing programs were considered. This was so that users who may have never used any type of circuit simulator could still find the interface familiar. These had a number of similarities that are listed below.

- A toolbar allowing placement of components is on the left-hand side.
- If an information panel is included it is on the right-hand side.
- The main drawing area takes up most of the screen, allowing users to interact with as much of the diagram as possible.
- An options toolbar is at the top, as with most desktop computer programs.
- Scroll bars and zooming functionality allow users to easily navigate a large circuit onscreen.

The drawing programs studied are listed here: Ltd (2019) GIMP (2018) Photopea (2013) .

All of these features are included in the program's design shown in the wire-frame in figure 4.1.

This technique of taking inspiration from existing programs was decided as a good starting point to designing the user interface after a review of literature describing best practice in design.

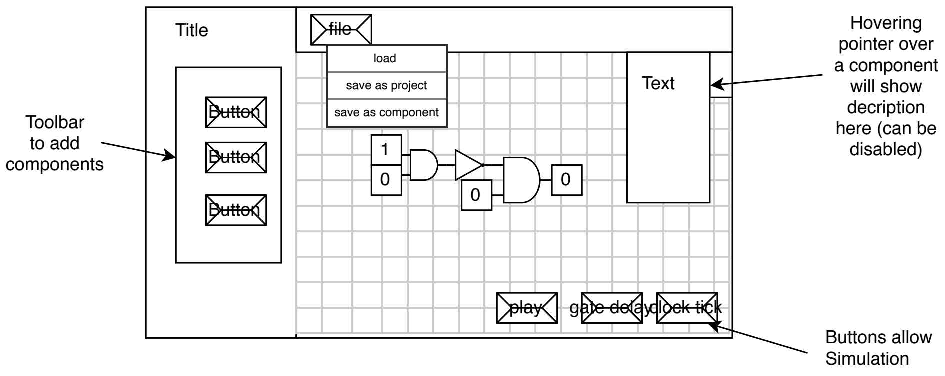


***Figure 4.1:*** *Initial program wire-frame showing the key features and some possible user interactions. This design was revised during requirements gathering and implementation to create to the final design described in the implementation chapter, so it does not represent the final design.*

External consistency allows users to understand a program more easily as it is similar in design to a program they have used in the past. Schlatter and Levinson (2013) This technique is aimed to ensure that users could easily start to use the program and focus on circuit design, not problems such as how to add components onscreen.

### 4.2.2 Constraints in design

One key difference between the design of the program created for this project and many of the programs studied in the background is that there are two clear modes of operations. Users start in a build mode where they can create circuits and press a button to move into a second mode to simulate the circuit. This was decided so that users can only simulate complete circuits, and error messages can be shown to the user when they attempt to simulate something incorrect, for example a circuit that has components with ports that are not connected to anything. It also leads to a less cluttered interface where only the buttons that are required for the current mode of operation are shown onscreen. This is an example of a physical constraint, users are prevented from accessing buttons that would perform an incorrect action. Norman (2013) Another example of the use of physical constraints is when users press a button to delete components or wires they can only perform that action as all other buttons are disabled until they press finish to end this mode of interaction.

## 4.3 Technologies

Time was taken at the beginning of this project to choose appropriate technologies. The technologies used and the reasons behind those choices are described here.

### 4.3.1 Java

Java was chosen as it is multi-platform and this was considered useful as computer science students are likely to use a range of operating systems, including Linux. Java is also easy to use and has a large number of supporting libraries and frameworks including well-developed build tools. This allowed for a focus on the given problem and an easy start to development.

### 4.3.2 Javafx

Javafx is a modern framework that is quickly becoming the standard for GUI in java. It uses an XML format for building screens, allows CSS styling and provides a good separation of concerns between a model, view and controller. It also has a range of libraries making it easy to solve common problems, for example, creating a standard screen layout. While this project will only focus on building a desktop application, it can also be used to create rich internet applications, for both web and mobile, from the same java code base. This makes the code highly portable.

### 4.3.3 Desktop application

It was decided that development should be focused on building a desktop application. There were two main reasons for this decision. Firstly, the goal of the application is for users to build, view, save and load existing circuits. This has no requirement for user accounts or external information sharing, which are made easier by the use of web frameworks. It also has a focus on access to user file systems, which is made easier by a desktop application that can store information locally. Also, there is no requirement to run a server for a desktop application. Secondly, as the program will be used within the University of Glasgow, it can simply be downloaded on to the appropriate machines.

# 5 Implementation

This chapter describes the steps taken to implement the program described in previous chapters and the technical decisions made during this process of implementation.

## 5.1 User Interface

The user interface was implemented in Javafx with an aim to make it simple and easy to use.

### 5.1.1 Model View Controller

Javafx presents both useful features and challenges for creating a program with a well-designed separation of concerns between the user interface and the core functionality of the program. A model view controller design pattern is built into the framework.

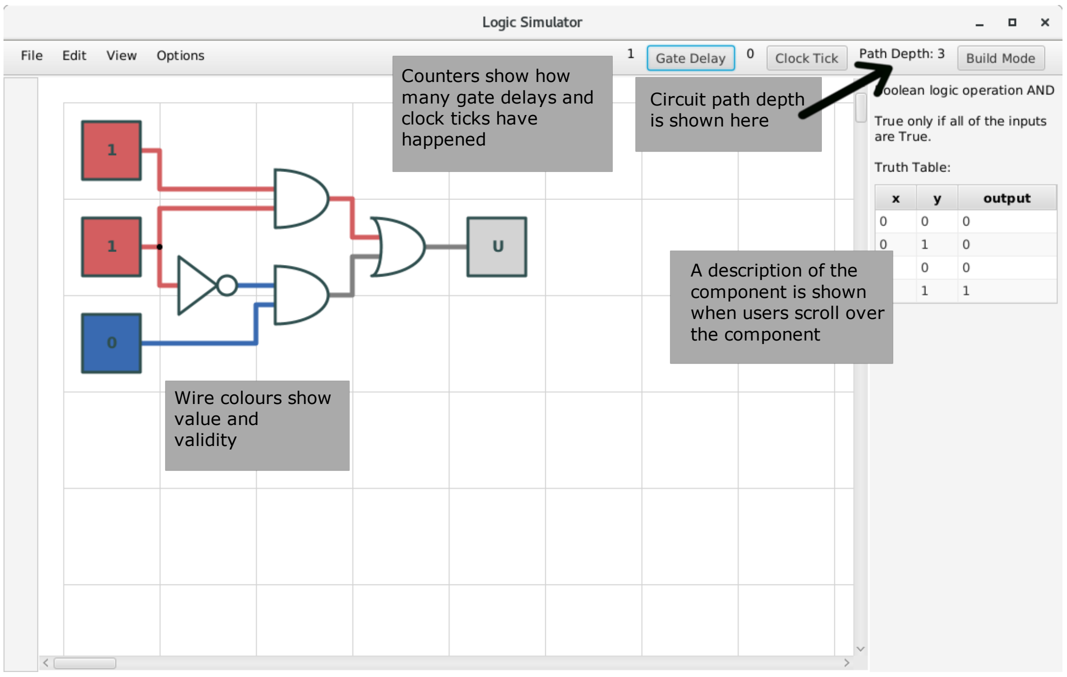
**Model** The model classes within the application are designed to perform any functions that are not related to UI components. As an example, each of the circuit components has a model to store its data and perform functions such as a logic gate operating on its input bits to produce an output. The controller class for the component can call these functions and access data from the model.

**View** The onscreen interface in javafx is built up using a custom XML format, called FXML, as well as CSS styling. This is well used within the program, building up almost all of the onscreen objects. It has a key advantage over some XML screen building formats in that it allows FXML to be inserted at run-time and allows the same object in FXML, e.g. an AND gate to be added many times. However, it does lack useful features that exist in some web frameworks, like allowing loops in XML. It was, therefore, far easier to build the background of the screen in its controller class, where the vector lines making it up could be built in a java for loop.

One good use of inserting FXML at run-time in the program is for the component descriptions shown in figure 5.1. The description changes very quickly, for example, if the user moves their mouse over the toolbar each component description is shown in turn. It is important that the file I/O required to load FXML was not part of this transition. Each description is loaded at program start and the onscreen component is added to a hash-map. These can then quickly be switched in and out at run-time. The XML is also built into the jar file, so will likely load faster than doing actual file I/O.

**Controller** There are two options in Javafx to automatically create a controller object for each FXML view that is loaded.

1. The Java class used as the controller can be specified in the FXML. When the component is loaded, a controller is created. The loading class can then access the controller and the view component from the loading class. This method is used for the main screen controller. However, it was found to have a major flaw in that the controller must have a no-parameter constructor. Initial versions of the program had a difficult to understand pattern of loading a controller and then setting its parameters. This also prevented the use



Counters show how many gate delays and clock ticks have happened

Circuit path depth is shown here

oolean logic operation AND

True only if all of the inputs are True.

Truth Table:

| x | y | output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| [illegible] | 0 | 0 |
| [illegible] | 1 | 1 |

A description of the component is shown when users scroll over the component

Wire colours show value and validity

***Figure 5.1:*** *Screenshot showing some key features of the program.*

of a controller class that could decide which FXML to load to based on parameters, making a ComponentController super-class very difficult to create.

2. The pattern used for most of the objects in the program allows a controller class to load its own FXML, passing its self to the loader as the controller class. This allows for a complex controller class that can choose FXML based on parameters. These controller objects can then be created with a single call to new like any Java class. However, with the limitation that the link between the FXML and its controller exists only at run-time, preventing compile-time checking that linked values are correct.

At the start of the project, background research was conducted on the Javafx design patterns used by other programmers. Many of the frameworks suggested provide only a limited amount of communication between controllers. Others used dependency injection, where controllers were singleton objects accessible in any part of the code for easy communication. Kruk et al. (2018) Fedortsova (2014) Bien (2014) Hommel (2014) However, a framework like this would make it impossible to have the recursive structure of a circuit simulation that has many components, some of which can have other circuit simulations, with more components and internal simulations. The pattern used in the Logic Simulator program allows controllers to hold references to all of the controllers for view objects that are inside their view object. This allows for any number of levels of recursion.

### 5.1.2 OS native appearance

Another advantage of the Javafx framework is that it allows the creation of components, such as a file load window, that use existing operating system features to give a native appearance on different operating systems. This gives the interface a familiar look, improving usability. It also handles issues, such as checking to see if a file exists when loading.

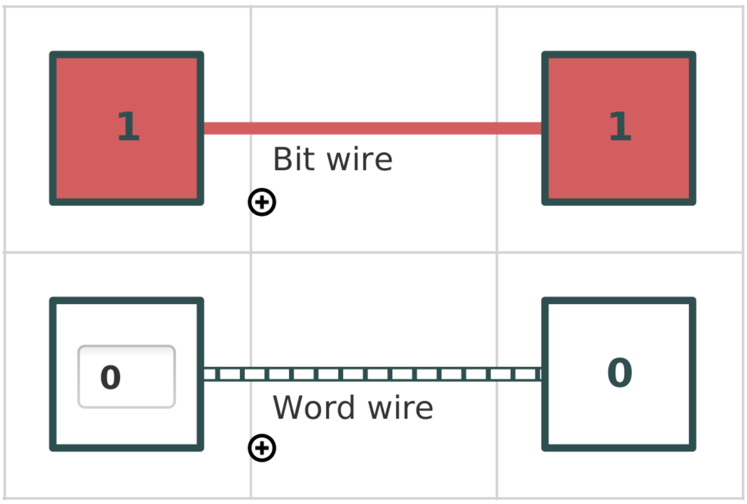


***Figure 5.2:*** *Screenshots showing the two different types of wires used.*

## 5.2 Aesthetics

It was decided that colours and basic design of common on-screen objects, such as buttons would use the standard Javafx styling. These were found to be user-friendly and were not highlighted as an issue in the user study. It also allowed the project to focus on problems specific to the task given, not on creating a unique user experience.

### 5.2.1 Component aesthetics

Components in the program use standard circuit symbols to ensure consistency with other examples, such as in textbooks or lecture notes. They also had to be easily differentiated from each other. The complete set of circuit symbols used in the program can be seen in table 5.1. Zooming on the program also added greater flexibility for users viewing components. This made the use of vector graphics essential to preserve definition at different zoom levels. The use of colour was also important. Blue, red and grey were chosen to represent different values on wires and components. Blue represents a 0 value, red represents 1 and grey represents an undefined value. These colours were chosen to contrast with each other to make the differences between them easier to see and understand. Vector graphics were created with a CSV style notation within the FXML defining the component. Creating them in a graphics program and then importing CSV files would have been an easier and more maintainable way of creating graphics. However, importing from a CSV file was not a feature in Javafx and would have been unreasonably difficult to implement just for this project.

### 5.2.2 Wire Aesthetics

Two different wire designs are used. Wires carrying a single bit are just a line, with the colour depending on the value going across the wire. Word wires are black with white stripes as shown in the screenshot 5.2. This is designed to show a clear difference from the other wire type and to look similar to a real component in electronics that provides a casing around multiple wires.

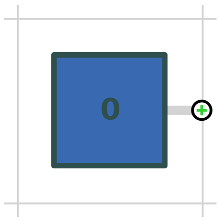
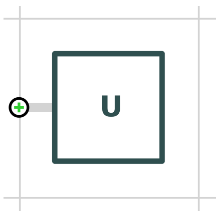
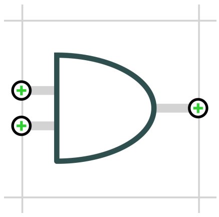
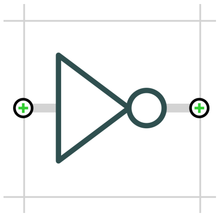
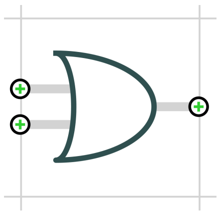
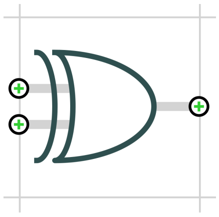
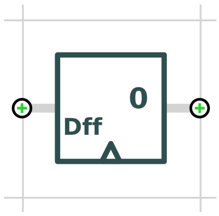
| | | | |
|---|---|---|---|
|  | Bit Input |  | Bit Output |
|  | AND gate |  | NOT gate |
|  | OR gate |  | XOR gate |
|  | Delay Flip Flop | | |

***Table 5.1:*** *Screenshot of components designed for the Logic Simulator program.*

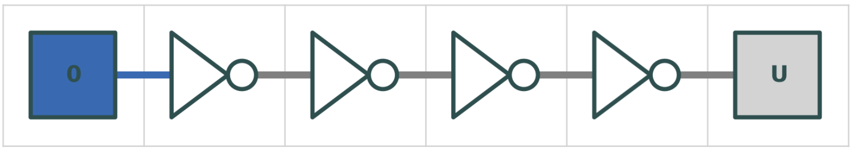
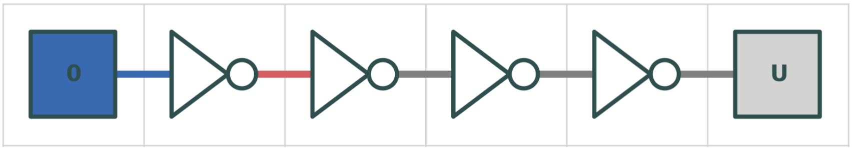
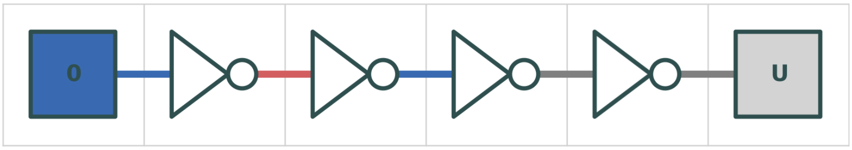
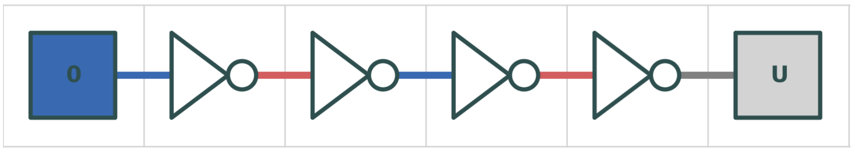
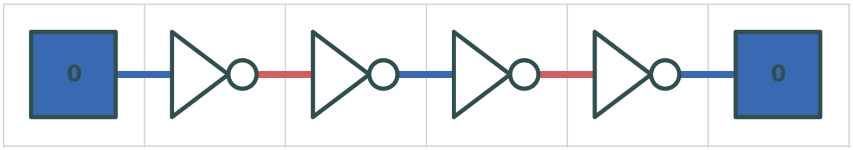
| | |
|---|---|
|  | 0 gate delays |
|  | 1 gate delay |
|  | 2 gate delays |
|  | 3 gate delays |
|  | 4 gate delays |

***Table 5.2:*** *Screenshots showing a valid signal moving through a circuit on each gate delay.*

## 5.3 Features

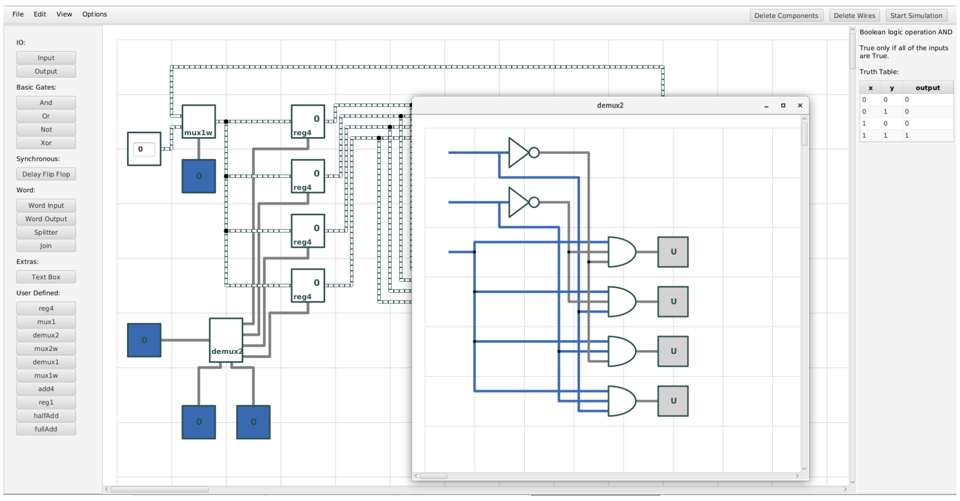
This section describes how the functional requirements for the system were implemented and lists some of most important and relevant features.

### 5.3.1 Signal validity

It is important for those learning about circuits to understand that it takes time for a signal to have a correct, stable value. Users of the program can press a button to see the resulting circuit after a gate delay has passed. They can repeat this process until all the signals are valid and see that more gate delays will not have any impact on circuit values. It can also be seen that if an input is changed or a clock tick happens then all of the signals in the circuit become invalid. Colours in wires and components make it clear to users whether or not signals are valid. This is useful to allow users to understand the concept of gate delays, which is fundamental to good digital hardware design. Table 5.2 shows how validity propagates through a simple circuit that has a row of NOT gates.

### 5.3.2 Simulation modes

The ability to simulate the functionality of a circuit was the key functional requirement for this project. One thing that makes this project stand out from similar existing programs is that this can be done in two different ways. Firstly, users can simulate just one gate delay on a circuit. This is particularly useful for users who are beginners and do not understand concepts such as what a gate delay is. It also helps to show why some circuits take more gate delays than others. For example, a 4-bit adder could be created with the same components to take either 2 or 3



Boolean logic operation AND

True only if all of the inputs are True.

Truth Table:

| x | y | output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

***Figure 5.3:*** *Screenshot showing a user opening a second screen allowing them to see the components inside one of the black box components on the main circuit.*

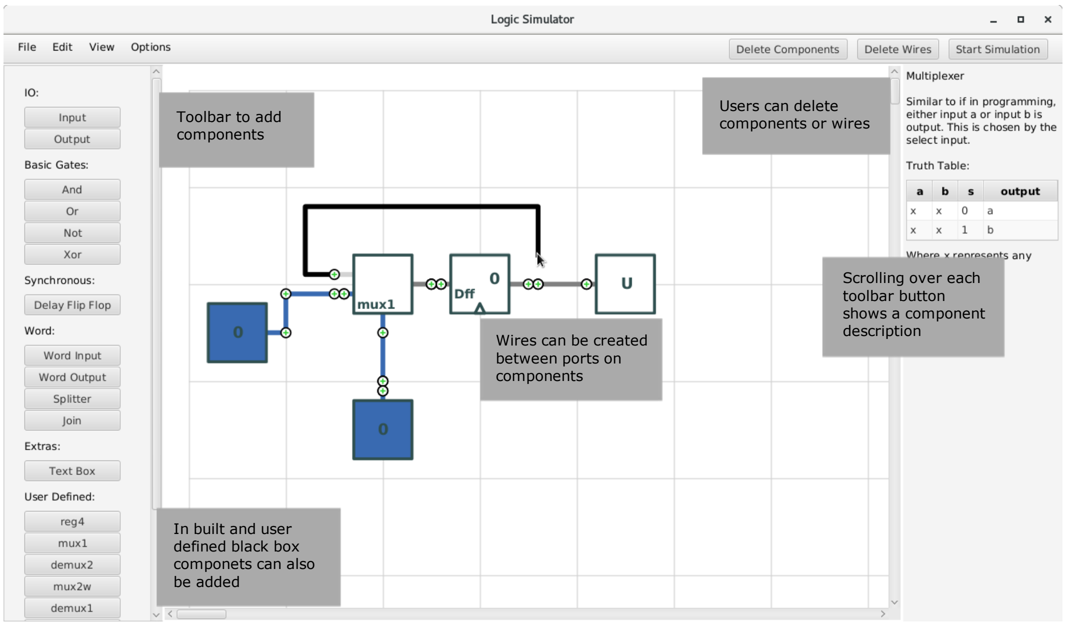
gate delays. Users can also change an option to simulate an entire clock cycle in one step. The program counts the path depth of the onscreen circuit and performs that many gate delays and then a clock tick. This is useful when simulating larger circuits that contain registers as loading a value into a register can be done with a single button press. Users are also able to individually perform gate delays and then a full clock cycle.

### 5.3.3 Counting path depth

When starting to simulate a circuit the total path depth for the circuit is calculated. Users can see the value onscreen 5.1. This allows users to gain an understanding of what path depth is, as well as compare the path depth of circuits. The implementation simply reuses the feature to simulate a gate delay on the circuit as a whole and counts the number of gate delays before all of the output components and flip flops have a valid signal on their input. This reuse of code avoids building a potentially complex algorithm walking through the circuit to count the gate delays required.

### 5.3.4 Black box components

Black box components are simply existing circuits built as a single component of a larger circuit. They are loaded from save files which are the same as regular save files with a small amount of metadata that determines how they connect to the outer circuit. It was decided that the internal structure of a black box in the program would be the same as the full simulation. The two structures inherit from the same super-class to allow the code to be reused. Then this internal simulation is simply a field of a sub-class of the component super class. This allows the reusable component to exist as any other and be processed in the same way. It was also then easy to allow users to see inside the circuits, as this interface already exists within memory. However, building the internals does come at a cost of time taken to load a given circuit, as described in section 6.4. This time could be reduced as described in the future work section 7.2. This was seen not to be a huge issue, as looking inside circuits is a useful feature. An example screenshot of a user looking inside a circuit is shown 5.3.



Toolbar to add components

Users can delete components or wires

Wires can be created between ports on components

Scrolling over each toolbar button shows a component description

In built and user defined black box componets can also be added

Multiplexer

Similar to if in programming, either input a or input b is output. This is chosen by the select input.

Truth Table:

| a | b | s | output |
|---|---|---|---|
| x | x | 0 | a |
| x | x | 1 | b |

Where x represents any

*Figure 5.4: Screenshot showing key features of the build tool.*

### 5.3.5 Building circuits

An onscreen circuit editor is included to allow users to build circuits to simulate. Figure 5.4 shows a screenshot of this aspect of the program. The main features of this are listed below.

**Adding components** Users can pick from a range of primitive components as well as pre-defined black box circuits. Components that users have created and saved into the component library will also be available in the toolbar to add as new components. Components can be dragged around the screen to position them. They also snap into position so that users can find it easier to create nice looking, easy to read circuits, for example, a circuit with components in a row would be much easier to create.

**Rotating components** Users can rotate components 90 degrees while clicking on them by pressing R. This allows for circuits that are more readable as inputs and outputs can come from any direction, as well as allowing circuits that move up or down the page as well as across. A good example of this in the 4-bit adder circuit, figure 5.11. This circuit has a 4-bit word with the least significant bit at the right and the most significant bit at the left. This is logical for a human reader.

**Creating wires** Each port on a component has a connector that can be clicked on to start building a wire. Users can left click to create a corner on the wire or to connect the wire to another port. If users right click, or move the cursor off the circuit part of the screen, they will stop creating the wire. If the chosen port is a word port a word wire will be created. The wires also snap into position, with their possible positions an equal fraction of the positions possible for the components. All of the ports are carefully positioned so that they are in line with the possible positioning of a wire. An example of a user building a wire can be seen in figure 5.5.

**Deletion** Users can also press buttons that will allow them to delete wires, as shown in figure 5.7, or components as shown in figure 5.6. The objects that can be deleted will be highlighted in red and will be deleted when clicked on. All other buttons are disabled when in delete mode, so users know that they will have to leave this mode to perform any other action.

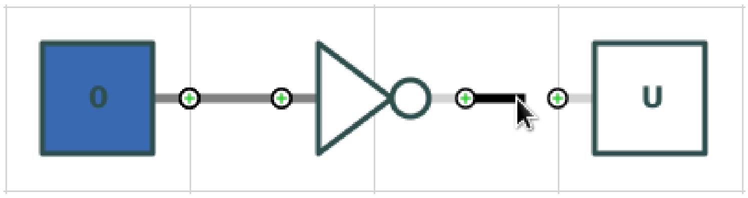


***Figure 5.5:*** *Screenshot showing a user building a wire between 2 components.*

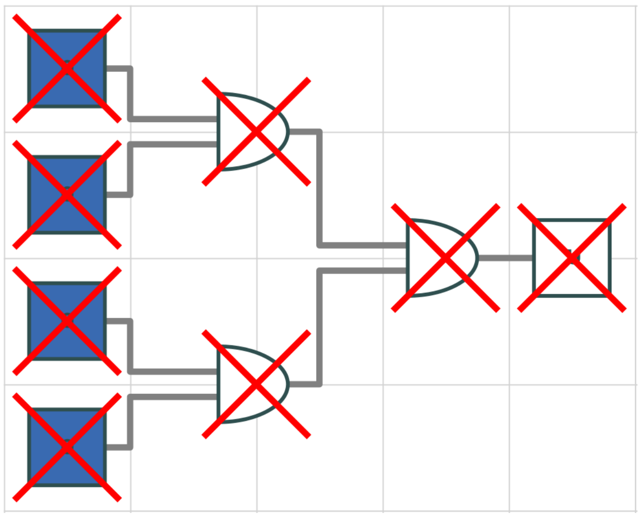


***Figure 5.6:*** *Screenshot showing component deletion mode.*

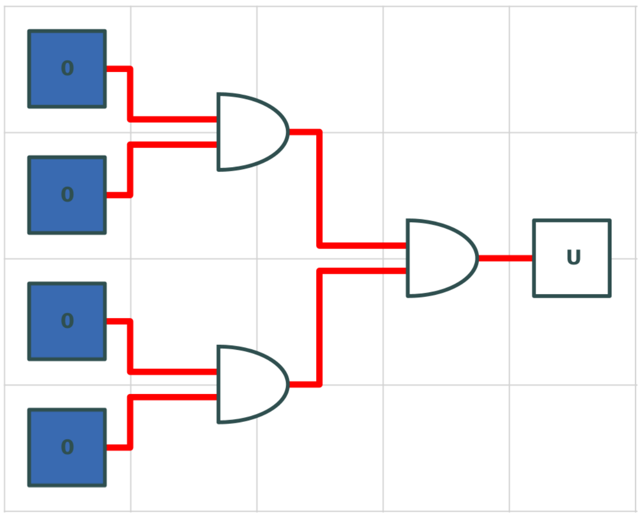
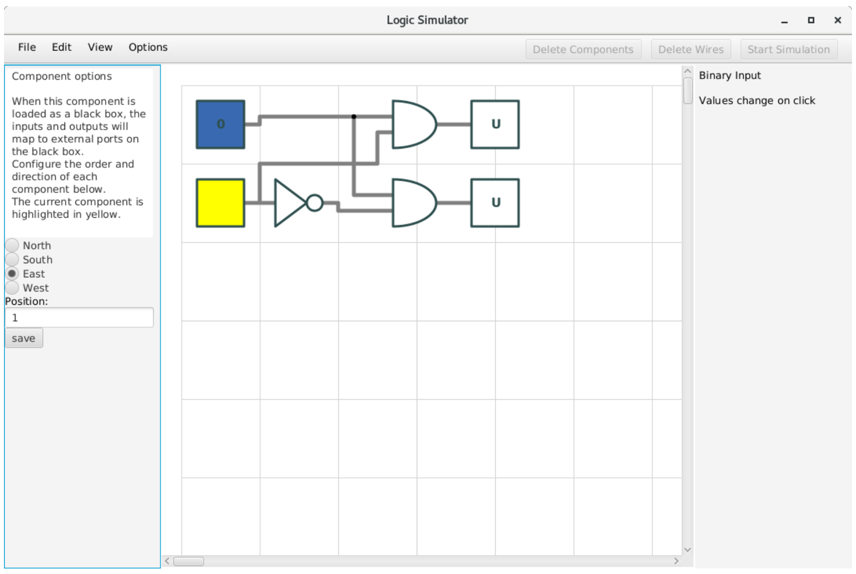


***Figure 5.7:*** *Screenshot showing wire deletion mode.*



Component options

When this component is loaded as a black box, the inputs and outputs will map to external ports on the black box.
Configure the order and direction of each component below.
The current component is highlighted in yellow.

North
South
[✓ East]
West

Position: 1

save

Binary Input

Values change on click

*Figure 5.8: Screenshot showing a user saving details of how a black box circuit will connect to a larger circuit.*

**Saving a circuit as a component** When saving a circuit for use as a black box inside a larger circuit, users can choose where the internal inputs and outputs in the circuit will be on the new component. When saving the circuit, each I/O component is highlighted in turn and users set two options, the direction that the port will be on the new component and the position on that direction. This can be seen in figure 5.8. Setting the position allows the ports to be ordered correctly. For example, in a multiplexer the ordering of the input wires is important.

**Text boxes** Users can add text boxes onto the circuit diagram. This could be useful to name components or to give any future users of the circuit further information. This could be particularly useful for a lecturer building circuits to be loaded on the program by students. Text boxes are used in the screenshot in figure 5.2.

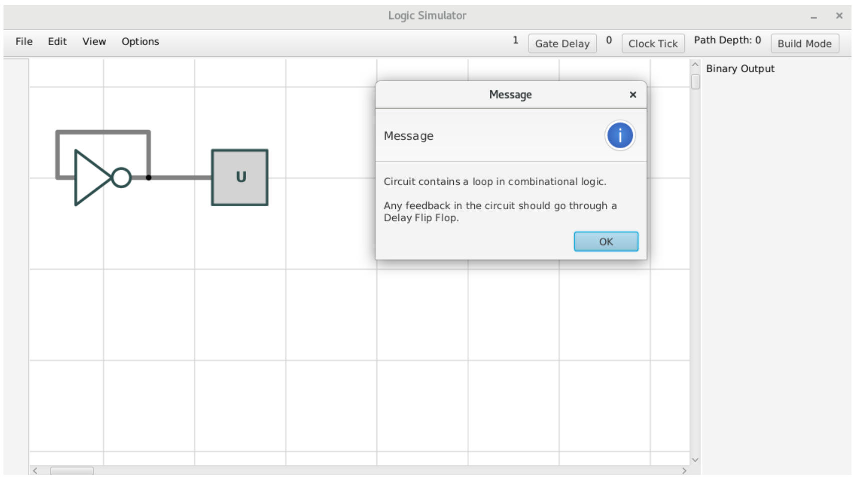
### 5.3.6 Word wires

As word wires are represented as a list as described in the design section 4.1.3, a subclass of a Java ArrayList was chosen to implement them. To avoid the constant creation of logic objects for this list, new functions were created to copy values between these objects and between lists of the objects.

### 5.3.7 Correctness checking

To improve the knowledge that users have of circuits it is important that users understand when and why a circuit they have built is incorrect. This should allow them to know how to change the circuit so that it is able to work correctly. They may have made an error because of their poor knowledge of circuits, or a poor understanding of how the program works. It was decided that the program should warn users when they attempt to simulate a circuit that is not suitable, so that they know how to change it. Some examples of this are listed below.

**Asynchronous circuits** While calculating the path depth of the circuit, the program checks to see if performing a gate delay changes any values in the circuit. If this is not the case, and not all signals are valid, the circuit will never become valid because there is a feedback loop in a



Message

Message

Circuit contains a loop in combinational logic.

Any feedback in the circuit should go through a Delay Flip Flop.

***Figure 5.9:*** *Screenshot showing an example of an asynchronous circuit and the error message shown when a user starts to simulate the circuit. The program will change back to build mode when the user presses ok or cancels the message. All of the program error messages use the same format as this.*

combinational circuit. The path depth function terminates and users are warned that the circuit is asynchronous. A screenshot with an example of an asynchronous circuit is shown in figure 5.9. As described in the limitations section 3.2 the program is not expected to simulate these circuits. Users should instead learn that this is not good circuit design, with the exception of building flip flops and these are already given to users as primitive components.

**Unconnected circuits** Users are also informed if they have components that have ports not connected to anything. Components with unconnected inputs could never give a valid output, and unconnected outputs may suggest an oversight by the user. It was, therefore, decided that it should not be possible to simulate these circuits.

### 5.3.8 Range of pre-defined circuits

Circuit design reuses common components that must be well understood by users. To gain an understanding of computers at a very low level, understanding certain circuits is particularly important. A list was drawn up in a meeting with a computer systems lecturer of required circuits.

- Multiplexers - Three multiplexer example circuits have been included. The first has a control signal that chooses between two single bit signals, shown in 5.10. The second also has a single bit signal that chooses between two words. Finally, a multiplexer that has a two-bit control signal that can choose between four different word inputs was included as this is part of the RTM circuit.
- Demultiplexers - Two demultiplexers were included in the program. They have 1 and 2 control signals respectively and both only output a single bit on the chosen output line.
- Adders - A half adder, that adds two bits, a full adder that adds three bits and a 4-bit adder that adds two 4 bit words, were included. A 4-bit adder is shown in 5.11.
- Registers - Single bit and 4-bit registers were included. Each has a data input and a load control.
- Register Transfer Machine - This is an important circuit in the University of Glasgow systems course. It has 4 word registers. Values can be loaded into any of these chosen by control signals. Values from these registers can also be input into an adder and the final result can be loaded back into one of the registers. A 4-bit version of the circuit was

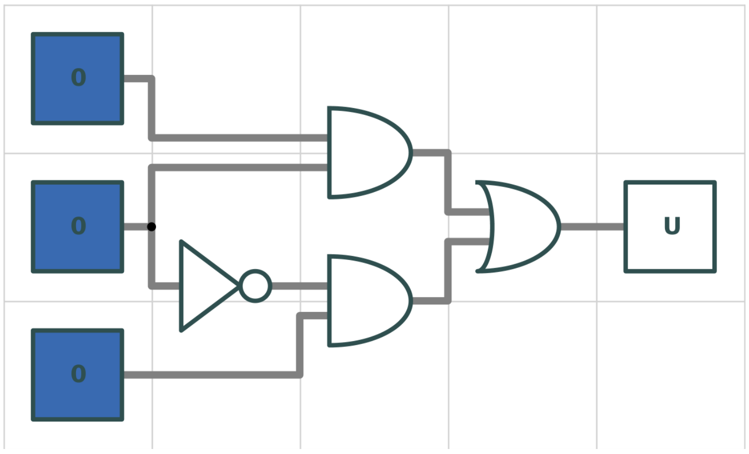


*Figure 5.10:* *Screenshot showing a 1 bit multiplexer circuit.*

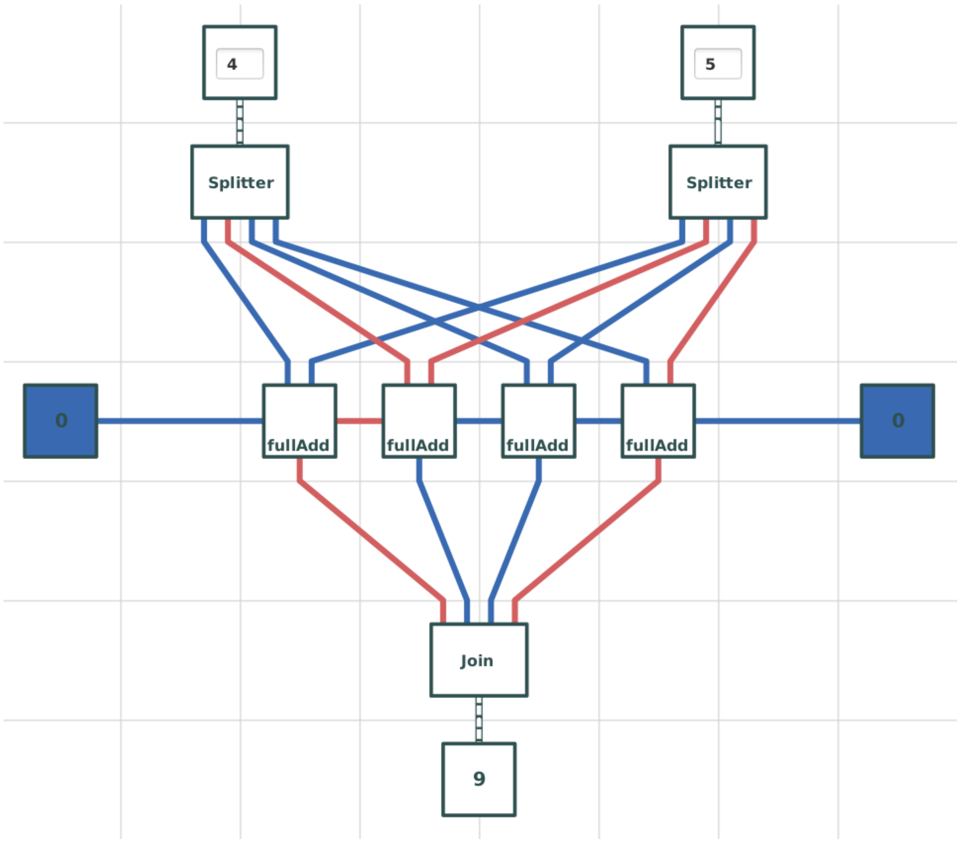


*Figure 5.11:* *Screenshot showing a 4 bit adder.*

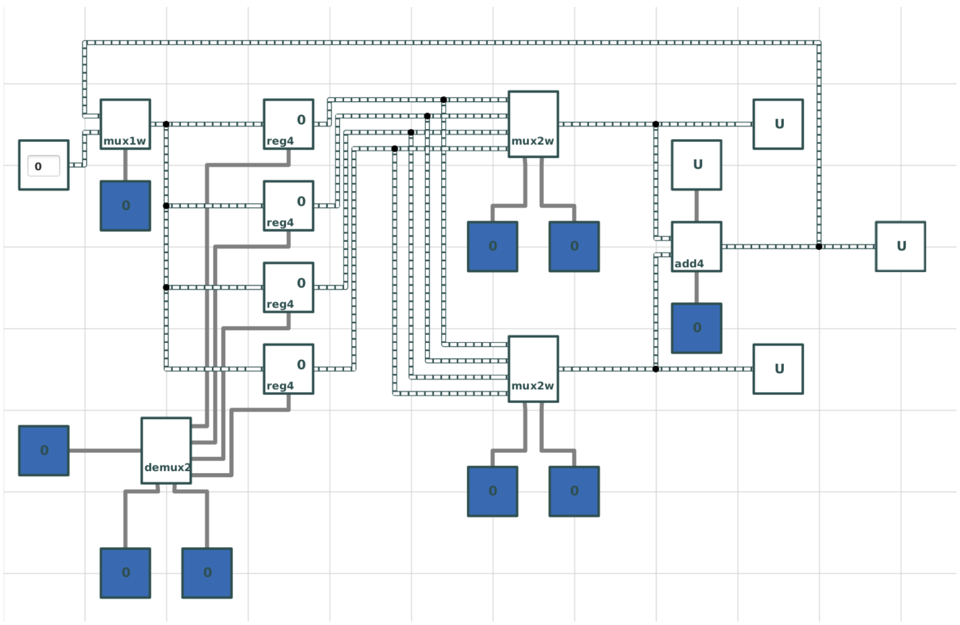


*Figure 5.12: Screenshot showing a Register Transfer Machine.*

chosen as this makes it clear that word wires can carry integer signals without having too many bits to be easily shown on-screen. Several versions of this circuit are included as pre-defined circuits within the program. Each adds more complexity, and the final one has all the functionality of the full circuit, shown in 5.12. This makes it easier for users to slowly build up a complete understanding of this circuit.

### 5.3.9 Integration with existing systems

This project was completed primarily for a University of Glasgow computer science course. To allow the program to integrate with the existing course, it was decided that there should be consistency across the tools used. To achieve this, naming conventions for in-build components in the program were chosen to be the same as those used in hydra, which is used to simulate more advanced circuits in courses at the university. These names follow common conventions from textbooks and real-world applications of circuits, so will also be relevant to users with no connection to the university.

### 5.3.10 Descriptions

Each of the primitive components, as well as the pre-defined black box components, has a description that users can see either by moving their cursor over the component within the circuit or moving their cursor over the toolbar button allowing them to add that component. This allows users to have a greater understanding of how the component works. Where it was seen to be appropriate, a truth table showing the outputs for all possible input values is shown, figure 5.1. Users cannot currently create descriptions for components that they build themselves. This is listed as future work for the project in section 7.2.

## 5.4 Deployment

When deciding on methods of deployment, the target users were considered. There were two main groups, students in a computer science department and lecturers in computer science. Firstly, in a computer science department, it was likely that an up to date version of Java was installed, however not necessarily the Javafx framework. It was also likely that users or system administrators would prefer that any new program have a small file size. To meet this need, a jar file was created which packaged the Javafx framework. This was fairly small at approximately 10MB. It could also run on Java 8 that was installed on in the University of Glasgow first year computer science lab.

Secondly, to ensure that it would run on the personal machine of most users, a version of the program was created using jlink. This tool packages all of the relevant java modules into a custom Java Runtime Environment. This can then be packaged with the tool to run natively on a chosen platform. The program was packaged like this for windows, including a .exe file that could be easily run. This second version of the program was easier to use and could run on windows machines without the need for external software. However, this came at a cost of space as files were approximately 100MB. It also was not platform agnostic. The future work section describes how similar versions of the program could have been created to run on macOS or Linux.

## 5.5 Maintainability

An important goal for the project was to allow future developers to maintain the project and to add new features. Steps that were taken to ensure that the code-base was of a high enough quality to allow this are listed below.

### 5.5.1 Documentation

**Issue tracking** Issue tracking was used from the beginning of the project on git hub. Each code commit referenced an issue and had a short message describing what the code was intended to do, and/or what goal it was trying to achieve. This aimed to help future developers understand the code-base, fix bugs and add new features. A list of objectives for the project are also included in the repository.

**Developer readme** A readme for developers is also included in the project. This explains how to set up the project for development and two possible ways to deploy the project. This should make it easier for future developers to create new runnable versions of the program.

### 5.5.2 Continuous integration

A continuous integration server was set up to remotely run unit tests and UI tests on each commit. Information on the test results and the test coverage is shown on the project repository. This allows the project to be maintainable by future developers.

# 6 Evaluation

Several evaluations were carried out to assess the effectiveness of the program. These focused on assessing how well the program meets the goals described in the introduction 1.2. They also highlighted some potential areas for improvement. Some of these were implemented and can be seen in the program as described in the implementation section 5. Others are listed in the future work section 7.2 either due to time constraints, or because they were seen to be outside the scope of the project.

## 6.1 Use within a lecture

John O'Donnell, a University of Glasgow lecturer agreed to use the program in a lecture to test out its capabilities as a demonstration tool. It was used in 2 different lectures on circuits for first-year university students.

**First lecture** The first was a lecture explaining what gate delays are and how they affect circuit design, as well as looking at some standard circuits. The lecturer started by loading a circuit that simply has a row of NOT gates. By repeatedly pressing the gate delay button, he could show that it takes time for a signal to move through each gate. This can be seen in table 5.2. Next, a 4-bit AND was loaded. This was to show the students that gates can work in parallel, so a circuit with 3 gates will not necessarily have 3 gate delays. Then he loaded a 4-bit ripple carry adder. This demonstrated how this circuit works and the time taken for the carry bit to move between components. Finally, he loaded a register circuit to show how this works, as well as demonstrating the functionality and usefulness of a delay flip flop. These circuits all worked as expected during the lecture.

**Second lecture** The second lecture used screenshots of circuits made using the simulation program. These were incorporated into the lecture slides. The diagrams could be seen clearly on the projector and were readable. A range of different circuits were shown including an adder and a multiplexer.

### 6.1.1 Results

**Benefits** The lecturer described the program as a helpful tool to be used in lectures. He highlighted a feature that he believed to be unique to this program, which was a focus on showing when signals became valid and the ability to simulate circuits both at a gate delay level, and showing a full clock cycle after a single button press. He also suggested that level of abstraction created by having black box components and word wires was useful for building and demonstrating larger circuits. As the lecturer felt that the program was useful to show a range of different circuits, and it was able to simulate these correctly, it can be seen to be useful for a lecturer demonstrating in a beginners lecture on computer hardware. This meets one of the goals described in the introduction 1.2.

**Suggested improvements** The lecturer also suggested some improvements that he felt would make the program a more effective demonstration tool. These improvements are listed below:

- Zooming takes steps that are too large making it difficult to fit the circuit exactly onto a screen.
- The buttons positioned at the bottom of the screen take up too much space, again making it difficult to fit a circuit on the screen. It was suggested that these should be moved into the menu bar at the top.
- The blue colour used to show the value did not seem substantially different from the grey used to show invalid. This was particularly apparent when projected onto a screen.
- Operations involving flip flops and registers were sometimes not clear. The lecturer suggested that the values inside these could be shown on the components.
- When simulating a clock cycle, the program first did a clock tick and then some gate delays. It was suggested that this should be reversed so that a value from the input can move into a register on a single clock cycle.

Each of these changes was made, in an aim to improve the usability and effectiveness of the software for future users.

## 6.2 Monitored user test

The monitored user test had the aim of allowing users to give feedback on how easy the interface was to use. It was useful for this test to be monitored by the program developer to note any issues that users had with the software. It also provided users with the opportunity to ask questions if they were stuck on any of the tasks. This made it easier to get feedback on all parts of the program. Users for this study were level 4 computer science students, this meant that most users had a good basic understanding of circuits, and would be able to test the features required to build more complex circuits. The task sheet used was kept fairly short and did not give huge levels of detail. For example, users were asked to create wires between components but they were not given an explanation of how to do this. This was because the test was designed to show that the program could be used without a very detailed explanation. It was also to reduce the time users had to spend reading the sheet and allow them to focus on using the program. The explanation of tasks can be seen in the appendix A.2.

### 6.2.1 Task explanation

Users were given a total of 4 tasks.

**Task 1** Task 1 looked at building a very simple circuit. This circuit had just 3 components, an input, a not gate and an output. This allowed users to familiarise themselves with the basics of how to build circuits on the program.

**Task 2** Users were asked to load a multiplexer circuit and attempt to understand it through simulation. This allowed users to visualise the effect of gate delays, so they could give feedback on this part.

**Task 3** Users were asked to build a circuit given an example drawing. This is realistic of use by a more advanced user who is likely to understand exactly what they want to build, but may not be familiar with the program.

**Task 4** Task 4 looked at simulating a 4-bit adder. This tested the users' ability to understand more complex features of the program such as word wires.

### 6.2.2 Monitoring

Users were observed while they performed the tasks. Notes were taken on any issues that users had or any difficulties that they discussed with the observer. It was also noted when a user attempted to do an action unsuccessfully. For example, attempting to drag a component from the toolbar when this is not possible. Users could also ask the observer for help if they were stuck on a task. The number of times the user had to ask for help to successfully complete the tasks was noted for each user.

### 6.2.3 Evaluation questions

For each task users were asked if they found the interface, easy, OK or difficult to use. They were then asked if they could think of any changes that could be made to the program to make completion of the task easier. For all tasks that involved simulating a specific circuit, or using a specific feature of circuits such as a gate delay, users were asked if they had a better understanding of the circuit or feature after using the program. Finally, for the program as a whole, users were asked to suggest new features and ways of improving the usability and aesthetics of the program. The full survey is included in the appendix section A.4.

### 6.2.4 Results

Copies of all of the users' answers are listed in the appendix section A.6. A list of the user issues recorded while observing the user test is shown in section A.7.

**Suggested improvements to the program:**

- Showing error messages when wires cannot be created between two ports.
- Preventing components from being placed on top of each other.
- More information about each component when hovering over it, as they felt that there was not enough detail in the descriptions.
- Ability to delete components and wires by right-clicking on them.
- Adding an interactive tutorial on program start-up.

**Possible improvements recognised when observing the user tests:**

- Several users attempted to drag items from the toolbar onscreen, and pointed out that this is more intuitive. This suggests that the interface should be changed to allow dragging components from the toolbar.
- Users often accidentally moved components when they were trying to build wires. This could be solved by only allowing dragging of components when selecting the main body of the component. Allowing wires to be created by dragging the cursor from the port as well as by clicking and then moving the cursor could also improve this.
- Users were often confused that the black box components take more than one gate delay, information on this could be added to a tutorial in future.
- Several users placed components so their ports were on-top of each other and expected them to automatically connect. This should be added as a feature.
- Users who got error messages when attempting something incorrect all seemed to find them useful and informative, suggesting that more error messages could be of benefit.
- Several users looked for a button allowing adding wires. Information on how to add wires could be added to a tutorial or help menu.

- Users attempted to right click on a component to rotate it. This feature could be added. Users also had difficulty with the current method of rotation where users had to both be clicking on a component and press R. This could be simplified by highlighting components when they have been clicked on and allowing keyboard short cuts such as rotating and deleting components.
- Users attempted to use the delete key while clicking on a component to delete it, this could be added as a feature.
- A user did not recognise a difference between input and output components, a change could be made to this, for example, having square inputs and round outputs to prevent this confusion.
- A user was confused that they must press a button to start the simulation. This could be included in a tutorial in future.
- A user saw only a blank screen after loading a circuit because they had scrolled to the middle of the build area. The program should automatically move to show the new circuit on load to prevent confusion.

All of these changes make sense to be added to the program, and can be seen as future work.

**Ease of use** When learning to use the interface for building circuits in the first task only one of the five users suggested that the interface was easy to use and the others said that it was OK to use. For the third task building circuits two of the five users said that it was easy to use and the rest said it was OK to use. This shows that users generally had difficulties when using the build tool. This was likely because they expected the interface to work differently, for example dragging components from the toolbar and dragging from ports to create wires. This shows that some improvements are necessary to make the circuit builder aspect of the program easier to use. These are listed above and in the section on future work 7.2.

In the second task when simulating a slightly more complex circuit, four of the five users suggested that the program was easy to use. For the most complex circuit, including word wires and word input and outputs all of the four users that attempted this task suggested that the interface was easy to use. This suggests that the simulation tool is easy to use for simple circuits and for slightly more complex circuits.

Four of the users five managed to complete the four main tasks requiring help no more than two times. The other user completed three tasks and required help seven times. This suggests that the program could be used in a lab environment where users can ask tutors for help if they are stuck on a given task. It also meets two of the goals listed in the goals section 1.2 as users were able to load, build and simulate a range of circuits.

**Usefulness as a learning tool** All of the five users said that they had an improved understanding of multiplexers after the second task. Three of five users said that they had an improved knowledge of registers after the third task and three of four users said that their knowledge of adders was improved in the final task. This suggests that the program can improve users' knowledge about a range of different digital circuits, as for each task more than half of those who attempted it stated that their knowledge had improved as a result of using the program. This shows that the program does meet the main goal of the project, section 1.2, allowing students to improve their knowledge of circuits.

### 6.2.5 Limitations

The users for this study are not really the target users for the product. Some of these users had more experience of circuits, as they had studied honours computer science courses, or electronic engineering courses, looking at digital circuits in greater detail than in the first year computer science course. Other users had not studied any relevant topics since first year and may have forgotten some of this information. This was considered to be ok as the focus of the test was to find out how easy the program is to use. However, this could have affected how users felt the program impacted on their knowledge of circuits. Users could have said that there was no change in their knowledge if they already had a very good understanding of the circuit or could have said that they have an improved knowledge as they remembered nothing about the circuit and learned only a small amount. This could mean that the results are not a true representation of expected users.

Users may have been able to complete a task, but chose to ask for help to finish more quickly, or because they did not want to be observed repeatedly failing the task. This could have affected the number of times that users asked for help. It was also decided that if the observer simply suggested that they re-read the task sheet to find a solution to the problem then this would not be marked as a requirement for help. This was decided because it is likely that if the user was not being observed, they would have read over the task again in an attempt to complete it. These factors could have affected the accuracy of the count of how many times users asked for help.

## 6.3 Unmonitored user test

A user test was created to see how easily users were able to follow a step by step guide on using the software to improve their knowledge of circuits without any input from someone with more knowledge of the circuits given or of the program its self. This is representative of a realistic user of the software, for example in a university course on circuits. The task sheet gave a very detailed explanation of how to use the software with screenshots at each step of each task. This was decided because users would not be able to ask for help if they had any confusion with the tasks so it was preferred that users were given a lot of detail to allow them to complete the task.

### 6.3.1 Task explanation

Users were given a total of 5 tasks. Task 0 was simply starting the software. The complete task sheet can be seen in the appendix section A.1.

**Task 1** Task 1 was to look at a simple circuit that could demonstrate gate delays within a circuit. The example used was a row of not gates, and users could see that the signal moved to the next gate on each gate delay to improve their knowledge of validity within a circuit.

**Task 2** This task looked at the ripple carry adder circuit, and encouraged users to add two numbers together. Users could then see how the signal moved between the full adders making up the circuit. The aim of this task was to find out if the software could help users to understand a more complex circuit that they may not have seen before. It also helped them understand that a complex component like a full adder is made up of many gates and could take more than one clock cycle to produce output.

**Task 3** Users were asked to load the register transfer machine circuit. They were asked to load two variables into the register file. They were then asked to load the result of this back into one of the registers. This was aimed at more advanced users who were confident with the basics of circuits and could provide feedback on how well the program was able to simulate complex circuits.

**Task 4** Task 4 looked at building a very simple circuit. This circuit had just 3 components, an input, a not gate and an output. This was described as an extra task as it was not really the focus of this evaluation. It was decided that building circuits should not be the focus for complete beginners. Users familiar with the idea of logic gates would be better suited for an evaluation of user interface for the circuit builder part of the application.

### 6.3.2 Evaluation questions

Evaluation questions were the same as described above for the monitored user test in section 6.2.3, edited to consider that different tasks were used. The complete list can be seen in appendix section A.3.

### 6.3.3 Results

A complete list of answers to the evaluation questions can be found in A.5.

**Suggested improvements to the program:**

- Option to view circuits as a running simulation of gate delays or clock ticks.
- Highlighting to show when a value changed on a wire, as this was not always noticed by the user. Similar highlighting was suggested for when a gate is added onscreen.
- Ability to rotate components, this was actually part of the program, although not well documented.
- Adding images of the gates to the buttons on the toolbar.
- Add a help sheet.
- Better error messaging.
- Improved wire builder.

Most of these changes make sense to be added to the program. Highlighting when gates are added might be useful, however, a better solution could be to drag items from the toolbar so it is obvious to users when an item is created.

**Issues users had with the program** A user suggested that when they tried to load two of the circuits they did not load properly. This may have been because they had scrolled to a different area of the screen without noticing. This could be changed by moving the screen back to the top left when a circuit is loaded.

**Ease of use** For the first task, loading a simple circuit and simulating it, 3 of the 4 users found the interface easy to use and one user highlighted the fact that they liked the ability to perform individual gate delays. This suggests that the software is usable when loading simple circuits, even without any help from someone more experienced with the software.

However, none of the students found the software easy to use for the second task and one user said that they found it difficult to use as they were unable to load the circuit. This could mean that the use of black box components and word wire confuse beginners and make the program less accessible to this group. One solution to this could be building up the user's understanding more slowing by asking them to simulate many circuits with slowly progressing complexity. This is more realistic of use of the software in a tutorial, where users will have more time than in a user test.

Just two of the students evaluated the RTM circuit. Both of the students described the interface as difficult to use. Again one of the students failed to load the circuit. This could mean that the program does not allow beginners to understand and simulate such a complex circuit.

Only two users attempted the final task building a simple circuit. One user found the program difficult to use and one user said that they found it ok to use. This suggests that changes are needed to the interface to allow new users to easily begin to build circuits.

**Usefulness as a learning tool** For the first task, two out of four users who answered said that their knowledge of gate delays had improved as a result of using the program. This suggests that the program is useful as a tool to teach users about this aspect of circuits. The other users said that their knowledge on the topic did not change. This could mean that the program was not helpful, or simply that they already had a good understanding of the topic.

Two of the three users that responded found that simulating the 4 bit adder improved their knowledge of the circuit. One user suggested that they saw no change. This suggests that the program is also able to help users understand more complex circuits such as a ripple carry adder.

None of the users found that the program improved their knowledge of the RTM circuit. This is likely because they struggled to load or simulate it as described above.

### 6.3.4 Limitations

One major limitation of this study is that it does not consider cases where users stopped doing the study before starting the evaluation stage. Users may have been unable to even start the software or may have given up before completing any tasks. This could mean that the study provides an unrealistically good view of the software. However, it removes the observer effect. This is where study participants may work unusually hard given that they are observed. Foundation (2018) Foundation (2017)

Another limitation is that few users decided to take part, and those who did, did not complete many of the tasks, or give detailed feedback. Also if users had any problems they would not have been able to continue the experiment. A realistic use of the program might be in a lab where tutors can offer help to students having difficulties. However, this was not allowed for this experiment.

It could be argued that the RTM circuit was not a good choice for the evaluation and that it would be more suitable for users with more experience of the program and of circuits. This circuit includes 5 different 1 or 2 bit control signals that must be set on each clock tick and 4 different 4-bit registers for values to be loaded into. Users would require a very good knowledge of topics covered in the first year computer systems course to understand the circuit.

## 6.4 Verification of program efficiency

One important aim of the project was to simulate larger circuits up to the size of the RTM circuit. For this to be useful to users of the program, the circuit should run at an acceptable speed.

### 6.4.1 Method

**Load timing** Testing was carried out using the RTM circuit to ensure that it worked at an acceptable speed. The time taken to load the circuit was tested, and compared to the time to load a very simple circuit. The load times can be seen in table 6.1.

**Clock cycle timing** The time taken to simulate the circuit was compared to the simulation of a much simpler circuit. To achieve this, the program was set to perform path depth gate delays per clock tick. For the RTM circuit, this meant that pressing the clock tick button would cause 17 gate delays and then a clock tick. This would mean that the 170 components in the circuit processed a gate delay 17 times. For the simple notnotnotnot circuit, the 4 components would each be processed 4 times. Each test was performed 3 times and an average time was calculated. The results for this are shown in table 6.2.

| | AND | RTM |
|---|---|---|
| Test 1 | 120 | 5316 |
| Test 2 | 23 | 3822 |
| Test 3 | 37 | 6032 |
| Average | 60 | 5057 |

***Table 6.1:*** *Measurements of the time taken to load two different files on the Logic Simulator program. Timings are in milliseconds. The java function System.currentTimeMillis() was called before and after the code performing the load and the difference between these was taken as the load time.*

| | notnotnotnot | RTM |
|---|---|---|
| Test 1 | 4 | 153 |
| Test 2 | 0 | 207 |
| Test 3 | 3 | 153 |
| Average | 2 | 171 |

***Table 6.2:*** *Measurements of the time taken to perform a full clock cycle on two different circuits. Timings are in milliseconds. The java function System.currentTimeMillis() was called before and after the code performing the load and the difference between these was taken as the load time.*

### 6.4.2 Results

**Load timing** It is clear from table 6.1 that there is a noticeable delay when loading a large circuit like the RTM with an average loading time of 5 seconds compared to loading a simple 1 gate circuit with an average of 60 milliseconds. This was not decided to be a huge issue as loading a large circuit is not a very common action so uses should not mind a small delay. The future work section 7.2 gives a reason for this and suggests changes that could be made to improve it.

**Clock cycle timing** It is important for users that a clock cycle does not have a noticeable delay. Users could press the clock tick button twice, if the delay was too long. From table 6.2, it is clear that larger circuits have a longer delay before this action can complete. However, even the longest timing of 0.2 second should not be noticeable or annoying to users. Again, the future work section 7.2 describes why this might be and ways of improving this. As suggested in the design section, it was decided that a more efficient algorithm would not be required for processing gate delays.

### 6.4.3 Limitations

Timings do not consider other things that might have been taking up CPU time while the test was running. However, all the values seem to be similar over the 3 tests, suggesting that the timings are accurate enough. Another issue is that performance could be different for a different machine or method of deployment. However, the program has been run, without formal testing, on several different University of Glasgow computers on both Windows and Linux and no timing concerns were noticed. The machine used for testing is of a similar specification to what would be expected for a computer science lab.

## 6.5 Overall

### 6.5.1 Verification for functional requirements

Most of the functional requirements described in section 3.1 were evidenced by being performed in the users tests described above. However, some of the more complex features were not tested.

Users did not save any circuits or save circuits as components to be used in a larger circuit. The first of these should be easy, and like the loading of circuits uses a standard file chooser window. Saving circuits as components that could be added to other circuits as described in section 5.3.5, was decided to be too complex for user testing, and would likely only be used by those who spent more than the 15 minutes required for the user tests using the program. The fact that there have been many component save files created on this interface by the program developer makes it clear that this feature works correctly.

Simulating a full clock cycle in a single click was also not performed for the user test as the two users who attempted the RTM task, which involved this both suggested that they were not able to complete it correctly. Screenshots from the explanation of how to do this task.A.1 give evidence that this feature does work correctly, and that the program allows values to be loaded into the RTM with a single button press.

It is clear that all of the functional requirements listed in 3.1 are met by the program.

### 6.5.2 Verification for non-functional requirements

This section looks at whether the program meets the non-functional requirements listed in section 3.1.

**Easy of use by beginners** The user studies above suggest that for all of the circuits used, with the exception of the most complex circuit, the RTM, were able to be built and simulated by students testing the program. Of the feedback from all the tasks over the two user tests, excluding the RTM circuit, at least 50% of users found the program to be easy or ok to use. This was 100% for most of the tasks. It was found that the simulation aspect of the program was easiest to use, with most of the tasks that only involved circuit simulation rated easy to use by 75% or more of users.

**Use without large amounts of documentation** The monitored user test suggests that most users were able to complete all of the tasks given with only an minimal task sheet and receiving help no more than two times during the test. This suggests that the program is usable without large amounts of documentation.

**Run on University of Glasgow machines** The user tests for the program were run on University of Glasgow machines, evidencing that the program works correctly on these.

**Efficiency on large circuits** This is clearly evidenced in section 6.4.

### 6.5.3 Meeting program goals

This section considers if the program meets the overall program goals given in the introduction 1.2.

**Simulate synchronous digital logic** It is clear from the range of different example circuits that have been made on this program that any circuit involving logic gates and flip flops, as well as I/O devices can be created using this program. This allows for the simulation of any digital logic circuit assuming some of the limitations in section 3.2, for example, circuits with multiple clocks cannot be displayed.

**Use by students and lecturers** The user study made it clear that students are able to use the program to build and simulate circuits. It can also be seen by the fact that the program was used successfully in lectures that it can also be used for some of the purposes required by lecturers on computer hardware.

**Users can load and build circuits** The user study shows that users are able to load, build and simulate a range of different circuits using this program.

**Improve users' circuit knowledge** As stated in both of the user evaluations with students, the students said that their knowledge of circuits had improved as a result of using the program. For all of the tasks excluding the RTM circuit, as all users who tried this task found it too difficult to simulate, at least 50% of users said their knowledge of the circuit or concept being asked about had improved.

**Different simulation modes** Users were able to simulate gate delays on the circuit, and screenshots of the program performing a full clock cycle in a single step can be seen in the appendix.

## 6.6 Unit testing

Unit testing for this project focuses on testing the functionality of under-lying parts of the program. It was important to know that these were working correctly to build on top of them. For example, testing if the onscreen gate delay button is working correctly becomes much easier if it can be assumed that each gate works correctly.

Features that have been unit tested:

- Function on gate delay/ clock tick of each gate.
- Wires passing signals between components.
- Loading a save file.
- Building the JSON for saving a file.
- Transferring the internal representation of a word into hexadecimal digits and vice versa.
- Building a component, given parameters to create it, i.e. what would happen when loading components from a file or clicking the button to create them.
- Performing gate delays and clock ticks.
- Counting path depth for a circuit.
- Loading all of the descriptions and swapping one out for another.

## 6.7 Automated user interface tests

Automated user interface tests were useful to ensure that changes to parts of the UI did not break existing functionality. For example, wires could still be added correctly after rotation of components was made possible. The focus for adding UI tests was on code in the controller classes for objects. It was also focused on the most commonly used or most likely to be extended classes.

Almost all of these tests could more accurately be described as black box integration tests, as they test a wide range of program features and do not rely on specific calls to code, just pressing onscreen buttons. This was made possible using the Testfx UI testing framework for Javafx. Testfx (2019) This has simple commands to perform user operations such as clicking on specific onscreen objects and key presses. The framework proved useful for creating a wide range of automated user tests.

The following automated UI tests have been added:

- Building simple, and then more complex circuits which included feedback across flip flops and rotation.
- Deleting components and wires.

- Loading circuits and testing the effect of gate delays and clock ticks on them. Three circuits were tested, an OR gate, a delay flip flop and a 4-bit adder. These were designed to test both the simplest circuits and complex ones involving word wires and black box components.

# 7 Conclusion

## 7.1 Summary

This project looked at building a software tool to teach computer science students about important concepts of computer hardware. It was decided that only synchronous digital logic circuits would be simulated as these make up most of modern computers and are therefore most relevant to computer science students. Some limitations were also decided on for this reason, for example, not allowing circuits that have multiple clock components. The program was designed as a simulation tool for digital circuits, considering that both lecturers and students with little knowledge of circuits would be expected to be able to use the program. There was a focus on creating readable circuits that an inexperienced user could easily understand and simulate. A circuit building tool as well as a range of example circuits were added to allow users to have a wide range of circuits to simulate.

It was decided that the concepts of a signal validity and clock cycles were important to be understood by users of the program. As a result of this, two simulation modes are offered to users. They can simulate the circuit gate delay by gate delay in order to understand that time is required to allow a circuit signal to become valid. This also allows them to consider efficiency in terms of the path depth of the circuit and load values into flip flops by performing a single clock tick. Users can also simulate a full clock cycle where enough gate delays will happen so that all signals are valid and then a clock tick will happen. This allows the simulation of more complex circuits involving registers.

The evaluation concluded that all of the functional requirements listed in section 3.1 were met as users were able to test most of these features, and others were clearly evidenced by the program developer. The non functional requirements were also shown to be met at an acceptable level. The key goals for the program were also evidenced by the user evaluation and by further testing of the program. The overall goal of this project was to create a program that was able to improve the users' knowledge of circuits and evidence from the user evaluation shows that it is able to do that. The main drawback of the program that was found during testing was that the circuit builder aspect of the program was difficult to use, particularly for those using it for the first time and this is the main area suggested to be worked on for a future version of this software.

## 7.2 Future work

Given more time on the project, the following changes have been suggested to improve the program.

### 7.2.1 Circuit Builder Improvements

The main goal of this project was to create a program that could simulate some simple and more complex circuits in a way that would make the program useful as a learning tool for computer science students. Another important goal was to ensure that circuits could be built that would be easy to see and understand. This was more important than creating an easy to use build tool.

The results of the user studies highlight many changes that could make the circuit builder easier to use. On top of these changes, some important features to add to the build tool are listed below.

- Undo and Redo
- Easy way to change the positioning of wires after they have been created
- Allow components to be moved with wires attached to them
- Make it easier to connect a wire to any point on another wire, currently this can only be done at corners or points defined specifically by the user
- It should not be possible to create wires laying on-top of each other. This makes it difficult for users to see where wires are connected.
- Duplicate components with copy/ paste keyboard shortcuts.

### 7.2.2 Variable word size

In the current version of this program, creating components that have a word size other than 4 can only be done by creating or editing a save file. There is support internally for larger components. For example, the split and join components can resize when their word size is larger than 4. A feature allowing users to choose the word size when creating components could be added as well as allowing users to specify sizes for other components e.g. a 3 input AND gate.

### 7.2.3 User defined descriptions

Currently users can add new components, but unlike the built in components these would not have a description attached to them. It was decided that the existing description files should not be editable by users as they are FXML files that could cause errors if they were edited incorrectly. A simpler interface allowing users to add text and tables to descriptions of their components could be added. This would be particularly useful for more advanced users, such as a lecturer building files for students to use.

### 7.2.4 Performance increase

Initially loading a large circuit that has many internal black box components has a noticeable delay. 6.1 This is because the interface to view these components internals is created when the circuit is loaded. This could be improved by loading this only when a user looks inside the circuit. These could then be stored so they do not have to be reloaded every time the internal circuit is shown onscreen.

Performing an entire clock cycle is inefficient. Each component performs a gate delay for every gate delay in the full clock cycle. This gives a $O(pathdepth * numberofgates)$ complexity. Each gate only acts on one of the gate delays in the clock cycle. This could be pre-computed when the circuit is loaded and gates could be processed in a given order. This would give a $O(numberofgates)$ and allow the program to scale to larger circuits. As there was not a long delay in the largest circuit chosen for this project 6.2, the existing algorithm was decided to be suitable.

### 7.2.5 Circuit animation

Being able to view changes in validity as an animation could make it easier for users to see a gate delay as a unit of time and better understand why the path depth of a circuit should be kept short. Allowing clock ticks to be shown as an animation would require users to be able to list the values of each input at each clock tick and could help simulate more complex circuits as well as allowing a repeatable animation of a complex action, for example loading 2 numbers, adding them together and loading the result back into memory.

### 7.2.6 Alternative methods of deployment

An easy to use program runner was created for windows that has a JDK built in and is runnable by clicking on a .exe file. Similar runners could be created for Mac OS and Linux using the same tool to improve portability and ease of use.

## 7.3 Reflection

Overall, this project was a useful learning opportunity, particularly in terms of good practice for coding a user interface. Separations of concerns between the model, view and controller were found to be particularly important to allow for code that was easy to edit and maintain. The use of automated user interface testing was also found be very helpful and time saving for the project. The difficulties of designing an interface that users would immediately understand how to use was made clear during development and user testing of the program.

On reflection, a more feature rich framework for developing graphics such as Unity Unity (2019) may have been a better choice for development. Over the course of this project, Javafx was found to lack support for many common components of a graphics application. One example of this is a component that allowed both scrolling and zooming. This had to be custom built for the program. This took time away from solving problems specific to the project and hindered development.

# A | Appendices

## A.1 User task sheet for unmonitored evaluation.

This can be seen in figure A.1.

## Introduction and Ethics

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less than 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

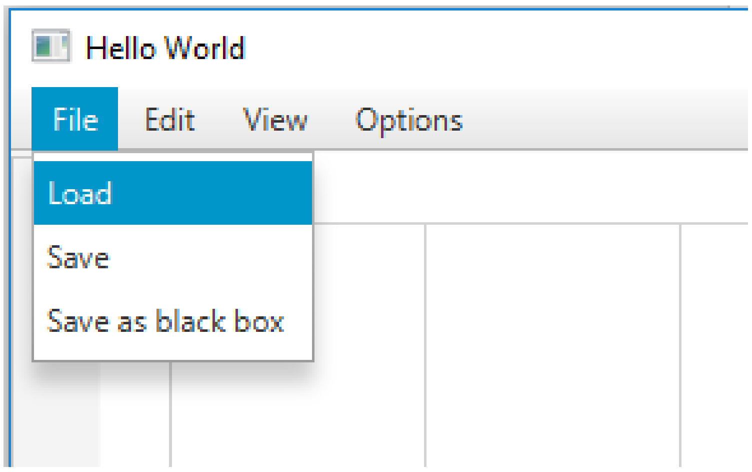
If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O’Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Task 0, Install and run

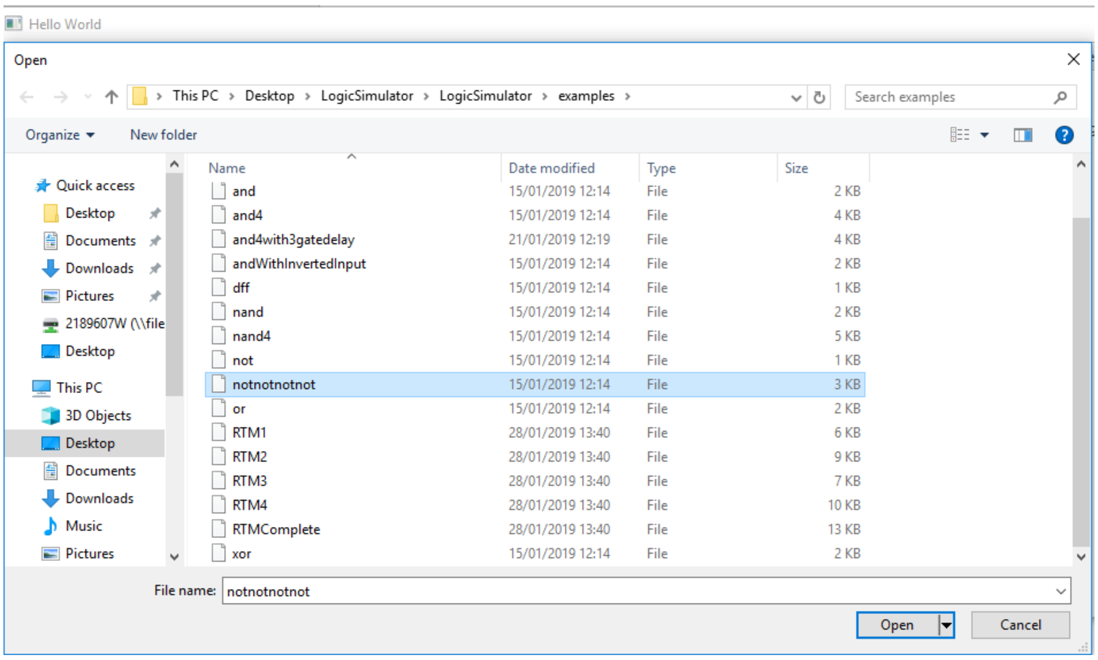
Download and unzip the LogicSimulator folder.
Right click inside the new folder and select Git Bash Here.
In the window that opens run the command:
java -jar all-in-one-jar-0.6.jar

## Task 1, Simulate notnotnotnot circuit

Click on the load button in the file menu.



Select the notnotnotnot circuit.

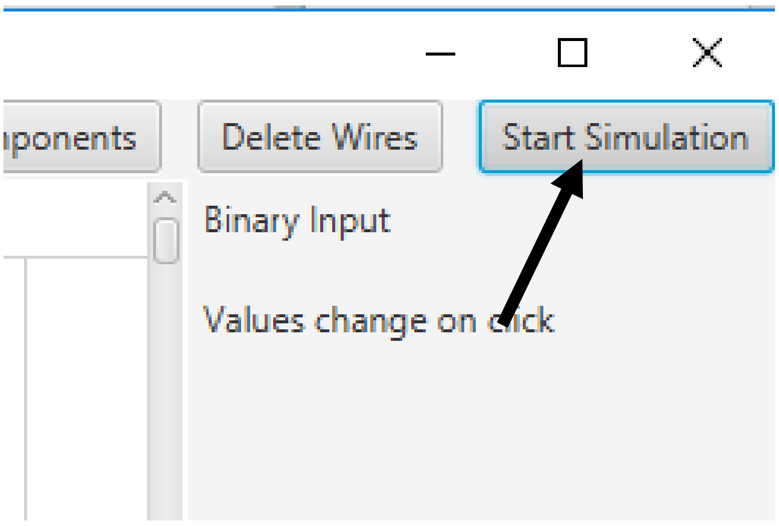


| Name | Date modified | Type | Size |
|---|---|---|---|
| and | 15/01/2019 12:14 | File | 2 KB |
| and4 | 15/01/2019 12:14 | File | 4 KB |
| and4with3gatedelay | 21/01/2019 12:19 | File | 4 KB |
| andWithInvertedInput | 15/01/2019 12:14 | File | 2 KB |
| dff | 15/01/2019 12:14 | File | 1 KB |
| nand | 15/01/2019 12:14 | File | 2 KB |
| nand4 | 15/01/2019 12:14 | File | 5 KB |
| not | 15/01/2019 12:14 | File | 1 KB |
| notnotnotnot | 15/01/2019 12:14 | File | 3 KB |
| or | 15/01/2019 12:14 | File | 2 KB |
| RTM1 | 28/01/2019 13:40 | File | 6 KB |
| RTM2 | 28/01/2019 13:40 | File | 9 KB |
| RTM3 | 28/01/2019 13:40 | File | 7 KB |
| RTM4 | 28/01/2019 13:40 | File | 10 KB |
| RTMComplete | 28/01/2019 13:40 | File | 13 KB |
| xor | 15/01/2019 12:14 | File | 2 KB |

File name: notnotnotnot

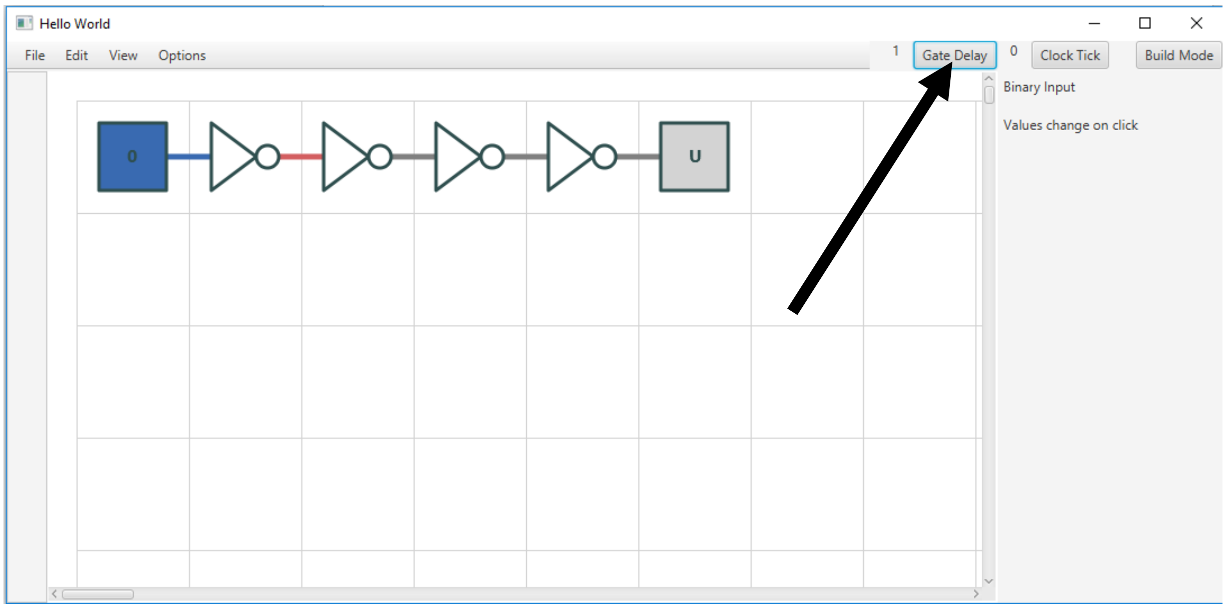
And click open.

Click start simulation.



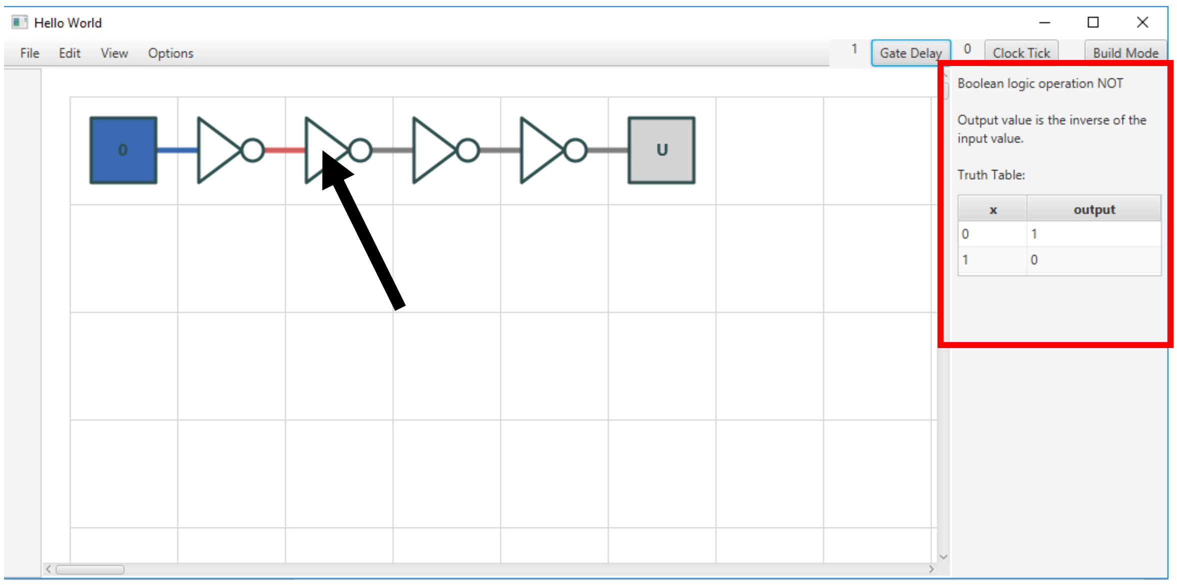
You will notice that some of the components and wires change colour. This is to show their values.
Red – 1
Blue - 0.
Grey - U
U stands for undefined. This means that the signal is invalid at that given point in time. Time must pass before the logic gate will definitely have the correct value on its output.

Now press the gate delay button.



This simulates a gate delay, this is the small amount of time that it takes for a signal to pass through a logic gate.

Now try hovering the mouse over one of the not gates. You should notice that an explanation of how the gate works is shown in the right hand panel.



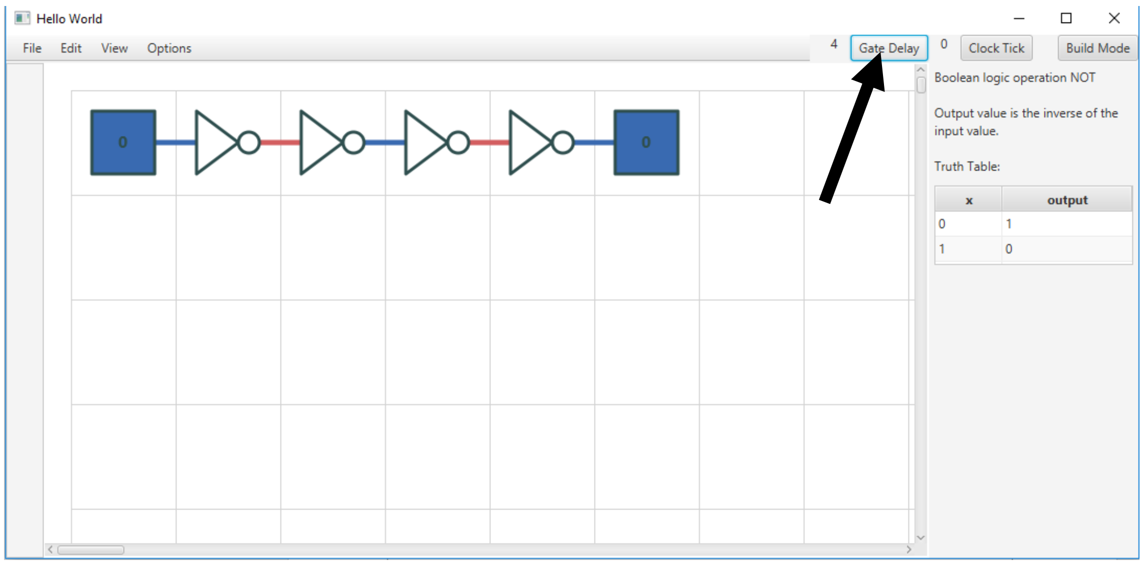
Boolean logic operation NOT

Output value is the inverse of the input value.

Truth Table:

| x | output |
|---|---|
| 0 | 1 |
| 1 | 0 |

Continue pressing the gate delay button until the output box at the right changes value.



Boolean logic operation NOT

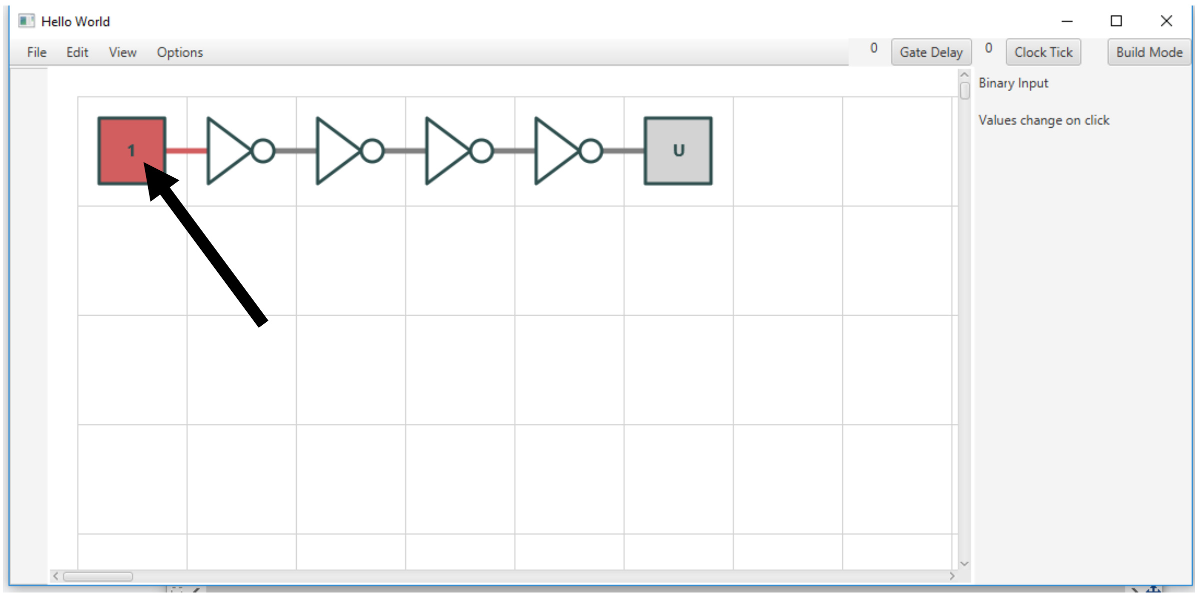
Output value is the inverse of the input value.

Truth Table:

| x | output |
|---|---|
| 0 | 1 |
| 1 | 0 |

This means that enough time has passed so that the signal has moved through every gate.

Press the input, you should notice it change from 0 to 1.

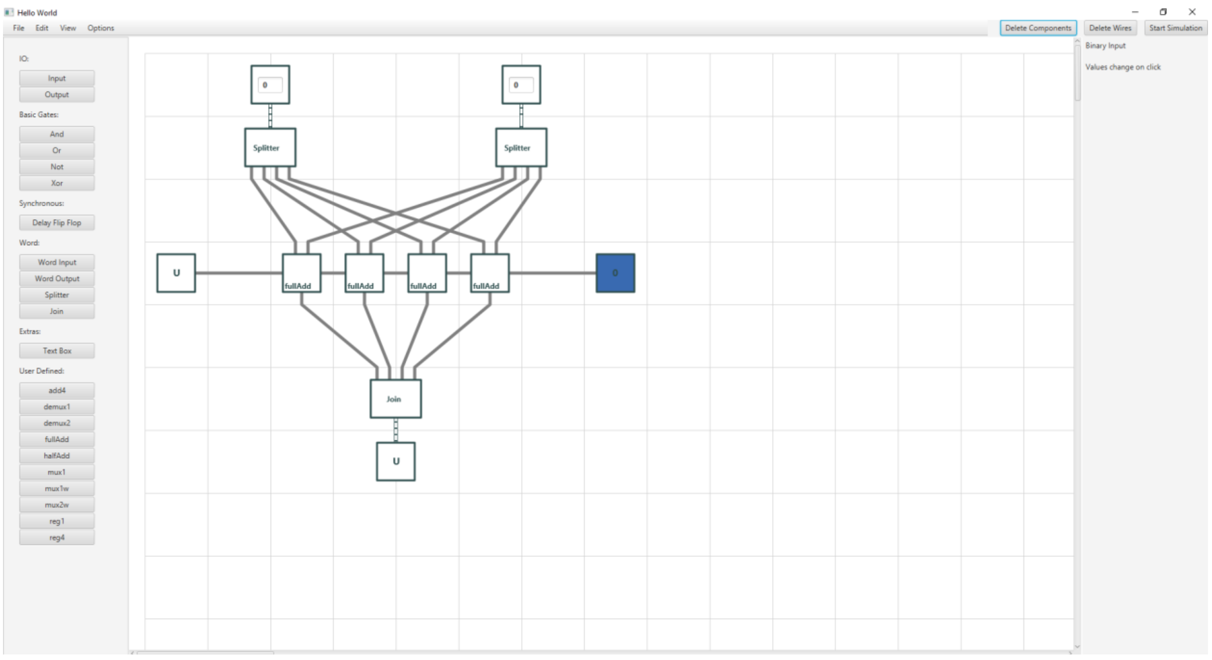


Using your knowledge of logic gates, think about which value will be shown on the output of the first not gate after a clock tick. (0 is shown as blue and 1 is shown as red.)

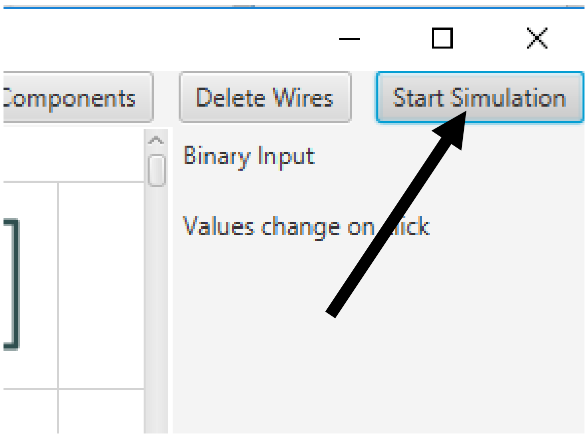
Press the gate delay button until the output shows a value and notice the changes in value.

## Task 2, add 2 numbers

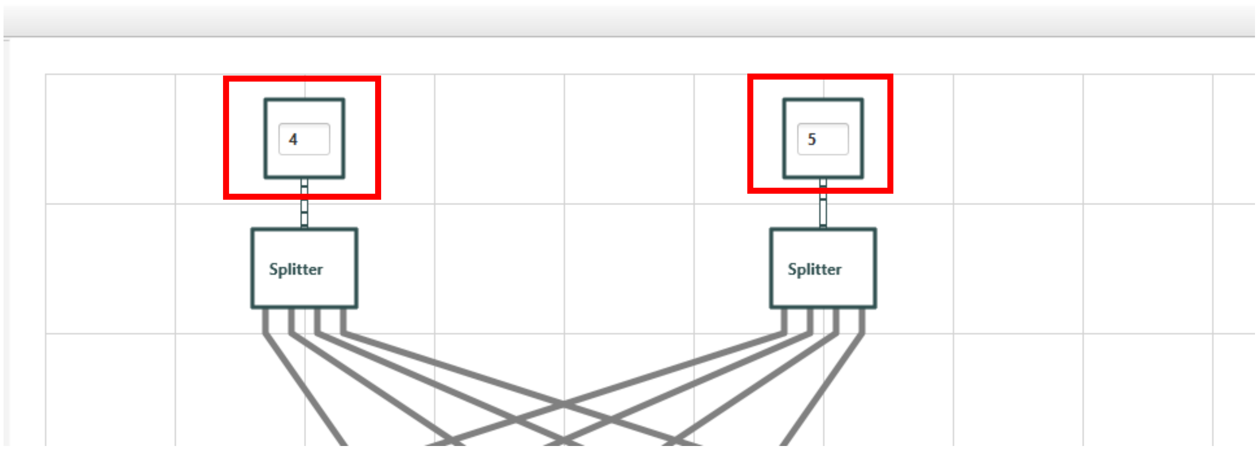
Load the circuit, add4 from the reusables folder, in the same way that you loaded the notnotnotnot circuit.



Click start simulation.

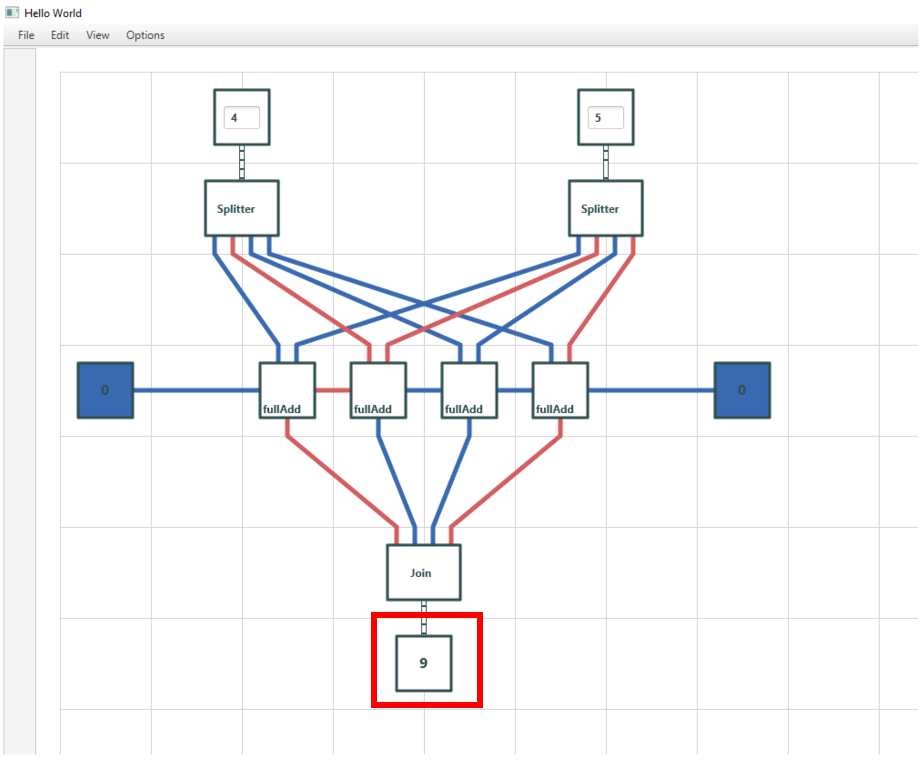


Inputs to the adder can be changed by clicking on them and typing in the new value.



Enter any two hexadecimal digits, say 4 and 5.

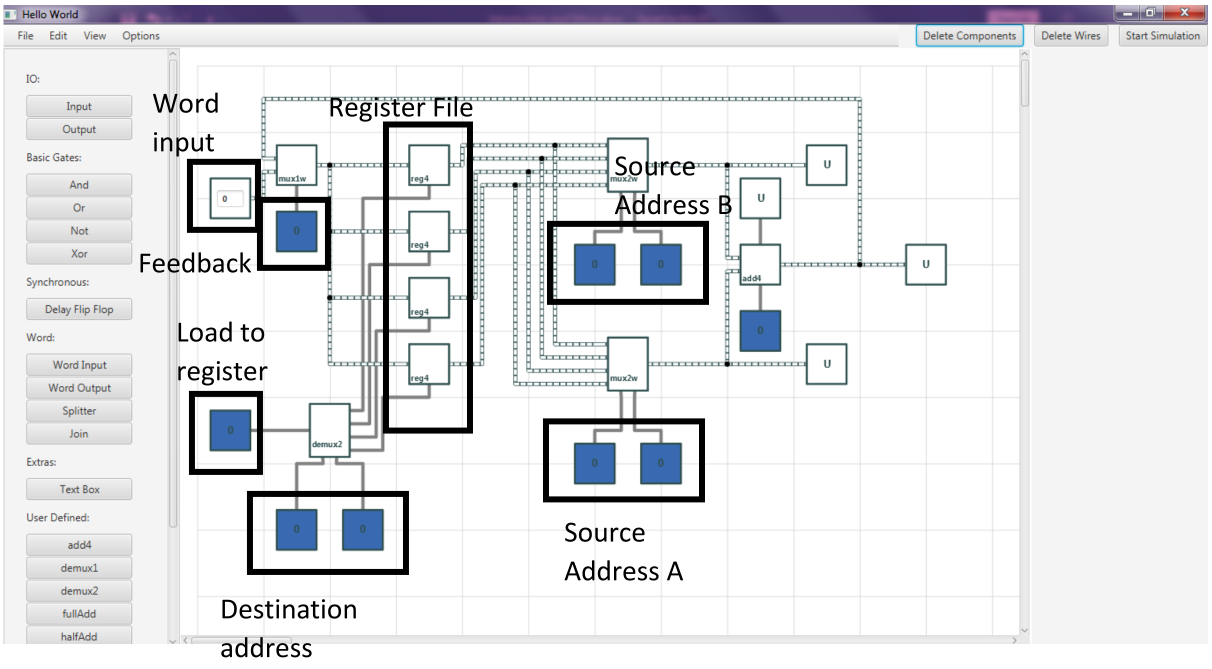
As with the not gate repeatedly press the gate delay button to see the signal move through the circuit and see the value of the addition in the bottom box.



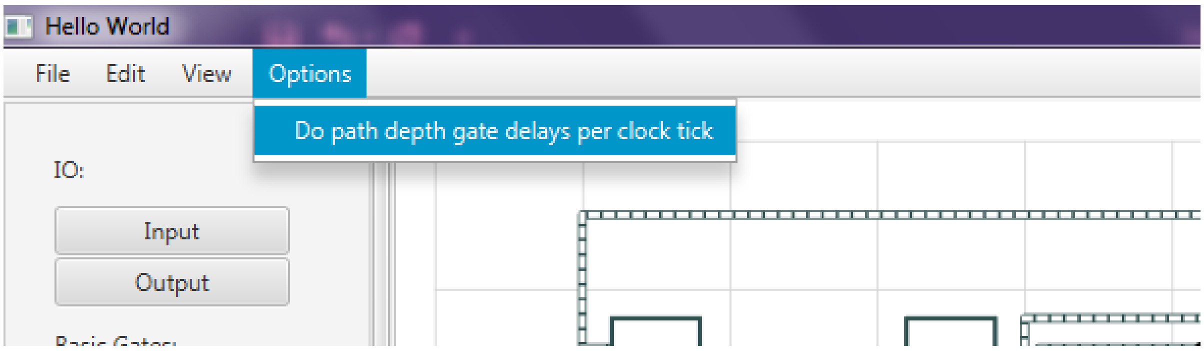
## Task 3, Register Transfer Machine

If you find this too complex, skip to the extra task or to the evaluation.

Load the file RTMComplete. This includes a register file with 4, 4 bit registers, these can each store a single hexadecimal digit.



First press the "Do path depth gate delays per clock tick" option from the options menu. This means that when you press the clock tick button enough gate delays will happen before the clock tick to allow all signals to become valid.



Then click start simulation.

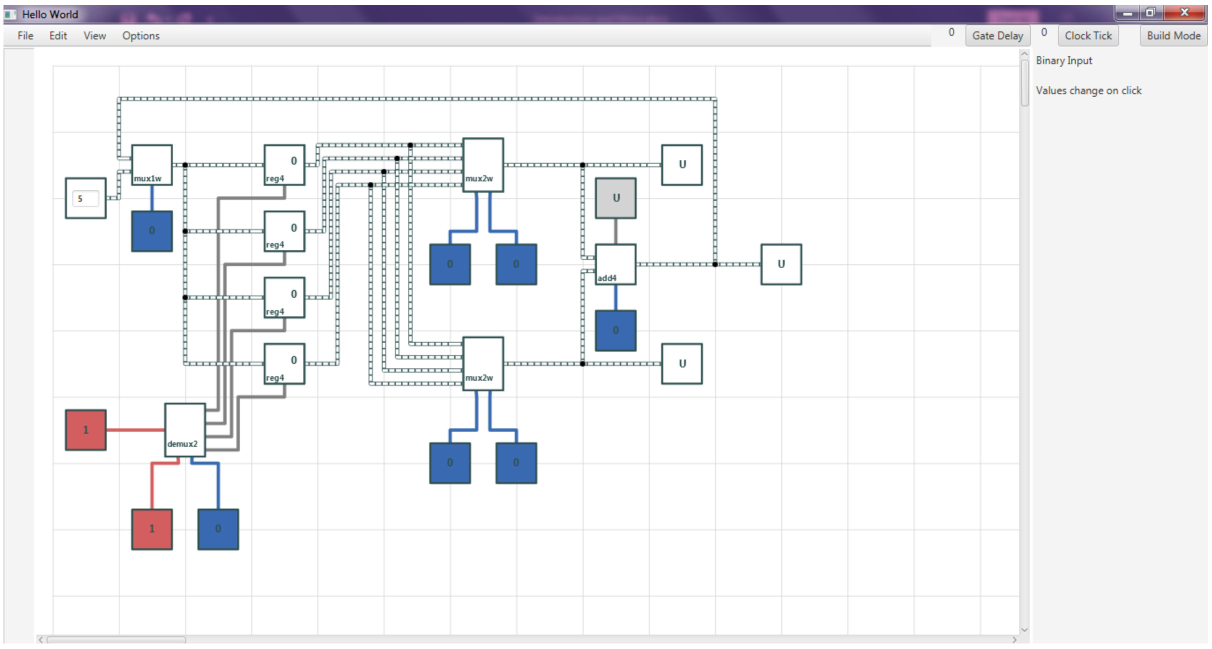
The steps below explain how to execute the following instructions on the RTM.

R2 := 5,

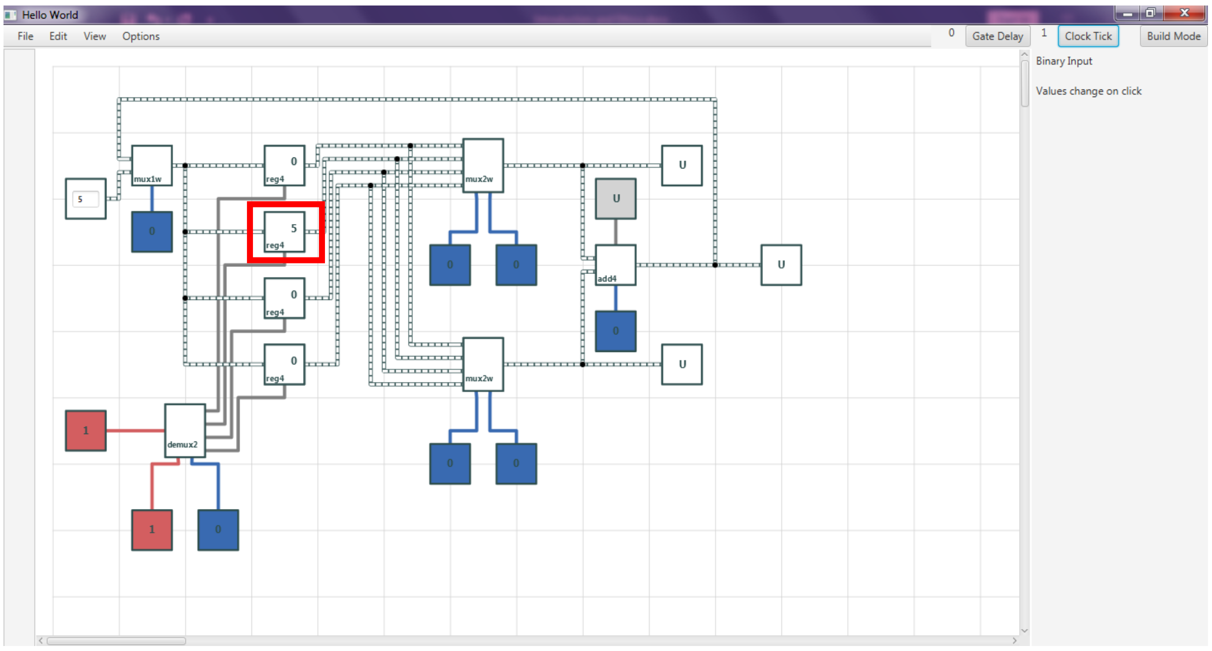
R0 := 3,

R3 := R2 + R0.

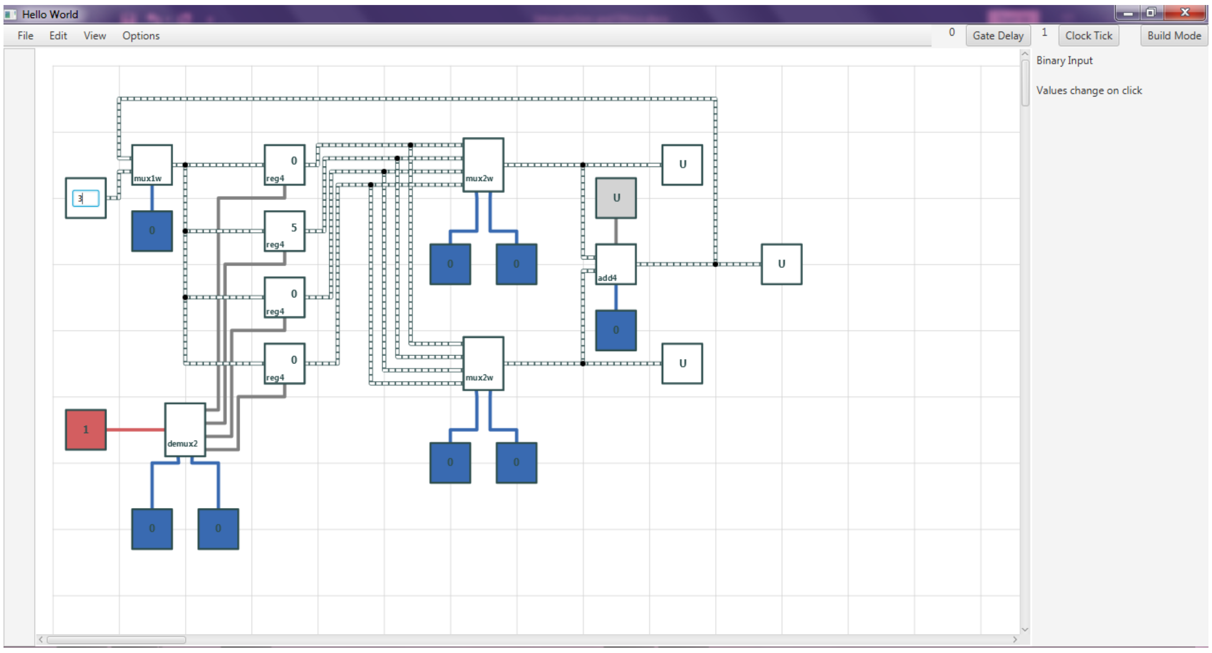
Start by loading 5 into R2. The value 5 should go in the word input box. The load to register control (shown in the image above) should be set to 1. The destination address should be set to 2. The image below shows this set up.



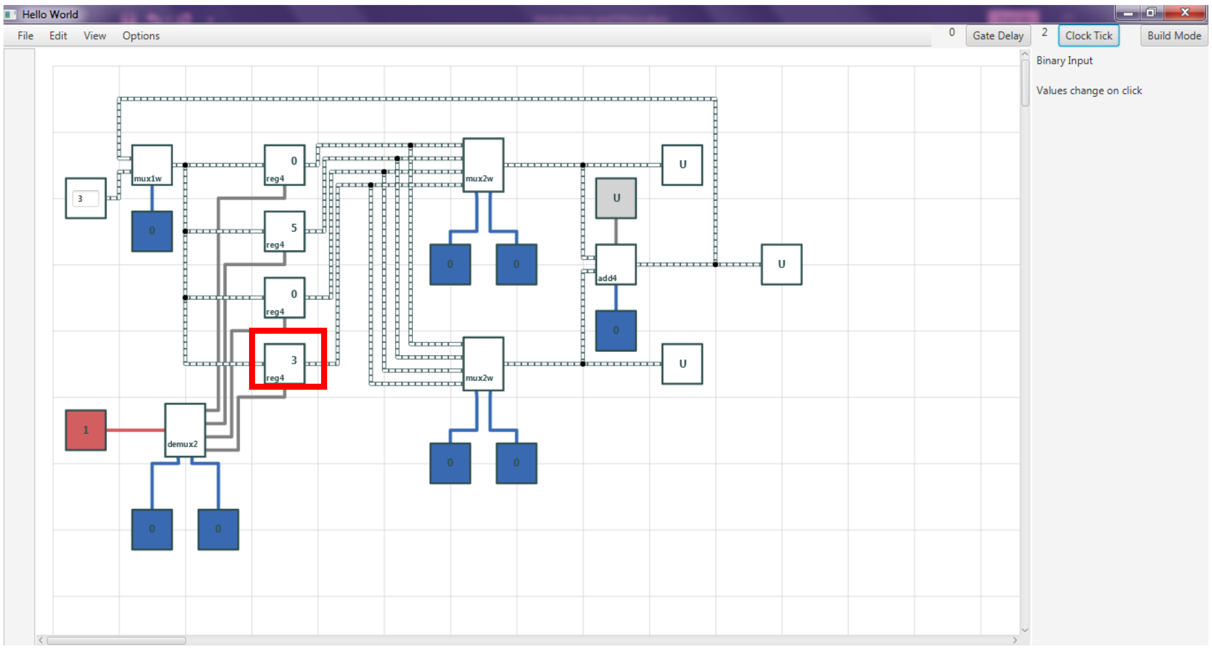
Press the clock tick button and you should see the value 5 going into register 2.



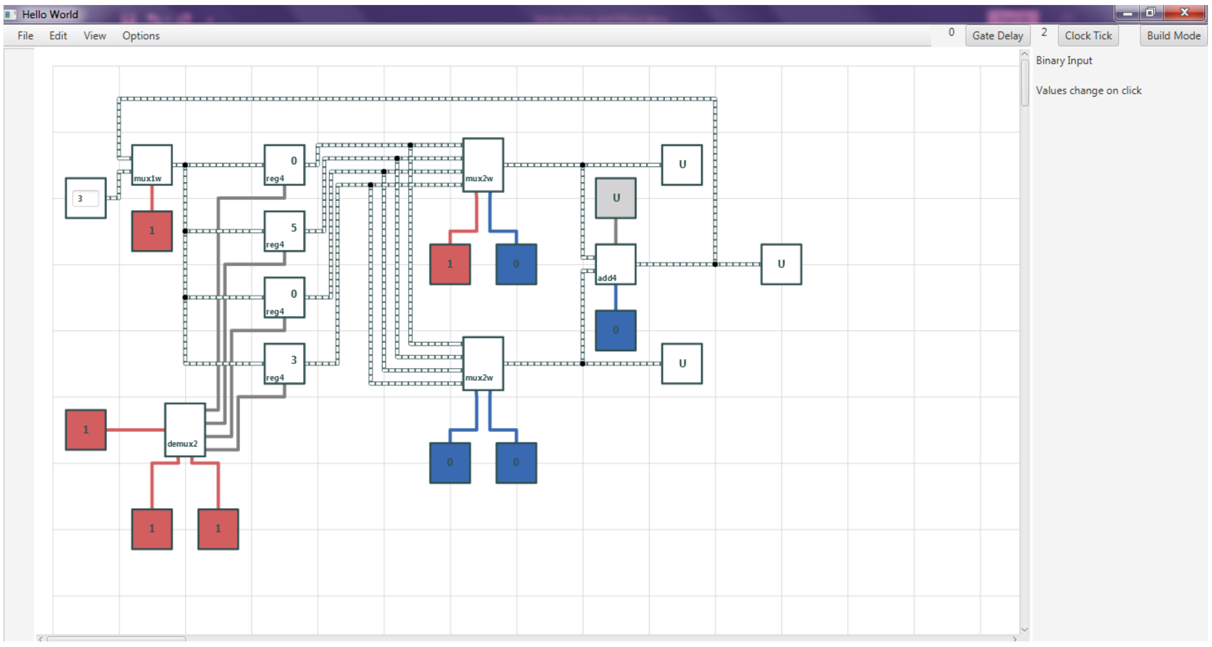
Now load 3 into R0. Change the word input to 3 and the destination address to 0 as shown below.



Again press the clock tick button. 3 should be shown in R0.



To add the two numbers, the source addresses should be set to 0 and 2. The destination address should be set to 3 to load the result of the addition to the R3. Finally, the feedback control should be set to 1 so that the value put in R3 is the result of the adder, not the value in the word input. The full setup is shown below.



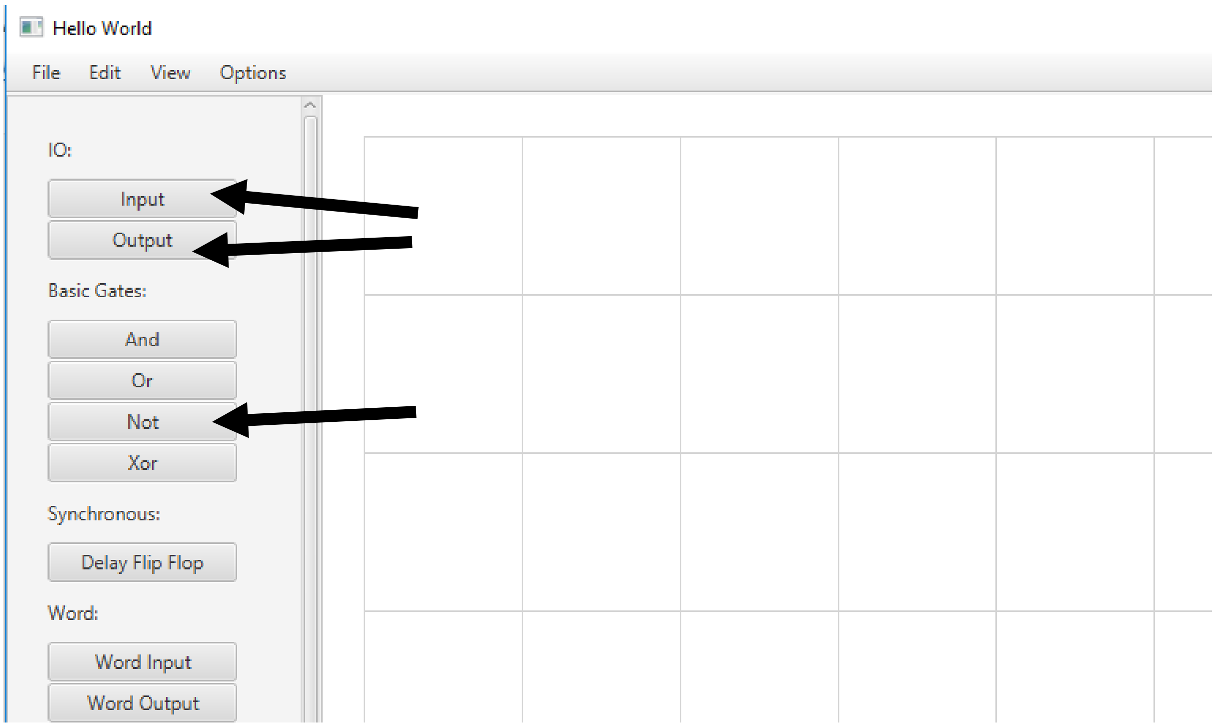
Press the clock tick button to load the result of the addition into R3.

## Extra task, Build and test not gate

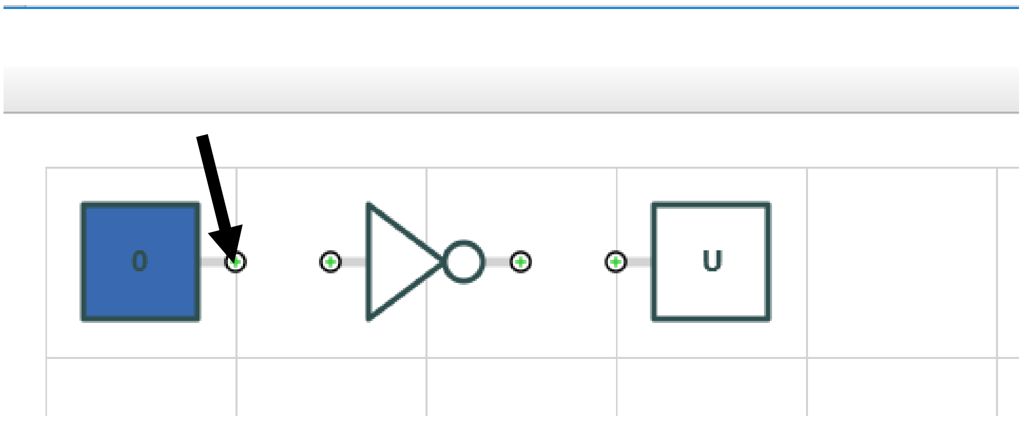
You will need to add 3 components:

- An input
- A not gate
- An output

Components can be added by clicking their name in the panel on the left. They can then be dragged around the screen.



You will then need to add wires between these components. Start building a wire by clicking on the green plus on the wires coming out of components.



Complete the wire by connecting it to another connection with a green plus.

Click start simulation.

Try simulating the circuit with 2 different input values, 1 and 0.

Use the gate delay button to simulate the gate working.

## Evaluation

Follow the evaluation form link.

## A.2 User task sheet for monitored evaluation.

This can be seen in figure A.2.

## Introduction and Ethics

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at anytime without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Task 1, Build and test not gate

You will need to add 3 components:

- An input
- A not gate
- An output

You will then need to add wires between these components.

Start the simulation.

Try simulating the circuit with 2 different input values, 1 and 0. Values can be changed by clicking on the input.

Use the gate delay button to simulate the gate working.

## Task 2, Simulate multiplexer

Load the save file called mux1, from the reusables folder.

Try to understand the circuit by simulating it.

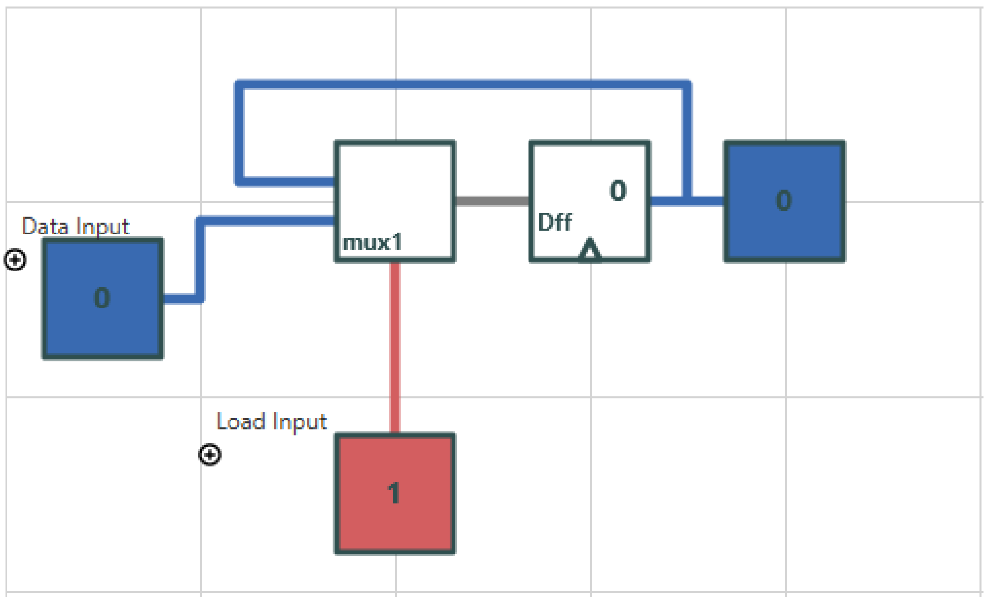
Note, you can hover over any component to see a description of it in the panel at the right. This also works for the buttons on the toolbar, hover over the mux1 button on the toolbar to see it’s description.

## Task 3, Build and test register

Clear the screen using the option under edit in the top menu.

The dff component is a delay flip flop.

Build the circuit shown here. (To rotate components, press R while clicking on them.)



Again enter simulation mode.

The value on the input marked as the data input above is only loaded into the dff on a clock tick and when the load signal is 1. This is similar to a single bit of computer memory.

Try to load the value 1 into the register, and show the result on the output.
Hint: you will need to perform both gate delays and a clock tick.

## Task 4, add 2 numbers

Load the circuit, add4 from the reusables folder.

Try to add two numbers, say 4 and 5

Look at how the signal moves between the components on each gate delay.

## Extra Task, testing the build tool

Feel free to build any other circuit you like

## Evaluation

Follow the evaluation form link.

## A.3 User evaluation form for unmonitored evaluation.

This can be seen in figure A.3.

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O’Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

* Required

1. **Ethical Consent** *
   *Mark only one oval.*

   ◯ I have read this information sheet, and agree to voluntarily take part in this experiment

   ◯ I do not agree *Stop filling out this form.*

---

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Simulate notnotnotnot circuit

2. **How did you did you find the interface during task 1?**
*Mark only one oval.*

- Easy to use
- Ok to use
- Difficult to use

3. **Describe any ways in which you think the software could be improved to make the completion of task 1 easier.**

4. **How has this changed your understanding of gate delays?**
*Mark only one oval.*

- Just more confused
- No change
- I have a better understaning of gate delays
- I have a much better understanding of gate delays

5. *Mark only one oval.*

- Continue to next task
- I did not complete any more tasks *Skip to question 16.*

## Task 2, add two numbers

6. **How did you did you find the interface during the task 2?**
*Mark only one oval.*

- Easy to use
- Ok to use
- Difficult to use

7. **Describe any ways in which you think the software could be improved to make the completion of task 2 easier.**

8. **How has this changed your understanding of the 4 bit adder?**
*Mark only one oval.*

- Just more confused
- No change
- I have a better understaning of the adder
- I have a much better understanding of the adder

9. *Mark only one oval.*

- Continue to next task
- I did not complete any more tasks *Skip to question 16.*

## Task 3, Register Transfer machine

10. **How did you did you find the interface during the task 3?**
*Mark only one oval.*

- Easy to use
- Ok to use
- Difficult to use

11. **Describe any ways in which you think the software could be improved to make the completion of task 3 easier.**

12. **How has this changed your understanding of the RTM circuit?**
*Mark only one oval.*

- Just more confused
- No change
- I have a better understaning of the RTM
- I have a much better understanding of the RTM

13. *Mark only one oval.*

- Continue to next task
- I did not complete any more tasks *Skip to question 16.*

## Extra Task, Build and test not gate

14. **How did you did you find the interface during this task?**
*Mark only one oval.*

- Easy to use
- Ok to use
- Difficult to use

15. **Describe any ways in which you think the software could be improved to make the completion of this task easier.**

## Overall

16. **Are there any new features that you would like to see in the program?**

17. **Are there any changes that you think would improve the usability of the interface?**

18. **Are the any changes that you think would improve the aesthetics of the the interface?**

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## A.4 User evaluation form for monitored evaluation.

This can be seen in figure A.4.

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

* Required

1. **Ethical Consent** *
*Mark only one oval.*

◯ I have read this information sheet, and agree to voluntarily take part in this experiment

◯ I do not agree *Stop filling out this form.*

---

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Build and test not gate

2. **How did you did you find the interface during task 1?**
   *Mark only one oval.*

   - Easy to use
   - Ok to use
   - Difficult to use

3. **Describe any ways in which you think the software could be improved to make the completion of task 1 easier.**

4. *Mark only one oval.*

   - Continue to next task
   - I did not complete any more tasks *Skip to question 19.*

## Task 2, Simulate multiplexer

5. **How did you did you find the interface during the task 2?**
   *Mark only one oval.*

   - Easy to use
   - Ok to use
   - Difficult to use

6. **Describe any ways in which you think the software could be improved to make the completion of task 2 easier.**

7. **How has this changed your understanding of multiplexers?**
*Mark only one oval.*

- Just more confused
- No change
- I have a better understaning of multiplexers
- I have a much better understanding of multiplexers

8. *Mark only one oval.*

- Continue to next task
- I did not complete any more tasks *Skip to question 19.*

## Task 3, Build and test register

9. **How did you did you find the interface during the task 3?**
*Mark only one oval.*

- Easy to use
- Ok to use
- Difficult to use

10. **Describe any ways in which you think the software could be improved to make the completion of task 3 easier.**

11. **How has this changed your understanding of the register circuit?**
*Mark only one oval.*

- Just more confused
- No change
- I have a better understaning of registers
- I have a much better understanding of registers

12. *Mark only one oval.*

- Continue to next task
- I did not complete any more tasks *Skip to question 19.*

## Task 4, add two numbers

13. **How did you did you find the interface during the task 4?**
*Mark only one oval.*

- ( ) Easy to use
- ( ) Ok to use
- ( ) Difficult to use

14. **Describe any ways in which you think the software could be improved to make the completion of task 4 easier.**

15. **How has this changed your understanding of the 4 bit adder?**
*Mark only one oval.*

- ( ) Just more confused
- ( ) No change
- ( ) I have a better understaning of the adder
- ( ) I have a much better understanding of the adder

16. *Mark only one oval.*

- ( ) Continue to next task
- ( ) I did not complete any more tasks *Skip to question 19.*

## Extra Task, Building circuits

17. **How did you did you find the interface during this task?**
*Mark only one oval.*

- ( ) Easy to use
- ( ) Ok to use
- ( ) Difficult to use

18. **Describe any ways in which you think the software could be improved to make the completion of this task easier.**

## Overall

19. **Are there any new features that you would like to see in the program?**

20. **Are there any changes that you think would improve the usability of the interface?**

21. **Are the any changes that you think would improve the aesthetics of the the interface?**

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## A.5 Results for unmonitored evaluation.

This can be seen in figure A.5.

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Simulate notnotnotnot circuit

How did you did you find the interface during task 1?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

Add automatic gate delay simulation, so circuit can for example advance 1 gate delay every one second (so you don't have to keep clicking it). The change could also be highlighted more somehow, for example add a glow effect outline on the change. When simulation is complete, this could be shown somehow.

How has this changed your understanding of gate delays?

Just more confused

No change

I have a better understaning of gate delays

[✓ I have a much better understanding of gate delays]

[✓ Continue to next task]

I did not complete any more tasks

## Task 2, add two numbers

How did you did you find the interface during the task 2?

- ○ Easy to use
- [✓ Ok to use]
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

It was a bit hard to follow what the change after gate delay was as there were many wires. Changes could be highlighted a bit better.

How has this changed your understanding of the 4 bit adder?

- ○ Just more confused
- ○ No change
- ○ I have a better understaning of the adder
- [✓ I have a much better understanding of the adder]

- ○ Continue to next task
- [✓ I did not complete any more tasks]

## Task 3, Register Transfer machine

How did you did you find the interface during the task 3?

- ○ Easy to use
- ○ Ok to use
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

How has this changed your understanding of the RTM circuit?

- ○ Just more confused
- ○ No change
- ○ I have a better understaning of the RTM
- ○ I have a much better understanding of the RTM

- ○ Continue to next task
- ○ I did not complete any more tasks

## Extra Task, Build and test not gate

How did you did you find the interface during this task?

◯ Easy to use

◯ Ok to use

◯ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of this task easier.

## Overall

Are there any new features that you would like to see in the program?

Highlighting the wires after gate delay. Right click to delete certain components instead/in addition to delete mode. Some pictures could be added to names of gates in build interface.

## Are there any changes that you think would improve the usability of the interface?

As above

## Are the any changes that you think would improve the aesthetics of the the interface?

change the hello world label hehe

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Simulate notnotnotnot circuit

How did you did you find the interface during task 1?

[✓ Easy to use]
Ok to use
Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

How has this changed your understanding of gate delays?

Just more confused
[✓ No change]
I have a better understaning of gate delays
I have a much better understanding of gate delays

[✓ Continue to next task]
I did not complete any more tasks

## Task 2, add two numbers

How did you did you find the interface during the task 2?

- ○ Easy to use
- ○ Ok to use
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

How has this changed your understanding of the 4 bit adder?

- ○ Just more confused
- ○ No change
- ○ I have a better understaning of the adder
- ○ I have a much better understanding of the adder

- ○ Continue to next task
- [✓ I did not complete any more tasks]

## Task 3, Register Transfer machine

How did you did you find the interface during the task 3?

○ Easy to use

○ Ok to use

○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

.................................................................................................................................

How has this changed your understanding of the RTM circuit?

○ Just more confused

○ No change

○ I have a better understaning of the RTM

○ I have a much better understanding of the RTM

○ Continue to next task

○ I did not complete any more tasks

## Extra Task, Build and test not gate

How did you did you find the interface during this task?

- ○ Easy to use
- ○ Ok to use
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of this task easier.

## Overall

Are there any new features that you would like to see in the program?

-

Are there any changes that you think would improve the usability of the interface?

the connection between elements acted a bit weird sometimes, so maybe that?

Are the any changes that you think would improve the aesthetics of the the interface?

-

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Simulate notnotnotnot circuit

How did you did you find the interface during task 1?

- [✓ Easy to use]
- Ok to use
- Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

n/a the step through of gate delays is particularly nice

How has this changed your understanding of gate delays?

- Just more confused
- [✓ No change]
- I have a better understaning of gate delays
- I have a much better understanding of gate delays

- Continue to next task
- I did not complete any more tasks

## Task 2, add two numbers

How did you did you find the interface during the task 2?

- ○ Easy to use
- ○ Ok to use
- [✓ Difficult to use]

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

add4 didn't display when I loaded it. I tried and failed to recreate it myself as I couldn't load and replicate the fullAdd component

How has this changed your understanding of the 4 bit adder?

- ○ Just more confused
- [✓ No change]
- ○ I have a better understaning of the adder
- ○ I have a much better understanding of the adder

- ○ Continue to next task
- ○ I did not complete any more tasks

## Task 3, Register Transfer machine

How did you did you find the interface during the task 3?

○ Easy to use

○ Ok to use

[✓ Difficult to use]

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

Same problem loading the RTM as with the previous task. Pity because this is the one I'm still confused by

How has this changed your understanding of the RTM circuit?

○ Just more confused

[✓ No change]

○ I have a better understaning of the RTM

○ I have a much better understanding of the RTM

○ Continue to next task

○ I did not complete any more tasks

## Extra Task, Build and test not gate

How did you did you find the interface during this task?

- Easy to use
- [✓ Ok to use]
- Difficult to use

Describe any ways in which you think the software could be improved to make the completion of this task easier.

Loading components works great. The connection with the wires worked strangely, producing tight twists and double backs I didn't think I'd created. I was running on Java 11 and I wonder if this caused the error - will try with the lab computers

## Overall

## Are there any new features that you would like to see in the program?

When loading components they appear immediately upon clicking the botton in the top left corner. This confused me occasionally as I was left wondering where they were. Perhaps they could change colour when they've just been placed - or similar to attract attention.

It'd be nice to change the orientation of components.

## Are there any changes that you think would improve the usability of the interface?

When I opened save a black box to see what it was the dialogue box gave a really good description of how it worked, but it didn't give me an easy way to return to the previous menu having decided it wasn't what I was after.

## Are the any changes that you think would improve the aesthetics of the the interface?

I think it looks really good

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Simulate notnotnotnot circuit

How did you did you find the interface during task 1?

- ○ Easy to use
- [✓ Ok to use]
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

How has this changed your understanding of gate delays?

- ○ Just more confused
- ○ No change
- [✓ I have a better understaning of gate delays]
- ○ I have a much better understanding of gate delays

- [✓ Continue to next task]
- ○ I did not complete any more tasks

## Task 2, add two numbers

How did you did you find the interface during the task 2?

- ○ Easy to use
- [✓ Ok to use]
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

.......................................................................................................................................

How has this changed your understanding of the 4 bit adder?

- ○ Just more confused
- ○ No change
- [✓ I have a better understaning of the adder]
- ○ I have a much better understanding of the adder

- ○ Continue to next task
- ○ I did not complete any more tasks

## Task 3, Register Transfer machine

How did you did you find the interface during the task 3?

- ○ Easy to use
- ○ Ok to use
- [✓ Difficult to use]

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

..................................................................................................................................

How has this changed your understanding of the RTM circuit?

- [✓ Just more confused]
- ○ No change
- ○ I have a better understaning of the RTM
- ○ I have a much better understanding of the RTM

- ○ Continue to next task
- ○ I did not complete any more tasks

## Extra Task, Build and test not gate

How did you did you find the interface during this task?

- ○ Easy to use
- ○ Ok to use
- [✓ Difficult to use]

Describe any ways in which you think the software could be improved to make the completion of this task easier.

Have the truth tables auto populate other wise a very good program THANK YOU!!

## Overall

Are there any new features that you would like to see in the program?

Show you where the error is

Are there any changes that you think would improve the usability of the interface?

Have a help sheet

Are the any changes that you think would improve the aesthetics of the the interface?

I like the colours make componets smaller

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## A.6 Results for monitored evaluation.

This can be seen in figure A.6.

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Build and test not gate

How did you did you find the interface during task 1?

○ Easy to use

[✓ Ok to use]

○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

Can't think of any, it was just a tiny learning curve to figure out how to connect wires

[✓ Continue to next task]

○ I did not complete any more tasks

## Task 2, Simulate multiplexer

How did you did you find the interface during the task 2?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

How has this changed your understanding of multiplexers?

Just more confused

No change

[✓ I have a better understaning of multiplexers]

I have a much better understanding of multiplexers

[✓ Continue to next task]

I did not complete any more tasks

## Task 3, Build and test register

How did you did you find the interface during the task 3?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

How has this changed your understanding of the register circuit?

Just more confused

No change

[✓ I have a better understaning of registers]

I have a much better understanding of registers

[✓ Continue to next task]

I did not complete any more tasks

## Task 4, add two numbers

How did you did you find the interface during the task 4?

- [✓ Easy to use]
- Ok to use
- Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 4 easier.

..............................................................................................................

How has this changed your understanding of the 4 bit adder?

- Just more confused
- [✓ No change]
- I have a better understaning of the adder
- I have a much better understanding of the adder

- Continue to next task
- [✓ I did not complete any more tasks]

## Extra Task, Building circuits

How did you did you find the interface during this task?

- ○ Easy to use
- ○ Ok to use
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of this task easier.

**Overall**

Are there any new features that you would like to see in the program?

Are there any changes that you think would improve the usability of the interface?

Are the any changes that you think would improve the aesthetics of the the interface?

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O’Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Build and test not gate

How did you did you find the interface during task 1?

- ○ Easy to use
- [✓ Ok to use]
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

It feels more natural to me to drag and drop things than to click a button and then appear. This was a recurring theme for me.

- [✓ Continue to next task]
- ○ I did not complete any more tasks

## Task 2, Simulate multiplexer

How did you did you find the interface during the task 2?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

It was easy to find the file I needed, but had same issue of wanting to drag and drop.

How has this changed your understanding of multiplexers?

Just more confused

No change

I have a better understaning of multiplexers

[✓ I have a much better understanding of multiplexers]

[✓ Continue to next task]

I did not complete any more tasks

## Task 3, Build and test register

How did you did you find the interface during the task 3?

- ○ Easy to use
- [✓ Ok to use]
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

Issues may have been down to my poor circuit knowledge, btu I felt like I had some issues getting the circuit to work at first.

How has this changed your understanding of the register circuit?

- ○ Just more confused
- ○ No change
- [✓ I have a better understaning of registers]
- ○ I have a much better understanding of registers

- [✓ Continue to next task]
- ○ I did not complete any more tasks

## Task 4, add two numbers

How did you did you find the interface during the task 4?

- [✓ Easy to use]
- Ok to use
- Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 4 easier.

How has this changed your understanding of the 4 bit adder?

- Just more confused
- No change
- [✓ I have a better understaning of the adder]
- I have a much better understanding of the adder

- [✓ Continue to next task]
- I did not complete any more tasks

## Extra Task, Building circuits

How did you did you find the interface during this task?

Easy to use [✓ Ok to use] Difficult to use

Describe any ways in which you think the software could be improved to make the completion of this task easier.

Again, I wanted to drag and drop, but overall trying out some circuits helped me better understand how a demultiplexer worked.

## Overall

Are there any new features that you would like to see in the program?

More information about things on hover.

Are there any changes that you think would improve the usability of the interface?

rather than clicking a button and going into a new mode to delete things, maybe double click them, or right click

Are the any changes that you think would improve the aesthetics of the the interface?

I think it looks very clean and professional

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Build and test not gate

How did you did you find the interface during task 1?

○ Easy to use

[✓ Ok to use]

○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

[✓ Continue to next task]

○ I did not complete any more tasks

## Task 2, Simulate multiplexer

How did you did you find the interface during the task 2?

- Easy to use
- [✓ Ok to use]
- Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

How has this changed your understanding of multiplexers?

- Just more confused
- No change
- [✓ I have a better understaning of multiplexers]
- I have a much better understanding of multiplexers

- [✓ Continue to next task]
- I did not complete any more tasks

## Task 3, Build and test register

How did you did you find the interface during the task 3?

- ○ Easy to use
- [✓ Ok to use]
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

How has this changed your understanding of the register circuit?

- [✓ Just more confused]
- ○ No change
- ○ I have a better understaning of registers
- ○ I have a much better understanding of registers

- [✓ Continue to next task]
- ○ I did not complete any more tasks

## Task 4, add two numbers

How did you did you find the interface during the task 4?

- ◯ Easy to use
- ◯ Ok to use
- ◯ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 4 easier.

How has this changed your understanding of the 4 bit adder?

- ◯ Just more confused
- ◯ No change
- ◯ I have a better understaning of the adder
- ◯ I have a much better understanding of the adder

- ◯ Continue to next task
- ◯ I did not complete any more tasks

## Extra Task, Building circuits

How did you did you find the interface during this task?

- ◯ Easy to use
- ◯ Ok to use
- ◯ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of this task easier.

**Overall**

Are there any new features that you would like to see in the program?

Are there any changes that you think would improve the usability of the interface?

Are the any changes that you think would improve the aesthetics of the the interface?

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Build and test not gate

How did you did you find the interface during task 1?

- ○ Easy to use
- [✓ Ok to use]
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

Don't spawn items of each other.

- [✓ Continue to next task]
- ○ I did not complete any more tasks

## Task 2, Simulate multiplexer

How did you did you find the interface during the task 2?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

How has this changed your understanding of multiplexers?

Just more confused

No change

[✓ I have a better understaning of multiplexers]

I have a much better understanding of multiplexers

Continue to next task

I did not complete any more tasks

## Task 3, Build and test register

How did you did you find the interface during the task 3?

- ○ Easy to use
- [✓ Ok to use]
- ○ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

..........

How has this changed your understanding of the register circuit?

- ○ Just more confused
- [✓ No change]
- ○ I have a better understaning of registers
- ○ I have a much better understanding of registers

- [✓ Continue to next task]
- ○ I did not complete any more tasks

## Task 4, add two numbers

How did you did you find the interface during the task 4?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 4 easier.

How has this changed your understanding of the 4 bit adder?

Just more confused

No change

I have a better understaning of the adder

[✓ I have a much better understanding of the adder]

Continue to next task

[✓ I did not complete any more tasks]

## Extra Task, Building circuits

How did you did you find the interface during this task?

- ◯ Easy to use
- ◯ Ok to use
- ◯ Difficult to use

Describe any ways in which you think the software could be improved to make the completion of this task easier.

**Overall**

Are there any new features that you would like to see in the program?

No

Are there any changes that you think would improve the usability of the interface?

Some kind tutorial which highlights on certain parts

Are the any changes that you think would improve the aesthetics of the the interface?

None

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

# Circuit Builder User Evaluation

The aim of this experiment is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use. This requires user testing, as the program creator cannot give an impartial evaluation of usability.
The experiment is expected to take less that 30 minutes to complete.
At the start of the experiment, you will be given a list of tasks to complete using the program. You will then be given access to the program.

At the end of the experiment, you will be asked to complete a questionnaire.
All results will be held in strict confidence, ensuring the privacy of all participants. No personal participant information will be stored with the data. Online data will be stored in a password protected computer account; paper data will be kept securely.

Your participation in this experiment will have no effect on your marks for any subject at this, or any other university. Please note that it is the program, not you, that is being evaluated. You may withdraw from the experiment at any time without prejudice, and any data already recorded will be discarded.

If you have any further questions regarding this experiment, please contact:
Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

## Ethical Consent *

[✓ I have read this information sheet, and agree to voluntarily take part in this experiment]

I do not agree

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster <stephen@dcs.gla.ac.uk>.

## Task 1, Build and test not gate

How did you did you find the interface during task 1?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 1 easier.

Error messages on incorrect actions.

[✓ Continue to next task]

I did not complete any more tasks

## Task 2, Simulate multiplexer

How did you did you find the interface during the task 2?

- [✓ Easy to use]
- Ok to use
- Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 2 easier.

Nope, was very straight forward.

How has this changed your understanding of multiplexers?

- Just more confused
- No change
- [✓ I have a better understaning of multiplexers]
- I have a much better understanding of multiplexers

- [✓ Continue to next task]
- I did not complete any more tasks

## Task 3, Build and test register

How did you did you find the interface during the task 3?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 3 easier.

Everything was intuitive

How has this changed your understanding of the register circuit?

Just more confused

No change

[✓ I have a better understaning of registers]

I have a much better understanding of registers

[✓ Continue to next task]

I did not complete any more tasks

## Task 4, add two numbers

How did you did you find the interface during the task 4?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of task 4 easier.

N/A

How has this changed your understanding of the 4 bit adder?

Just more confused

No change

[✓ I have a better understaning of the adder]

I have a much better understanding of the adder

[✓ Continue to next task]

I did not complete any more tasks

## Extra Task, Building circuits

How did you did you find the interface during this task?

[✓ Easy to use]

Ok to use

Difficult to use

Describe any ways in which you think the software could be improved to make the completion of this task easier.

None, was lots of fun to throw lots of components at it and see what happened.

## Overall

Are there any new features that you would like to see in the program?

Error messages on clicking incorrect buttons.

Are there any changes that you think would improve the usability of the interface?

None come to mind, the interface seemed very good.

Are the any changes that you think would improve the aesthetics of the the interface?

Nope.

## Ethics Debrief

As stated at the beginning of the experiment, the aim is to test out a circuit building program to provide evidence of its effectiveness, as well as how easy it is to use.

Do you have any questions relating to the program or the experiment?

If you would like to discuss the project at any time, with myself or my project advisor, we can be contacted at,

Brodie West, Program Creator
2189607w@student.gla.ac.uk
John O'Donnell, Project Advisor
john.odonnell@glasgow.ac.uk

| Participant | Help required | Issues/ suggestions |
|---|---|---|
| 1 | 1 | Often accidentally dragged components when trying to start building wires |
| | | Confusing that some components take multiple gate delays |
| 2 | 0 | Wires should be created when build icons are ontop of eachother. |
| | | Would like to drag from buttons |
| | | Connected message was useful |
| 3 | 7 | Looked for wire button |
| | | Would like to right click to delete compoents |
| | | Suggested it would be easier to not have file, edit etc. and just have all options separate. |
| | | Right click to rotate |
| | | Confused between outputs and inputs |
| | | Message when wire cannot be created with explaination |
| 4 | 2 | Bug found, decriptions do not work for every component |
| | | Confusion when components appeared ontop of eachother |
| 5 | 2 | Confused about having to start simulation |
| | | Confused when loading circuits when not at the edge of the screen, could move to there on load |
| | | Should show error when wires cannot be built. |

***Table A.1:*** *Users were monitored during this test and any difficulties they found with the program were noted during the test.*

## A.7 Notes from monitored evaluation.

Users were monitored during this test and any difficulties they found with the program were noted during the test. This can be seen in table A.1.

# 7 Bibliography

I. I. I. T. Bangalore. Circuitverse. https://circuitverse.org/, 2019.

A. Bien. Structuring complex javafx 8 applications for productivity. https://www.oracle.com/technetwork/articles/java/javafx-productivity-2345000.html, 2014.

DesignWorks. Logic works. https://designworkssolutions.com/logicworks/, 2019.

I. Fedortsova. Mastering fxml: Creating a custom control with fxml. https://docs.oracle.com/javafx/2/fxml_get_started/custom_control.htm, 2014.

I. D. Foundation. Hawthorne effect. https://www.interaction-design.org/literature/article/light-up-your-user-research-understanding-the-hawthorne-effect 2017.

I. D. Foundation. User observations. https://www.interaction-design.org/literature/article/how-to-conduct-user-observations, 2018.

O. S. GIMP. Gimp. https://www.gimp.org/, 2018.

S. Hommel. Implementing javafx best practices. https://docs.oracle.com/javafx/2/best_practices/jfxpub-best_practices.htm, 2014.

S. Kollmansberger. Logic gate simulator. https://www.kolls.net/gatesim/, 2011.

G. Kruk, O. Da Silva Alves, L. Molinari, and E. Roux. Best practices for efficient development of javafx applications. *Proceedings of the 16th Int. Conf. on Accelerator and Large Experimental Control Systems*, ICALEPCS2017, 2018. doi: 10.18429/jacow-icalepcs2017-thapl02.

B. H. LLC. Logicly. https://logic.ly/, 2019.

J. Ltd. draw.io. https://www.draw.io/, 2019.

D. Norman. *The design of Everydaty Things*. Basic Books, 2013.

Photopea. Online image editor. https://www.photopea.com/, 2013.

T. Schlatter and D. Levinson. *Visual Usability*. Morgan Kaufmann, 2013.

O. S. Testfx. *Testfx, Simple and clean testing for JavaFX*. TestFX, Open Source, 2019.

Unity. Unity. https://unity3d.com, 2019.