Douglas Finlay

Beauchamp College

A2 Computing Logic Simulator

2012/2013

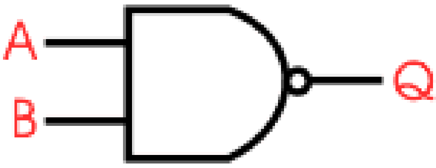
Table of Contents

ANALYSIS

BACKGROUND

Teachers at Beauchamp College are looking for a new method of teaching digital logic to their GCSE and A-level Electronics and Computing students in a manner which is easier to grasp than the current system. It is a requirement of both syllabuses for digital logic to be taught, hence why is important for the departments to obtain an accurate and intuitive method which is easily understandable by students.

A requirement for both subjects is for the students to be able have an understanding of the fundamental logic gates: NOT; AND; OR; NAND; NOR (and possibly XOR/XNOR). It is also a requirement for the students to be able to construct truth tables for each of these gates; showing that they have a thorough understanding of boolean logic. Students are also expected to learn the conventional symbols used for the gates. An example of the symbol used for a NAND logic gate (and its corresponding truth table) is shown below.



A	B	$Q = \overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

It is usual for an entire class to be taught this topic together by a single teacher. In electronics lessons at Beauchamp, the classes are built up from roughly 30 students, but significantly fewer (4 – 12) in computing lessons. In electronics, the students work individually to build and experiment with digital logic circuits to enhance their practical skills. Along with this practical aspect, students are also taught about the theory; how to use and simplify logic-based algebraic equations.

IDENTIFICATION OF THE PROBLEM(S)

One of the main disadvantages with an initially practical learning method is the cost and the waste of resources. This high cost is caused is mainly caused by wastage of materials. It is very easy for a student to accidentally destroy many logic integrated circuits by applying a current or voltage which is too large, or from handling the integrated circuits without a suitable level of care.

On the other hand, a purely theoretical method (such as the one used in computing) can be problematic for visual learners, who aren't easily able to picture the flow of combined logic gates to give them a clear understanding of the necessary materials. This regularly leads to time-consuming lessons in which a lot of time could be spent on other syllabus modules, making the students feel more comfortable with what they are learning. Having more constructive teaching time would be of great benefit to the teachers, as more time could be spent running planned revision lessons.

The practical method in electronics is beneficial in the long-run; however, it can initially be very costly for the department as many delicate components are damaged due to insufficient knowledge of the gates. The department would benefit for a system that eliminates the high cost earlier on in the course, but still provides the students with a clear knowledge of digital logic.

Description of the Current System

Currently, computing students use truth tables as a means to learn about the various logic gates required by the syllabus. Each truth table corresponds to a single logic gate, giving the students an overview the gate's unique operation. With a knowledge of these gates, the students advance onto combining the gates into simple circuits to achieve specific tasks. For example, one task may be the control of a car's interior lighting depending on the state of the ignition and doors.

Electronics students are taught very similar topics, however, they are not taught using as much theory as computing students. They are taught using practical methods, which enhances other skills required for their particular course.

Identification of the Prospective User(s)

The main users of the system would be students who are studying GCSE and A-level Computing or Electronics. The students usually work in small groups (or alone, if resources allow this), to enhance their knowledge of digital logic. Every student has access to IT equipment at any time during college hours, and will also have access to IT equipment outside of college due to the recent rise in demand for computerised systems. As a result of this, most students will be computer literate, although it cannot be assumed that everybody will have the same level of competence.

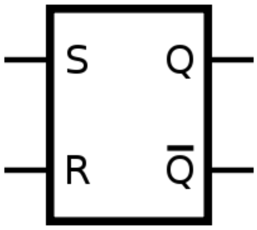
It is likely that teachers would also have a benefit from a new system. Currently, teaching is performed on a whiteboard in which it is hard to interactively simulate circuits and logic gates. Simulating logic would be a huge benefit for teachers as class presentations would be more understandable for students and easier to carry out.

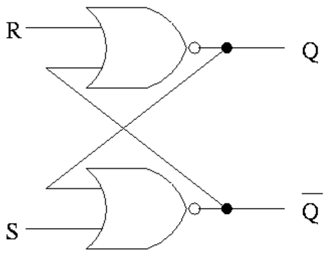
Both teachers and students who are part of the college have access to recent versions of the Windows operating system. This means that every student and teacher will have access to very similar systems including similar hardware and architectures, no matter where they are in the college.

Identification of User Needs and Acceptable Limitations

Almost every system has to be easy and intuitive for people to use. This is because the new system will be used by students and teachers, with a wide range of technical ability, so simplicity is one of the most important requirements for this new, computerised system. Factors that could make the system simple would be the addition of drag-and-drop logic gates when creating circuits. It will be necessary that a user is able to drag the desired logic gate from a toolbar, and drop it on the schematic with ease. A search function for logic gates is not necessary, as there won't be an incredibly large selection of gates available. It is also important that the user is able to draw connecting wires between the inputs and outputs of each gate with minimal confusion. There must also be restrictions on the connection of logic gates. The output of any gate or switch must be allowed to connect to more than one input. However, the input of any device should not be connected to more than one output.

The schematics should be editable by the users. This involves the selection of gates in order to move them to a new location and also delete them. When the gates are clicked and dragged, the wires will need to stay attached to eliminate the frustration of reconnecting gates. It is also necessary for the user to be able to delete wires in order to reconnect them in a different configuration. The ability to edit schematics is essential, as it will allow the users to prove their skill to examiners by creating an easily understandable circuit diagram for their coursework. Users must have NOT, AND, OR, NAND, NOR and possibly XOR and XNOR gates available for use.





Some combinations of gates are grouped as separate devices. For example, an RS latch (left) is made using two NOR or NAND gates, but can be expressed on a schematic with a single symbol. It is not necessary for students to learn how to use symbols for composite gates like this, so they are not going to be implemented in the final system.

Including theory, electronics students are required to produce practical pieces of work using computer aided design (CAD) tools to aid with the creation of custom circuit boards. Since various CAD systems which support boolean logic integrated circuits are already in place, the ability to produce circuit board layouts and information regarding specific manufacturers' components is not required in the new system.

Although the new system will be required to simulate logic circuits, a method to simulate complex circuits may not be implementable within the given time period. These circuits contain gates whose outputs eventually link back to themselves as inputs (whether directly or via other gates).

Users will require the ability to view the state of individual wires and outputs during simulation. It is recommended that users are able to place output indicators (the equivalent of lamps) to any output. They must also be able to have control over the inputs if desired, so input buttons which produce single pulses and switches which are able to be toggled are necessary components.

DATA VOLUMES

Data volumes in this system will be minimal, thus not being relevant.

OBJECTIVES FOR THE PROPOSED SYSTEM

The new system must provide and easy method to simulate a specific boolean logic circuit. End users must be able to connect indicators to any wire in the circuit if desired, clearly changing visually as the wire changes state. As well as this, it would be beneficial for each wire to be colour coded; for example, when the wire's state is low (0/off), it is black, and when it is high (1/on), it is coloured red. The users must also be able to add interactive input buttons to wires in the circuit. Logic simulation is the main requirement for the system. If the system allows visual and interactive simulation of logic circuits, then this objective has been achieved.

To allow users to create and edit boolean logic circuits, the new system must be able to:

- create a new, blank schematic;
- place logic gates and wires on the schematic;
- connect any desired logic gates together with wires;
- move (drag) existing gates to new positions, in order to visually simply the circuit;
- remove logic gates and wires from the schematic.

The goal mentioned above is realistic and will be attainable. It is of great importance to the end user as it will give them the freedom to experiment with various combinations of logic gates. Movement, addition and deletion of components on a blank canvas (schematic) will be simple to implement, so this goal will be achievable within the time available.

End users will need a selection of logic gates to choose from when designing their own circuit. Students are expected to have an understanding of NOT, AND, OR, NAND, NOR, XOR and XNOR gates. So these gates are required to be in the new system. These should be implementable within the available time, as each gate is simply a slight variation of existing gates. If time does not allow them implementation of every gate listed here, it will be acceptable to not include the XOR and XNOR gates.

Each logic gate will need to be situated on an easily accessibly panel. This will make it easier for the user to find the required logic gates to add to their design. The panel must clearly be a collection of the available logic gates to avoid confusion when the user is trying to create or edit a design. This requirement will have been met if there is a panel containing various selectable logic gates. This objective will be attainable within the time period available, and it important as it will provide an easy method for users to interact with the new system.

The new system will need to include a simple means of navigation including a heavy amount of icons to cater for the varying skill of users. This involves icons for each type of logic gate, and also icons for printing, simulating and possibly loading and saving the logic circuits. Easy navigation is useful for less competent users of software. Although easy navigation is relevant to the system, it is not vital to be implemented within the available time.

Printing schematics is another requirement for the new system. The user should be able to obtain a hard copy of the information (schematic) shown on the screen. If a specific number of copies of the schematic can be printed onto various sizes of paper, then this objective has been achieved. Providing a method of printing the user’s work will be worthwhile. For example, students will need the ability to add a hard copy of their work to their coursework, and teachers might print out example circuits for an entire class. This goal will be achievable within the time available, as printing an image of what the user can see (without any extra formatting apart from scaling to cater for different paper sizes) is a simple feature to implement, due to numerous operating system libraries.

Useful error/warning message will have to be provided when:

- the outputs from more than one device are connected together;
- the circuit cannot be simulated;
- a file cannot be loaded;
- an existing file will be replaced when saving the schematic;
- any unexpected error occurs.

This objective will have been achieved if these trivial errors are handled in an understandable manner (so the user is able to rectify them). The error checking will be attainable within the time available, as error checking will play a large part in the simulation of logic circuits. This is important to the system, as errors may frequently be made, especially when students are learning about logic circuits.

Another requirement is for the system to include some form of help or reference system. This will contain information on each logic gate (including truth tables) and also basic usage and simulation instructions for the system. A full help system covering every logic gate might not be attainable within the available time;

however, basic information will be included and could be expanded at a later date. A detailed help system will not be vital to the end users, as they will have their own notes and knowledge about logic gates.

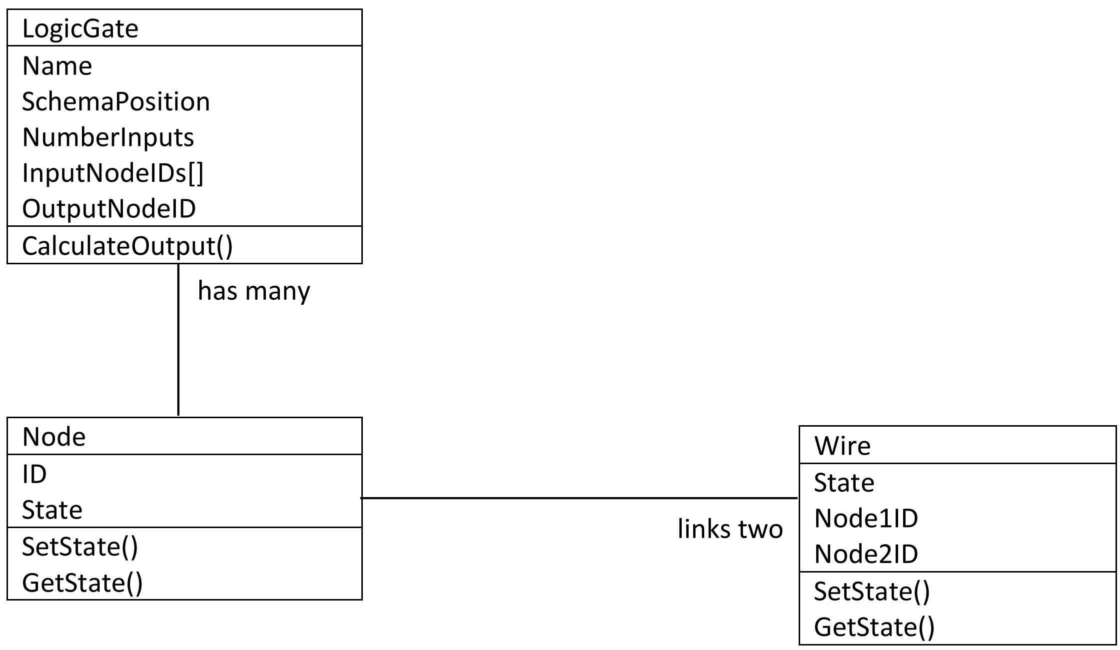
JUSTIFICATION OF CHOSEN SOLUTION

I believe a computerised system will be of great benefit to both teachers and students, to aid the learning of digital logic. A computerised system provides a cheap and easy way for teachers to present and teach vital concepts required by the syllabus. It would also allow students to work independently – whether at home or at college – to enhance their knowledge of the topic. This would be an important asset to students as it would allow them to increase their knowledge of the topic any time they wish.

One potential solution would be to write the source code from scratch. However, this method would take a long time due to the large scale of the project. Another solution would be to use Free Pascal with Lazarus (an open source integrated development environment) to develop the new system. The rapid application development aspect of this method will allow quick and easily development of the software, while still being able to have complete control over every component.

As writing the source code from scratch is not a feasible option, I will be using Free Pascal and Lazarus to implement this system.

IDENTIFICATION OF OBJECTS AND OBJECT ANALYSIS DIAGRAMS



Design

I intend to produce a logic circuit simulator with a form-based interface. In order to be intuitive, menus and toolbars will be included as part of the user interface, to allow selection of tools and gates. The software will also have the ability to save and load existing circuits, with toolbar buttons providing access to these features.

Software Tools

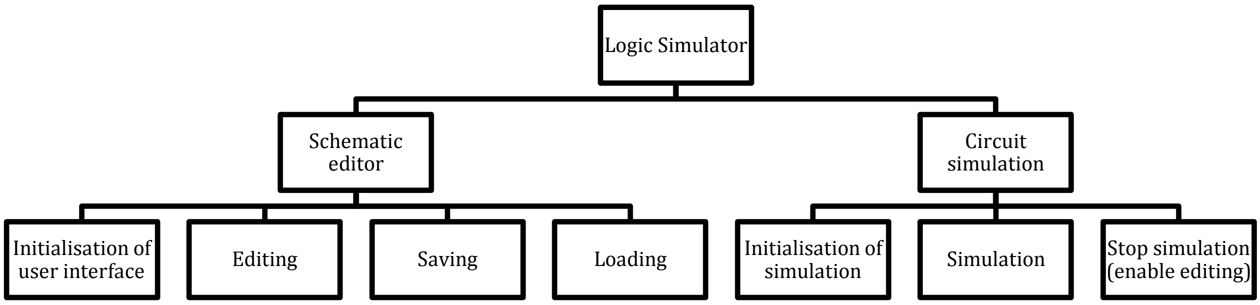
The Free Pascal Lazarus IDE will be used for the implementation of the system due to its included rapid application development features. These will allow me to use pre-written form components without having to write my own components from scratch, which would be a long and unnecessary task.

Development System

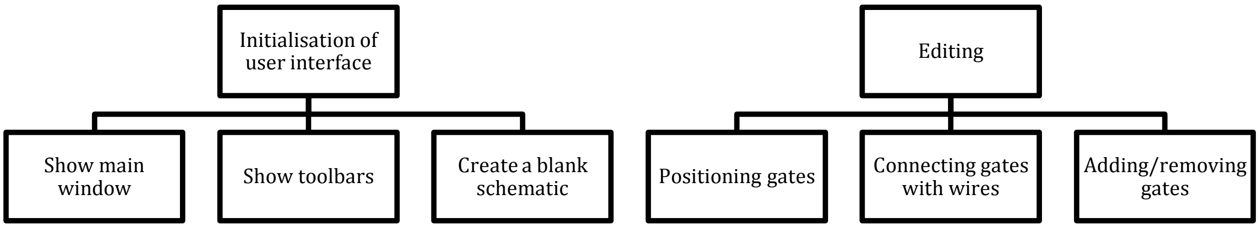
This logic simulator will be developed on Windows 7, and is intended to run on both 32 and 64 bit architectures. Systems intending the run the software will require a clock speed of at least 2GHz, at least 1GB of memory, and a screen resolution of at least 800x600 pixels.

Top-Down Design

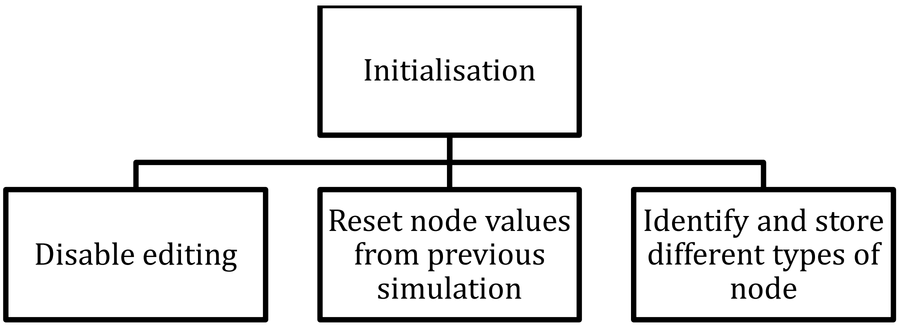
Main System



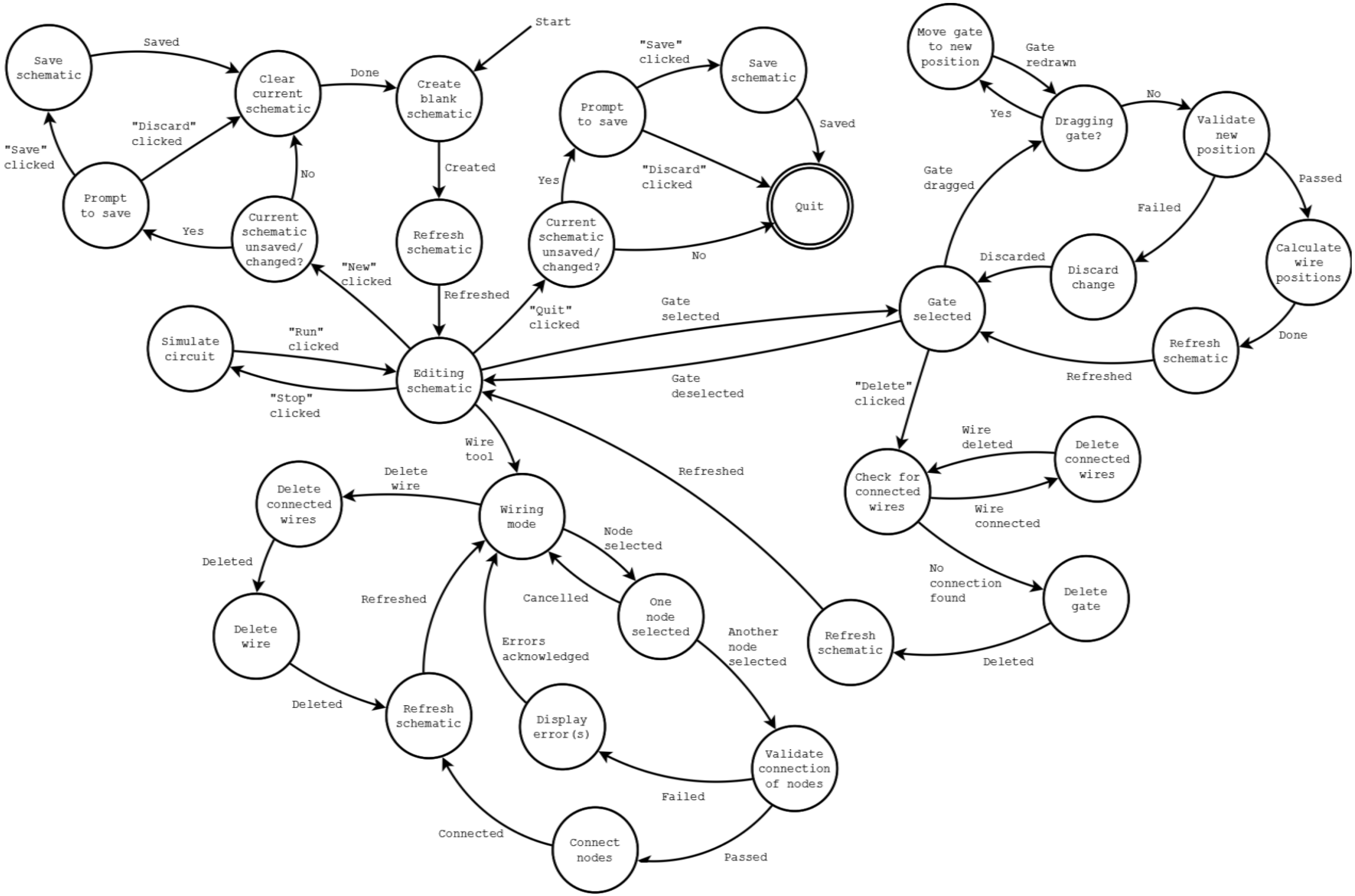
Schematic Editor

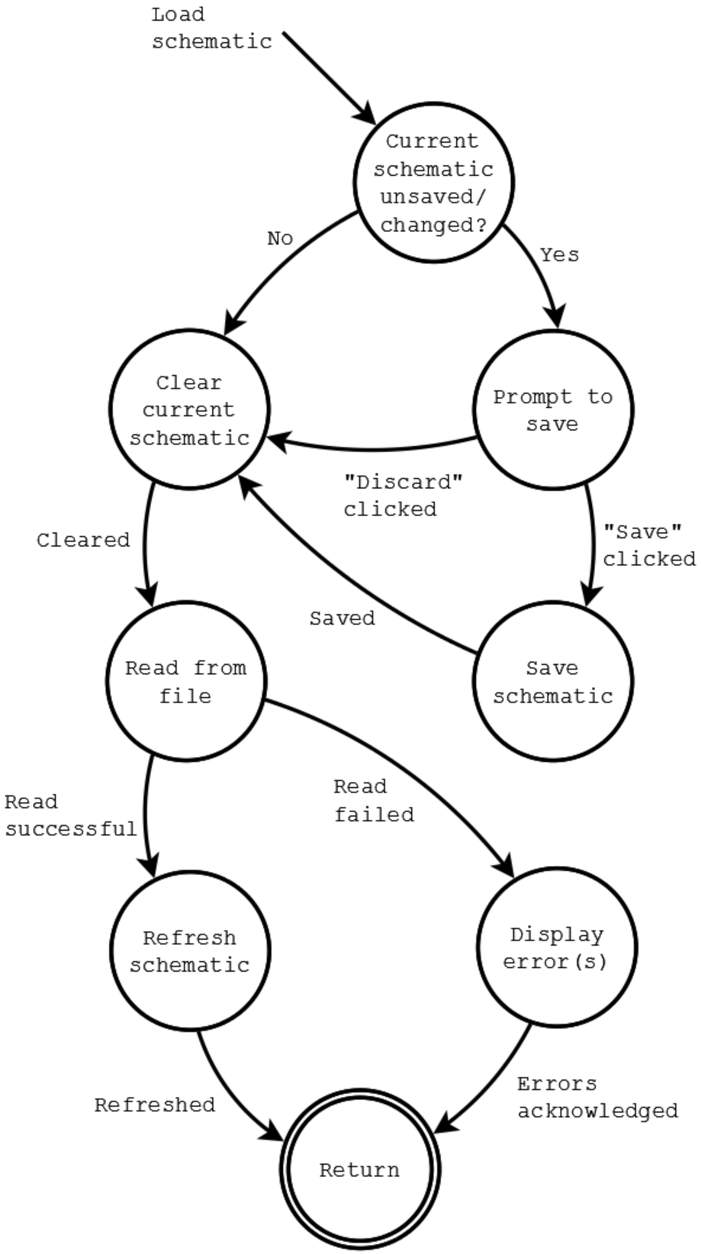


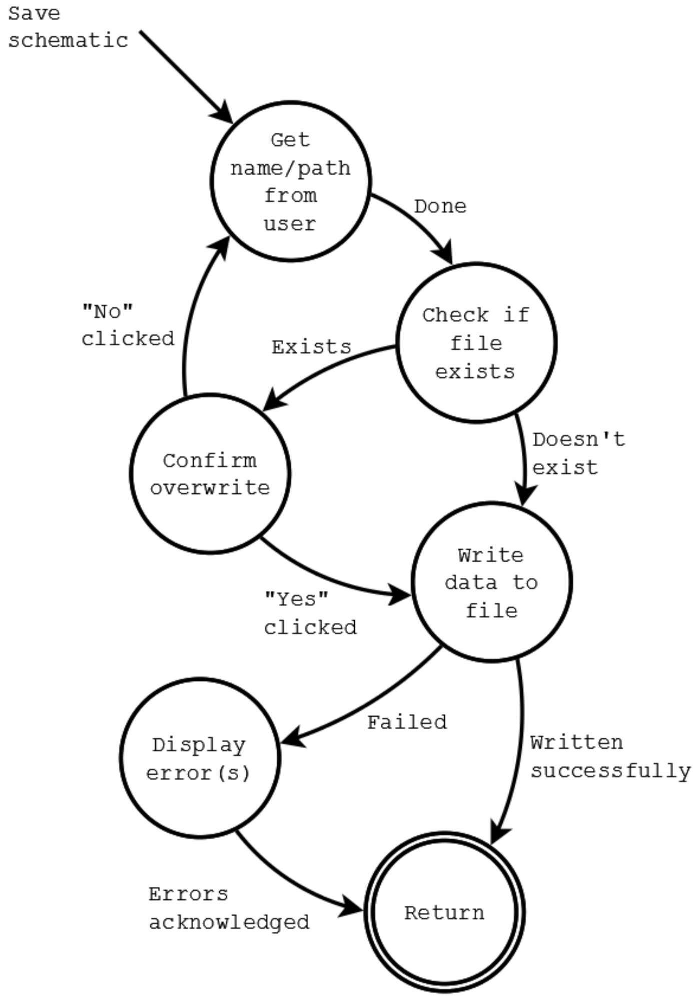
Circuit Simulation



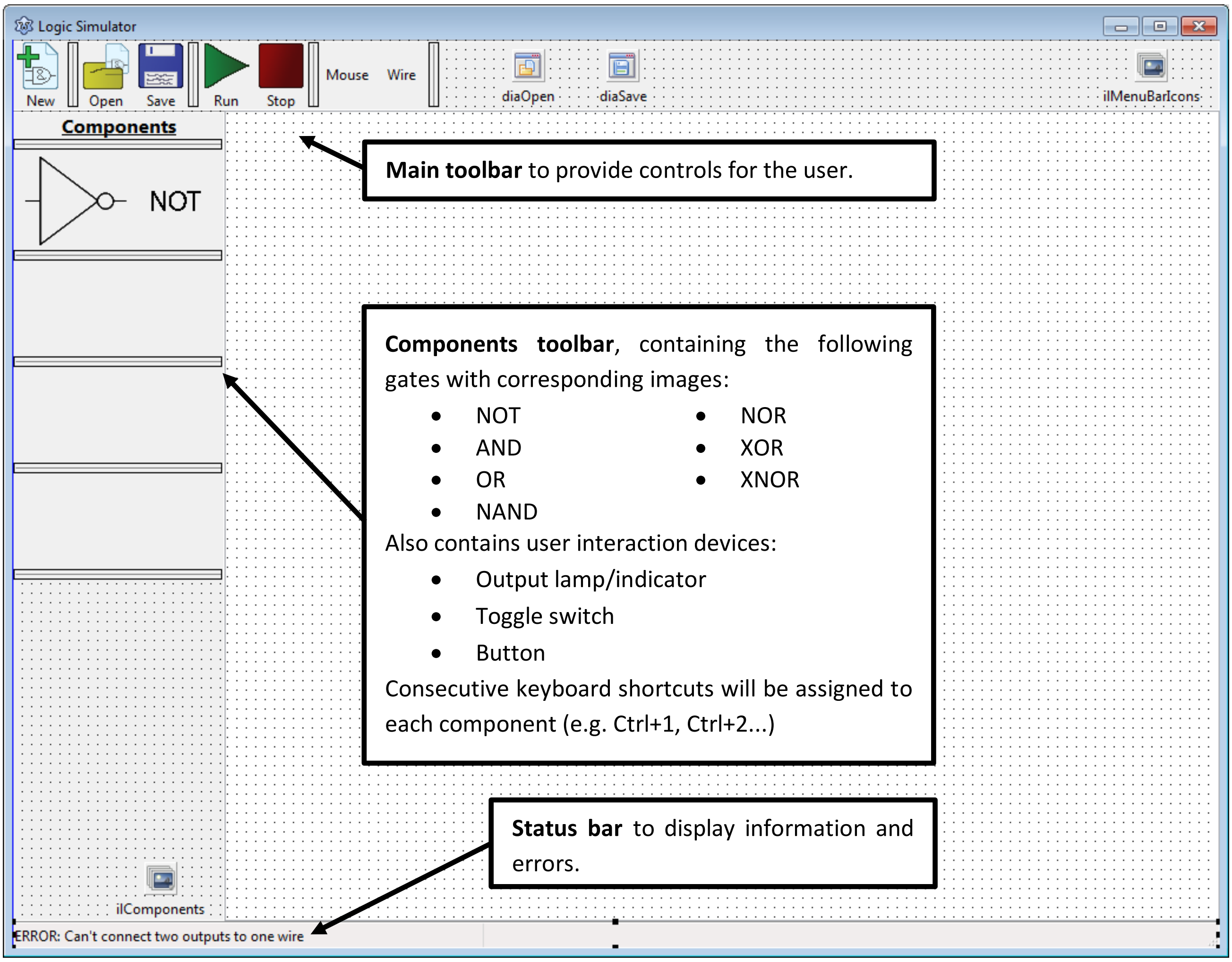
SYSTEM STATE DIAGRAM (EDITING SCHEMATIC)




Load
schematic
Current
schematic
unsaved/
changed?
No
Yes
Clear
current
schematic
Prompt to
save
"Discard"
clicked
Cleared
"Save"
clicked
Saved
Read from
file
Save
schematic
Read
successful
Read
failed
Refresh
schematic
Display
error(s)
Refreshed
Errors
acknowledged
Return


Save
schematic
Get
name/path
from
user
Done
"No"
clicked
Check if
file
exists
Exists
Confirm
overwrite
Doesn't
exist
Write
data to
file
"Yes"
clicked
Display
error(s)
Failed
Written
successfully
Errors
acknowledged
Return

HUMAN CONTROL INTERFACE (MAIN FORM)



Purpose

This will be the main form of the new system, and will provide an intuitive interface in order for users to easily create and edit boolean logic circuits. There is a large schematic view which will provide a method of designing and visualising the schematics. This form will also provide the interface for simulating the boolean logic circuits.

Components

Name (component type)	Properties	Action
frmLogicSim (TForm)	Caption ← "Logic Simulator" Width ← 1024 Height ← 768 Position ← poScreenCenter Constraints.MinWidth ← 800 Constraints.MinHeight ← 600	N/A
pnlSchema (TPanel)	Align ← alClient BevelOuter ← bvNone BorderStyle ← bsSingle Color ← clWhite	N/A

Name (component type)	Properties		Action
tbComponents (TToolBar)	Align ButtonWidth ButtonHeight Images Enabled	← alLeft ← 178 ← 80 ← ilComponents ← *False when simulating, true otherwise.*	N/A
tbMenuBar (TToolBar)	Align ButtonHeight ButtonWidth Images ShowCaptions	← alTop ← 56 ← 46 ← ilMenuBarIcons ← True	N/A
btnNew (TToolButton)	Caption ImageIndex Enabled	← "&New" ← *(corresponding image)* ← *False when simulating, true otherwise.*	A new, blank schematic is created. However, if the schematic has changed since the last save, an option for the user to save is displayed. Keyboard shortcut: Ctrl+N
btnOpen (TToolButton)	Caption ImageIndex Enabled	← "&Open" ← *(corresponding image)* ← *False when simulating, true otherwise.*	An existing schematic is loaded from a file. Keyboard shortcut: Ctrl+O
btnSave (TToolButton)	Caption ImageIndex Enabled	← "&Save" ← *(corresponding image)* ← *False when simulating, true otherwise.*	The entire schematic is written to a file. Keyboard shortcut: Ctrl+S
btnRun (TToolButton)	Caption ImageIndex Enabled	← "&Run" ← *(corresponding image)* ← *True when simulation is stopped, false otherwise.*	The circuit is simulated. Keyboard shortcut: Spacebar
btnStop (TToolButton)	Caption ImageIndex Enabled	← "&Stop" ← *(corresponding image)* ← *True when simulation is running, false otherwise.*	The simulation is stopped. Keyboard shortcut: Spacebar
btnMouse (TToolButton)	Caption ImageIndex Enabled	← "&Mouse" ← *(corresponding image)* ← *False when simulating, true otherwise.*	Switches to mouse mode, allowing the user to select, drag and delete components and wires). Keyboard shortcut: Ctrl+M
btnWire (TToolButton)	Caption ImageIndex Enabled	← "&Wire" ← *(corresponding image)* ← *False when simulating, true otherwise.*	Switches to wire mode, allowing the user to connect inputs, outputs and gates togethers. Keyboard shortcut: Ctrl+W
btnNOT, btnOR, btnAND, btnNOR, btnNAND, btnXOR, btnXNOR, btnOutputLamp, btnInputButton,	ImageIndex	← *(corresponding image)*	

Name (component type)	Properties	Action
btnInputSwitch (TToolButton)		
lblComponents (TLabel)	Align ← alTop Alignment ← taCenter Font.Style ← [fsBold, fsUnderline]	N/A
sbStatus (TStatusBar)	Align ← alBottom SimplePanel ← False	N/A
ilMenuBarItems (TImageList)	Width ← 36 Height ← 36	N/A
ilComponents (TImageList)	Width ← 178 Height ← 80	N/A

Component Hierarchy

- frmLogicSim
  - pnlSchema
  - tbMenuBar
    - btnNew
    - btnOpen
    - btnSave
    - btnRun
    - btnStop
    - btnMouse
    - btnWire
  - tbComponents
    - btnNOT
    - btnOR
    - btnAND
    - btnNOR
    - btnNAND
    - btnXOR
    - btnXNOR
    - btnOutputLamp
    - btnInputButton
    - btnInputSwitch
  - sbStatus
  - ilMenuBarItems
  - ilComponents

Notes

- Captions beginning with ‘&’ denote accelerator (shortcut) keys. For example, “&New” can be triggered by pressing Alt+n.

File Organisation and Processing

This system will only be required to write and read a single file upon loading or saving a user's schematic. Files will be saved with the extension 'lsch' (logic schematic).

When saving, each instance of node, wire and gate objects will be written directly to the schematic file.

Object Overview

Note: for a NOT gate, InputNode2 will not be used.

Gate
XPosition YPosition Type InputNode1 InputNode2 OutputNode
UpdateOutputValue Move Delete

Node
ID Type State Timestamp
SetState GetState

UIDevice
XPosition YPosition Type NodeID
Move

IMPORTANT ALGORITHMS

ADDING WIRES

When a wire is added to the schematic, the connection will need to be validated. The connection will be accepted only if the two nodes are of opposite type (one input and one output). Any other combination will be rejected. If the IDs of the two nodes are the same, then the wire will not be added as it already exists. If the connection is accepted, the two nodes will be assigned the same ID, allowing them to share the same state.

Pseudo Code

```
if firstNode selected then
      if secondNode selected then
            if firstNode.ID = secondNode.ID then
                  error "Wire already exists."
                  abort
            endif

            if (firstNode.type = "output") and
               (secondNode.type = "output") then
                  error "Cannot connect two outputs together."
                  abort
            endif

            if (firstNode.type = "input") and
               (secondNode.type = "input") then
                  error "Cannot connect two inputs together."
                  abort
            endif

            { No errors found }
            firstNode.ID ← secondNode.ID
            redraw schematic view
      endif
endif
```

REMOVING WIRES

When the user wants to remove a wire from the schematic, each connected node will need to be assigned a unique ID.

Pseudo Code

```
procedure removeWire(wireID)
      numberOfNodes ← number of nodes in schematic

      for i ← 0 to numberOfNodes do
            currentNode ← next node

            if currentNode.ID = wireID then
                  currentNode.ID ← next unique node ID in schematic
            endif
      endfor

      redraw schematic view
end
```

SIMULATION

When simulating, all editing features will be disabled. Every node in the circuit will have its value reset, so that values from the previous simulation do not exist.

Pseudo Code

```
disable editing

{ Reset value (state) of every node in circuit. }
for each node in circuit do
      node.state ← false
      node.timestamp ← 0
endfor

{ List of output nodes from user interaction devices
  (switches/buttons). }
outputNodes ← each output node in schematic

{ Repeatedly update values, but only if outputNodes has changed. }
finished ← false

{ Initially set changed to true, to iterate once and set initial node
  values. }
changed ← true

repeat
      { Update the relevant gates if any of the user interaction devices
        have been changed. }
      if changed then
            { Update all gates related to the node that has changed. }
            for changedNode in outputNodes do
                  tmpNode ← changedNode
                  stop ← false

                  { Create an list of every gate that this node affects,
                    ordered from closest to changedNode to furthest away. }
                  repeat
                        find all gates with same ID as tmpNode
                        if gates found then
                              updateList.add(gates)
                              tmpNode ← output node from gate
                        else
                              { Stop if no gates are found. }
                              stop ← true
                        endif
                  until stop
            endfor

            { Update the output node value of each gate. }
            for each gate in updateList do
                  gate.updateOutputValue
            endfor

            redraw schematic view
      endif
```

```
    { Check to see if the any user interaction devices have
      been changed. }
    if outputNodes changed since last iteration then
        changed ← true
    else
        changed ← false
    endif

    if stop button pressed then
        finished ← true
    endif
until finished

enable editing

--------------------------------------------------------------------------

procedure calculateGateOutputValue(gate)
    case gate.type of
        "NOT" do
            { InputNode2 is not used for a NOT gate. }
            gate.outputNode.value = not gate.InputNode1.value
        end
        "OR" do
            gate.outputNode.value = gate.InputNode1.value
                                    or gate.InputNode2.value
        end
        "AND" do
            gate.outputNode.value = gate.InputNode1.value
                                    and gate.InputNode2.value
        end
        "NOR" do
            gate.outputNode.value = not (gate.InputNode1.value
                                    or gate.InputNode2.value)
        end
        "NAND" do
            gate.outputNode.value = not (gate.InputNode1.value
                                    and gate.InputNode2.value)
        end
        "XOR" do
            gate.outputNode.value = gate.InputNode1.value
                                    xor gate.InputNode2.value
        end
        "XNOR" do
            gate.outputNode.value = not (gate.InputNode1.value
                                    xor gate.InputNode2.value)
        end
    endcase

    gate.outputNode.timestamp ← current time
end
```

Security and Integrity of Data

Security

Since this system will not hold any sensitive information, security is not a concern.

Integrity

Consequences will not be large in the event of data loss. The only data that will be stored are the logic circuit schematic files, to allow saving and loading. No means of protection or backup will be necessary for these files as they are under control of the users.

Overall Test Strategy

Bottom-up testing involves testing algorithms for correct functionality as they are being written, and will be used frequently during the creation of the new system. This testing strategy will mainly be used for aspects such as the click-and-drag functionality for adding logic gates and user interaction devices. It will also be used for testing functionality such as saving or loading schematics, adding or deleting wires and moving or removing components from the circuit.

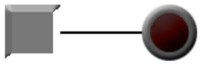
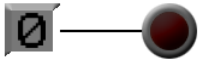
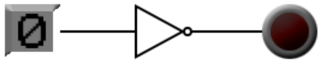
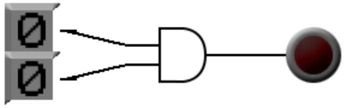
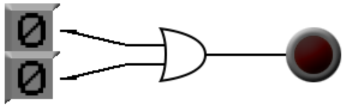
Black box testing will be used as soon as the new system is built. It will be used to ensure that the new system conforms to the specified details desired by the end user. During this phase, various logic circuits will be tested to ensure correct operation of simulations. Aspects such as file saving and loading will also be tested.

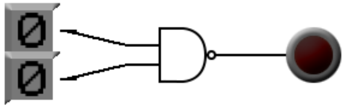
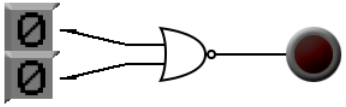
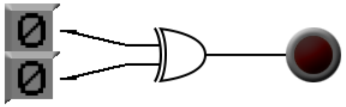
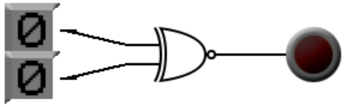
As well as the previously mentioned testing strategies, copies of the system will be distributed to a number of end users to ensure it suitably matches their criteria. This section will primarily focus on the system's ease of use.

System Testing

Test Plan

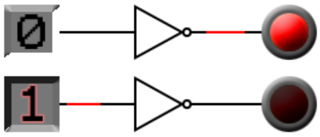
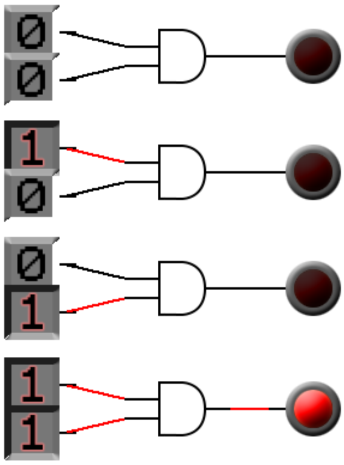
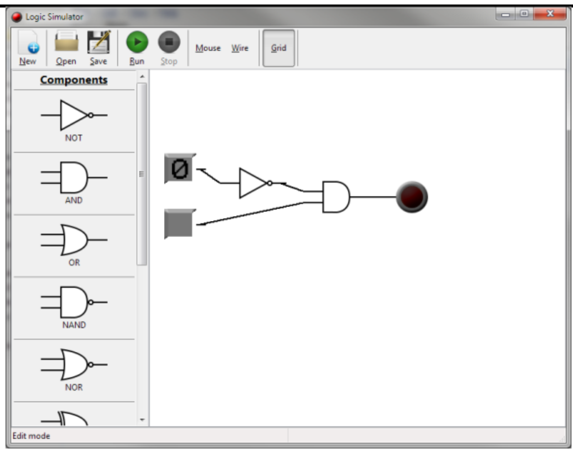
- In each test, wires should change colour according to the current state. (Red = high, black = low).
- Truth tables should be adhered to in each test (where present).

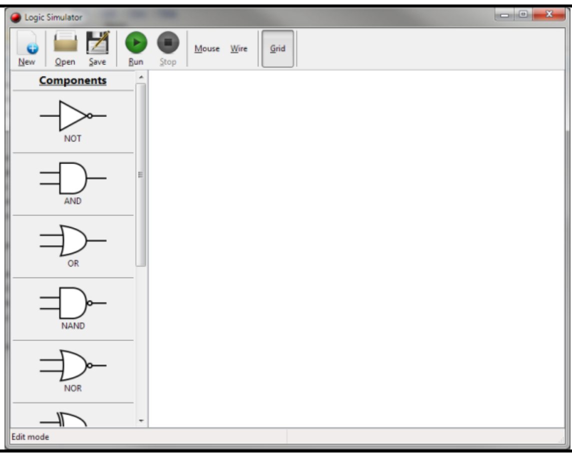
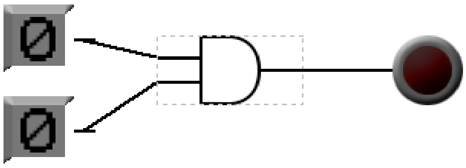
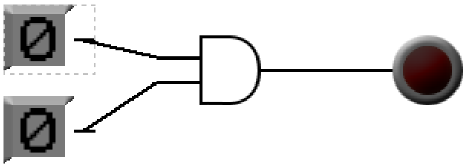
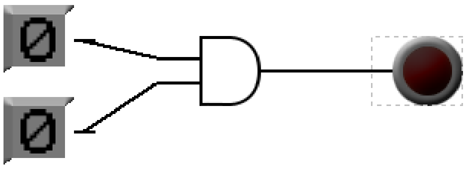
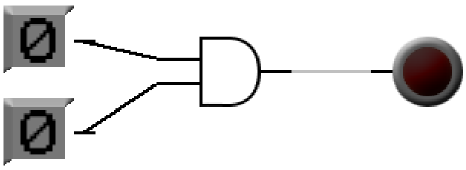
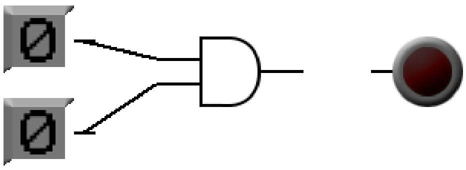
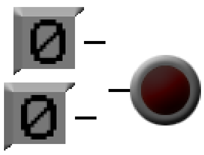
<table>
<tr><th>#</th><th>Description And Setup</th><th>Expected Outcome</th><th>Outcome</th></tr>
<tr><td>1</td><td>Push button and output indicator.
</td><td>Once clicked, turns off after 0.5 seconds. Output indicator should turn on and off accordingly.</td><td>PASSED</td></tr>
<tr><td>2</td><td>Toggle switch and output indicator.
</td><td>Toggle switch should change to opposite state when clicked. Output indicator should turn on and off accordingly.</td><td>PASSED</td></tr>
<tr><td>3</td><td>NOT gate should invert the input.
</td><td>The output indicator should represent the opposite state of the toggle switch.<table><tr><th>A</th><th>Q</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td><td>PASSED</td></tr>
<tr><td>4</td><td>AND gate.
</td><td><table><tr><th>A</th><th>B</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>PASSED</td></tr>
<tr><td>5</td><td>OR gate.
</td><td><table><tr><th>A</th><th>B</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>PASSED</td></tr>
</table>

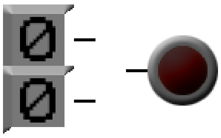
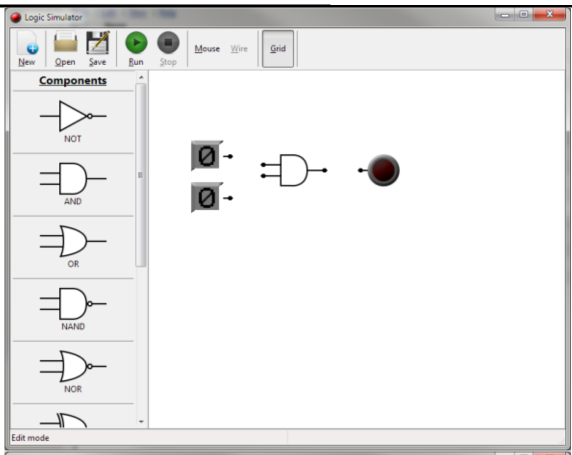
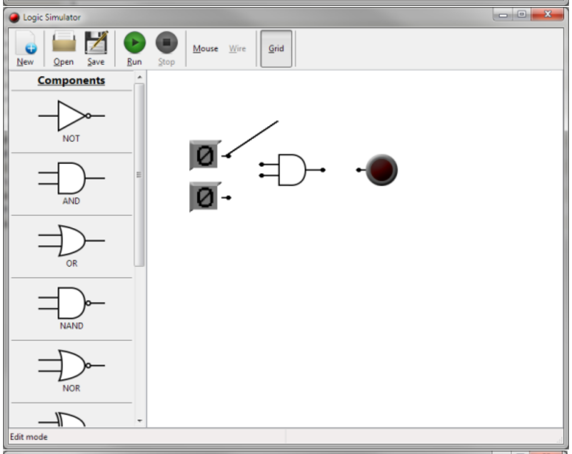
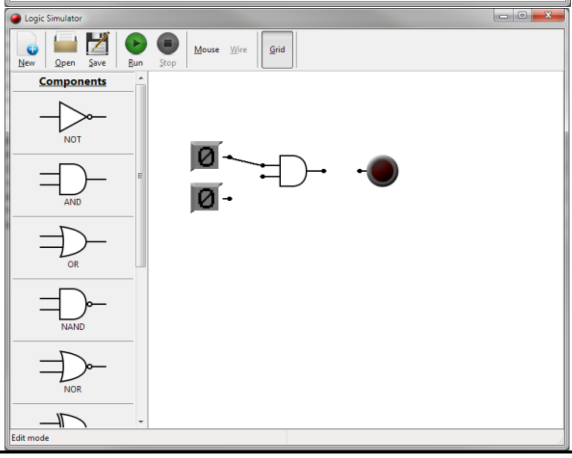
<table>
<tr><th>#</th><th>Description And Setup</th><th>Expected Outcome</th><th>Outcome</th></tr>
<tr><td>6</td><td>NAND gate.
</td><td><table><tr><th>A</th><th>B</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td><td>PASSED</td></tr>
<tr><td>7</td><td>NOR gate.
</td><td><table><tr><th>A</th><th>B</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td><td>PASSED</td></tr>
<tr><td>8</td><td>XOR gate.
</td><td><table><tr><th>A</th><th>B</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td><td>PASSED</td></tr>
<tr><td>9</td><td>XNOR gate.
</td><td><table><tr><th>A</th><th>B</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>PASSED</td></tr>
<tr><td>10</td><td>‘New’ button.</td><td>Clicking this button should exit the simulation and clear the entire schematic.</td><td>PASSED</td></tr>
<tr><td>11</td><td>Saving a schematic.</td><td></td><td>FAILED</td></tr>
<tr><td>12</td><td>Loading a schematic.</td><td></td><td>FAILED</td></tr>
<tr><td>13</td><td>Mouse mode.</td><td>Devices/wires should be selected when clicked. Any currently selected device should be deselected.
Devices should move when dragged. Wires should still be connected.</td><td>PASSED</td></tr>
<tr><td>14</td><td>Removing devices/wires.</td><td>Selected device/wire should be removed (along with connected wires) when the delete key or backspace key is pressed.</td><td>PASSED</td></tr>
</table>

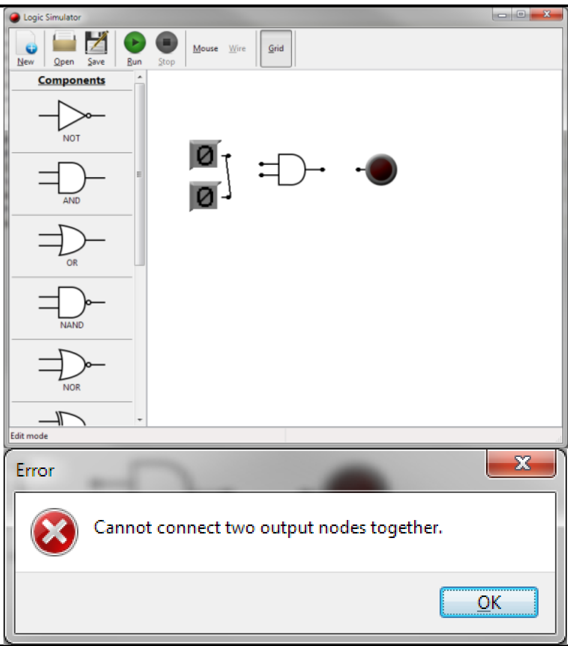
#	Description And Setup	Expected Outcome	Outcome
15	Grid snapping.	Gates should snap to a virtual 20x20 pixel grid when moved with mouse mode.	**PASSED**
16	Wire mode.	Nodes should appear where wires can be connected to gates and other devices. Clicking a node should start drawing a wire. Clicking another node should insert a permanent wire. An error message should be shown if the wire cannot be added (see tests 17-20).	**PASSED**
17	Error when wiring two output nodes together.	Error dialog will be shown with message "Cannot connect two output nodes together."	**PASSED**
18	Error when wiring two input nodes together.	Error dialog will be shown with message "Cannot connect two input nodes together."	**PASSED**
19	Error when wire already exists between two nodes.	Error dialog will be shown with message "Wire already exists."	**PASSED**
20	Error when node is being wired to itself.	Error dialog will be shown with message "Cannot connect a node to itself."	**PASSED**

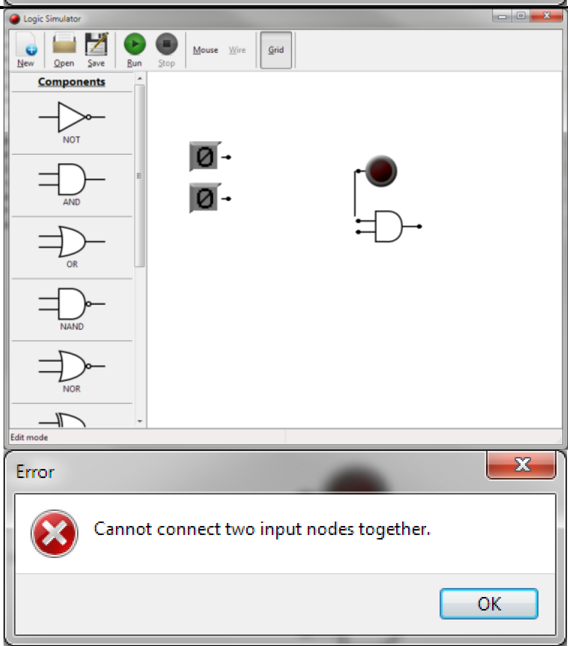
Test Evidence

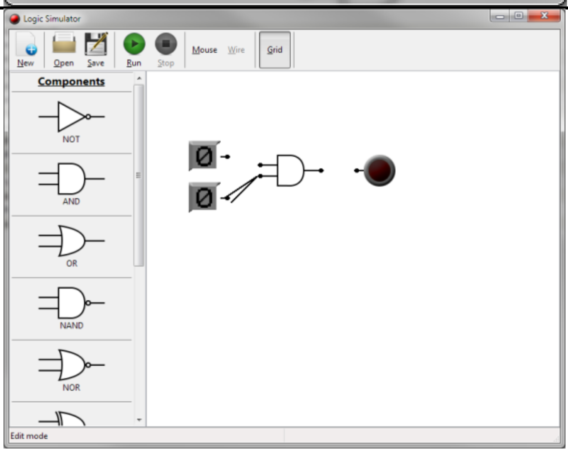
#	Screenshots	Notes
1		Button and output successfully turned off after 0.5 seconds.
2		
3		
4		
5	Substituted corresponding gate and tested (as shown in test 4).	
6		
7		
8		
9		
10		

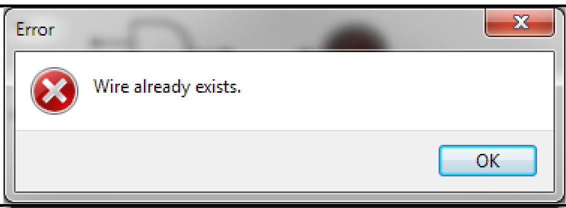
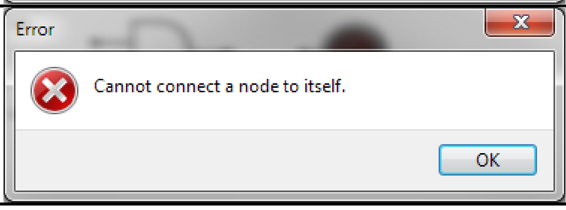
		Schematic cleared completely.
11		
12		
13	   	AND gate clicked. Box drawn to show that it is currently selected. Input switch clicked and selected. AND gate is automatically deselected. Output lamp clicked and selected. Input switch is automatically deselected. Wire clicked and selected. Output lamp is automatically deselected.
14	 	Delete/backspace key pressed with wire selected. Wire is removed from the schematic. Delete/backspace key pressed with AND gate selected. Gate is removed from the schematic, along with connected wires.
15		Devices can be placed anywhere with the grid turned off.

<table>
<tr><td></td><td></td><td>A strict layout is followed with the grid turned on.</td></tr>
<tr><td rowspan="3">16</td><td>
</td><td>Nodes appear in correct positions.</td></tr>
<tr><td>
</td><td>Temporary wire drawn to location of mouse cursor when node is clicked.</td></tr>
<tr><td>
</td><td>Clicking another node creates a permanent wire/connection.</td></tr>
</table>

<table>
<tr><td>17</td><td></td><td>Two output nodes clicked.

Correct error message shown.</td></tr>
<tr><td>18</td><td></td><td>Two input nodes clicked.

Correct error message shown.</td></tr>
<tr><td>19</td><td></td><td>Wire drawn over an existing wire.</td></tr>
</table>

		Correct error message shown.
20		Node clicked twice. Correct error message shown.

MAINTENANCE

FILE LIST

File	Description
logic_simulator.lpr	Lazarus project file.
logic_simulator_form.lfm	Main form design.
logic_simulator_form.pas	Main form implementation.
logic_simulator_common.pas	Unit containing common functionality for editing.
logic_simulator_simulation.pas	Unit containing functionality for simulation.
logic_simulator_globals.pas	Unit containing global variables and constants.

Annotated Program Listing

File: logic_simulator_common.pas

```
unit logic_simulator_common;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, Menus,
  ExtCtrls, LCLType, TypInfo, DateUtils;

type
  TToolType = (TOOL_MOUSE, TOOL_WIRE);

  TGateType = (GATE_NOT, GATE_OR, GATE_AND, GATE_NOR, GATE_NAND, GATE_XOR,
               GATE_XNOR);

  TOutputDeviceType = (OD_INDICATOR);

  TInputDeviceType = (ID_SWITCH, ID_BUTTON);

  TSchematicDevice = class;

  TNodeType = (NODE_UNUSED, NODE_INPUT, NODE_OUTPUT);

  { TNode }
  TNode = record
    NodeType: TNodeType;
    PositionRelativeToDevice: TPoint;
    State: Boolean;
    LastUpdate: Integer;
    ConnectedWires: TList;
    ParentSchematicDevice: TSchematicDevice;
  end;

  { TNodePtr }
  TNodePtr = ^TNode;

  { TSchematicDevice }
  TSchematicDevice = class(TCustomImage)
  private
    Selected: Boolean;
  public
    constructor Create(AOwner: TComponent); override;
    procedure DrawNode(ANode: TNode);
    procedure Reset; virtual;
    procedure UpdateImage; virtual;
    procedure UpdateDevice; virtual;

    { Event handlers }
    procedure OnPaintHandler(Sender: TObject); virtual;
    procedure OnMouseDownHandler(Sender: TObject; {%H-}Button: TMouseButton;
                                 {%H-}Shift: TShiftState; {%H-}X: longint;
                                 {%H-}Y: longint); virtual;
    procedure OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
                                 {%H-}X: longint; {%H-}Y: longint); virtual;
  end;

```

```
{ TWire }
TWire = class(TCustomImage)
private
  Selected: Boolean;
  PStartNode: ^TNode;
  PEndNode: ^TNode;
  EndPoint: TPoint;
  FlippedX: Boolean;
  FlippedY: Boolean;
public
  constructor Create(AOwner: TComponent); override;
  procedure SetStart(var ANode: TNode);
  procedure SetEnd(const ANode: TNode); overload;
  procedure SetEnd(const APoint: TPoint); overload;
  procedure SetEnd(const X: Integer; const Y: Integer); overload;
  procedure CalculateNewSize;
  procedure AffectInputNode;

  { Event handlers }
  procedure OnPaintHandler(Sender: TObject);
  procedure OnMouseDownHandler(Sender: TObject; {%H-}Button: TMouseButton;
                               {%H-}Shift: TShiftState; {%H-}X: longint;
                               {%H-}Y: longint);
  procedure OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
                               {%H-}X: longint; {%H-}Y: longint);
end;

{ TLogicGate }
TLogicGate = class(TSchematicDevice)
private
  GateType: TGateType;
  InputNode1: TNode;
  InputNode2: TNode;
  OutputNode: TNode;
public
  constructor Create(AOwner: TComponent; NewGateType: TGateType); reintroduce;
  destructor Destroy; override;
  procedure Reset; override;
  procedure UpdateDevice; override;
  function GetOutputNodeState: Boolean;

  { Event handlers }
  procedure OnPaintHandler(Sender: TObject); override;
  procedure OnMouseDownHandler(Sender: TObject; Button: TMouseButton;
                               Shift: TShiftState; X: longint; Y: longint);
                               override;
  procedure OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
                               {%H-}X: longint; {%H-}Y: longint); override;
end;

{ TOutputDevice }
TOutputDevice = class(TSchematicDevice)
private
  OutputDeviceType: TOutputDeviceType;
  ImageIDOn: Integer;
  ImageIDOff: Integer;
  InputNode: TNode;
public
  constructor Create(AOwner: TComponent;
                     NewOutputDeviceType: TOutputDeviceType); reintroduce;
  destructor Destroy; override;
```

```
    procedure Reset; override;
    procedure UpdateImage; override;
    procedure UpdateDevice; override;

    { Event handlers }
    procedure OnPaintHandler(Sender: TObject); override;
    procedure OnMouseDownHandler(Sender: TObject; Button: TMouseButton;
                               Shift: TShiftState; X: longint; Y: longint);
                               override;
    procedure OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
                               {%H-}X: longint; {%H-}Y: longint); override;
  end;

  { TInputDevice }
  TInputDevice = class(TSchematicDevice)
  private
    InputDeviceType: TInputDeviceType;
    OutputNode: TNode;
    ImageIDOn: Integer;
    ImageIDOff: Integer;
    ButtonPressedTime: TDateTime;
  public
    constructor Create(AOwner: TComponent; NewInputDeviceType: TInputDeviceType);
                       reintroduce;
    destructor Destroy; override;
    procedure Reset; override;
    procedure UpdateDevice; override;
    procedure UpdateImage; override;
    function GetInputDeviceType: TInputDeviceType;

    { Event handlers }
    procedure OnPaintHandler(Sender: TObject); override;
    procedure OnMouseDownHandler(Sender: TObject; Button: TMouseButton;
                               Shift: TShiftState; X: longint; Y: longint);
                               override;
    procedure OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
                               {%H-}X: longint; {%H-}Y: longint); override;
  end;


procedure InitialiseProgram;
procedure UninitialiseProgram;
procedure ClearSchematic;
procedure SetToolType(const NewToolType: TToolType);
function WriteSchematicToFile(const Location: string): Boolean; experimental;
function LoadSchematicFromFile(const Location: string): Boolean; experimental;
procedure DisplayError(const Text: string);
procedure DisplayWarning(const Text: string);
procedure UpdateStatusBar(const Text: string);
procedure AddSchematicDevice(const GateType: TGateType; X, Y: Integer); overload;
procedure AddSchematicDevice(const InputDeviceType: TInputDeviceType;
                             X, Y: Integer); overload;
procedure AddSchematicDevice(const OutputDeviceType: TOutputDeviceType;
                             X, Y: Integer); overload;
procedure RemoveCurrentlySelected;
procedure DeselectCurrentlySelected;
procedure RepaintAllSchematicDevices;
function AskToSave: Integer;
function AskToOverwrite: Integer;
procedure UpdateTmpWire;
procedure RemoveTmpWire;
```

```
procedure RedrawAllWires;
procedure UpdateNodeState(PANode: TNodePtr; const NewState: Boolean);


implementation

uses
  logic_simulator_form, logic_simulator_globals, logic_simulator_simulation;

type
  { TSaveData }
  TSaveData = record
    ObjectType: String[20];
    ObjectSpecificType: Integer;
    Top: Integer;
    Left: Integer;
  end;


var
  CurrentTool: TToolType;
  CurrentlySelectedDevice: TSchematicDevice;
  CurrentlySelectedWire: TWire;
  PLastNodeClicked: ^TNode;
  TmpWire: TWire;


// InitialiseProgram
//
// Allocates memory to lists, sets up basic user interface variables and enters
// edit mode.
procedure InitialiseProgram;
begin
  GatesList := TList.Create;
  InputDevicesList := TList.Create;
  OutputDevicesList := TList.Create;
  WiresList := TList.Create;

  frmLogicSim.SimulationTimer.Enabled := False;
  frmLogicSim.SimulationTimer.Interval := SIMULATION_INTERVAL;

  // There is no device currently selected/wire, as none exist!
  CurrentlySelectedDevice := Nil;
  CurrentlySelectedWire := Nil;
  PLastNodeClicked := Nil;
  TmpWire := Nil;
  SetToolType(TOOL_MOUSE);
  UpdateStatusBar('Edit mode');
end;

// UnitialiseProgram
//
// Frees any allocated memory back to the OS.
procedure UninitialiseProgram;
begin
  StopSimulation;
  ClearSchematic;

  GatesList.Free;
  InputDevicesList.Free;
  OutputDevicesList.Free;
```

```
  WiresList.Free;
end;

// ClearSchematic
//
// Removes any schematic devices from the schematic, and frees any schematic
// devices.
procedure ClearSchematic;
var
  Tmp: Pointer;
begin
  DeselectCurrentlySelected;

  for Tmp in GatesList do
  begin
    TLogicGate(Tmp).Free;
  end;
  GatesList.Clear;

  for Tmp in InputDevicesList do
  begin
    TInputDevice(Tmp).Free;
  end;
  InputDevicesList.Clear;

  for Tmp in OutputDevicesList do
  begin
    TOutputDevice(Tmp).Free;
  end;
  OutputDevicesList.Clear;

  for Tmp in WiresList do
  begin
    TWire(Tmp).Free;
  end;
  WiresList.Clear;
end;

// SetToolType
procedure SetToolType(const NewToolType: TToolType);
begin
  CurrentTool := NewToolType;
  DeselectCurrentlySelected;
  RemoveTmpWire;
  RepaintAllSchematicDevices;

  case NewToolType of
    TOOL_WIRE:
    begin
      frmLogicSim.btnWire.Enabled := False;
      frmLogicSim.btnMouse.Enabled := True;
    end;
    TOOL_MOUSE:
    begin
      frmLogicSim.btnWire.Enabled := True;
      frmLogicSim.btnMouse.Enabled := False;
    end;
  end;
end;

// WriteSchematicToFile
```

```
function WriteSchematicToFile(const Location: string): Boolean;
var
  SaveFile: file of TSaveData;
  TmpSaveRecord: TSaveData;
  Tmp: Pointer;
begin
  Result := True;
  AssignFile(SaveFile, Location);

  try
    // Open the file for writing.
    Rewrite(SaveFile);

    for Tmp in GatesList do
    begin
      TmpSaveRecord.ObjectType := 'TLogicGate';
      TmpSaveRecord.ObjectSpecificType := Ord(TLogicGate(Tmp).GateType);
      TmpSaveRecord.Top := TLogicGate(Tmp).Top;
      TmpSaveRecord.Left := TLogicGate(Tmp).Left;
      Write(SaveFile, TmpSaveRecord);
    end;

    for Tmp in InputDevicesList do
    begin
      TmpSaveRecord.ObjectType := 'TInputDevice';
      TmpSaveRecord.ObjectSpecificType := Ord(TInputDevice(Tmp).InputDeviceType);
      TmpSaveRecord.Top := TInputDevice(Tmp).Top;
      TmpSaveRecord.Left := TInputDevice(Tmp).Left;
      Write(SaveFile, TmpSaveRecord);
    end;

    for Tmp in OutputDevicesList do
    begin
      TmpSaveRecord.ObjectType := 'TOutputDevice';
      TmpSaveRecord.ObjectSpecificType :=
        Ord(TOutputDevice(Tmp).OutputDeviceType);
      TmpSaveRecord.Top := TOutputDevice(Tmp).Top;
      TmpSaveRecord.Left := TOutputDevice(Tmp).Left;
      Write(SaveFile, TmpSaveRecord);
    end;

    for Tmp in WiresList do
    begin
      TmpSaveRecord.ObjectType := 'TWire';
      TmpSaveRecord.ObjectSpecificType := -1;
    end;
  except
    on E: EInOutError do
    begin
      Result := False;
    end;
  end;

  CloseFile(SaveFile);
end;

// LoadSchematicFromFile
function LoadSchematicFromFile(const Location: string): Boolean;
var
  LoadFile: file of TSaveData;
  TmpLoadRecord: TSaveData;
```

```
begin
  Result := True;

  AssignFile(LoadFile, Location);

  try
    // Open the file for reading.
    Reset(LoadFile);

    repeat
      Read(LoadFile, TmpLoadRecord);

      case TmpLoadRecord.ObjectType of
        'TLogicGate':
        begin
          AddSchematicDevice(TGateType(TmpLoadRecord.ObjectSpecificType),
                             TmpLoadRecord.Left, TmpLoadRecord.Top);
        end;
        'TInputDevice':
        begin
          AddSchematicDevice(TInputDeviceType(TmpLoadRecord.ObjectSpecificType),
                             TmpLoadRecord.Left, TmpLoadRecord.Top);
        end;
        'TOutputDevice':
        begin
          AddSchematicDevice(TOutputDeviceType(TmpLoadRecord.ObjectSpecificType),
          TmpLoadRecord.Left, TmpLoadRecord.Top);
        end;
      end;
    until EOF(LoadFile);
  except
    on E: EInOutError do
    begin
      Result := False;
    end;
  end;

  CloseFile(LoadFile);
end;

// DisplayError
//
// Displays Text in a pop-up message with an error icon.
procedure DisplayError(const Text: string);
begin
  MessageDlg(Text, mtError, [mbOK], 0);
end;

// DisplayWarning
//
// Displays Text in a pop-up message with a warning icon.
procedure DisplayWarning(const Text: string);
begin
  MessageDlg(Text, mtWarning, [mbOK], 0);
end;

// UpdateStatusBar
//
// Updates the status bar with Text.
procedure UpdateStatusBar(const Text: string);
begin
```

```
  frmLogicSim.sbStatus.Panels[0].Text := Text;
end;

// AddSchematicDevice
//
// Overloaded procedure that will add a specific type of device to the schematic
// at the specified position. Adds the schematic device to the corresponding
// list (GatesList, InputDevicesList, OutputDevicesList).
procedure AddSchematicDevice(const GateType: TGateType; X, Y: Integer); overload;
var
  Tmp: TLogicGate;
begin
  Tmp := TLogicGate.Create(frmLogicSim.pnlSchema, GateType);
  Tmp.Left := X;
  Tmp.Top := Y;
  GatesList.Add(Tmp);
end;

procedure AddSchematicDevice(const InputDeviceType: TInputDeviceType;
                             X, Y: Integer); overload;
var
  Tmp: TInputDevice;
begin
  Tmp := TInputDevice.Create(frmLogicSim.pnlSchema, InputDeviceType);
  Tmp.Left := X;
  Tmp.Top := Y;
  InputDevicesList.Add(tmp);
end;

procedure AddSchematicDevice(const OutputDeviceType: TOutputDeviceType;
                             X, Y: Integer); overload;
var
  Tmp: TOutputDevice;
begin
  Tmp := TOutputDevice.Create(frmLogicSim.pnlSchema, OutputDeviceType);
  Tmp.Left := X;
  Tmp.Top := Y;
  OutputDevicesList.Add(Tmp);
end;
// END AddSchematicDevice (overloaded)

// RemoveWire
//
// Removes AWire from the schematic, updates ConnectedWires of the affected
// nodes and frees the AWire's memory. Also removes the wire from WiresList.
procedure RemoveWire(AWire: TWire);
begin
  AWire.PStartNode^.ConnectedWires.Remove(AWire);
  AWire.PEndNode^.ConnectedWires.Remove(AWire);

  WiresList.Remove(AWire);
  AWire.Free;
end;

// RemoveSchematicDevice
//
// Removes ADevice from the schematic, along with any connected wires. Also
// removes ADevice from the corresponding list (GatesList, InputDevicesList,
// OutputDevicesList) and frees memory used by ADevice.
procedure RemoveSchematicDevice(ADevice: TSchematicDevice);
begin
```

```
  case ADevice.ClassName of
    'TLogicGate':
    begin
      GatesList.Remove(ADevice);

      while TLogicGate(ADevice).OutputNode.ConnectedWires.Count <> 0 do
      begin
        RemoveWire(TWire(TLogicGate(ADevice).OutputNode.ConnectedWires[0]));
      end;

      while TLogicGate(ADevice).InputNode1.ConnectedWires.Count <> 0 do
      begin
        RemoveWire(TWire(TLogicGate(ADevice).InputNode1.ConnectedWires[0]));
      end;

      while TLogicGate(ADevice).InputNode2.ConnectedWires.Count <> 0 do
      begin
        RemoveWire(TWire(TLogicGate(ADevice).InputNode2.ConnectedWires[0]));
      end;
    end;
    'TInputDevice':
    begin
      InputDevicesList.Remove(ADevice);

      while TInputDevice(ADevice).OutputNode.ConnectedWires.Count <> 0 do
      begin
        RemoveWire(TWire(TInputDevice(ADevice).OutputNode.ConnectedWires[0]));
      end;
    end;
    'TOutputDevice':
    begin
      OutputDevicesList.Remove(ADevice);

      while TOutputDevice(ADevice).InputNode.ConnectedWires.Count <> 0 do
      begin
        RemoveWire(TWire(TOutputDevice(ADevice).InputNode.ConnectedWires[0]));
      end;
    end;
  end;
  ADevice.Free;
end;

// SelectDevice
//
// Overloaded procedure that will mark AGate or AWire as the
// CurrentlySelectedDevice. Sets the Selected property of the device to TRUE and
// causes the device to be redrawn.
procedure SelectDevice(AGate: TSchematicDevice); overload;
begin
  DeselectCurrentlySelected;
  CurrentlySelectedDevice := AGate;
  CurrentlySelectedDevice.Selected := True;
  CurrentlySelectedDevice.Invalidate;
end;

procedure SelectDevice(AWire: TWire); overload;
begin
  DeselectCurrentlySelected;
  CurrentlySelectedWire := AWire;
  CurrentlySelectedWire.Selected := True;
  CurrentlySelectedWire.Invalidate;
```

```
end;
// END SelectDevice (overloaded)

// RemoveCurrentlySelected
//
// Removes the currently selected device or wire from the schematic.
procedure RemoveCurrentlySelected;
var
  TmpDevice: TSchematicDevice;
  TmpWire: TWire;
begin
  if CurrentlySelectedDevice <> Nil then
  begin
    TmpDevice := CurrentlySelectedDevice;
    DeselectCurrentlySelected;
    RemoveSchematicDevice(TmpDevice);
  end;

  if CurrentlySelectedWire <> Nil then
  begin
    TmpWire := CurrentlySelectedWire;
    DeselectCurrentlySelected;
    RemoveWire(TmpWire);
  end;
end;

// DeselectCurrentlySelected
//
// Deselects the currently selected device or wire.
procedure DeselectCurrentlySelected;
begin
  if CurrentlySelectedDevice <> Nil then
  begin
    CurrentlySelectedDevice.Selected := False;
    CurrentlySelectedDevice.Invalidate;
    CurrentlySelectedDevice := Nil;
  end;

  if CurrentlySelectedWire <> Nil then
  begin
    CurrentlySelectedWire.Selected := False;
    CurrentlySelectedWire.Invalidate;
    CurrentlySelectedWire := Nil;
  end;
end;

// RepaintAllSchematicDevices
//
// Calls Invalidate on every device in the schematic, causing them to be
// redrawn.
procedure RepaintAllSchematicDevices;
var
  Tmp: Pointer;
begin
  for Tmp in GatesList do
  begin
    TLogicGate(Tmp).Invalidate;
  end;

  for Tmp in InputDevicesList do
  begin
```

```
    TInputDevice(Tmp).Invalidate;
  end;

  for Tmp in OutputDevicesList do
  begin
    TOutputDevice(Tmp).Invalidate;
  end;
end;

// AskToSave
//
// Prompts the user with a message dialog box asking if they want to save the
// schematic to a file, with yes/no/cancel as response options.
function AskToSave: integer;
begin
  Result := MessageDlg('Would you like to save the current schematic?',
    mtConfirmation, mbYesNoCancel, 0);
end;

// AskToOverwrite
//
// Prompts the user with a message dialog box asking if they want to overwrite
// an existing file, with yes/no/cancel as response options.
function AskToOverwrite: Integer;
begin
  Result := MessageDlg('File exists. Overwrite?', mtConfirmation,
    mbYesNoCancel, 0);
end;

// UpdateTmpWire
//
// Causes the temporary wire to be redrawn. TmpWire connects the last node
// clicked to the current mouse position.
procedure UpdateTmpWire;
begin
  if (ApplicationState = EDITING) and (CurrentTool = TOOL_WIRE) and
      (PLastNodeClicked <> Nil) then
  begin
    // Create an instance of TWire if it doesn't already exist.
    if TmpWire = Nil then
    begin
      TmpWire := TWire.Create(frmLogicSim.pnlSchema);
      // Set the start position of TmpWire.
      TmpWire.SetStart(PLastNodeClicked^);
    end;
    // Set the end position of TmpWire as the cursor position.
    TmpWire.SetEnd(frmLogicSim.pnlSchema.ScreenToClient(Mouse.CursorPos));

    TmpWire.Invalidate;
  end
  else
  begin
    // Destroy the old TmpWire if it exists.
    if TmpWire <> Nil then
    begin
      TmpWire.Free;
      TmpWire := Nil;
    end;
  end;
end;
```

```
// RemoveTmpWire
procedure RemoveTmpWire;
begin
  PLastNodeClicked := Nil;
  UpdateTmpWire;
end;

// RedrawAllWires
//
// Causes all wires to recalculate their size and redraws them.
procedure RedrawAllWires;
var
  Tmp: Pointer;
begin
  for Tmp in WiresList do
  begin
    TWire(Tmp).CalculateNewSize;
    TWire(Tmp).Invalidate;
  end;
end;

// UpdateNodeState
//
// Updates the state of a node and sets LastUpdate to the current time.
procedure UpdateNodeState(PANode: TNodePtr; const NewState: Boolean);
begin
  if PANode^.State <> NewState then
  begin
    PANode^.State :=  NewState;
    PANode^.LastUpdate := Round(Now * 24 * 60 * 60 * 1000);
  end;
end;

// InitialiseNewNode
//
// Sets up basic initial values for a new, uninitialised node, and allocate
// memory for the ConnectedWires list.
procedure InitialiseNewNode(var ANode: TNode; const Parent: TSchematicDevice;
                            const NewNodeType: TNodeType; const X: Integer;
                            const Y: Integer);
begin
  ANode.ParentSchematicDevice := Parent;

  ANode.PositionRelativeToDevice.x := X;
  ANode.PositionRelativeToDevice.y := Y;

  ANode.NodeType := NewNodeType;
  ANode.State := False;
  ANode.LastUpdate := 0;

  ANode.ConnectedWires := TList.Create;
end;

// UnitialiseNode
// Free the memory used for ConnectedWires list.
procedure UninitialiseNode(var ANode: TNode);
begin
  ANode.ConnectedWires.Free;
end;

// IsMouseInNodeArea
```

```
//
// Returns TRUE if the mouse is currently sitting over a node. Otherwise,
// returns FALSE.
function IsMouseInNodeArea(const MouseX, MouseY: Integer;
                           const ANode: TNode): Boolean;
var
  TopLeftX, TopLeftY, BottomRightX, BottomRightY: Integer;
begin
  TopLeftX := ANode.PositionRelativeToDevice.x;// - (NODE_WIDTH div 2);
  TopLeftY := ANode.PositionRelativeToDevice.y - (NODE_HEIGHT div 2);
  BottomRightX := TopLeftX + NODE_WIDTH;
  BottomRightY := TopLeftY + NODE_HEIGHT;

  if (MouseX >= TopLeftX) and (MouseX <= BottomRightX) and
     (MouseY >= TopLeftY) and (MouseY <= BottomRightY) then
  begin
    Result := True;
  end
  else
  begin
    Result := False;
  end;
end;

// NodeClicked
//
// Processes the event of a node being clicked by the user. If a node has
// already been clicked, validation will be performed and the two nodes will be
// connected with a wire if passed. Useful error messages are displayed
// if validation fails.
procedure NodeClicked(PANode: TNodePtr);
var
  InvalidConnection: Boolean;
  TmpNewWire: TWire;
  Tmp: Pointer;
begin
  InvalidConnection := False;

  if PLastNodeClicked = Nil then
  begin
    // The first node has been clicked.
    PLastNodeClicked := PANode;
  end
  else
  begin
    // The second node has been clicked.

    // Check to see if the two nodes are the same.
    if PLastNodeClicked = PANode then
    begin
      DisplayError('Cannot connect a node to itself.');
      InvalidConnection := True;
    end
    else
    begin
      // Check to see if two input nodes are trying to be connected together.
      if (PANode^.NodeType = NODE_INPUT) and
         (PLastNodeClicked^.NodeType = NODE_INPUT) then
      begin
        DisplayError('Cannot connect two input nodes together.');
        InvalidConnection := True;
```

```
      end
      else
      begin
        // Check to see if two output nodes are trying to be connected together.
        if (PANode^.NodeType = NODE_OUTPUT) and
           (PLastNodeClicked^.NodeType = NODE_OUTPUT) then
        begin
          DisplayError('Cannot connect two output nodes together.');
          InvalidConnection := True;
        end;
      end;
    end;

    // Check to see if more than one output can't connect to the same input.
    for Tmp in WiresList do
    begin
      if (TWire(Tmp).PEndNode = PANode) then
      begin
        DisplayError('Only one wire can be connected to a single input.');
        InvalidConnection := True;
      end;
    end;

    // Check to see if the two nodes are already connected.
    for Tmp in WiresList do
    begin
      if ((TWire(Tmp).PStartNode = PANode) and
          (TWire(Tmp).PEndNode = PLastNodeClicked)) or
         ((TWire(Tmp).PStartNode = PLastNodeClicked) and
          (TWire(Tmp).PEndNode = PANode)) then
      begin
        DisplayError('Wire already exists.');
        InvalidConnection := True;
      end;
    end;

    if not InvalidConnection then
    begin
      // Connect the two nodes, since no error has been found.
      TmpNewWire := TWire.Create(frmLogicSim.pnlSchema);
      TmpNewWire.SetStart(PLastNodeClicked^);
      TmpNewWire.SetEnd(PANode^);

      // Add TmpNewWire to ConnectedWires list of both nodes.
      PANode^.ConnectedWires.Add(TmpNewWire);
      PLastNodeClicked^.ConnectedWires.Add(TmpNewWire);

      WiresList.Add(TmpNewWire);

      RemoveTmpWire;
    end;
  end;
end;

// TWire
// Create
constructor TWire.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Parent := frmLogicSim.pnlSchema;

```

```
  Selected := False;
  PStartNode := Nil;
  PEndNode := Nil;

  // Should be able to see through the bounding box of a wire.
  Transparent := True;
  Picture.Bitmap.Transparent := True;
  Picture.Bitmap.TransparentColor := clFuchsia;
  Picture.Bitmap.TransparentMode := tmFixed;

  // Assign event handlers.
  OnPaint := @OnPaintHandler;
  OnMouseDown := @OnMouseDownHandler;
  OnMouseMove := @OnMouseMoveHandler;
end;

// SetStart
procedure TWire.SetStart(var ANode: TNode);
begin
  PStartNode := @ANode;
  CalculateNewSize;
end;

// SetEnd
//
// Overloaded function that can take either a TNode, TPoint or separate X/Y
// values (for setting the end position to the cursor position).
procedure TWire.SetEnd(const ANode: TNode);
begin
  PEndNode := @ANode;
  CalculateNewSize;
end;

procedure TWire.SetEnd(const APoint: TPoint);
begin
  PEndNode := Nil;
  EndPoint.x := APoint.x;
  EndPoint.y := APoint.y;
  CalculateNewSize;
end;

procedure TWire.SetEnd(const X: Integer; const Y: Integer);
begin
  PEndNode := Nil;
  EndPoint.x := X;
  EndPoint.y := Y;
  CalculateNewSize;
end;
// END SetEnd (overloaded)

// CalculateNewSize
//
// Calculates the new canvas size for the wire, based on the distance between
// the start node and end node.
procedure TWire.CalculateNewSize;
var
  StartCoord, EndCoord: TPoint;
begin
  StartCoord.x := PStartNode^.ParentSchematicDevice.Left +
                  PStartNode^.PositionRelativeToDevice.x;
  StartCoord.y := PStartNode^.ParentSchematicDevice.Top +
```

```
                     PStartNode^.PositionRelativeToDevice.y;

  if PEndNode = Nil then
  begin
    // Use EndPoint.
    EndCoord := EndPoint;
  end
  else
  begin
    // Use PEndNode.
    EndCoord.x := PEndNode^.ParentSchematicDevice.Left +
                  PEndNode^.PositionRelativeToDevice.x;
    EndCoord.y := PEndNode^.ParentSchematicDevice.Top +
                  PEndNode^.PositionRelativeToDevice.y;
  end;

  Width := abs(StartCoord.x - EndCoord.x);;
  Height := abs(StartCoord.y - EndCoord.y);;

  Top := StartCoord.y;
  Left := StartCoord.x;

  FlippedX := False;
  if EndCoord.x < StartCoord.x then
  begin
    // EndCoord is to left of StartCoord.
    Left := Left - Width;
    FlippedX := True;
  end;

  FlippedY := False;
  if EndCoord.y < StartCoord.y then
  begin
    // EndCoord is above StartCoord.
    Top := Top - Height;
    FlippedY := True;
  end;

  // Adjust canvas size if thinner than the wire's thickness.
  if Height < WIRE_THICKNESS then
  begin
    Height := WIRE_THICKNESS;
    Top := Top - (WIRE_THICKNESS div 2);
  end;
  if Width < WIRE_THICKNESS then
  begin
    Width := WIRE_THICKNESS;
    Left := Left - (WIRE_THICKNESS div 2);
  end;
end;

// AffectInputNode
//
// Updates the connected input node with the signal from the connected output
// node.
procedure TWire.AffectInputNode;
begin
  if PStartNode^.NodeType = NODE_INPUT then
  begin
    UpdateNodeState(PStartNode, PEndNode^.State);
  end
```

```
  else
  begin
    UpdateNodeState(PEndNode, PStartNode^.State);
  end;
end;

// OnPaintHandler
//
// Repaints the wire inside its canvas with the correct orientation based on
// FlippedX and FlippedY. Also sets the colour of the wire depending on the
// state of the connected output node.
procedure TWire.OnPaintHandler(Sender: TObject);
var
  // Points relative to TWire's canvas.
  StartPointRel, EndPointRel: TPoint;
  LineColour: TColor;
begin
  if PStartNode^.NodeType = NODE_OUTPUT then
  begin
    if PStartNode^.State = True then
    begin
      LineColour := WIRE_HIGH_COLOUR;
    end
    else
    begin
      LineColour := WIRE_LOW_COLOUR;
    end;
  end
  else
  begin
    if PStartNode^.State = True then
    begin
      LineColour := WIRE_HIGH_COLOUR;
    end
    else
    begin
      LineColour := WIRE_LOW_COLOUR;
    end;
  end;

  if (not FlippedY) and FlippedX then
  begin
    StartPointRel.x := Width-1;
    StartPointRel.y := 0;
    EndPointRel.x := 0;
    EndPointRel.y := Height-1;
  end;

  if FlippedY and (not FlippedX) then
  begin
    StartPointRel.x := 0;
    StartPointRel.y := Height-1;
    EndPointRel.x := Width-1;
    EndPointRel.y := 0;
  end;

  if FlippedX and FlippedY then
  begin
    StartPointRel.x := Width-1;
    StartPointRel.y := Height-1;
    EndPointRel.x := 0;
```

```
    EndPointRel.y := 0;
  end;

  if (not FlippedX) and (not FlippedY) then
  begin
    StartPointRel.x := 0;
    StartPointRel.y := 0;
    EndPointRel.x := Width-1;
    EndPointRel.y := Height-1;
  end;

  with Canvas do
  begin
    // Draw the wire.
    Pen.Width := WIRE_THICKNESS;

    if Selected then
    begin
      Pen.Color := WIRE_SELECTED_COLOUR;
    end
    else
    begin
      Pen.Color := LineColour;
    end;

    Line(StartPointRel, EndPointRel);
  end;
end;

// OnMouseDownHandler
procedure TWire.OnMouseDownHandler(Sender: TObject; Button: TMouseButton;
                                   Shift: TShiftState; X: longint; Y: longint);
begin
  if ApplicationState = EDITING then
  begin
    SelectDevice(self);
  end;
end;

// OnMouseMoveHandler
procedure TWire.OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
                                   X: longint; Y: longint);
begin
  // Pass the MouseMove event to pnlSchema, so wires are redrawn correctly.
  frmLogicSim.pnlSchemaMouseMove(Sender, Shift, X, Y);
end;
// END TWire

// TInputDevice
// Create
constructor TInputDevice.Create(AOwner: TComponent;
                                NewInputDeviceType: TInputDeviceType);
begin
  inherited Create(AOwner);
  InputDeviceType := NewInputDeviceType;

  InitialiseNewNode(OutputNode, self, NODE_OUTPUT, 60 - NODE_WIDTH, 23);

  SetBounds(10, 10, 60, 46);

  case InputDeviceType of
```

```
    ID_BUTTON: begin
                 ImageIDOff := BUTTON_RELEASED_IMG_ID;
                 ImageIDOn := BUTTON_PRESSED_IMG_ID;
               end;
    ID_SWITCH: begin
                 ImageIDOff := SWITCH_RELEASED_IMG_ID;
                 ImageIDOn := SWITCH_PRESSED_IMG_ID;
               end;
  end;

  ButtonPressedTime := 0;
  UpdateImage;
end;

// Destroy
destructor TInputDevice.Destroy;
begin
  UninitialiseNode(OutputNode);

  inherited Destroy;
end;

// Reset
procedure TInputDevice.Reset;
begin
  UpdateNodeState(@OutputNode, False);
  ButtonPressedTime := 0;

  inherited Reset;
end;

// Update
procedure TInputDevice.UpdateDevice;
var
  OldNodeState: Boolean;
begin
  inherited UpdateDevice;

  OldNodeState := OutputNode.State;

  if InputDeviceType = ID_BUTTON then
  begin
    if MilliSecondsBetween(Now, ButtonPressedTime)
       >= SIMULATION_BUTTON_TIME then
    begin
      UpdateNodeState(@OutputNode, False);

      // If the node state has changed, set InputDevicesChanged to true to
      // execute a pass of the simulator.
      if OldNodeState <> OutputNode.State then
      begin
        InputDevicesChanged := True;
      end;

      UpdateImage;
    end;
  end;
end;

// UpdateImage
procedure TInputDevice.UpdateImage;
```

```
begin
  if OutputNode.State = True then
  begin
    frmLogicSim.ilComponents.GetBitmap(ImageIDOn, Picture.Bitmap);
  end
  else
  begin
    frmLogicSim.ilComponents.GetBitmap(ImageIDOff, Picture.Bitmap);
  end;
end;

// GetInputDeviceType
function TInputDevice.GetInputDeviceType: TInputDeviceType;
begin
  Result := InputDeviceType;
end;

// MouseDownHandler
procedure TInputDevice.OnMouseDownHandler(Sender: TObject; Button: TMouseButton;
                                          Shift: TShiftState; X: longint;
                                          Y: longint);
var
  OldNodeState: Boolean;
begin
  inherited OnMouseDownHandler(Sender, Button, Shift, X, Y);

  if CurrentTool = TOOL_WIRE then
  begin
    if IsMouseInNodeArea(X, Y, OutputNode) then
    begin
      NodeClicked(@OutputNode);
    end;
  end;

  if ApplicationState = SIMULATING then
  begin
    OldNodeState := OutputNode.State;

    case InputDeviceType of
      ID_SWITCH: begin
                   UpdateNodeState(@OutputNode, not OutputNode.State);
                 end;
      ID_BUTTON: begin
                   UpdateNodeState(@OutputNode, True);
                   ButtonPressedTime := Now;
                 end;
    end;

    UpdateImage;

    // If the node state has changed, set InputDevicesChanged to true to execute
    // a pass of the simulator.
    if OldNodeState <> OutputNode.State then
    begin
      InputDevicesChanged := True;
    end;
  end;
end;

// OnMouseMoveHandler
procedure TInputDevice.OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
```

```
                                                X: longint; Y: longint);
begin
  inherited OnMouseMoveHandler(Sender, Shift, X, Y);

  // If cursor is within the bounds of any node, change cursor to crCross, else
  // change it to crArrow (the default cursor).
  if Cursor = crCross then
  begin
    Cursor := crArrow;
  end;
  if CurrentTool = TOOL_WIRE then
  begin
    if IsMouseInNodeArea(X, Y, OutputNode) then
    begin
      Cursor := crCross;
    end;
  end;
end;

// OnPaintHandler
procedure TInputDevice.OnPaintHandler(Sender: TObject);
begin
  inherited OnPaintHandler(Sender);

  if CurrentTool = TOOL_WIRE then
  begin
    DrawNode(OutputNode);
  end;
end;
// END TInputDevice

// TOutputDevice
// Create
constructor TOutputDevice.Create(AOwner: TComponent;
  NewOutputDeviceType: TOutputDeviceType);
begin
  inherited Create(AOwner);
  OutputDeviceType := NewOutputDeviceType;

  InitialiseNewNode(InputNode, self, NODE_INPUT, 0, 23);

  SetBounds(0, 0, 60, 46);

  case OutputDeviceType of
    OD_INDICATOR:
    begin
      ImageIDOff := OUTPUT_INDICATOR_OFF_IMG_ID;
      ImageIDOn := OUTPUT_INDICATOR_ON_IMG_ID;
    end;
  end;

  frmLogicSim.ilComponents.GetBitmap(ImageIDOff, Picture.Bitmap);
end;

// Destroy
destructor TOutputDevice.Destroy;
begin
  UninitialiseNode(InputNode);

  inherited Destroy;
end;
```

```
// Reset
procedure TOutputDevice.Reset;
begin
  UpdateNodeState(@InputNode, False);

  inherited Reset;
end;

// UpdateImage
procedure TOutputDevice.UpdateImage;
begin
  if InputNode.State = True then
  begin
    frmLogicSim.ilComponents.GetBitmap(ImageIDOn, Picture.Bitmap);
  end
  else
  begin
    frmLogicSim.ilComponents.GetBitmap(ImageIDOff, Picture.Bitmap);
  end;
end;

procedure TOutputDevice.UpdateDevice;
begin
  UpdateImage;
end;

// OnPaintHandler
procedure TOutputDevice.OnPaintHandler(Sender: TObject);
begin
  inherited OnPaintHandler(Sender);

  if CurrentTool = TOOL_WIRE then
  begin
    DrawNode(InputNode);
  end;
end;

// OnMouseDownHandler
procedure TOutputDevice.OnMouseDownHandler(Sender: TObject; Button: TMouseButton;
                                          Shift: TShiftState; X: longint;
                                          Y: longint);
begin
  inherited OnMouseDownHandler(Sender, Button, Shift, X, Y);

  if CurrentTool = TOOL_WIRE then
  begin
    if IsMouseInNodeArea(X, Y, InputNode) then
    begin
      NodeClicked(@InputNode);
    end;
  end;
end;

// OnMouseMoveHandler
procedure TOutputDevice.OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
                                          X: longint; Y: longint);
begin
  inherited OnMouseMoveHandler(Sender, Shift, X, Y);

  if Cursor = crCross then
```

```
  begin
    Cursor := crArrow;
  end;
  if CurrentTool = TOOL_WIRE then
  begin
    if IsMouseInNodeArea(X, Y, InputNode) then
    begin
      Cursor := crCross;
    end;
  end;
end;
// END TOutputDevice

// TLogicGate
// Create
constructor TLogicGate.Create(AOwner: TComponent; NewGateType: TGateType);
var
  ImageID: Integer;
begin
  inherited Create(AOwner);
  GateType := NewGateType;
  SetBounds(10, 10, 96, 46);

  // Initialise the nodes.
  InitialiseNewNode(OutputNode, self, NODE_OUTPUT, 96 - NODE_WIDTH, 23);
  if GateType = GATE_NOT then
  begin
    // GATE_NOT does not use the second input node.
    InitialiseNewNode(InputNode1, self, NODE_INPUT, 0, 23);
    InitialiseNewNode(InputNode2, self, NODE_UNUSED, 0, 31);
  end
  else
  begin
    // Gates other than GATE_NOT use both input nodes.
    InitialiseNewNode(InputNode1, self, NODE_INPUT, 0, 15);
    InitialiseNewNode(InputNode2, self, NODE_INPUT, 0, 31);
  end;

  case GateType of
    GATE_NOT: ImageID := GATE_NOT_IMG_ID;
    GATE_AND: ImageID := GATE_AND_IMG_ID;
    GATE_OR: ImageID := GATE_OR_IMG_ID;
    GATE_NAND: ImageID := GATE_NAND_IMG_ID;
    GATE_NOR: ImageID := GATE_NOR_IMG_ID;
    GATE_XOR: ImageID := GATE_XOR_IMG_ID;
    GATE_XNOR: ImageID := GATE_XNOR_IMG_ID;
  end;
  frmLogicSim.ilComponents.GetBitmap(ImageID, Picture.Bitmap);
end;

// Destroy
destructor TLogicGate.Destroy;
begin
  UninitialiseNode(OutputNode);
  UninitialiseNode(InputNode1);
  UninitialiseNode(InputNode2);

  inherited Destroy;
end;

// Reset
```

```
procedure TLogicGate.Reset;
begin
  UpdateNodeState(@InputNode1, False);
  UpdateNodeState(@InputNode2, False);
  UpdateNodeState(@OutputNode, False);

  inherited Reset;
end;

// Update
procedure TLogicGate.UpdateDevice;
var
  NewOutputNodeState: Boolean;
begin
  // Special case for GATE_NOT - only uses InputNode1.
  if GateType = GATE_NOT then
  begin
    NewOutputNodeState := not InputNode1.State;
  end;

  // Any other gate uses both InputNode1 and InputNode2.
  case GateType of
    GATE_AND: NewOutputNodeState := InputNode1.State and InputNode2.State;
    GATE_OR: NewOutputNodeState := InputNode1.State or InputNode2.State;
    GATE_NAND: NewOutputNodeState := not (InputNode1.State and InputNode2.State);
    GATE_NOR: NewOutputNodeState := not (InputNode1.State or InputNode2.State);
    GATE_XOR: NewOutputNodeState := InputNode1.State xor InputNode2.State;
    GATE_XNOR: NewOutputNodeState := not (InputNode1.State xor InputNode2.State);
  end;

  UpdateNodeState(@OutputNode, NewOutputNodeState);
end;

// GetOutputNodeState
function TLogicGate.GetOutputNodeState: Boolean;
begin
  Result := OutputNode.State;
end;

// OnPaintHandler
procedure TLogicGate.OnPaintHandler(Sender: TObject);
begin
  inherited OnPaintHandler(Sender);

  if CurrentTool = TOOL_WIRE then
  begin
    DrawNode(OutputNode);
    DrawNode(InputNode1);
    if GateType <> GATE_NOT then
    begin
      DrawNode(InputNode2);
    end;
  end;
end;

// OnMouseDownHandler
procedure TLogicGate.OnMouseDownHandler(Sender: TObject; Button: TMouseButton;
                                        Shift: TShiftState; X: longint;
                                        Y: longint);
begin
  inherited OnMouseDownHandler(Sender, Button, Shift, X, Y);
```

```
  if CurrentTool = TOOL_WIRE then
  begin
    if IsMouseInNodeArea(X, Y, OutputNode) then
    begin
      NodeClicked(@OutputNode);
    end;
    if IsMouseInNodeArea(X, Y, InputNode1) then
    begin
      NodeClicked(@InputNode1);
    end;
    if IsMouseInNodeArea(X, Y, InputNode2) and (GateType <> GATE_NOT) then
    begin
      NodeClicked(@InputNode2);
    end;
  end;
end;

// OnMouseMoveHandler
procedure TLogicGate.OnMouseMoveHandler(Sender: TObject; Shift: TShiftState;
                                        X: longint; Y: longint);
begin
  inherited OnMouseMoveHandler(Sender, Shift, X, Y);

  if Cursor = crCross then
  begin
    Cursor := crArrow;
  end;
  if CurrentTool = TOOL_WIRE then
  begin
    if IsMouseInNodeArea(X, Y, OutputNode) or
       IsMouseInNodeArea(X, Y, InputNode1) or
       (IsMouseInNodeArea(X, Y, InputNode2) and (GateType <> GATE_NOT)) then
    begin
      Cursor := crCross;
    end;
  end;
end;
// END TLogicGate

// TSchematicDevice
// Create
constructor TSchematicDevice.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Parent := frmLogicSim.pnlSchema;

  // Make the schematic device selected when first created.
  DeselectCurrentlySelected;
  CurrentlySelectedDevice := self;
  Selected := True;

  // Assign event handlers.
  OnPaint := @OnPaintHandler;
  OnMouseDown := @OnMouseDownHandler;
  OnMouseMove := @OnMouseMoveHandler;
end;

// DrawNode
//
// Draws the nodes as black ellipses on the schematic device's corresponding
```

```
// pins.
procedure TSchematicDevice.DrawNode(ANode: TNode);
begin
  with Canvas do
  begin
    Pen.Color := clBlack;
    Brush.Style := bsSolid;
    Brush.Color := clBlack;

    EllipseC(ANode.PositionRelativeToDevice.x + NODE_WIDTH div 2,
             ANode.PositionRelativeToDevice.y,
             NODE_WIDTH div 2, NODE_HEIGHT div 2);
  end;
end;

// Reset
procedure TSchematicDevice.Reset;
begin
  UpdateImage;
end;

// UpdateImage
procedure TSchematicDevice.UpdateImage;
begin

end;

// Update
procedure TSchematicDevice.UpdateDevice;
begin

end;

// OnPaintHandler
procedure TSchematicDevice.OnPaintHandler(Sender: TObject);
begin
  if Selected then
  begin
    with Canvas do
    begin
      Brush.Style := bsClear;
      Pen.Width := 1;
      Pen.Color := clSilver;
      Pen.Style := psDot;
      Rectangle(0, 0, self.Width, self.Height);
    end;
  end;
end;

// OnMouseDownHandler
procedure TSchematicDevice.OnMouseDownHandler(Sender: TObject;
                                             Button: TMouseButton;
                                             Shift: TShiftState;
                                             X: longint; Y: longint);
begin
  if (ApplicationState = EDITING) and (CurrentTool = TOOL_MOUSE) then
  begin
    SelectDevice(self);
  end;
end;
```

```
// OnMouseMoveHandler
procedure TSchematicDevice.OnMouseMoveHandler(Sender: TObject;
                                              Shift: TShiftState; X: longint;
                                              Y: longint);
var
  CursorPositionRelativeToSchematic: TPoint;
begin
  if (ApplicationState = EDITING) and (CurrentTool = TOOL_MOUSE) then
  beginXX
    CursorPositionRelativeToSchematic :=
      frmLogicSim.pnlSchema.ScreenToClient(Mouse.CursorPos);

    if ssLeft in Shift then
    begin
      if SnapToGrid then
      begin
        // Snapping enabled.
        Left := ((CursorPositionRelativeToSchematic.x - Width div 2)
                 div GRID_SIZE) * GRID_SIZE;
        Top := ((CursorPositionRelativeToSchematic.y - Height div 2)
                 div GRID_SIZE) * GRID_SIZE;
      end
      else
      begin
        // Snapping disabled.
        Left := CursorPositionRelativeToSchematic.x - Width div 2;
        Top := CursorPositionRelativeToSchematic.y - Height div 2;
      end;
    end;
  end;

  RedrawAllWires;
end;
// END TSchematicDevice

end.
```

File: logic_simulator_simulation.pas

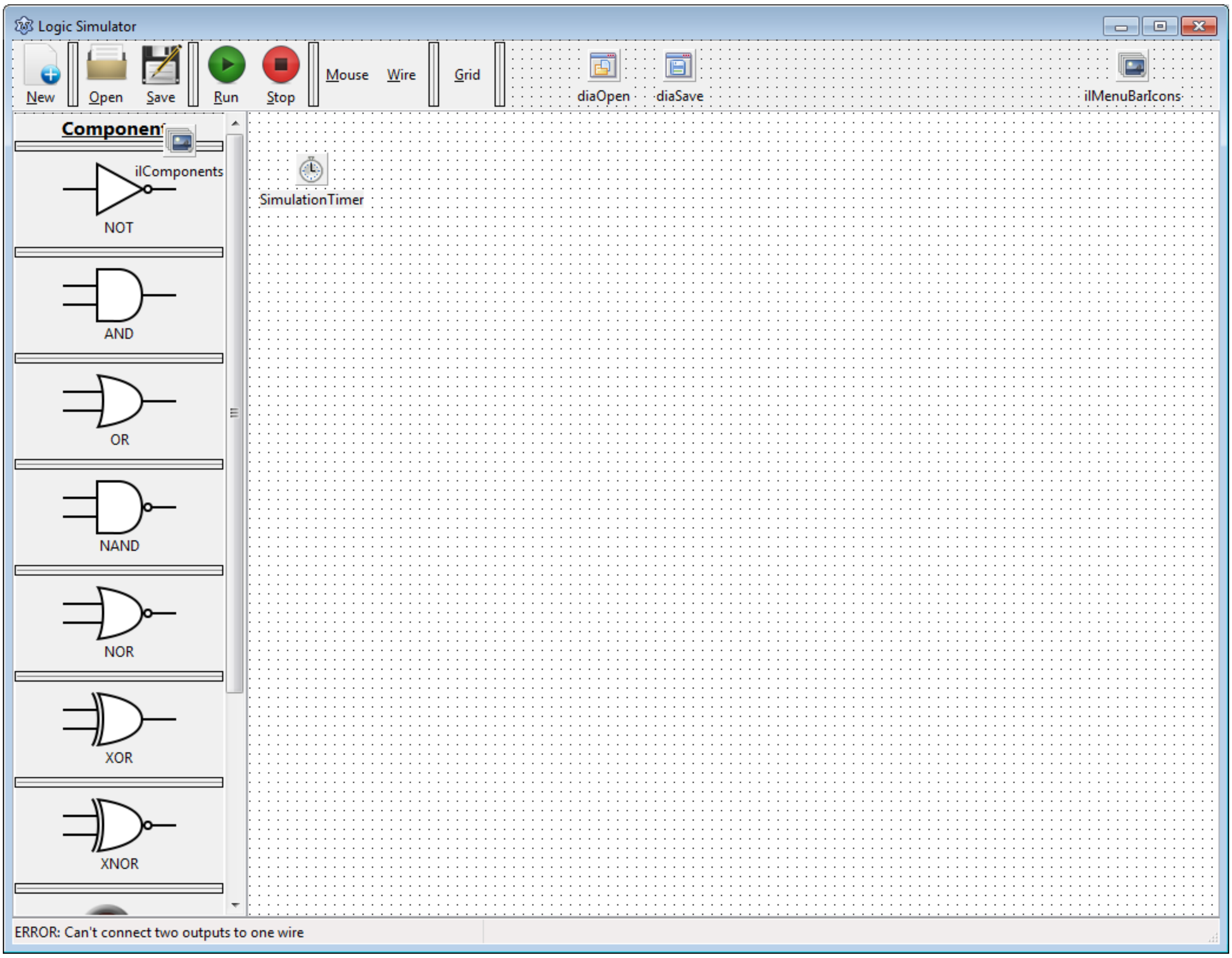
```
unit logic_simulator_simulation;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, Forms, ExtCtrls, Controls;

procedure StartSimulation;
procedure StopSimulation;
procedure SimulateOnce;


implementation

uses
  logic_simulator_form, logic_simulator_globals, logic_simulator_common;

var
  OldGateValues:     array of Boolean;
  CurrentGateValues: array of Boolean;

// SchematicDevicesChanged
//
// Returns TRUE if the output node of any schematic device has changed since the
// last call. Otherwise, returns FALSE.
function SchematicDevicesChanged: Boolean;
var
  i: Integer;
  Tmp: Pointer;
begin
  // Copy CurrentGateValues into OldGateValues.
  for i := 0 to Length(CurrentGateValues)-1 do
  begin
    OldGateValues[i] := CurrentGateValues[i];
  end;

  // Update CurrentGateValues with the current gate (output node) values.
  i := 0;
  for Tmp in GatesList do
  begin
    CurrentGateValues[i] := TLogicGate(Tmp).GetOutputNodeState;
    Inc(i);
  end;

  // Return TRUE if there is a difference between CurrentGateValues and
  // OldGateValues.
  Result := False;
  for i := 0 to Length(CurrentGateValues)-1 do
  begin
    if CurrentGateValues[i] <> OldGateValues[i] then
    begin
      Result := True;
      Exit;
    end;
  end;
end;

```

```
// ResetSchematicDevices
//
// Calls the function 'Reset' on every device in the schematic.
procedure ResetSchematicDevices;
var
  Tmp: Pointer;
begin
  for Tmp in InputDevicesList do
  begin
    TInputDevice(Tmp).Reset;
  end;

  for Tmp in OutputDevicesList do
  begin
    TOutputDevice(Tmp).Reset;
  end;

  for Tmp in GatesList do
  begin
    TLogicGate(Tmp).Reset;
  end;
end;

// StartSimulation
//
// Disables all editing functionality, sets up variables and timers for use
// while simulating and executes an initial simulation pass.
procedure StartSimulation;
var
  Tmp: Pointer;
begin
  DeselectCurrentlySelected;
  SetToolType(TOOL_MOUSE);

  // Disable editing.
  with frmLogicSim do
  begin
    btnRun.Enabled := False;
    btnStop.Enabled := True;
    btnMouse.Enabled := False;
    btnWire.Enabled := False;
    scrollComponents.Hide;
  end;
  ApplicationState := SIMULATING;
  UpdateStatusBar('Simulating');

  ResetSchematicDevices;
  frmLogicSim.SimulationTimer.Enabled := True;
  SimulationStartTime := Now;

  // Change the size of OldGateValues and CurrentGateValues to accommodate every
  // gate in the circuit.
  SetLength(OldGateValues, GatesList.Count);
  SetLength(CurrentGateValues, GatesList.Count);

  // Set to true, to execute an initial simulation.
  InputDevicesChanged := True;

  // Trigger the input device buttons, to simulate some initial use of loopback
  // circuits such as latches.
  for Tmp in InputDevicesList do
```

```
  begin
    if TInputDevice(Tmp).GetInputDeviceType = ID_BUTTON then
    begin
      TInputDevice(Tmp).OnMouseDownHandler(Nil, mbLeft, [], 10, 10);

      // Sleep, to ensure that each device is triggered in its own complete
      // cycle of the simulation timer.
      Sleep(SIMULATION_INTERVAL*2);
    end;
  end;
end;

// StopSimulation
//
// Clears up any resources used for the simulation and enables editing of the
// schematic.
procedure StopSimulation;
begin
  ResetSchematicDevices;
  frmLogicSim.SimulationTimer.Enabled := False;

  // Enable editing.
  with frmLogicSim do
  begin
    btnRun.Enabled := True;
    btnStop.Enabled := False;
    btnMouse.Enabled := True;
    btnWire.Enabled := True;
    scrollComponents.Show;
  end;
  ApplicationState := EDITING;
  UpdateStatusBar('Edit mode');
end;

// SimulateOnce
//
// Updates all devices and passes signals between devices until the schematic is
// no longer changing or until LoopCounter exceeds 100.
//
// When the schematic is no longer changing, it can be assumed that the circuit
// is in a stable state and can be presented to the user.
//
// LoopCounter is used to prevent infinite loops, such as when loopback circuits
// are created (e.g. SR flip-flops).
procedure SimulateOnce;
var
  Tmp: Pointer;
  LoopCounter: Integer;
begin
  LoopCounter := 0;
  repeat
    for Tmp in WiresList do
    begin
      TWire(Tmp).AffectInputNode;
    end;

    for Tmp in GatesList do
    begin
      TLogicGate(Tmp).UpdateDevice;
    end;

```

```
    Inc(LoopCounter);
  until (not SchematicDevicesChanged) or (LoopCounter >= 100);

  RedrawAllWires;

  for Tmp in OutputDevicesList do
  begin
    TOutputDevice(Tmp).UpdateDevice;
  end;

  InputDevicesChanged := False;
end;

end.
```

Form: logic_simulator_form.lfm



Object List

Name (component type)	Properties	
btnGateAND (TToolButton)	Caption ImageIndex Style	← "AND" ← 1 ← tbsButton
btnGateNAND (TToolButton)	Caption ImageIndex Style	← "NAND" ← 3 ← tbsButton
btnGateNOR (TToolButton)	Caption ImageIndex Style	← "NOR" ← 4 ← tbsButton
btnGateNOT (TToolButton)	Caption ImageIndex Style	← "NOT" ← 0 ← tbsButton
btnGateOR (TToolButton)	Caption ImageIndex Style	← "OR" ← 2 ← tbsButton
btnGateXNOR (TToolButton)	Caption ImageIndex Style	← "XNOR" ← 6 ← tbsButton

Name (component type)	Properties	
btnGateXOR (TToolButton)	Caption ImageIndex Style	← “XOR" ← 5 ← tbsButton
btnInputButton (TToolButton)	Caption ImageIndex Style	← “Button (Momentary)" ← 9 ← tbsButton
btnInputSwitch (TToolButton)	Caption ImageIndex Style	← “Switch (Toggle)" ← 11 ← tbsButton
btnMouse (TToolButton)	Caption Style	← “&Mouse” ← tbsButton
btnNew (TToolButton)	Caption ImageIndex Style	← “&New” ← 0 ← tbsButton
btnOpen (TToolButton)	Caption ImageIndex Style	← “&Open” ← 1 ← tbsButton
btnOutputIndicator (TToolButton)	Caption ImageIndex Style	← “Output Lamp" ← 7 ← tbsButton
btnRun (TToolButton)	Caption ImageIndex Style	← “&Run” ← 3 ← tbsButton
btnSave (TToolButton)	Caption ImageIndex Style	← “&Save” ← 2 ← tbsButton
btnSnapToGrid (TToolButton)	Caption Style	← “&Grid” ← tbsCheck
btnStop (TToolButton)	Caption ImageIndex Style	← “&Stop” ← 4 ← tbsButton
btnWire (TToolButton)	Caption Style	← “&Wire” ← tbsButton
diaOpen (TOpenDialog)	Options	← [ofPathMustExist, ofFileMustExist, ofEnableSizing, ofViewDetail]
diaSave (TSaveDialog)	DefaultExt Filter Options	← “.lsch” ← “Logic simulator schematic|*.lsch” ← [ofPathMustExist, ofEnableSizing, ofViewDetail]
frmLogicSim (TForm)	Caption Height Position Width	← “Logic Simulator” ← 768 ← poScreenCenter ← 1024
ilComponents (TImageList)	Height Width	← 46 ← 96
ilMenuBarIcons (TImageList)	Height Width	← 36 ← 36

Name (component type)	Properties	
lblComponents (TLabel)	Align	← alTop
	Alignment	← toCenter
	Font.Height	← 21
	Font.Size	← -16
	Font.Style	← [fsBold, fsUnderline]
	ParentColor	← False
	ParentFont	← False
pnlSchema (TPanel)	Align	← alClient
	BevelOuter	← bvNone
	BorderStyle	← bsSingle
	Color	← clWhite
	ParentColor	← False
sbStatus (TStatusBar)	Align	← alBottom
	Height	← 23
	SimplePanel	← False
scrollComponents (TScrollBox)	Align	← alLeft
	AutoSize	← True
SimulationTimer (TTimer)	Enabled	← False
tbComponents (TToolBar)	Align	← alLeft
	AutoSize	← True
	ButtonHeight	← 80
	ButtonWidth	← 178
	Images	← ilComponents
	Indent	← 0
	Width	← 178
	Wrapable	← False
tbMenuBar (TToolBar)	Align	← alTop
	ButtonHeight	← 56
	ButtonWidth	← 46
	Height	← 60
	Images	← ilMenuBarIcons
	ShowCaptions	← True

File: logic_simulator_form.pas

```
unit logic_simulator_form;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, Menus,
  ComCtrls, ExtCtrls, StdCtrls, LCLType;

type

  { TfrmLogicSim }

  TfrmLogicSim = class(TForm)
```

```
    btnGateAND: TToolButton;
    btnGateNAND: TToolButton;
    btnGateNOR: TToolButton;
    btnGateNOT: TToolButton;
    btnGateOR: TToolButton;
    btnGateXNOR: TToolButton;
    btnGateXOR: TToolButton;
    btnInputButton: TToolButton;
    btnInputSwitch: TToolButton;
    btnMouse: TToolButton;
    btnNew: TToolButton;
    btnOpen: TToolButton;
    btnOutputIndicator: TToolButton;
    btnRun: TToolButton;
    btnSave: TToolButton;
    btnSnapToGrid: TToolButton;
    btnStop: TToolButton;
    btnWire: TToolButton;
    componentSeparator1: TToolButton;
    componentSeparator2: TToolButton;
    componentSeparator3: TToolButton;
    componentSeparator4: TToolButton;
    componentSeparator5: TToolButton;
    componentSeparator6: TToolButton;
    componentSeparator7: TToolButton;
    componentSeparator8: TToolButton;
    componentSeparator9: TToolButton;
    componentSeparator10: TToolButton;
    diaOpen: TOpenDialog;
    diaSave: TSaveDialog;
    ilComponents: TImageList;
    ilMenuBarIcons: TImageList;
    lblComponents: TLabel;
    menuSeparator1: TToolButton;
    menuSeparator2: TToolButton;
    menuSeparator3: TToolButton;
    menuSeparator4: TToolButton;
    menuSeparator5: TToolButton;
    pnlSchema: TPanel;
    sbStatus: TStatusBar;
    scrollComponents: TScrollBox;
    SimulationTimer: TTimer;
    tbComponents: TToolBar;
    tbMenuBar: TToolBar;
    procedure btnGateANDClick(Sender: TObject);
    procedure btnGateNANDClick(Sender: TObject);
    procedure btnGateNORClick(Sender: TObject);
    procedure btnGateNOTClick(Sender: TObject);
    procedure btnGateORClick(Sender: TObject);
    procedure btnGateXNORClick(Sender: TObject);
    procedure btnGateXORClick(Sender: TObject);
    procedure btnInputButtonClick(Sender: TObject);
    procedure btnInputSwitchClick(Sender: TObject);
    procedure btnMouseClick(Sender: TObject);
    procedure btnNewClick(Sender: TObject);
    procedure btnOpenClick(Sender: TObject);
    procedure btnOutputIndicatorClick(Sender: TObject);
    procedure btnRunClick(Sender: TObject);
    procedure btnSaveClick(Sender: TObject);
    procedure btnSnapToGridClick(Sender: TObject);
    procedure btnStopClick(Sender: TObject);
```

```
    procedure btnWireClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCloseQuery(Sender: TObject; var CanClose: boolean);
    procedure FormCreate(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: word;
                           {%H-}Shift: TShiftState);
    procedure pnlSchemaMouseDown({%H-}Sender: TObject; {%H-}Button: TMouseButton;
                                  {%H-}Shift: TShiftState;
                                  {%H-}X, {%H-}Y: integer);
    procedure pnlSchemaMouseMove(Sender: TObject; {%H-}Shift: TShiftState;
                                  {%H-}X, {%H-}Y: integer);
    procedure SimulationTimerTimer(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  frmLogicSim: TfrmLogicSim;


implementation

uses
  logic_simulator_common, logic_simulator_globals, logic_simulator_simulation;

{$R *.lfm}

{ TfrmLogicSim }

// Form events.
procedure TfrmLogicSim.FormCreate(Sender: TObject);
begin
  InitialiseProgram;
  btnStop.Enabled := False;
  SchematicChanged := False;
  SnapToGrid := False;
  ApplicationState := EDITING;
end;

procedure TfrmLogicSim.FormKeyDown(Sender: TObject; var Key: word;
                                    Shift: TShiftState);
begin
  case Key of
    VK_BACK, VK_DELETE:
    begin
      RemoveCurrentlySelected;
      RedrawAllWires;
    end;
    VK_ESCAPE:
    begin
      DeselectCurrentlySelected;
      RemoveTmpWire;
    end;
  end;
end;

procedure TfrmLogicSim.pnlSchemaMouseDown(Sender: TObject; Button: TMouseButton;
                                           Shift: TShiftState; X, Y: integer);
begin
```

```
  DeselectCurrentlySelected;
  RemoveTmpWire;
end;

procedure TfrmLogicSim.pnlSchemaMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: integer);
begin
  UpdateTmpWire;
end;

procedure TfrmLogicSim.SimulationTimerTimer(Sender: TObject);
var
  SimulationElapsedTime: TDateTime;
  Tmp: Pointer;
begin
  SimulationElapsedTime := Now - SimulationStartTime;
  UpdateStatusBar('Elapsed time: ' +
                  FormatDateTime('nn:ss', SimulationElapsedTime));

  // Update the input devices - important for buttons which automatically turn
  // off.
  for Tmp in InputDevicesList do
  begin
    TInputDevice(Tmp).UpdateDevice;
  end;

  if InputDevicesChanged then
  begin
    SimulateOnce;
  end;
end;

procedure TfrmLogicSim.FormClose(Sender: TObject; var CloseAction: TCloseAction);
begin
  CloseAction := caFree;
  UninitialiseProgram;
end;

procedure TfrmLogicSim.FormCloseQuery(Sender: TObject; var CanClose: boolean);
begin
  if SchematicChanged then
  begin
    case AskToSave of
      ID_CANCEL: CanClose := False;
      ID_YES: btnSaveClick(nil);
    end;
  end;
end;
// END form events.

// Components toolbar buttons.
procedure TfrmLogicSim.btnGateNOTClick(Sender: TObject);
begin
  AddSchematicDevice(GATE_NOT, 10, 10);
end;

procedure TfrmLogicSim.btnGateORClick(Sender: TObject);
begin
  AddSchematicDevice(GATE_OR, 10, 10);
end;

```

```
procedure TfrmLogicSim.btnGateXNORClick(Sender: TObject);
begin
  AddSchematicDevice(GATE_XNOR, 10, 10);
end;

procedure TfrmLogicSim.btnGateXORClick(Sender: TObject);
begin
  AddSchematicDevice(GATE_XOR, 10, 10);
end;

procedure TfrmLogicSim.btnGateANDClick(Sender: TObject);
begin
  AddSchematicDevice(GATE_AND, 10, 10);
end;

procedure TfrmLogicSim.btnGateNANDClick(Sender: TObject);
begin
  AddSchematicDevice(GATE_NAND, 10, 10);
end;

procedure TfrmLogicSim.btnGateNORClick(Sender: TObject);
begin
  AddSchematicDevice(GATE_NOR, 10, 10);
end;

procedure TfrmLogicSim.btnOutputIndicatorClick(Sender: TObject);
begin
  AddSchematicDevice(OD_INDICATOR, 10, 10);
end;

procedure TfrmLogicSim.btnInputButtonClick(Sender: TObject);
begin
  AddSchematicDevice(ID_BUTTON, 10, 10);
end;

procedure TfrmLogicSim.btnInputSwitchClick(Sender: TObject);
begin
  AddSchematicDevice(ID_SWITCH, 10, 10);
end;
// END components toolbar buttons.

// Main toolbar buttons.
procedure TfrmLogicSim.btnNewClick(Sender: TObject);
begin
  if SchematicChanged then
  begin
    case AskToSave of
      ID_CANCEL: Exit;
      ID_YES: btnSaveClick(nil);
    end;
  end;

  ClearSchematic;
end;

procedure TfrmLogicSim.btnOpenClick(Sender: TObject);
var
  Location: string;
begin
  StopSimulation;

```

```
  if SchematicChanged then
  begin
    case AskToSave of
      ID_CANCEL: Exit;
      ID_YES: btnSaveClick(nil);
    end;
  end;

  // Show open file selection dialog
  // diaOpen handles non-existent files
  if diaOpen.Execute then
  begin
    Location := diaOpen.FileName;

    // Cancel button has been pressed.
    if Location = '' then
    begin
      Exit;
    end;
  end;

  ClearSchematic;
  if LoadSchematicFromFile(Location) then
  begin
    // Mark the schematic as unchanged since the last save.
    SchematicChanged := False;
    DeselectCurrentlySelected;
  end
  else
  begin
    DisplayError('Could not load the schematic.');
  end;
end;

procedure TfrmLogicSim.btnSaveClick(Sender: TObject);
var
  Location: string;
begin
  StopSimulation;

  // Show save file selection dialog.
  if diaSave.Execute then
  begin
    Location := diaSave.FileName;
  end;

  if FileExists(Location) then
  begin
    // Exit the procedure if the user does not wish to overwrite existing file.
    if AskToOverwrite = ID_CANCEL then
    begin
      Exit;
    end;
  end;

  if WriteSchematicToFile(Location) then
  begin
    // Mark the schematic as unchanged since the last save.
    SchematicChanged := False;
  end
  else
```

```
  begin
    DisplayError('Could not save the schematic.');
  end;
end;

procedure TfrmLogicSim.btnSnapToGridClick(Sender: TObject);
begin
  SnapToGrid := not SnapToGrid;
end;

procedure TfrmLogicSim.btnRunClick(Sender: TObject);
begin
  StartSimulation;
end;

procedure TfrmLogicSim.btnStopClick(Sender: TObject);
begin
  StopSimulation;
end;

procedure TfrmLogicSim.btnMouseClick(Sender: TObject);
begin
  SetToolType(TOOL_MOUSE);
end;

procedure TfrmLogicSim.btnWireClick(Sender: TObject);
begin
  SetToolType(TOOL_WIRE);
end;
// END main toolbar.

end.
```

Global Constants And Variables

File: logic_simulator_globals.pas

Anything declared or defined in the unit *logic_simulator_globals.pas* is intended to be accessible application-wide.

Constants

Name	Value	Description
GRID_SIZE	20	Size of grid squares (pixels).
NODE_HEIGHT	6	Physical dimensions of nodes (pixels).
NODE_WIDTH	8	
SIMULATION_BUTTON_TIME	1000	Time for buttons to stay pressed (milliseconds).
SIMULATION_INTERVAL	10	Time between each simulation pass (milliseconds).
WIRE_HIGH_COLOUR	clRed	Wire colours for simulation.
WIRE_LOW_COLOUR	clBlack	
WIRE_SELECTED_COLOUR	clSilver	Colour of selected wire.
GATE_NOT_IMG_ID	0	Index values for ilComponents image list.
GATE_AND_IMG_ID	1	
GATE_OR_IMG_ID	2	
GATE_NAND_IMG_ID	3	
GATE_NOR_IMG_ID	4	
GATE_XOR_IMG_ID	5	
GATE_XNOR_IMG_ID	6	
OUTPUT_INDICATOR_OFF_IMG_ID	7	
OUTPUT_INDICATOR_ON_IMG_ID	8	
BUTTON_RELEASED_IMG_ID	9	
BUTTON_PRESSED_IMG_ID	10	
SWITCH_RELEASED_IMG_ID	11	
SWITCH_PRESSED_IMG_ID	12	

Variables

Name	Type	Description
ApplicationState	TApplicationState	Current state of the application.
SchematicChanged	Boolean	Holds whether the schematic has been changed since the last save. Used when saving, loading and creating a new file.
SimulationDevicesChanged	Boolean	Save processor usage by only simulating if any of the devices have changed.
SimulationStartTime	TDateTime	Time at which the simulation started.
SnapToGrid	Boolean	Devices will snap to the grid if set to True.
GatesList	TList	Lists holding every schematic object created.
InputDevicesList	TList	
OutputDevicesList	TList	
WiresList	TList	

Procedures And Functions

Name	Declared In Unit	Used In Unit(s)
AddSchematicDevice	logic_simulator_common	logic_simulator_common logic_simulator_form
AskToOverwrite	logic_simulator_common	logic_simulator_form
AskToSave	logic_simulator_common	logic_simulator_form
ClearSchematic	logic_simulator_common	logic_simulator_common logic_simulator_form
DeselectCurrentlySelected	logic_simulator_common	logic_simulator_common logic_simulator_form
DisplayError	logic_simulator_common	logic_simulator_common logic_simulator_form
DisplayWarning	logic_simulator_common	
InitialiseNewNode	logic_simulator_common	logic_simulator_common
InitialiseProgram	logic_simulator_common	logic_simulator_form
IsMouseInNodeArea	logic_simulator_common	logic_simulator_common
LoadSchematicFromFile	logic_simulator_common	logic_simulator_form
NodeClicked	logic_simulator_common	logic_simulator_common
RedrawAllWires	logic_simulator_common	logic_simulator_common logic_simulator_form logic_simulator_simulation
RemoveCurrentlySelected	logic_simulator_common	logic_simulator_form
RemoveSchematicDevice	logic_simulator_common	logic_simulator_common
RemoveTmpWire	logic_simulator_common	logic_simulator_common logic_simulator_form
RemoveWire	logic_simulator_common	logic_simulator_common
RepaintAllSchematicDevices	logic_simulator_common	logic_simulator_common
ResetSchematicDevices	logic_simulator_simulation	logic_simulator_simulation
SchematicDevicesChanged	logic_simulator_simulation	logic_simulator_simulation
SelectDevice	logic_simulator_common	logic_simulator_common
SetToolType	logic_simulator_common	logic_simulator_common logic_simulator_form logic_simulator_simulation
SimulateOnce	logic_simulator_simulation	logic_simulator_form
StartSimulation	logic_simulator_simulation	logic_simulator_form
StopSimulation	logic_simulator_simulation	logic_simulator_common logic_simulator_form
UninitialiseNode	logic_simulator_common	logic_simulator_common
UninitialiseProgram	logic_simulator_common	logic_simulator_form
UpdateNodeState	logic_simulator_common	logic_simulator_common
UpdateStatusBar	logic_simulator_common	logic_simulator_common logic_simulator_form logic_simulator_simulation
UpdateTmpWire	logic_simulator_common	logic_simulator_common logic_simulator_form
WriteSchematicToFile	logic_simulator_common	logic_simulator_form

CLASS: TSCHEMATICDEVICE

Name	Unit	Used In
DrawNode	logic_simulator_common	
Reset	logic_simulator_common	
UpdateImage	logic_simulator_common	
UpdateDevice	logic_simulator_common	

CLASS: TLOGICGATE

Name	Unit	Used In
Reset	logic_simulator_common	
UpdateDevice	logic_simulator_common	
GetOutputNodeState	logic_simulator_common	

CLASS: TOUTPUTDEVICE

Name	Unit	Used In
Reset	logic_simulator_common	
UpdateImage	logic_simulator_common	
UpdateDevice	logic_simulator_common	

CLASS: TINPUTDEVICE

Name	Unit	Used In
Reset	logic_simulator_common	
UpdateDevice	logic_simulator_common	
UpdateImage	logic_simulator_common	
GetInputDeviceType	logic_simulator_common	

CLASS: TWIRE

Name	Unit	Used In
SetStart	logic_simulator_common	
SetEnd	logic_simulator_common	
CalculateNewSize	logic_simulator_common	
AffectInputNode	logic_simulator_common	

User Manual

Logic Simulator will provide you with the ability to

Minimum System Requirements

Operating System	Windows 7 (or newer)
Processor Clock Speed	2 GHz
Installed Memory	1 GB
Screen Resolution	800x600
Other	CD-ROM Drive

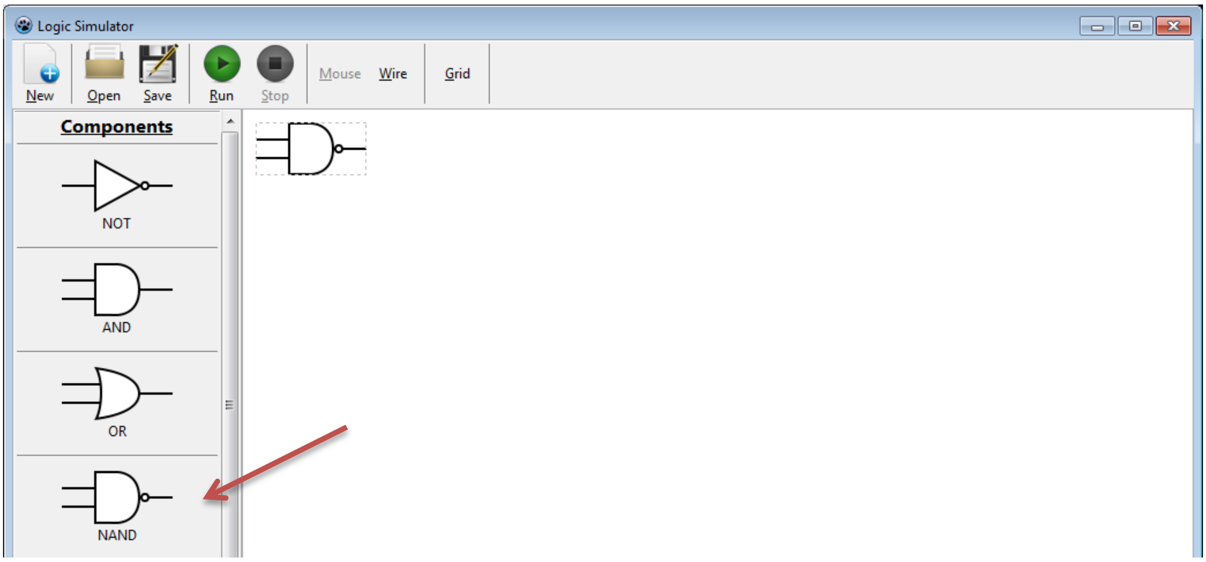
Installation And Running

Insert the CD provided and copy “Logic Simulator.exe” to your Desktop (or a preferred location). The CD can now be removed. To run, simply double-click the executable from its new location.

Schematic Editing

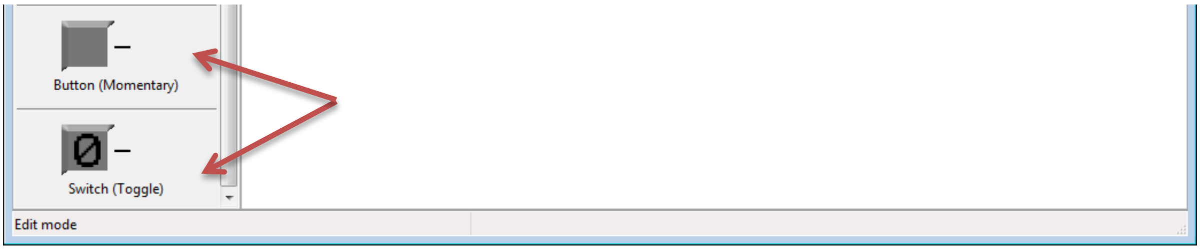
Adding Gates

1. Click on the desired logic gate from the components list at the left of the window.
2. The logic gate will appear at the top-left of the schematic view.



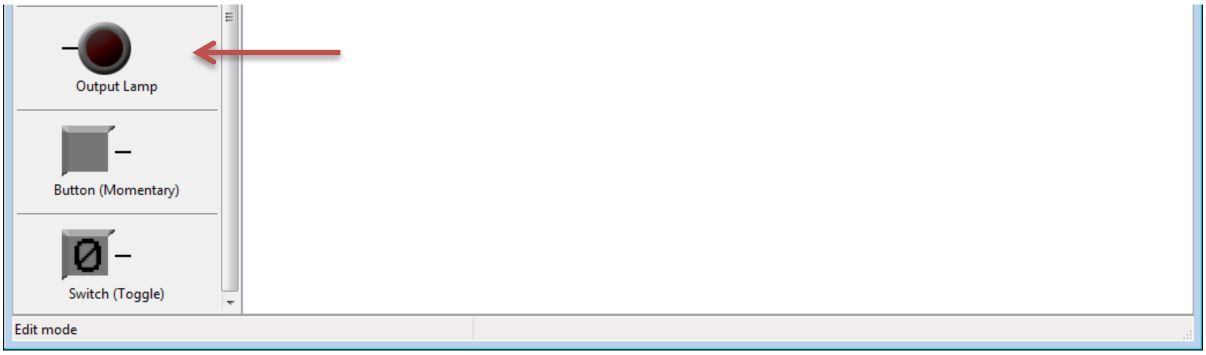
Adding Inputs

1. Scroll down to the bottom of the components list, and click on either a momentary button or a toggle switch.
2. The input device will appear at the top-left of the schematic view.



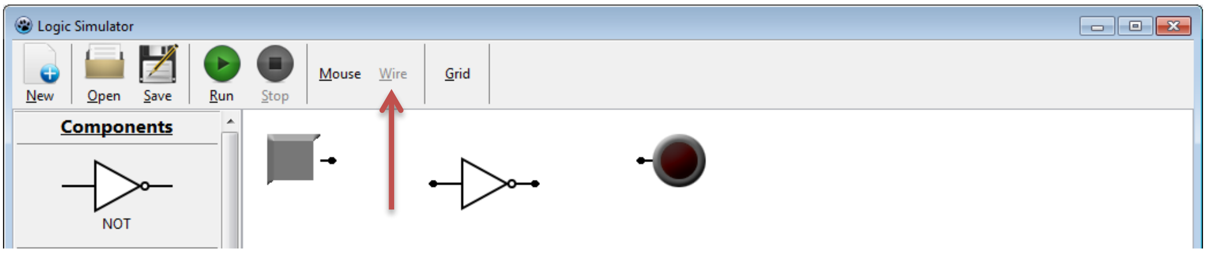
Adding Outputs

1. Scroll down to near the bottom of the components list, and select the output indicator.
2. The output device will appear at the top-left of the schematic view.

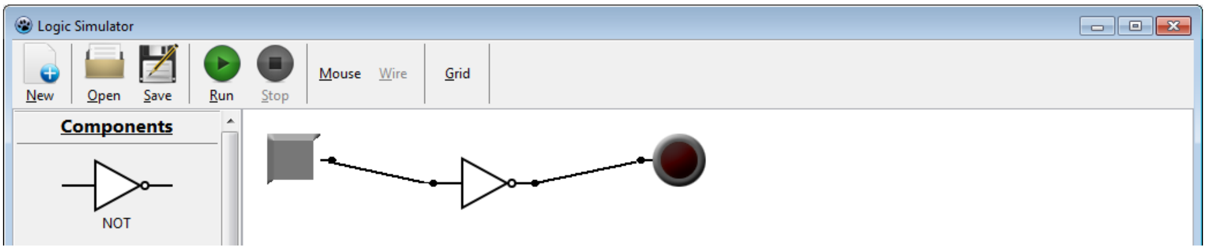


Adding Wires

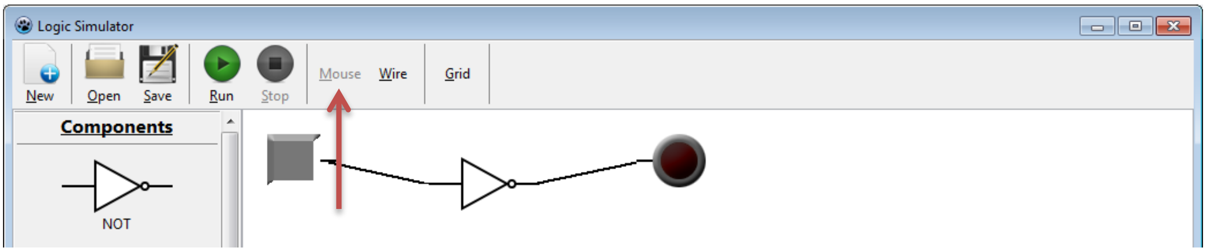
1. Click on the wire button, located on the top toolbar. Connection nodes will appear on each device in the schematic.



2. Click on the desired node to set the start point begin drawing a wire.
3. To set an end point, click on another node. If the connection is valid, the wire will be drawn. If not, an error message will appear. Please refer to the troubleshooting section of this manual for more information.



4. Click on the mouse button to exit out of wiring mode.



Removing Items

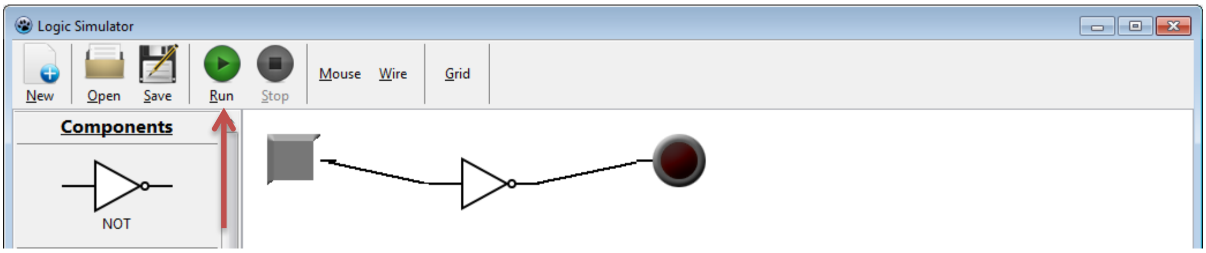
1. Select the desired device or wire by clicking on it. It will either change colour or an outline will appear around it.

2. Press the delete key or the backspace key on the keyboard.
   NOTE:
   - If a wire is selected, it will be removed.
   - If a device is selected, it will be removed along with any connected wires.

Circuit Simulation

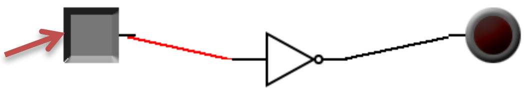
Starting A Simulation

1. Click the green 'run' button located on the top toolbar. The components list will disappear, and editing will be disabled.



Interacting With A Simulation

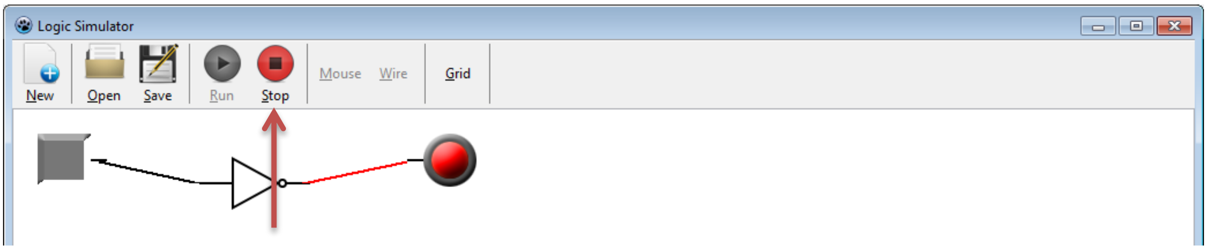
1. During simulation, input devices can be clicked in order to change the state of device inputs.



2. Output devices will change colour depending on the state of their inputs.
3. Wires will turn red if they are carrying a signal and black if there is no signal present.

Stopping A Simulation

1. Click the red 'stop' button located on the top toolbar. The components list will reappear, and editing will be enabled.



TROUBLESHOOTING

Error	Explanation
"Cannot connect a node to itself."	A wire would not have any useful effect if both ends were connected to the same node. Therefore, wires cannot connect a node to itself.
"Cannot connect two input nodes together."	
"Cannot connect two output nodes together."	
"Only one wire can be connected to a single input."	Only a single wire can be connected to an input node.
"Wire already exists."	When trying to connect two nodes with a wire, the connection already exists. A wire cannot be drawn here.
"Could not load the schematic."	
"Could not save the schematic."	

APPRAISAL

REVIEW OF OBJECTIVES

Fully Met

- ✓ **"End users must be able to connect indicators to any wire in the circuit if desired, clearly changing visually as the wire changes state."**
  The program has been implemented with output lamps that can connect to any output node in the circuit. They provide an easy way to determine the state of any wire connected to the corresponding output node by becoming illuminated when high (on), and dull when low (off).

- ✓ **"It would be beneficial for each wire to be colour coded…"**
  During simulation, the colour of a wire reflects the current state that it is holding. Depending on the wire's state (high/low), the wire will be coloured (red/black) accordingly.

- ✓ **"The user must also be able to add interactive input buttons…"**
  The logic simulator currently provides two interactive input devices; a momentary push button and a toggle switch. During simulation, the user is able to interact with these devices in order to make the simulation environment more approachable and generally easier to use.

- ✓ **"Movement, addition and deletion of components…"**
  The program's edit mode provides the user with an intuitive way to add components to the schematic. Clicking the corresponding device in the components toolbar will insert it in an accessible place in the schematic. The user is then able to click and drag the device to move it to the desired location. If the component is not required, it can be selected and deleted simply by clicking it once, and pressing either the backspace key or delete key on the keyboard.

- ✓ **"NOT, AND, OR, NAND, NOR, XOR and XNOR gates."**
  All of these logic gates are included in the final program, and function exactly how they are expected to.

- ✓ **"Each logic gate will need to be situated on an easily accessible panel."**
  The components toolbar on the left-hand side of the program is always accessible (provided the user is in edit mode) and contains every device that has been implemented.

- ✓ **"Useful error/warning messages will have to be provided…"**
  Error messages are programmed in areas where failsafe operation cannot be guaranteed. This includes the user connecting devices/nodes with wires (as an invalid circuit could easily be created), saving (when the user does not have the necessary permissions to write the file) and loading (when the user does not have the necessary permissions to read the file).

- ✓ Simulation.

Partially Met

- Icons.
- Saving/loading.

Not Met

- **"Printing schematics…"**
- **"…include some form of help or reference system…"**

EVALUATION FROM END USER

Overall, I believe that the Logic Simulator that has been produced is an effective tool for producing logic simulations. The ease of use of the program is one of its key strengths; the interface is simple, with graphics that are appropriate without being overly detailed or cluttered. The layout is such that components are easy to find, and the toolbar at the top of the window is very simple to use. Furthermore, the ability to snap components to a grid means that it is easy to structure the layout of components, thus making it easier to use than without a grid.

While testing, I found a multitude of disadvantages that – while not making the program impossible to use – it would benefit from these issues being rectified. These are outlined below.

Issue	Explanation
Show grid	The grid is not shown to the user when the grid interface is enabled. While this may be beneficial in some circumstances (to reduce the cluttering up of the screen), in others, it may not. Being able to toggle displaying the grid in addition to toggling the grid itself would be beneficial, to allow easier alignment of components.
Component addition	Currently, components are added by clicking on their icon. While this is easy to get used to, I believed it to be more intuitive to click and drag the components onto the canvas.
Wire drawing	Drawing wires is always done by connecting the two points directly. This means that there cannot be any elbow joins. Adding this functionality would be beneficial as it would allow the layout of the logic simulation to be more structured, in addition to giving it a greater resemblance to how these logic schematics may be drawn on paper.
Input to output connection bug	When trying to connect two output components to an output of a gate, the second component must be connected by drawing a wire from the output of the gate to the input of the component. Connecting it the other way around produces an error message that is irrelevant.
Hidden component list	Upon running the simulation, the component list disappears. This means that the entire canvas is shifted to the left, and this is slightly confusing as it changes the appearance of the schematic. I believe that it would equally convenient and easier to use if the component list was simply greyed out.
Drag components off screen	It is possible to drag components out of the window or behind the component list. These cannot be retrieved, and this is a bug that makes the program difficult to use as it is possible to lose a component.
Right click wiring functionality	Constantly switching modes to move components and wire gates together is a tedious process. As two button mice are standard, it may be more beneficial and easier to use if the left mouse button were to be used for moving components, and the right mouse button used to draw a wire between them.
Loading of wires	When saving a schematic and loading it again, the wires are not preserved. This means that the wires need to be reconnected every time the schematic is loaded. This is very inconvenient for the user and makes the saving feature rather redundant.
Design expansion	Currently, the schematic size is limited to the size of the canvas. While it is unlikely that many designs will be larger than the space provided, it is conceivable that the combination of a lower resolution monitor and a complex logic schematic would require a larger canvas. Therefore, being able to expand the size of the canvas would be a useful feature.
Variety of output components	Only one output component is currently available. While this works appropriately, it may be more convenient to be able to choose between different output components and/or change the colour of the lamp. This would mean that logic simulations could be easily understood if colours could be associated with certain input patterns, rather than having to trace through the network of logic gates each time.

Issue	Explanation
Cascade new components	When new components are added, they are added on top of each other. This can cause confusion as to how many components have been added, and it would be clearer if new components were cascaded so that as well as being able to individually pick out the components that have been added (otherwise they have to be selected in reverse order of addition), it makes it clear how many components have been added.
Multiple component selection	Selecting multiple components is currently not possible. This means that when new components are added, each one has to be dragged to the right area of the schematic before laying them out appropriately, especially as they are currently not cascaded. Being able to select multiple components (through use of a box drag selection) would allow components to be easily moved between areas of the schematic. For example, if a new gate needed to be added at the start of the schematic, it is currently not possible to move the entire schematic at once and add it, making it very inconvenient to implement it.

Possible Extensions

- Astable clock input devices.
- Wider selection of output devices.