# NWU®

# Faculty of Engineering

## REII 313
ANALOG PRACTICAL

Submitted to:        RUAN LUIES

Date:                17/06/2020

Author:              Chrizanne Pieterse 30171342

                     Ilanika Schalekamp 29955440

                     Walter Koekemoer  28464230

**North-West University: Potchefstroom Campus**

# Table of Contents

# 1. Contents

# List of Figures

## 2. Introduction

The purpose of an analog simulation is to calculate a circuit's voltages and currents in accordance with Kirchoff's Current Law (KCL) and Kirchhoffs Voltage Law (KVL). The analog model takes independent simulation variables as input and computes the dependent variables according to their behaviour. The results are then returned to the system matrix to determine if a solution has been achieved.

## 3. Background

An analog simulation is a process where a physical electric circuit is designed by the user and implemented into the program to virtually display and understand the circuit. Typically an analog simulation process starts with the network listing, that describes the various components in the simulation and how they are connected to each other, followed by the simulation process, which produces the voltage and current results as a function of time and/or frequency and lastly the post-processing tools that can be used to calculate quantities such as power.
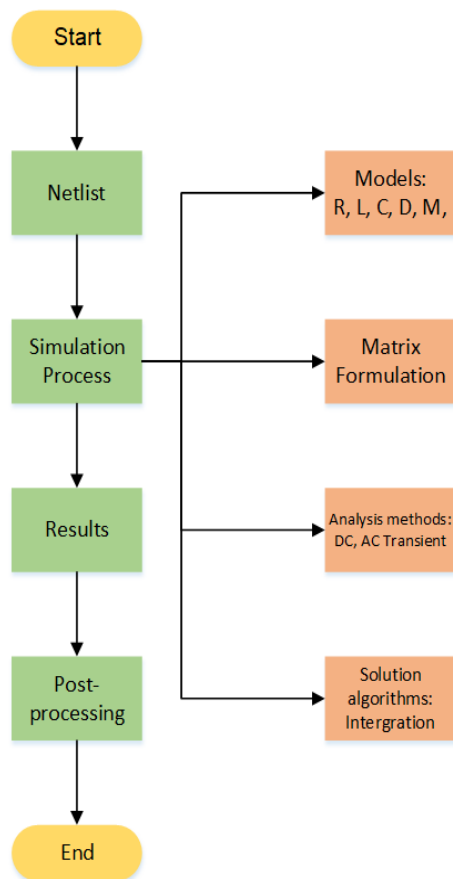


**FIGURE 2: FLOWCHART INDICATING AN OVERVIEW OF THE SIMULATION PROCESS.**

As seen in figure 1 the simulation process is able to instantiate component models. The models are then stamped into a matrix in order to perform various analyses. The order of the matrix is determined by the number of nodes in the circuit and is formed through matrix stamping. Stamping involves "stamping" the model's matrix element based on how the system is connected. The analysis requires solution algorithms for linear, non-linear and integration systems.

# 4. Methodology

The Waterfall software development model were used to describe the lifecycle of the analog practical because the requirements of the project are known in advance. Each phase in the model had to be completed before the next phase could start.

The lifecycle started with ideation where ideas were brainstormed between group members on how to solve this particular problem. Analog simulations replicate the behaviour of actual electronic circuits which in turn can solve various problems as it provides users with an analytical tool that makes it possible to visualize, test and evaluate electric circuits. When simulated, a circuit's efficiency can be improved by identifying faults in the design.

The practical's software requirements include any PC, and software such as LT SPICE, Qt and Git. To build the analog simulation from the start the simulator requires knowledge of Nodal analysis, Linearization, Numerical integration and matrix solving techniques. Nodal analysis is used to solve a circuit with linear, non-differential equations formulated in accordance with Kirchoff's current law. The simulator then uses matrix solving techniques in order to determine the solution to the set of linear equations. In the case were a user makes use of the analog simulator program the requirements from the user are to define voltages and currents as input into the simulation to be able to visualize the circuit.

The design stage included, creating the architecture of the software system and its elements. Figure 2, the UML (unified modelling language) visually represents the relationship between components making up the design of the analog electronic system and figure 1 gives an overview of the simulation process.

The development stage consisted of the development team building the program and writing the algorithm, the team used C++ as the main programming language in Qt and uploaded the program to Git. The coding activities included code reviews to ensure the code written by the members lived up to standards.

Testing the program included evaluating the quality of the software in order to identify defects and to see if the program were implemented correctly. Three types of tests were done namely the unit test, integrated test and system test. The Unit test were done by the group members, individually. After a piece of code were written the code were tested to see if the code satisfied the design. The integration test was conducted when the different pieces of code were put together to see if the code still executes correctly.  The system test was done to validate that the analog simulation implemented were in line with the system requirements. More specifically if Nodal analysis, Linearization, Numerical integration and matrix solving techniques were used to solve Kirchoff's voltage and current laws and if the user is able to define currents and voltages of their own in a GUI environment in order to visualize the circuit. A load performance test was conducted to ensure that the program is also able to run for small RLC, RL and RC circuits. The performance test will be discussed in more detail later in the report under heading: Tests. The user acceptance test was done by presenting the program to a user not part of the group.

Deployment included preparing our analog program to run in any environment by making final adjustments to the program and uploading our changes on GIT.

The last stage of the software development cycle is the maintenance of the program. This can only be done if the customer (lecturer) evaluated our program and had given us feedback. Then the team would be able to update the analog electronic simulation program
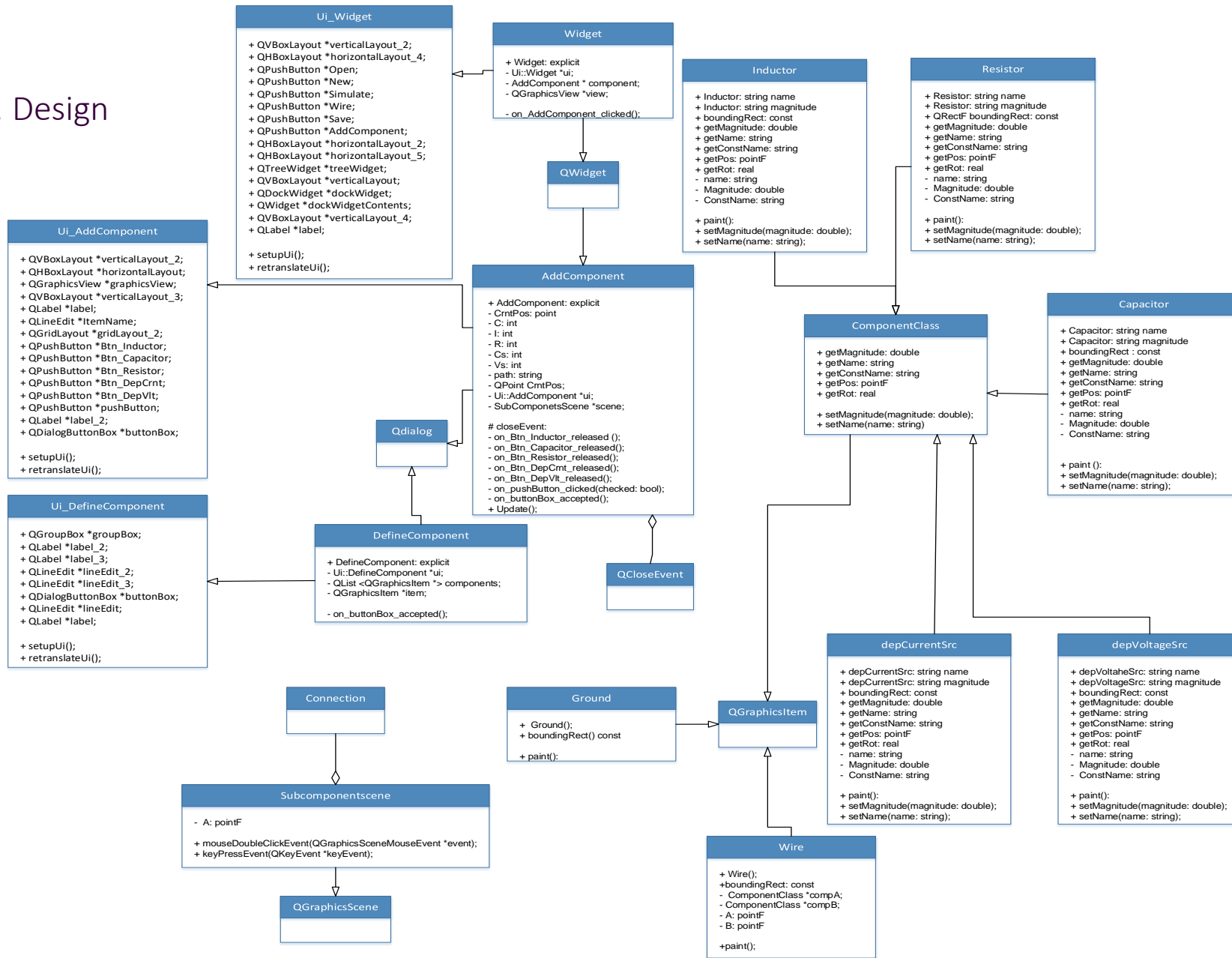
# 5. Design

**Ui_Widget**
- + QVBoxLayout *verticalLayout_2;
- + QHBoxLayout *horizontalLayout_4;
- + QPushButton *Open;
- + QPushButton *New;
- + QPushButton *Simulate;
- + QPushButton *Wire;
- + QPushButton *Save;
- + QPushButton *AddComponent;
- + QHBoxLayout *horizontalLayout_2;
- + QHBoxLayout *horizontalLayout_5;
- + QTreeWidget *treeWidget;
- + QVBoxLayout *verticalLayout;
- + QDockWidget *dockWidget;
- + QWidget *dockWidgetContents;
- + QVBoxLayout *verticalLayout_4;
- + QLabel *label;

- + setupUi();
- + retranslateUi();

**Widget**
- + Widget: explicit
- - Ui::Widget *ui;
- - AddComponent * component;
- - QGraphicsView *view;

- - on_AddComponent_clicked();

**QWidget**

**Inductor**
- + Inductor: string name
- + Inductor: string magnitude
- + boundingRect: const
- + getMagnitude: double
- + getName: string
- + getConstName: string
- + getPos: pointF
- + getRot: real
- - name: string
- - Magnitude: double
- - ConstName: string

- + paint();
- + setMagnitude(magnitude: double);
- + setName(name: string);

**Resistor**
- + Resistor: string name
- + Resistor: string magnitude
- + QRectF boundingRect: const
- + getMagnitude: double
- + getName: string
- + getConstName: string
- + getPos: pointF
- + getRot: real
- - name: string
- - Magnitude: double
- - ConstName: string

- + paint();
- + setMagnitude(magnitude: double);
- + setName(name: string);

**Ui_AddComponent**
- + QVBoxLayout *verticalLayout_2;
- + QHBoxLayout *horizontalLayout;
- + QGraphicsView *graphicsView;
- + QVBoxLayout *verticalLayout_3;
- + QLabel *label;
- + QLineEdit *ItemName;
- + QGridLayout *gridLayout_2;
- + QPushButton *Btn_Inductor;
- + QPushButton *Btn_Capacitor;
- + QPushButton *Btn_Resistor;
- + QPushButton *Btn_DepCrnt;
- + QPushButton *Btn_DepVlt;
- + QPushButton *pushButton;
- + QLabel *label_2;
- + QDialogButtonBox *buttonBox;

- + setupUi();
- + retranslateUi();

**AddComponent**
- + AddComponent: explicit
- - CrntPos: point
- - C: int
- - I: int
- - R: int
- - Cs: int
- - Vs: int
- - path: string
- - QPoint CrntPos;
- - Ui::AddComponent *ui;
- - SubComponetsScene *scene;

- # closeEvent:
- - on_Btn_Inductor_released ();
- - on_Btn_Capacitor_released();
- - on_Btn_Resistor_released();
- - on_Btn_DepCrnt_released();
- - on_Btn_DepVlt_released();
- - on_pushButton_clicked(checked: bool);
- - on_buttonBox_accepted();
- + Update();

**ComponentClass**
- + getMagnitude: double
- + getName: string
- + getConstName: string
- + getPos: pointF
- + getRot: real

- + setMagnitude(magnitude: double);
- + setName(name: string)

**Capacitor**
- + Capacitor: string name
- + Capacitor: string magnitude
- + boundingRect : const
- + getMagnitude: double
- + getName: string
- + getConstName: string
- + getPos: pointF
- + getRot: real
- - name: string
- - Magnitude: double
- - ConstName: string

- + paint ();
- + setMagnitude(magnitude: double);
- + setName(name: string);

**Qdialog**

**Ui_DefineComponent**
- + QGroupBox *groupBox;
- + QLabel *label_2;
- + QLabel *label_3;
- + QLineEdit *lineEdit_2;
- + QLineEdit *lineEdit_3;
- + QDialogButtonBox *buttonBox;
- + QLineEdit *lineEdit;
- + QLabel *label;

- + setupUi();
- + retranslateUi();

**DefineComponent**
- + DefineComponent: explicit
- - Ui::DefineComponent *ui;
- - QList <QGraphicsItem *> components;
- - QGraphicsItem *item;

- - on_buttonBox_accepted();

**QCloseEvent**

**Connection**

**Ground**
- + Ground();
- + boundingRect() const

- + paint():

**QGraphicsItem**

**depCurrentSrc**
- + depCurrentSrc: string name
- + depCurrentSrc: string magnitude
- + boundingRect: const
- + getMagnitude: double
- + getName: string
- + getConstName: string
- + getPos: pointF
- + getRot: real
- - name: string
- - Magnitude: double
- - ConstName: string

- + paint();
- + setMagnitude(magnitude: double);
- + setName(name: string);

**depVoltageSrc**
- + depVoltaheSrc: string name
- + depVoltageSrc: string magnitude
- + boundingRect: const
- + getMagnitude: double
- + getName: string
- + getConstName: string
- + getPos: pointF
- + getRot: real
- - name: string
- - Magnitude: double
- - ConstName: string

- + paint();
- + setMagnitude(magnitude: double);
- + setName(name: string);

**Subcomponentscene**
- - A: pointF

- + mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event);
- + keyPressEvent(QKeyEvent *keyEvent);

**QGraphicsScene**

**Wire**
- + Wire();
- +boundingRect: const
- - ComponentClass *compA;
- - ComponentClass *compB;
- - A: pointF
- - B: pointF

- +paint();

**Figure 2: UML Diagram**

The UML in figure 2 is a top-level view of the system. After writing the program the UML were drawn and is therefore a backward design.

# 6. Tests

The analog simulator were tested by critically evaluating the working of each component in the program. Black-box testing is the method that is used to test software, that examine the functionality of an application. The method of this testing can be functional to the unit, integration, system, and acceptance. Test cases are built around constraints and requirements. Black-box test technique include decision table testing, all-pair testing, equivalence partitioning, boundary value analysis, cause effect graph, use case testing and many more. The goal of the black-box penetration test is to simulate an external hacking or cyber warfare attack.

## 6.1. Display saved components

On the right of the displayed eBui window a list of the user's saved components should appear under the heading: My Components. The yellow envelope enables the user to open previously saved schematics.
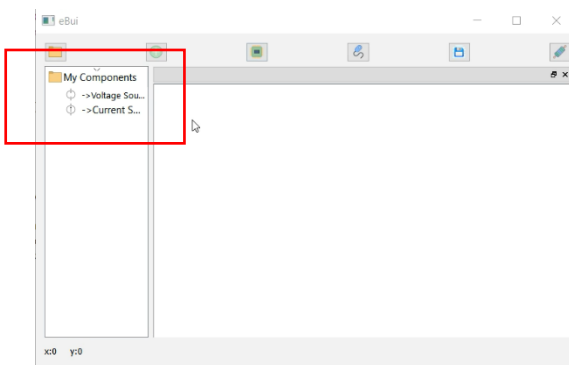


**FIGURE 3: SAVED CIRCUIT LIST**

As seen in figure 3 the list is displayed, and the user can open a component previously created by clicking on the saved item of choice.

## 6.2. Simulate
This icon enables the user to simulate their circuit.

## 6.3. Wire
The wire icon enables the user to connect all the components in the scene to each other.

## 6.4. Save
The blue envelope enables the user to save the circuit currently in the scene.

## 6.5. Clear view
The globe icon should be able to clear the whole scene view.

## 6.6. Build

The build icon should take the user to the dialog window where they should be able to create their own schematic of a circuit. A choice between six electric components will be displayed and the user should be able to click on either of the components to add them to the dialog scene. After the user clicked on the component, they want to add to the dialog the component should appear in the scene. If the user double clicks on the component in the scene a second, but smaller, dialog box should appear and the user are presented with: the component's default name, an input line where the name of the component can be changed, an input line asking the magnitude of the component and a "Dependent on" input line. The user should be able to give their own input of the specific component. If any other component but the dependent source the "Dependent on" input line would not be necessary to fill in. When the user clicks the OK button the selected components should be displayed in the dialog scene with the relevant information to the right of the component. If two or more components were selected and present in the dialog scene and the user wants to connect the components, they would click on the wire component and double click on the first component's edge to be connected followed by double clicking on the second component's edge.
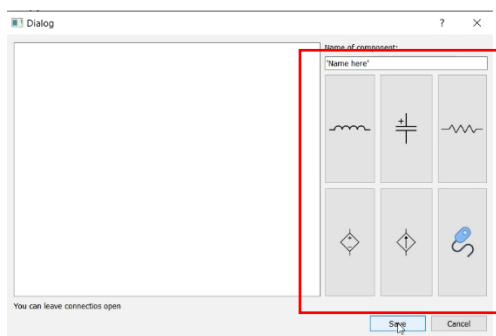


**FIGURE 4: COMPONENT CHOICES**
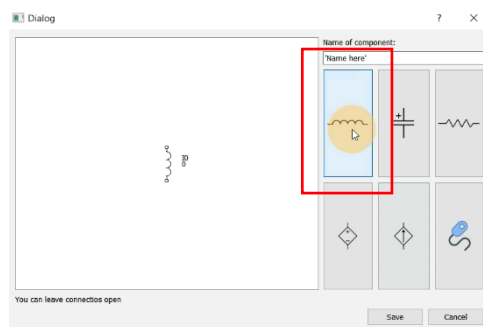


**FIGURE 5: USER IDENTIFYING WHICH COMPONENT TO ADD TO THE SCENE**

As seen in figure 4 the user is presented with 6 components to choose from to build a circuit. Figure 5 shows that the user can click on a chosen component and the component is displayed in the dialog scene. As showed in figure 7 the user can give their own input parameters of the specific component.
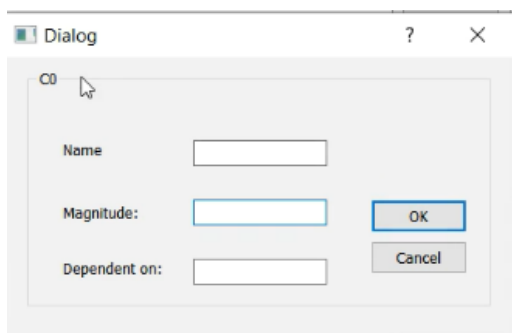


**FIGURE 6: SECOND DIALOG BOX**
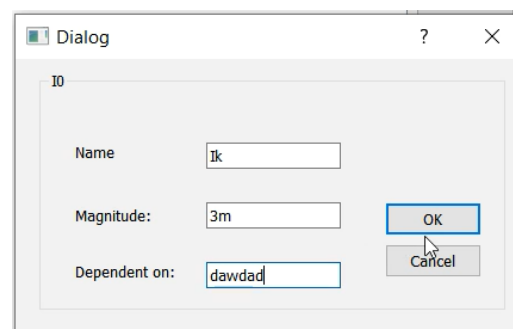


**FIGURE 7: USER INPUT**
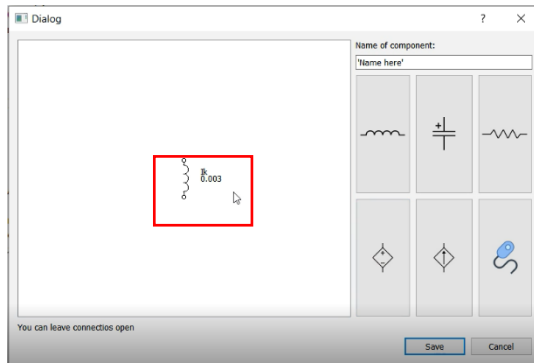
**FIGURE 8: COMPONENT DISPLAYED IN THE DIALOG SCENE WITH RELEVANT INFORMATION**
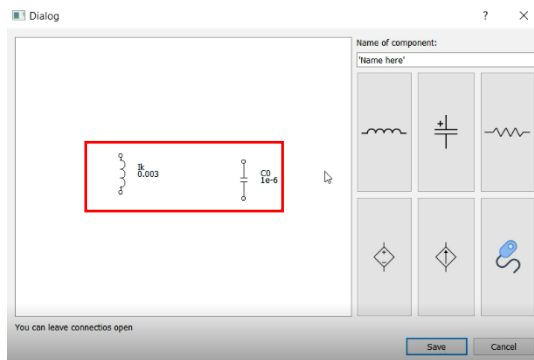


**FIGURE 9: SECOND COMPONENT ADDED TO THE DIALOG SCENE**

Figure 8 shows how the parameters the user had input in figure 7 displays to the right of the component in the dialog scene. Figure 9 shows that a second component can be added to the dialog scene.
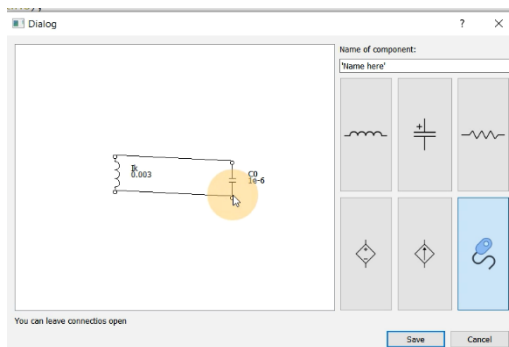


**FIGURE 10: COMPONENTS CONNECTED**

Figure 10 shows how the user was able to connect the two components to each other by using the wire component.

9

## 6.7. Moving components

The user should be able to click on a component they would like to move, and the wire should still be connected to each edge of the component.
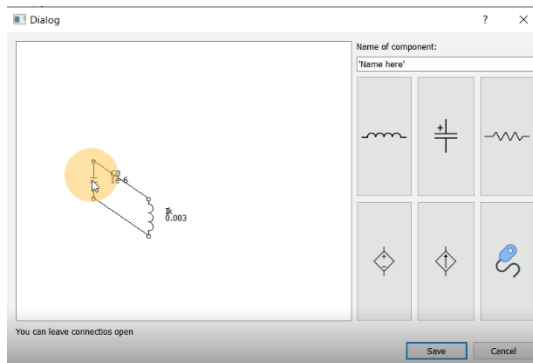


**FIGURE 11: MOVING COMPONENTS**

As seen in figure 11 the user can move the components around at any angle and the wire will remain connected to the edges of the components.

## 6.8. Changing component information

If a user wants to alter a component's information, they should be able to double click on the component and change the information.
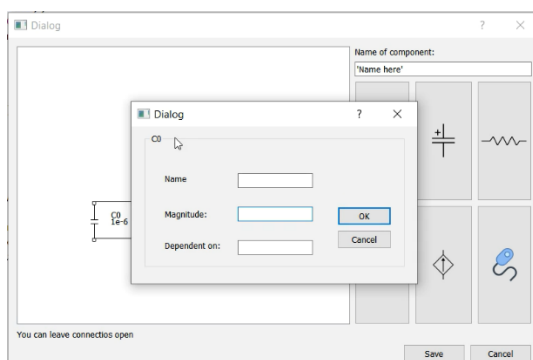


**FIGURE 12: CHANGING COMPONENT INFORMATION**

As seen in figure 12 when the user double clicks on the component they are able to alter the components information.

## 6.9. Open connections

If the user would like to leave connections open to return to their work later they should be able to do so.
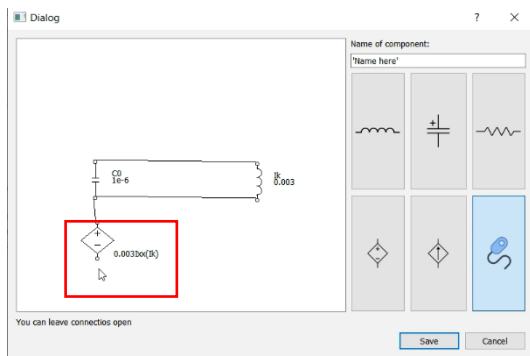
**FIGURE 13: OPEN CONNECTIONS**

As seen in figure 13 the user is able to leave connections open if they would like to return to the circuit at a later stage.

## 6.10. Save circuit

The user should be able to save their work. When the user clicks on save the "Override?" question should appear and they should be able to click on Yes to override the existing schematic or not to.
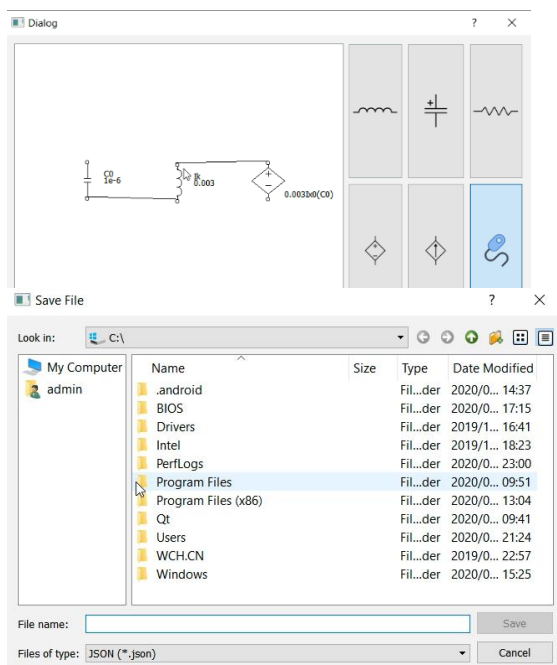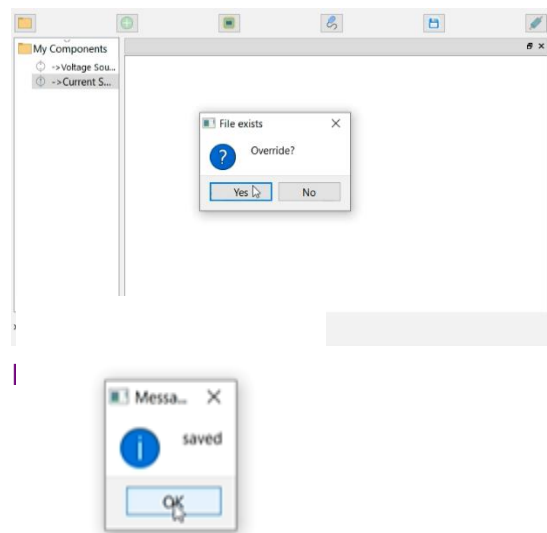


**FIGURE 16: SAVE SCHEMATIC TO COMPUTER**



**FIGURE 17: MESSAGE**

Figure 16 indicates that when the user followed the steps presented in figure 14 and 15 to save and override the schematic the user will have the choice where on their computer they would like to store their schematic and what its file name should be. After saving a message box will appear that notifies the user that their file was saved in figure 17.

## 6.11. Output

A text file of the saved circuit should appear in the program folder with the name the user saved the circuit as. Inside the text file all the components present in the circuit should be displayed in a matrix with the component's Username, Used Name, x and y coordinates, rotation angle and Magnitude. How the components were connected are displayed under the Connections heading.
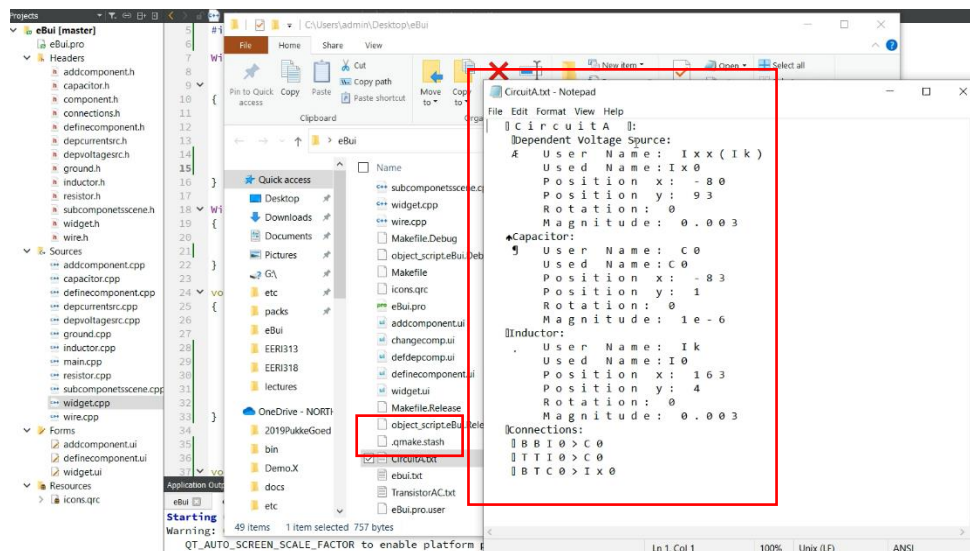
**FIGURE 18: OUTPUT TEXT FILE**

As seen in figure 18 the user can find the text file saved and are able to see the circuit's information in matrix form.
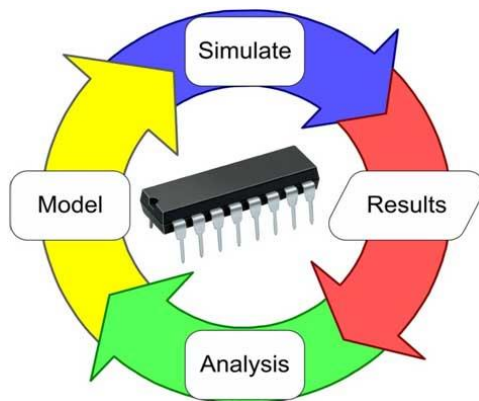
# 7. Conclusion



**FIGURE 19: ANALOG SIMULATION PROCESS**

In this analog simulation the user can create their own component, save it in their component library. Located to that directory and retrieve any saved item by selecting all files when in the component directory. The user can also connect their components, add a source, simulate, and have fun. The user then save when finished.