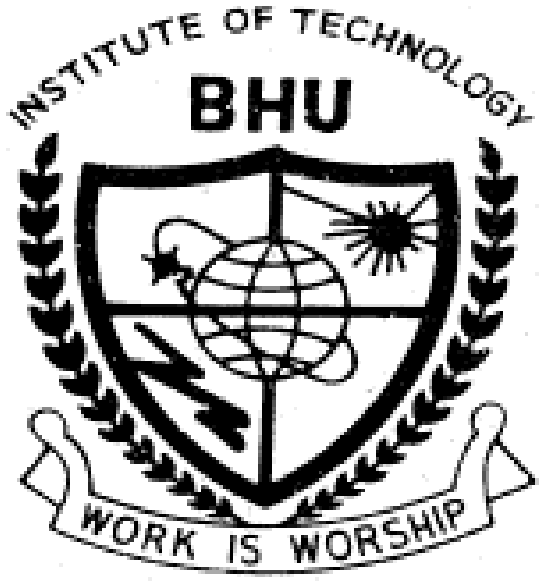
Logic Circuit Simulator



Done By,

Abhijith V Mohan

10400EN001

IDD 2nd CSE

ITBHU

Certificate

This is to certify that Abhijith V Mohan, Roll No. 10400EN001 of IDD Computer Science, 2nd Year, 4th Semester has successfully completed this project as part of their Software Project Lab (CS-2403).

Dr. Vinayak Srivastava
Asst. Professor

Acknowledgements

I would like to express my deepest gratitude Prof Vinayak Srivastava for his valuable guidance in the implementation of this project.

I also take this opportunity to thank Daniweb, the programming community , for answering my queries regarding the implementation of GUI features in JAVA.

Abstract

The objective of this project is to make a cross-platform GUI application in JAVA that allows the user to simulate simple combinational circuits having devices such as AND, OR, NOT, NAND, NOR, XOR, EXNOR etc. The user can drag and drop devices from a toolbar and connect them to create the circuit and the circuit behaviour will be simulated.

Table Of Contents

Introduction

Logic Circuits are the basic building blocks of any digital system such as a computer. A circuit designer may need to test his circuit before implementing it in hardware. A simulation software is used for this purpose. The goal of simulation is to avoid loss of time and money which occurs by testing circuits on hardware. Once the circuit behaves as intended in a simulation program, it can be implemented in hardware thus saving testing time and avoiding wastage of equipment.

Our project is a Logic Circuit Simulation Software in JAVA.The project can simulate any combinational circuit and has devices such as AND, OR, NOT, NAND, NOR, XOR, EXNOR. There are also devices for setting the input variables and seeing the output generated by the circuit.

Theoretical Background

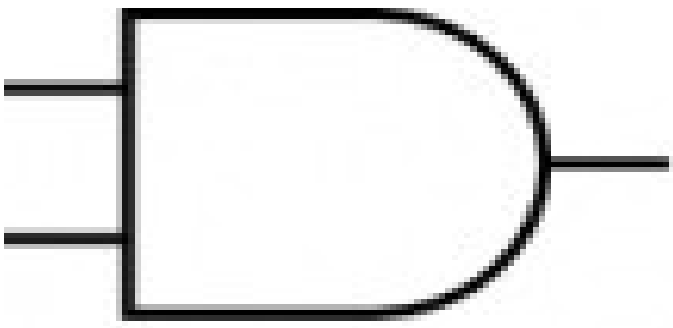
The logic circuits have devices that implement various boolean operations. The basic gates used in this application are

1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. EXNOR

AND Gate

The AND Gate implements the boolean AND function. Its symbol and truth table are as follows.

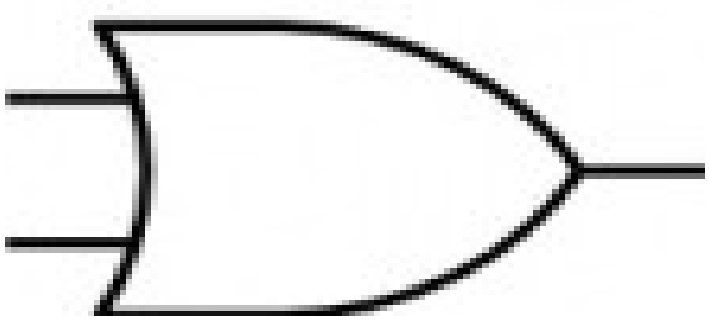
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



OR Gate

The OR Gate implements the boolean OR function. Its symbol and truth table are as follows.

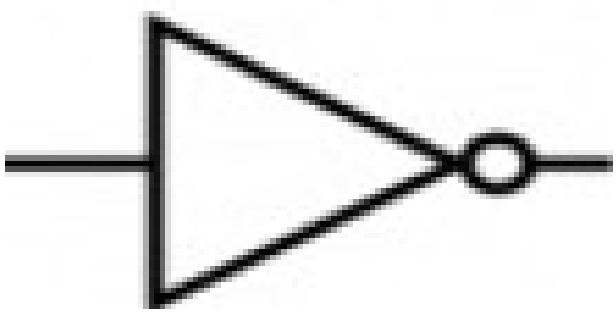
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



NOT Gate

The NOT Gate is a one input gate that implements the NOT function. Its symbol and truth table are as follows.

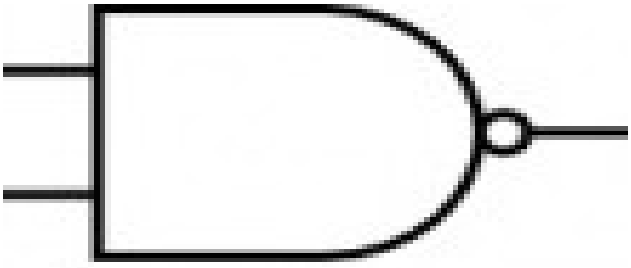
A	F
0	1
1	0



NAND Gate

The NAND gate is an AND gate followed by a NOT gate.

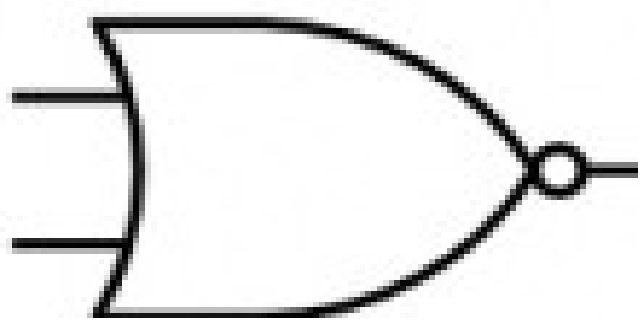
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



NOR Gate

The NOR gate is an OR gate followed by a NOT gate.

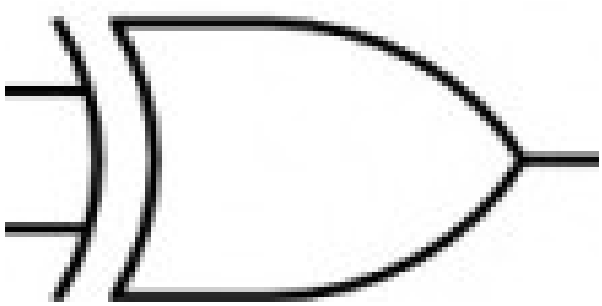
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



XOR Gate

The XOR gate implements the Exclusive OR function.

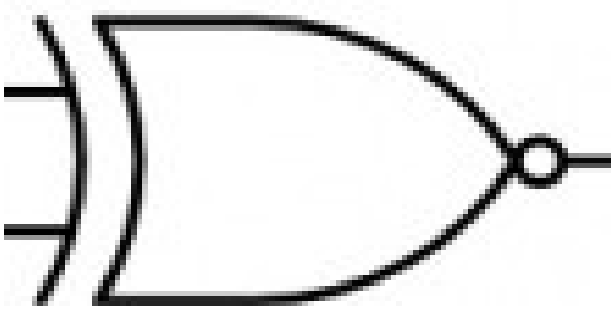
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



EXNOR Gate

The EXNOR gate is a XOR gate followed by a NOT gate.

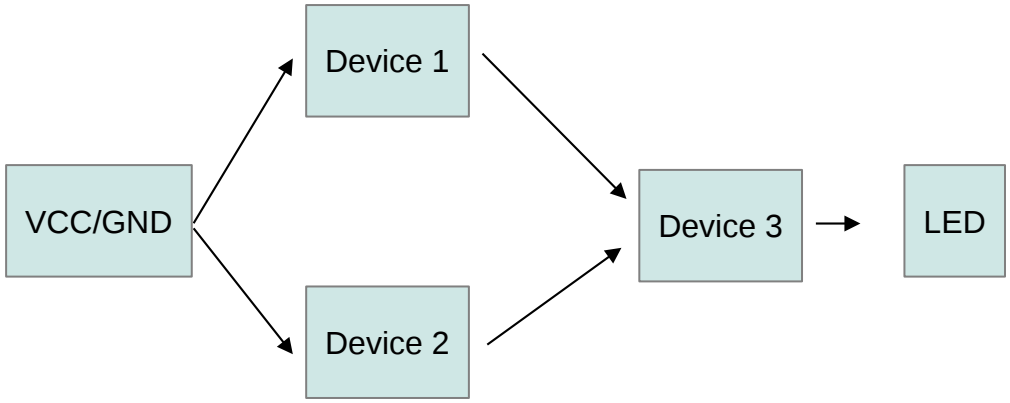
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1



Design

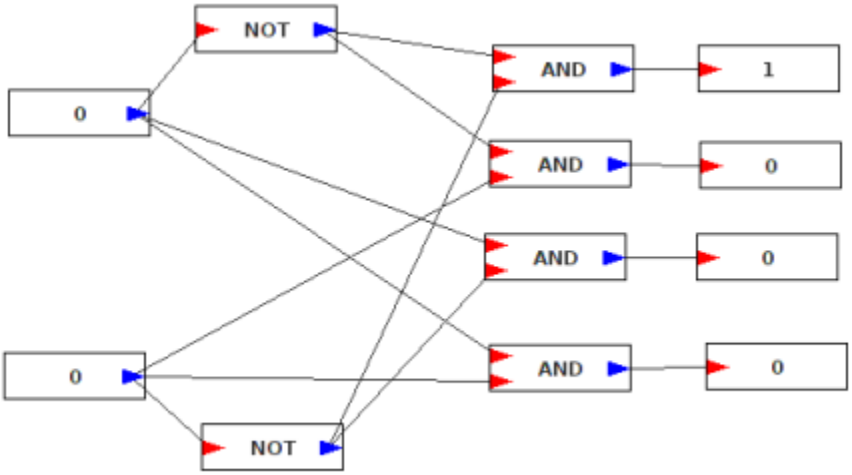
The simulation occurs in a chained fashion.The user connects the circuit and clicks on simulate button to simulate the circuit. The simulate button actionPerformed() calls the deviceFunction() methods of all LEDs in the circuit. Every LED will query its Input connector for its value. The input connector in turn will query the output of the device to which it is connected, the output calls the devicefunction() of that device and the process continues until it reaches VCC or GND which is a toggleable device that has output HIGH or LOW. The VCC or GND will return its user-set value which will propagate to the LED and the LED will change its state and the circuit behaviour can be observed.

Consider an abstract circuit as shown.



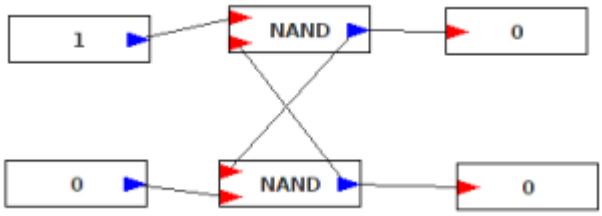
Here the LED will query Device 3. It will query Device 1 and Device 2, which in turn will query the VCC/GND. The value returned by the VCC/GND is acted upon by Devices 1 and Device 2 to produce some outputs. These will be returned to Device 3 which will perform its function on these inputs and return the value to the LED. The LED will change its state to reflect the value input to it. Hence, the circuit behaviour is simulated.

An example circuit of a 2x4 decoder simulation is given below.



The 4 LEDs are shown at the right end. The simulate button calls their deviceFunction() and they call the AND gates from which they are taking input. These calls propagate until they reach the VCC/GND toggleables at the left end of the circuit shown above. Then the values are propagated through the devices into the output.

The disadvantage of this method of simulation is that sequential circuits cannot be simulated. The reason is that the devices in the loop will keep calling the functions of each other infinitely, resulting in stack overflow. For example consider the circuit shown below:

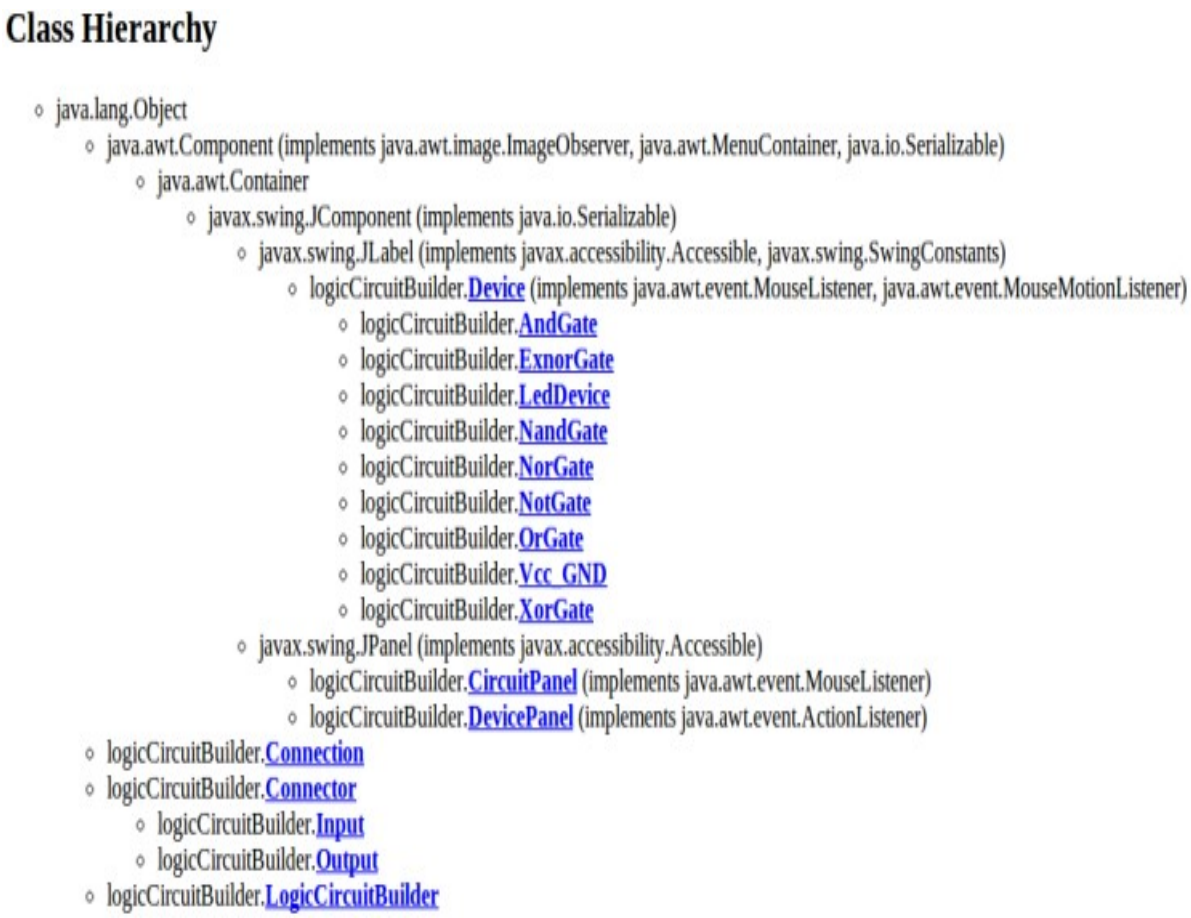


Implementation Details

A summary of the classes are shown in the following screenshot of the Javadoc generated from the project

Class Summary	
AndGate	An AND Gate does the boolean AND operation on 2 inputs
CircuitPanel	CircuitPanel is the panel where the circuit will be built.
Connection	Connection objects are the wires between the devices They are used just for drawing the lines no logic is implemented through them
Connector	Abstract Class for input and output connectors
Device	Device is any of the devices available to be implemented in the circuit.
DevicePanel	DevicePanel is the panel on the west where the device buttons are placed.
ExnorGate	EXNOR Gate implements the boolean complemented Exclusive OR (equivalence) function for 2 inputs
Input	Input connectors are used to give input to the gates
LedDevice	LED is used to see the output of the circuit.
LogicCircuitBuilder	Main class for the Logic Circuit Builder program.
NandGate	A NAND gate implements the boolean NAND function on 2 inputs
NorGate	A NOR gate implements the boolean NOR function for 2 inputs
NotGate	A NOT gate implements the boolean NOT function.
OrGate	An OR Gate implements the boolean OR function for 2 inputs
Output	Output connectors are used to take output from devices.
Vcc_GND	VCC is a device which gives a constant HIGH output
XorGate	A XOR gate implements the boolean Exclusive OR function for 2 inputs

The class hierarchy is shown in the following screenshot from the Javadoc generated from the project.



The main JFrame in the LogicCircuitBuilder class has two Jpanels- The DevicePanel to the left and the CircuitPanel to the right. Clicking on a Device button in the DevicePanel sets the static deviceAdded variable of CircuitPanel to any of the values specified as constants in the Device class. Now when the user clicks on the CircuitPanel, the mouseClicked() method of CircuitPanel checks the value of the deviceAdded variable, creates an instance of the particular device and adds it to the CircuitPanel. Also if the deviceAdded is an LED, the CircuitPanel will keep a reference to it inside an ArrayList leds.

Device is an abstract class that extends javax.swing.JLabel. It consists of an array of Input objects, an array of Output objects, and an abstract function `public abstract void deviceFunction()`. It also defines several constants which are the valid values of the deviceAdded variable. All devices are subclasses of Device. Each device implements its logic by overriding the `public void deviceFunction()`. The Device class also implements the drag & drop features and other interactions by mouse such as connecting two devices by clicking an output and then an input. It does this by making use of functions in the mouseMotionListener interface.

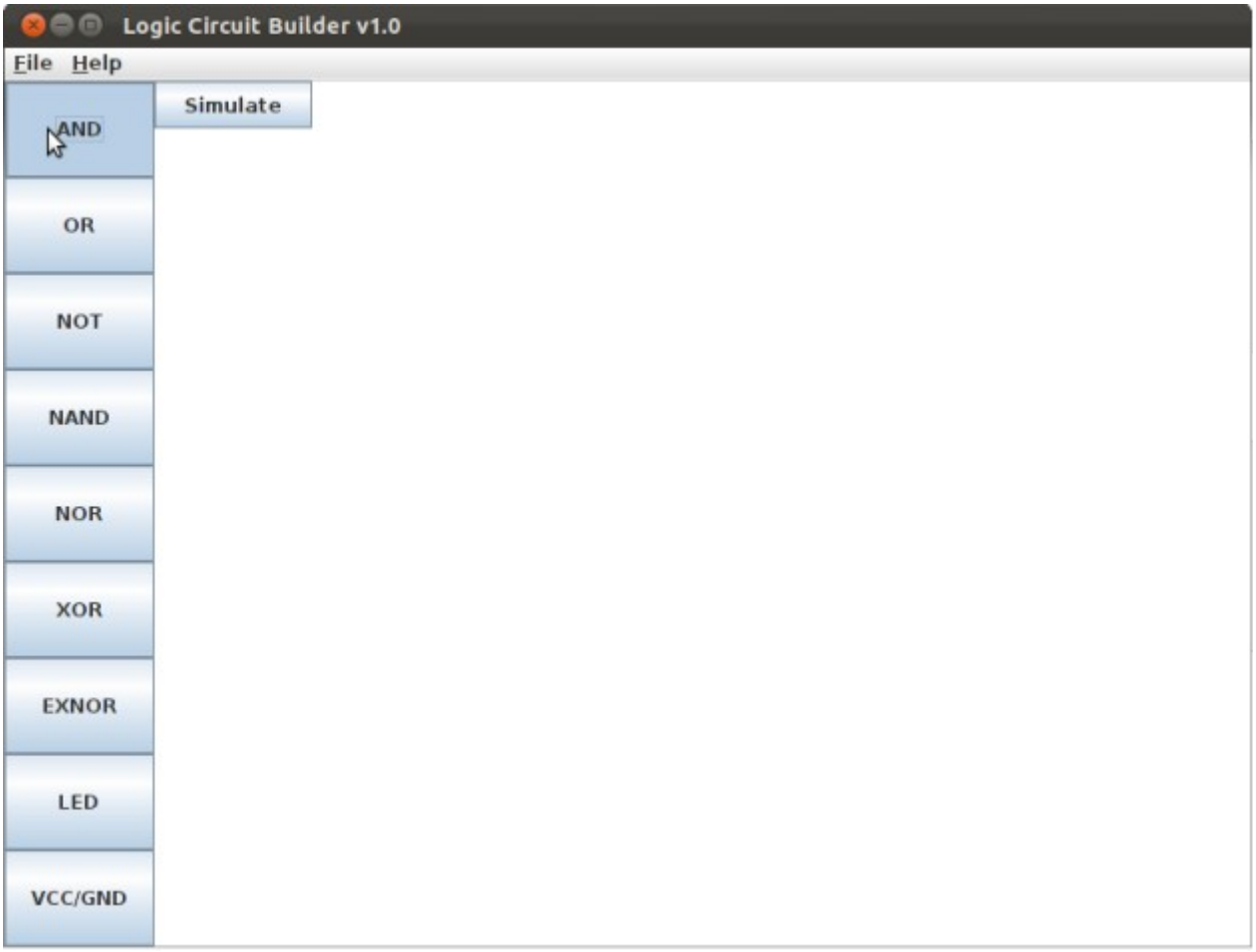
Connector is an abstract class from which the Input and Output classes inherit. It implements their common behaviour. The Input and Output objects are added to Devices in their inputs[] and outputs[] arrays. Their functions are important in implementing the circuit logic.

Connection is a class for storing an Input,Output pair and drawing a line between them. It has no significance in circuit logic. Its function is simply to display to the user the connections between the devices.

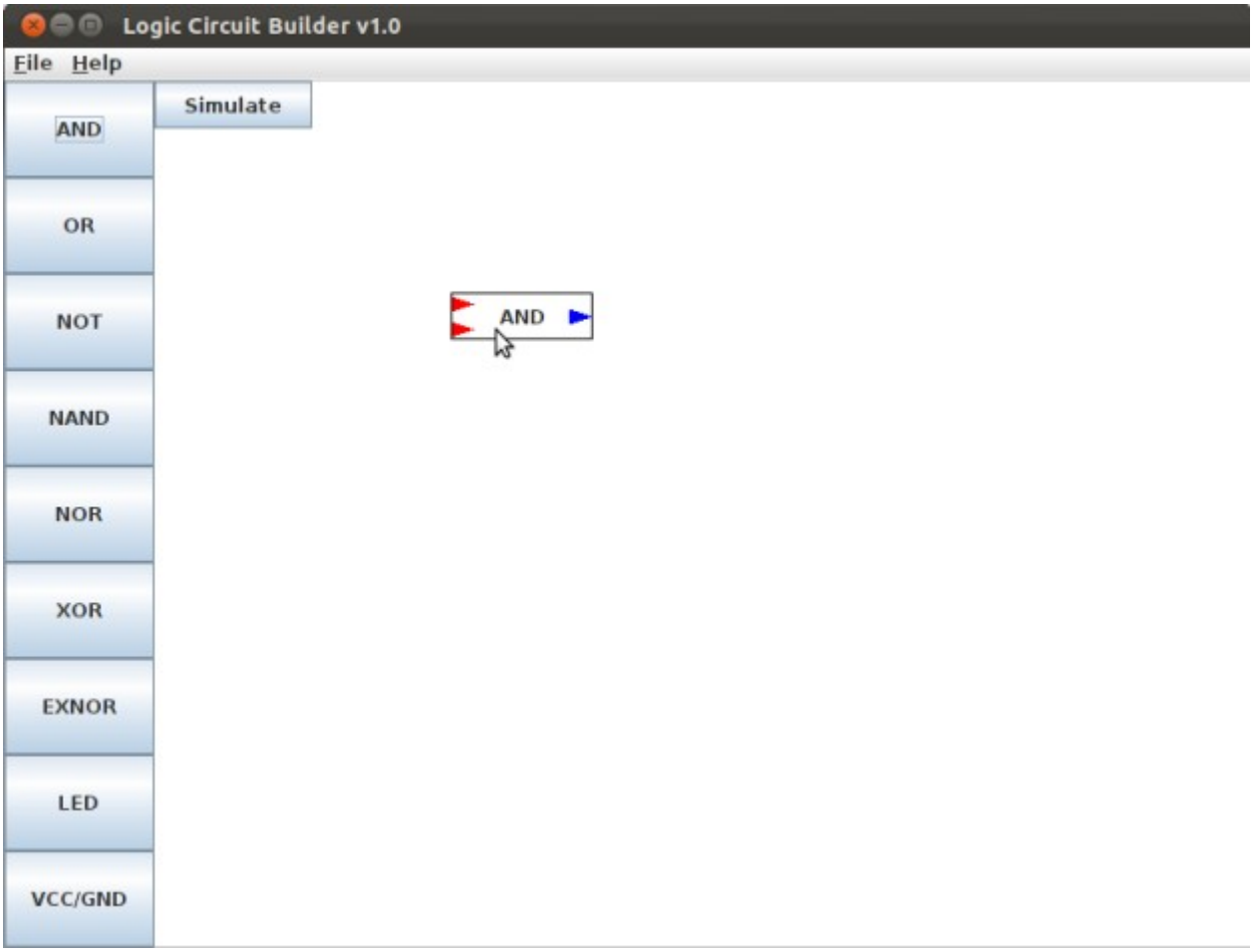
Manual

The steps to follow if a Circuit has to be simulated are as follows:

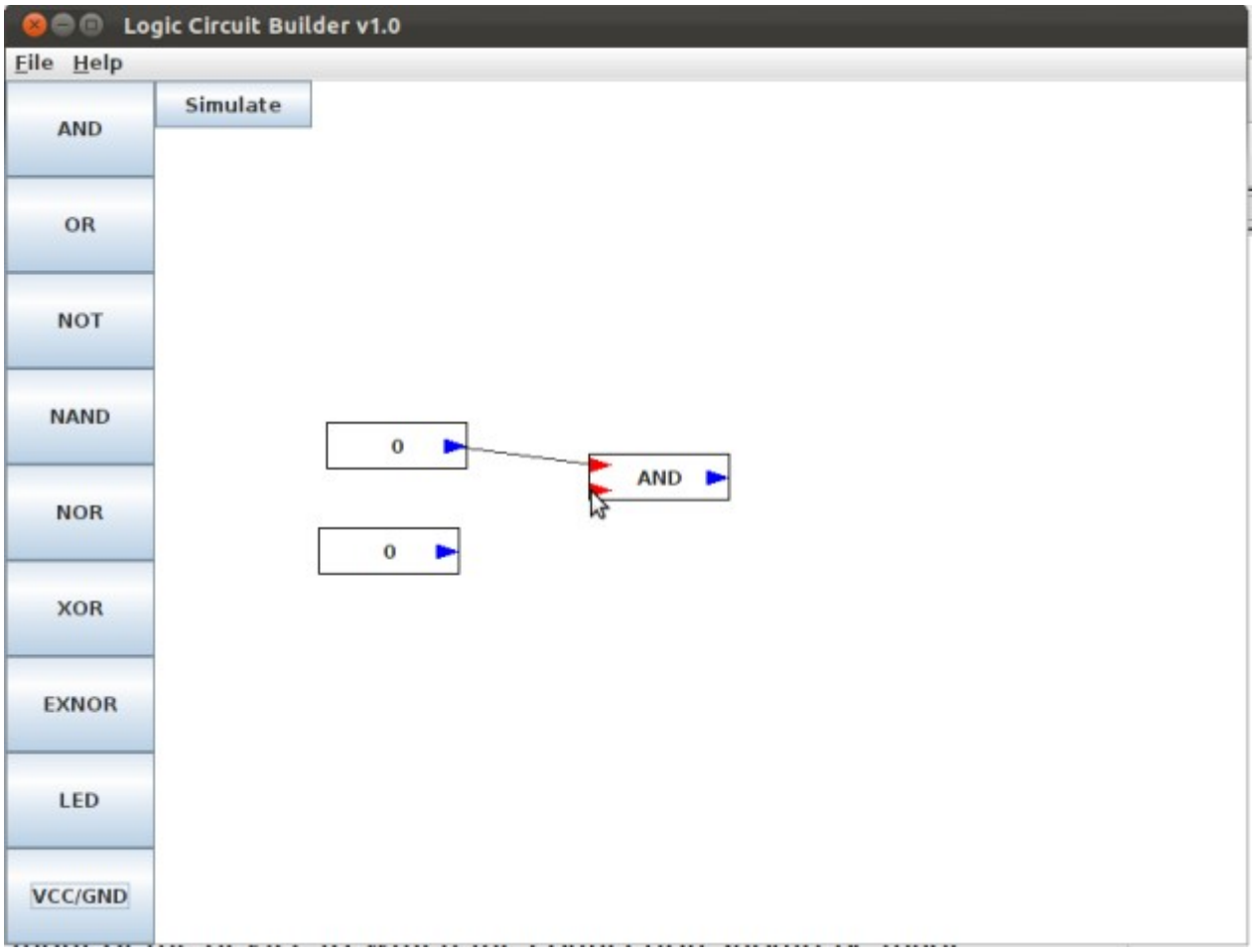
1. Click on a device name from the buttons to the left



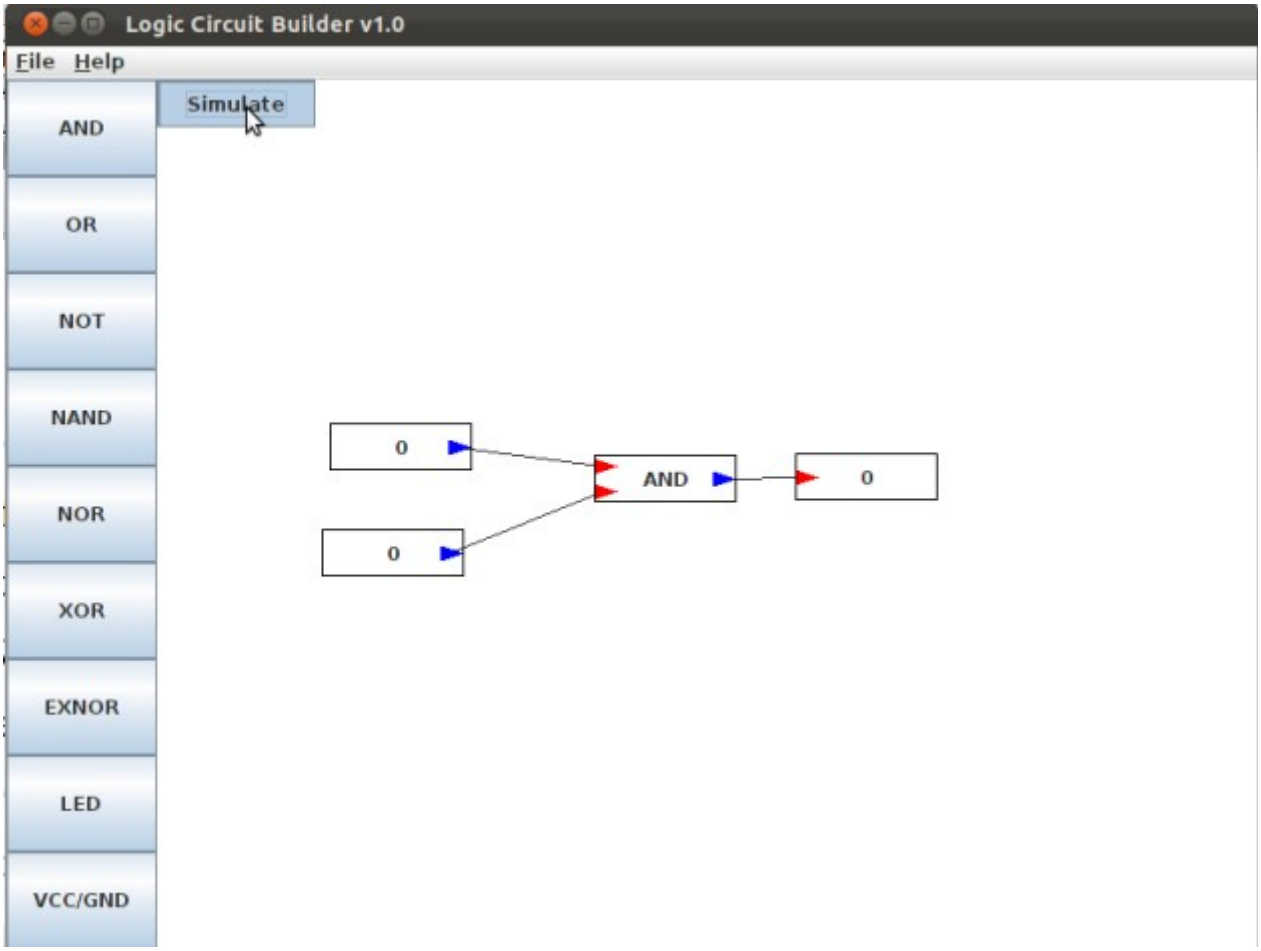
2. Click on the white circuit area in the center to add it to the circuit. Now the device can be freely dragged inside the circuit area.



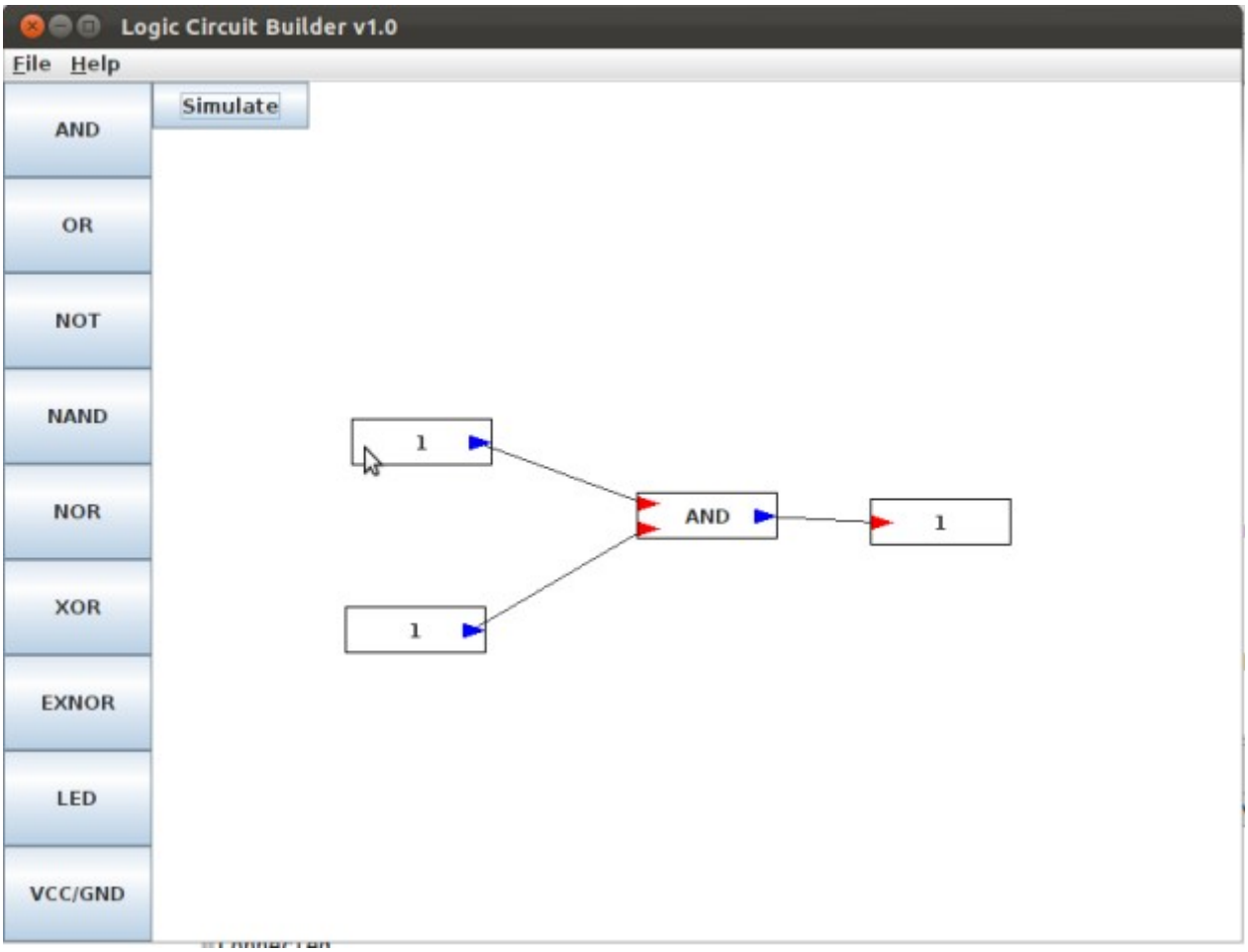
3. Once the devices have been added, connections are made between them by first clicking the output of a device and then clicking on the input of the device to which the connection should be made. VCC/GND toggleable devices are added to set the circuit input. LEDs are used to see the output from the circuit.



4. After the circuit is connected, the Simulate button is clicked.

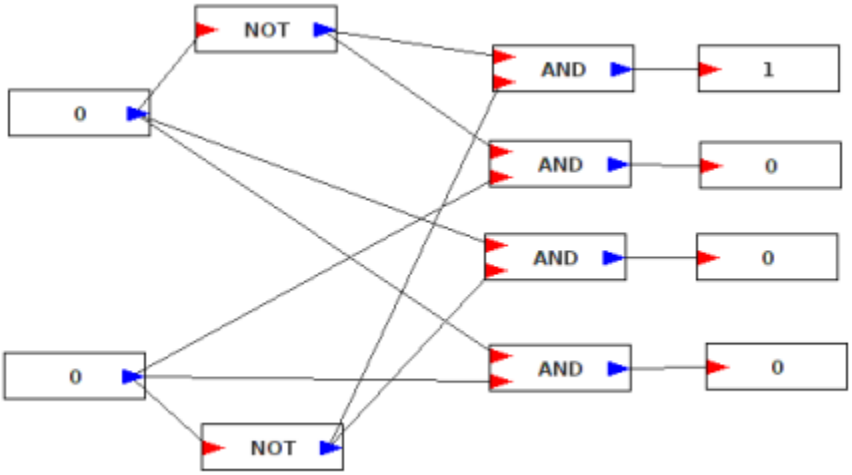


5. The input variables can be toggled by clicking on VCC/GND and the Simulation can be repeated.

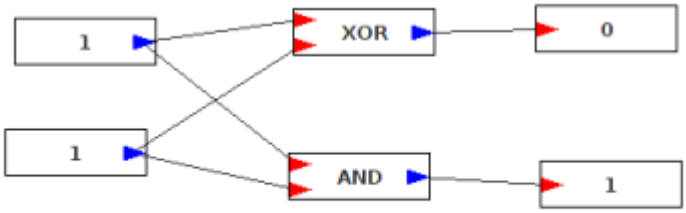


Some sample circuits are shown below.

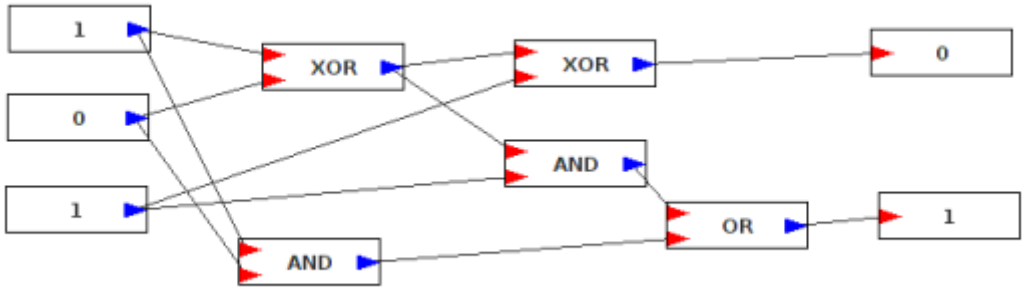
2x4 Decoder



Half Adder



Full Adder



Bibliography

1. **Core Java Volume-I Fundamentals Eighth Edition**
   Author:Cay S.Horstmann, Gary Cornell

2. **Daniweb Programming Community**
   URL:“http://www.daniweb.com/software-development/java/threads / 419761/logic-circuit-sim-dragdrop”

Tools Used

The following tools were used in the making of this project.

1. Eclipse IDE for Java, in coding.
2. Netbeans IDE for Java, in coding.
3. Ubuntu Linux 11.10 Oneiric Ocelot, as a testing platform.
4. LibreOffice Writer, for writing this report.
5. Okular document viewer, for cropping the screenshots for this report.

Code Listing

AndGate.java

```
package logicCircuitBuilder;

import java.awt.Component;


/**An AND Gate does the boolean AND operation on 2 inputs
 * @author vega
 *
 */
public class AndGate extends Device {

	/**
	 * Creates an ANDGate
	 * @param parent It is the CircuitPanel where the device is
added
	 */
	public AndGate(Component parent) {
		super("AND",2,1, parent);
		inputs[0]=new Input(0, 2, this);
		inputs[1]=new Input(0, 18, this);
		outputs[0]=new Output(75, 10,this);
	}

	/**
	 * Implements the AND function on the 2 inputs and sets the
value of outvalue.
	 */
	@Override
	public void deviceFunction() {
		outvalue=(inputs[0].getValue()&&inputs[1].getValue());
	}

}
```

CircuitPanel.java

```
package logicCircuitBuilder;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.ArrayList;

import javax.swing.JButton;
import javax.swing.JPanel;

/**CircuitPanel is the panel where the circuit will be built.
 * @author vega
```

```
 *
 */
public class CircuitPanel extends JPanel implements MouseListener
{
	static ArrayList<LedDevice> leds;
	static ArrayList<Connection> wires;
	static int deviceAdded;
	static boolean isConnecting=false;
	public CircuitPanel() {
		setLayout(null);
		setBackground(Color.white);
		leds=new ArrayList<LedDevice>();
		wires=new ArrayList<Connection>();
		deviceAdded=-1;
		addMouseListener(this);
		JButton simulate=new JButton("Simulate");
		simulate.addActionListener(new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent arg0) {
				for(LedDevice led:leds)
					led.deviceFunction();
			}
		});
		simulate.setBounds(new Rectangle(0,0,100,30));
		add(simulate);

	}
	/**
	 * Adds the devices depending on the value of the variable
deviceAdded.
	 */
	@Override
	public void mouseClicked(MouseEvent arg0) {
		Point P=arg0.getPoint();
		if(deviceAdded==-1)
			return;
		Device device;
		//		device=new AndGate();
		switch (deviceAdded) {
		case Device.AND:
			device=new AndGate(this);
			break;
		case Device.OR:
			device=new OrGate(this);
			break;
		case Device.NOT:
			device=new NotGate(this);
			break;
		case Device.NAND:
			device=new NandGate(this);
			break;
		case Device.NOR:
			device=new NorGate(this);
			break;
```

```
          case Device.XOR:
               device=new XorGate(this);
               break;
          case Device.EXNOR:
               device=new ExnorGate(this);
               break;
          case Device.LED:
               device=new LedDevice(this);
               leds.add((LedDevice)device);
               break;
          case Device.VCC_OR_GND:
               device=new Vcc_GND(this);
               break;
          default:
               device=new AndGate(this);
               break;
          }
          device.setHorizontalAlignment(0);
          add(device);
          device.setBounds(P.x, P.y, Device.devwidth,
Device.devheight);
          deviceAdded=-1;
          this.repaint();

     }


     @Override
     public void mouseEntered(MouseEvent arg0) {


     }
     @Override
     public void mouseExited(MouseEvent arg0) {

     }


     @Override
     public void mousePressed(MouseEvent arg0) {

     }

     @Override
     public void mouseReleased(MouseEvent arg0) {

     }

     /**
      * Paints the connections between the devices by calling the
paintConnection method
      * of the {@link Connection} class for each of the wires
      */
     @Override
     protected void paintComponent(Graphics g) {
```

```
          super.paintComponent(g);
          for(Connection c:wires)
               c.paintConnection((Graphics2D)g);
     }


}
```

Connection.java

```
package logicCircuitBuilder;

import java.awt.Graphics2D;

/**Connection objects are the wires between the devices
 * They are used just for drawing the lines no logic is
implemented through them
 * @author vega
 *
 */
public class Connection {
     Input input;
     Output output;

     /**
      * Creates a connection joining the Input input and Outputd
output
      * @param output The output from which the connection is drawn
      * @param input the input to which the connection is drawn
      */
     public Connection(Output output, Input input) {
          this.output = output;
          this.input = input;
          output.addConnection(this);
          input.addConnection(this);
     }

     /**
      * Paints the line between input and output connectors
      * This is called by the paintComponent() method of the
CircuitPanel
      * @param g2d The Graphics2D object whic is used to paint the
lines.
      */
     public void paintConnection(Graphics2D g2d) {
          g2d.drawLine(output.getX(), output.getY(), input.getX(),
input.getY());
     }

     /**
      * Used to remove a connection when one input has 2 incident
connections
      * @return the input connector to which the connector is
drawn.
      */
```

```
     public Input getInput()
     {
          return input;
     }



}
```

Connector.java

```
package logicCircuitBuilder;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Polygon;
import java.util.ArrayList;

/**Abstract Class for input and output connectors
 * @author vega
 *
 */
public abstract class Connector {
     ArrayList<Connection> cons;
     Device device;
     int x,y;
     Color c;
     Polygon poly;
     public static final int w=15,h=10;

     public Connector(int x,int y,Color c, Device device) {
          this.device=device;
          this.c=c;
          this.x=x;
          this.y=y;
          poly= new Polygon();
          poly.addPoint(x,y);
          poly.addPoint(x+w,y+h/2);
          poly.addPoint(x,y+h);
          cons=new ArrayList<Connection>();
     }
     public boolean contains(Point p){
          return (poly.contains(p));
     }
     public void addConnection(Connection c)
     {
          cons.add(c);
     }
     public void paintConnector(Device owner,Graphics2D g2d){
          g2d.setColor(c);
          g2d.fill(poly);
     }
     /**
      * @return the x
```

```
	 */
	public int getX() {
		return device.getX()+ x+5;
	}
	/**
	 * @return the y
	 */
	public int getY() {
		return device.getY()+ y+5;
	}



}
```

Device.java

```
package logicCircuitBuilder;

import java.awt.Color;
import java.awt.Component;
import java.awt.Cursor;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import javax.swing.JLabel;
import javax.swing.border.LineBorder;
/**Device is any of the devices available to be implemented in the
circuit.
 * The device queries its input which in turn queries the output
connected to it.
 * @author vega
 *
 */
public abstract class Device extends JLabel implements
MouseListener,MouseMotionListener {
	Component parent;
	static Output selectedOutput;//for connecting wires.
	private static final long serialVersionUID = 1L;
	int inno,outno;//number of outputs and inputs
	boolean outvalue;//the output value of the device
	int startDragX,startDragY;//for saving starting position of a
drag
	//so that offset wrt cursor can be maintained
	Input[] inputs;
	Output[] outputs;
	public static final int devwidth=90,devheight=30;
	/**
	 * The constants that are set as values for
CircuitPanel.deviceAdded by the {@link DevicePanel}
```

```
	 */
	public static final int AND=0, OR=1, NOT=2, NAND=3, NOR=4,
XOR=5, EXNOR=6, LED=7, VCC_OR_GND=8;

	/**
	 * Creates a device
	 * @param name The name of the device to be created
	 * @param inno the number of inputs in the device
	 * @param outno the number of outputs in the device
	 * @param parent the panel in which the device is created.
Here the CircuitPanel is given. This allows
	 *                  the device to call the functions of the
CircuitPanel.
	 */
	public Device(String name,int inno,int outno, Component
parent) {
		super(name);
		this.parent=parent;
		setBorder(new LineBorder(Color.black, 1));
		inputs=new Input[inno];
		outputs=new Output[outno];
		addMouseListener(this);
		addMouseMotionListener(this);
	}

	/**
	 * Implements the logic of the device.
	 * Every device will override this function in order to
implement its logic.
	 */
	public abstract void deviceFunction();

	/**
	 * Paints the input and output connectors of the device
	 */
	@Override
	protected void paintComponent(Graphics arg0) {
		for(Input in : inputs)
			in.paintConnector(this, (Graphics2D)arg0);
		for(Output out : outputs)
			out.paintConnector(this, (Graphics2D) arg0);
		super.paintComponent(arg0);
		parent.repaint();
	}

	/**
	 * Recognises mouse clicks on the input and output connectors
and performs necessary functions
	 */
	@Override
	public void mouseClicked(MouseEvent arg0) {
		Point p=arg0.getPoint();
		if(selectedOutput!=null)
			for(Input in:inputs)
				if(in.contains(p))
```

```
                    {
                         System.out.println("Connected");
                         for(Connection c:CircuitPanel.wires)
                              if(c.getInput()==in)
                              {
                                   CircuitPanel.wires.remove(c);
                              }

                         CircuitPanel.wires.add(new
Connection(selectedOutput, in));
                         in.connectTo(selectedOutput);
                         selectedOutput=null;
                         return;
                    }
               for(Output out:outputs)
                    if(out.contains(p))
                    {
                         System.out.println("Clicked on output");
                         selectedOutput=out;
                    }
          }
          @Override
          public void mouseEntered(MouseEvent arg0) {

          }

          @Override
          public void mouseExited(MouseEvent arg0) {

          }

          @Override
          /**
           * Sets the starting coordinates of the cursor relative to the
Device coordinates so that the offset with
           * respect to the cursor can be maintained. Also sets the
cursor to a hand to indicate dragging.
           */
          public void mousePressed(MouseEvent arg0) {
               //events for start of a drag that may or may not occur
               Point p=arg0.getPoint();
               startDragX = arg0.getX();
               startDragY = arg0.getY();

          setCursor(Cursor.getPredefinedCursor(Cursor.MOVE_CURSOR));

          }

          /**
           * Sets the cursor back to the default cursor
           */
          @Override
          public void mouseReleased(MouseEvent arg0) {
               setCursor(Cursor.getDefaultCursor());
```

```
     }

     /**Implements the device drag operation The mouse coordinates
are obtained and the device is
      * set at the new location using the setLocation() function.
      *
      */
     @Override
     public void mouseDragged(MouseEvent e) {
          int newX = getX() + (e.getX() - startDragX);
          int newY = getY() + (e.getY() - startDragY);
          setLocation(newX, newY);
          repaint();

     }

     /**
      * Sets the cursor to a crosshair when it is over the input
and output connectors for easy identification
      * by the user. Else sets the cursor to the default cursor.
      */
     @Override
     public void mouseMoved(MouseEvent arg0) {
          /*for setting crosshair cursor inside the connectors so
that they can be easily identified*/
          Point p=arg0.getPoint();
          for(Input in:inputs)
               if(in.contains(p))
               {


setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
                    return;
               }
          for(Output out:outputs)
               if(out.contains(p))
               {


setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
                    return;
               }
          setCursor(Cursor.getDefaultCursor());
     }
}
```

DevicePanel.java

```
package logicCircuitBuilder;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
```

```
import javax.swing.JPanel;
/**DevicePanel is the panel on the west where the device buttons
are placed.
 * Clicking on the button and clicking on the CircuitPanel adds
the corresponding device at the click location
 * @author vega
 *
 */
public class DevicePanel extends JPanel implements ActionListener
{
     JButton[] DeviceButtons;
     public DevicePanel() {
          DeviceButtons=new JButton[9];
          DeviceButtons[0]=new JButton("AND");
          DeviceButtons[1]=new JButton("OR");
          DeviceButtons[2]=new JButton("NOT");
          DeviceButtons[3]=new JButton("NAND");
          DeviceButtons[4]=new JButton("NOR");
          DeviceButtons[5]=new JButton("XOR");
          DeviceButtons[6]=new JButton("EXNOR");
          DeviceButtons[7]=new JButton("LED");
          DeviceButtons[8]=new JButton("VCC/GND");

          setLayout(new GridLayout(DeviceButtons.length,1));
          for(int i=0;i<DeviceButtons.length;i++){
               DeviceButtons[i].addActionListener(this);
               add(DeviceButtons[i]);
          }
     }

     /**
      * Clicking on a device button sets the value of {@link
CircuitPanel}.deviceAdded
      */
     @Override
     public void actionPerformed(ActionEvent arg0) {
          String name=((JButton)arg0.getSource()).getText();
          if(name.equals("AND"))
               CircuitPanel.deviceAdded=Device.AND;
          else if(name.equals("OR"))
               CircuitPanel.deviceAdded=Device.OR;
          else if(name.equals("NOT"))
               CircuitPanel.deviceAdded=Device.NOT;
          else if(name.equals("NAND"))
               CircuitPanel.deviceAdded=Device.NAND;
          else if(name.equals("NOR"))
               CircuitPanel.deviceAdded=Device.NOR;
          else if(name.equals("XOR"))
               CircuitPanel.deviceAdded=Device.XOR;
          else if(name.equals("EXNOR"))
               CircuitPanel.deviceAdded=Device.EXNOR;
          else if(name.equals("LED"))
               CircuitPanel.deviceAdded=Device.LED;
          else if(name.equals("VCC/GND"))
               CircuitPanel.deviceAdded=Device.VCC_OR_GND;
```

```
	}
}
```

ExnorGate.java

```
package logicCircuitBuilder;

import java.awt.Component;

/**EXNOR Gate implements the boolean complemented Exclusive OR
(equivalence) function for 2 inputs
 * @author vega
 *
 */
public class ExnorGate extends Device {
	/**
	 * Makes an EXNOR gate
	 * @param parent CircuitPanel object
	 */
	public ExnorGate(Component parent) {
		super("EXNOR", 2, 1, parent);
		inputs[0]=new Input(0, 2, this);
		inputs[1]=new Input(0, 18, this);
		outputs[0]=new Output(75,10, this);
	}

	@Override
	public void deviceFunction() {
		outvalue=(!(inputs[0].getValue()^inputs[1].getValue()));

	}

}
```

Input.java

```
package logicCircuitBuilder;

import java.awt.Color;

/**Input connectors are used to give input to the gates
 * @author vega
 *
 */
public class Input extends Connector{
	Output out;
	boolean state;
	/**
	 * @return the output to which the input is connected to.
	 */
	public Output getOut() {
		return out;
	}
	/**
```

```
      * @return the state of the Input by querying the output to
which it is connected
      */
     public boolean getValue() {
          state=out.getValue();
          return state;
     }

     /**
      * Makes an Input connector
      * @param x The x coordinate
      * @param y The y coordinate
      * @param device the parent device to which the Input belongs
      */
     public Input(int x,int y, Device device) {
          super(x,y,Color.red, device);
          out=null;
     }

     /**
      * Connects the Input
      * @param out the output to which the input must be connected
      */
     public void connectTo(Output out){
          this.out=out;
     }
}
```

LedDevice.java

```
package logicCircuitBuilder;

import java.awt.Component;

import javax.swing.Icon;

/**LED is used to see the output of the circuit. The simulate
button on the circuitpanel will
 * call the deviceFunction() method of the LED to simulate the
circuit
 * @author vega
 *
 */
public class LedDevice extends Device {
     boolean state=false;//to denote on/off state

     /**
      * makes an LED
      * @param parent CircuitPanel
      */
     public LedDevice(Component parent) {
          super("0", 1, 0, parent);
          inputs[0]=new Input(0, 10, this);
     }
     /**
```

```
     * Sets the state of the LED by querying its input.
     */
    @Override
    public void deviceFunction() {
        if(inputs[0].getValue())
            setText("1");
        else
            setText("0");
    }

}
```

LogicCircuitBuilder.java

```
package logicCircuitBuilder;

import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;

/**Main class for the Logic Circuit Builder program.
 * It makes the JFrame and has the main() function.
 * @author vega
 *
 */
public class LogicCircuitBuilder {
    /**
     * The main class
     * @param args No arguments are accepted
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable()
        {
            @Override
            public void run()
            {
                LogicCircuitBuilderFrame frame = new
LogicCircuitBuilderFrame();
                frame.setSize(800, 600);
                frame.setExtendedState(Frame.MAXIMIZED_BOTH);


frame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE
);//closing handled by windowClosing
                frame.setVisible(true);
                frame.showInstructions();
            }
        });
    }
```

```
}
class LogicCircuitBuilderFrame extends JFrame
{
     LogicCircuitBuilderFrame()
     {
          addWindowListener(new WindowAdapter()
          {
               @Override
               public void windowClosing(WindowEvent e) {
                    super.windowClosing(e);
                    showCloseWarning();
               }
          });
          setLocationRelativeTo(null);
          setTitle("Logic Circuit Builder v1.0");
          final JMenu fileMenu = new JMenu("File");
          fileMenu.setMnemonic('F');

          JMenuItem New = fileMenu.add("New");
          New.setAccelerator(KeyStroke.getKeyStroke("ctrl N"));
          New.setMnemonic('N');
          New.addActionListener(new ActionListener() {

               @Override
               public void actionPerformed(ActionEvent arg0) {
                    remove(eastPanel);
                    eastPanel=new CircuitPanel();
                    add(eastPanel,BorderLayout.CENTER);
                    validate();
               }
          });
          JMenuItem Exit = fileMenu.add("Exit");
          Exit.setMnemonic('X');
          Exit.setAccelerator(KeyStroke.getKeyStroke("ctrl X"));
          Exit.addActionListener(new ActionListener() {

               @Override
               public void actionPerformed(ActionEvent arg0) {
                    showCloseWarning();

               }
          });
          JMenu helpMenu=new JMenu("Help");
          helpMenu.setMnemonic('H');
          JMenuItem Instructions=new JMenuItem("Instructions");
          Instructions.addActionListener(new ActionListener() {
               @Override
               public void actionPerformed(ActionEvent arg0) {
                    showInstructions();

               }
          });
          Instructions.setMnemonic('I');

     Instructions.setAccelerator(KeyStroke.getKeyStroke("F1"));
```

```
          helpMenu.add(Instructions);
          JMenuItem About=new JMenuItem("About");
          About.addActionListener(new ActionListener() {
               @Override
               public void actionPerformed(ActionEvent arg0) {
                    String about="Made by Abhijith V Mohan-
10400EN001\n& Prakhar Jain-10400EN002\n"
                              +"IDD 2nd Year, Computer Science, IT-
BHU";

     JOptionPane.showMessageDialog(eastPanel,about,"About Logic
Circuit Simulator v.1.0",JOptionPane.INFORMATION_MESSAGE);
               }
          });
          About.setMnemonic('A');
          helpMenu.add(About);

          JMenuBar menubar = new JMenuBar();
          setJMenuBar(menubar);
          menubar.add(fileMenu);
          menubar.add(helpMenu);
          westPanel = new DevicePanel();
          eastPanel = new CircuitPanel();
          setLayout(new BorderLayout());
          add(westPanel, BorderLayout.WEST);
          add(eastPanel, BorderLayout.CENTER);
     }
     /**
      * Shows the instructions Dialog Box
      */
     public void showInstructions(){
          String instructions="-Click on device name in the left
toolbar\n"+
                    "-Click on Circuit Area to the right to add the
device\n"+
                    "-Click on input connector and connect them to
output of another device\n"+
                    "-Please don't make any loop structure--
sequential circuits are unsupported as of now\n"+
                    "-Click on Simulate after circuit is
complete\n";




JOptionPane.showMessageDialog(eastPanel,instructions,"Instructions
",JOptionPane.INFORMATION_MESSAGE);
     }
     /**
      * Shows a confirmation dialog box whether to exit
      */
     public void showCloseWarning(){
          String message="Are you sure you want to exit?";
          int choice=JOptionPane.showConfirmDialog(eastPanel,
message, "Confirm exit", JOptionPane.YES_NO_OPTION,
```

```
JOptionPane.QUESTION_MESSAGE);
          if(choice==JOptionPane.YES_OPTION)
               System.exit(0);
     }
     JPanel westPanel, eastPanel;
}
```

NandGate.java

```
package logicCircuitBuilder;

import java.awt.Component;

/**A NAND gate implements the boolean NAND function on 2 inputs
 * @author vega
 *
 */
public class NandGate extends Device {
     /**
      * Creates a NAND Gate
      * @param parent CircuitPanel
      */
     public NandGate(Component parent) {
          super("NAND", 2, 1, parent);
          inputs[0]=new Input(0, 2, this);
          inputs[1]=new Input(0, 18, this);
          outputs[0]=new Output(75, 10,this);
     }

     @Override
     public void deviceFunction() {
          outvalue=(!(inputs[0].getValue()&&inputs[1].getValue()));

     }

}
```

NorGate.java

```
package logicCircuitBuilder;

import java.awt.Component;

/**A NOR gate implements the boolean NOR function for 2 inputs
 * @author vega
 *
 */
public class NorGate extends Device {
     /**
      * Creates a NOR gate
      * @param parent CircuitPanel
      */
     public NorGate(Component parent) {
          super("NOR", 2, 1, parent);
```

```
          inputs[0]=new Input(0, 2, this);
          inputs[1]=new Input(0, 18, this);
          outputs[0]=new Output(75, 10,this);
     }

     @Override
     public void deviceFunction() {
          outvalue=(!(inputs[0].getValue()||inputs[1].getValue()));
     }

}
```

NotGate.java

```
package logicCircuitBuilder;

import java.awt.Component;

/**A NOT gate implements the boolean NOT function.
 * @author vega
 *
 */
public class NotGate extends Device{
     /**
      * Creates a NotGate
      * @param parent CircuitPanel
      */
     public NotGate(Component parent) {
          super("NOT",1,1, parent);
          inputs[0]=new Input(0, 10, this);
          outputs[0]=new Output(75, 10,this);
     }

     @Override
     public void deviceFunction() {
          outvalue=(!inputs[0].getValue());

     }

}
```

OrGate.java

```
package logicCircuitBuilder;

import java.awt.Component;

/**An OR Gate implements the boolean OR function for 2 inputs
 * @author vega
 *
 */
public class OrGate extends Device {
     /**Creates a NOR Gate
      *
```

```
     * @param parent CircuitPanel
     */
    public OrGate(Component parent) {
        super("OR", 2, 1, parent);
        inputs[0]=new Input(0, 2, this);
        inputs[1]=new Input(0, 18, this);
        outputs[0]=new Output(75, 10,this);

    }

    @Override
    public void deviceFunction() {
        outvalue=(inputs[0].getValue()||inputs[1].getValue());

    }

}
```

Output.java

```
package logicCircuitBuilder;

import java.awt.Color;

/**Output connectors are used to take output from devices. They
are queried by the inputs to which they are connected
 * and they in turn query the device for the value of output
 * @author vega
 *
 */
public class Output extends Connector{
    boolean value;
    /**
     * Creates an Output connector
     * @param x X coordinate
     * @param y Y coordinate
     * @param device the parent device
     */
    public Output(int x, int y,Device device) {
        super(x, y,Color.blue, device);
        value=false;
    }
    /**Next Input will get state from here
     * @return the state
     */
    public boolean getValue() {
        device.deviceFunction();
        value=device.outvalue;
        return value;
    }
}
```

Vcc_GND.java

```
package logicCircuitBuilder;

import java.awt.Component;
import java.awt.event.MouseEvent;

/**VCC_GND is a device which gives a toggleable HIGH/LOW output
 * @author vega
 *
 */
public class Vcc_GND extends Device {
     boolean state;
     /**
      * Creates a toggleable VCC/GND device
      * @param parent CircuitPanel
      */
     public Vcc_GND(Component parent) {
          super("1", 0, 1, parent);
          state=true;
          outputs[0]=new Output(75, 10,this);
     }

     @Override
     public void deviceFunction() {
          outvalue=state;
     }

     /**
      * Toggles VCC/GND
      */
     @Override
     public void mouseClicked(MouseEvent arg0) {
          super.mouseClicked(arg0);
          state=!state;
          if(state)
               setText("1");
          else
               setText("0");
     }


}
```

XorGate.java

```
package logicCircuitBuilder;

import java.awt.Component;

/**A XOR gate implements the boolean Exclusive OR function for 2
inputs
 * @author vega
 *
```

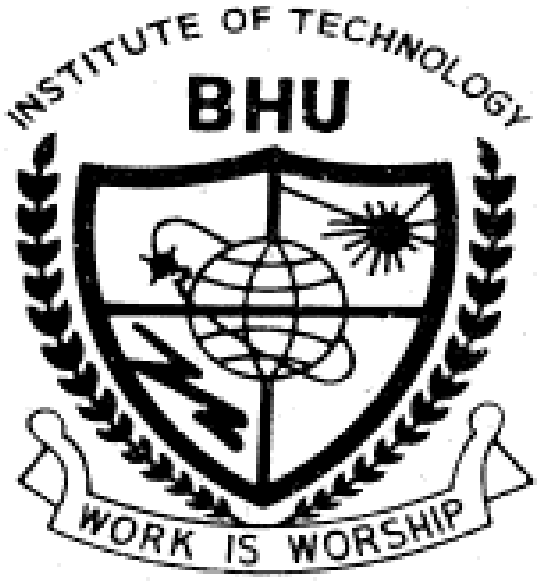
```
 */
public class XorGate extends Device {
	/**
	 * Creates a XOR Gate
	 * @param parent
	 */
	public XorGate(Component parent) {
		super("XOR", 2, 1, parent);
		inputs[0]=new Input(0, 2, this);
		inputs[1]=new Input(0, 18, this);
		outputs[0]=new Output(75, 10,this);
	}

	@Override
	public void deviceFunction() {
		outvalue=(inputs[0].getValue()^inputs[1].getValue());

	}

}
```

Logic Circuit Simulator



Done By,

Prakhar Jain

10400EN002

IDD 2nd CSE

ITBHU

Certificate

This is to certify that Prakhar Jain, Roll No. 10400EN002 of IDD Computer Science, 2nd Year, 4th Semester has successfully completed this project as part of their Software Project Lab (CS-2403).

Dr. Vinayak Srivastava
Asst. Professor