TOUCHSPICE VS REACTIVE-SPICE: A HUMAN-COMPUTER INTERACTION
PERSPECTIVE

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical Engineering

by
Joshua Martin O'Hara

August 2012

© 2012
Joshua Martin O'Hara
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: TouchSPICE vs. ReActive-SPICE: A Human-Computer Interaction Perspective

AUTHOR: Joshua Martin O'Hara

DATE SUBMITTED: August 2012

COMMITTEE CHAIR: Dr. John Oliver, Assistant Professor

COMMITTEE MEMBER: Dr. Dale Dolan, Assistant Professor

COMMITTEE MEMBER: Dr. Tina Smilkstein, Assistant Professor

ABSTRACT

TouchSPICE vs. ReActive-SPICE: A Human-Computer Interaction Perspective

Joshua Martin O'Hara

Traditional SPICE simulation tools and applications of circuit theory lack real-time interaction and feedback. The goal of this thesis was to create an interactive physical environment to allow the manipulation and simulation of discrete electrical components in near-real-time while optimizing and streamlining the human-computer interaction (HCI) elements to make the user experience as positive and transparent as possible. This type of HCI and near-real-time simulation feedback would allow for the instant realization of how the parameters of each discrete component or hardware module affect the overall simulation and response of the circuit.

The scope of this thesis is to research, design and develop two real-time interactive SPICE simulation tools and analyze the real-time benefits and HCI elements of both simulators, principally the user interface design itself. The first real-time interactive simulator (TouchSPICE) uses multiple embedded processors (touchscreen hardware blocks) and a host computer to build and simulate a circuit. The second real-time interactive simulator (ReActive-SPICE) uses a single host computer with integrated software to build and simulate a circuit, much like LTspice™ and PSpice™ without the real-time aspects.

As part of the study, 20 students were asked to create circuits utilized in undergraduate-level labs using TouchSPICE and ReActive-SPICE for the sole purpose of providing feedback on the two user interfaces. Students were asked to complete a survey before, during and after circuit creation to provide a basis for judging the intuitiveness, efficiency and overall effectiveness of the HCIs. Conclusions based-off the surveys support the hypothesis that both TouchSPICE and ReActive-SPICE were more intuitive and overall simpler than traditional SPICE simulation tools. Feedback collected showed TouchSPICE to have a more intuitive user interface while ReActive-SPICE proved to be more efficient. ReActive-SPICE was further developed and enhanced to improve the user interface as well as the overall circuit creation and real-time simulation processes.

Keywords: real-time, SPICE, simulation, human-computer interaction, HCI, learning
tools, graphical user interface, GUI, embedded processors, ARM, Linux, Qt,
TouchSPICE, ReActive-SPICE

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Oliver for the original thesis idea, project funding and guidance. I would also like to thank Dr. Dale Dolan and Dr. Tina Smilkstein for their participation on my thesis committee. For assistance with the host user interface development, I extend my thanks to Kevin Peters, a fellow graduate student. I would also like to thank the Graduate Programs Coordinator, Becky Powell, for all of her time and persistence in making sure that everything was taken care of for graduation.

I thank my parents, Patrick and Catherine, for mentally and financially supporting me and allowing me to pursue my ambitions. I thank my girlfriend, Courtney, for all of her support, inspiration and encouragement along the way. I thank my brother, Sean, for the much needed comic relief to keep me motivated. And above all, I thank God for giving me the strength and health to keep me focused and for blessing me with the resources and ability to complete this project.

TABLE OF CONTENTS

	Page
LIST OF TABLES.....	xi
LIST OF FIGURES.....	xii
CHAPTER	
I. INTRODUCTION.....	1
1.1. Statement of the Problem.....	2
1.2. Possible Solutions.....	4
1.3. Thesis Organization.....	5
II. PREVIOUS STUDIES AND RELATED WORKS.....	6
2.1. SPICE with Virtual Hardware Blocks.....	6
2.2. Virtual Lab Benches.....	7
2.3. Existing Real-time SPICE Simulators.....	16
2.4. Existing Physical-Virtual Interfaces.....	20
III. DESIGN CRITERIA.....	25
3.1. Requirements.....	27
3.1.1. Budget.....	27
3.1.2. Supported Circuit Elements.....	28
3.1.3. Form Factor.....	28
3.1.4. Interactivity and Play.....	29
3.1.5. Intuitive UI.....	29
3.1.6. Power Consumption.....	30
3.2. Scalability Analysis.....	30

IV. TOUCHSPICE SYSTEM DESIGN.....	31
4.1. TouchSPICE Hardware.....	31
4.1.1. Development Platform.....	32
4.1.1.1. FriendlyARM Mini2440/Mini6410 (ARM9/ARM11).....	33
4.1.1.2. MicroBuilder LPC1343 (ARM Cortex M3).....	34
4.1.1.3. OMAP Zoom eXperimenter (TI OMAP).....	35
4.1.1.4. Overall Development Board Comparison.....	36
4.1.1.4.1. Power Consumption Analysis.....	38
4.1.1.4.2. Development Platform Evaluation.....	39
4.1.2. Inter-node Communication.....	40
4.1.2.1. Banana-to-Banana Connectors (1-wire).....	40
4.1.2.2. Twisted Pair Connectors (2-wire).....	40
4.1.2.3. Wired Connections Evaluation.....	41
4.1.3. Wireless Connections.....	42
4.1.3.1. Zigbee Wireless.....	43
4.1.3.2. Bluetooth Wireless.....	44
4.1.3.3. Wireless Evaluation.....	45
4.2. TouchSPICE Software.....	46
4.2.1. Operating System and Programming Environment.....	46
4.2.2. Wireless Communication Protocol.....	48
4.2.2.1. Node-initiated Coordination (Scheduled Nodes).....	48
4.2.2.2. Host-initiated Coordination.....	49
4.2.2.3. Time Synchronization.....	50

4.2.2.4. Wireless Communication Protocol Evaluation.....	51
4.3. TouchSPICE High-level Concept.....	52
V. TOUCHSPICE FIRMWARE DESIGN.....	58
5.1. Minimum Fan-in and Fan-out Considerations.....	58
5.2. Two-Wire Communication Protocol Considerations.....	59
5.3. Two-Wire Communication Driver Design.....	59
5.4. Two-Wire Communication API Design.....	64
5.5. Bluetooth API.....	72
VI. TOUCHSPICE SOFTWARE/UI DESIGN.....	74
6.1. Target.....	75
6.1.1. Graphical User Interface.....	79
6.2. Host.....	82
6.2.1. Graphical User Interface.....	85
VII. REACTIVE-SPICE DESIGN.....	86
7.1. Porting TouchSPICE to ReActive-SPICE.....	86
7.2. Graphical User Interface.....	87
7.3. Creating Circuits via Net Matching.....	89
VIII. USER STUDY.....	92
IX. HCI ANALYSIS.....	104
9.1. User Study Summary.....	105
9.2. Learning Curves.....	106
9.3. Intuitive Interfaces and Real-time Features.....	108
9.4. Importance of the Physical-Virtual Interfaces.....	110

9.5. Shortcomings of TouchSPICE and ReActive-SPICE v1.0.....	112
9.6. ReActive-SPICE v2.0.....	113
X. CONCLUSION.....	120
FUTURE WORK.....	122
BIBLIOGRAPHY.....	124
APPENDICES.....	126
A. TouchSPICE Bill of Materials.....	126
B. Mini Bring-up / Kernel building / Environment setup.....	127
C. User Survey.....	137
D. UI Screenshots.....	145
E. Target Source Code.....	151
F. Host Source Code.....	181
G. ReActive-SPICE v1.0 Source Code.....	204
H. ReActive-SPICE v2.0 Source Code.....	261

LIST OF TABLES

Table	Page
2.1 Literary Search terms used to find Previous Studies.....	6
4.1 Development Board Selection Comparison.....	36
4.2 Development Board Power Consumption Summary.....	38
4.3 Development Board Evaluation	39
4.4 Wired Connections Evaluation.....	42
4.5 Bluetooth vs. Zigbee Features.....	45
4.6 Wireless Connections Evaluation.....	46
5.1 Target Software I/O Pair Mapping and Functions.....	61
8.1 User-base Grade Level.....	92
8.2 User-base SPICE Software Experience.....	93
9.1 User Study Tabulated Results Summary.....	105
9.2 Time to build a circuit using TouchSPICE Summary.....	106
9.3 Time to build a circuit using ReActive-SPICE v1.0 Summary.....	107

LIST OF FIGURES

Figure	Page
1.1 Evolution of the modeling and simulation of circuits.....	3
2.1 The virtual environment created in pure software	7
2.2 The simulation process in virtual experiment system.....	9
2.3 LabView Virtual Instrument (VI) HCI.....	10
2.4 MultiSim Simulated Tektronix Oscilloscope Interface.....	11
2.5 DC/AC Lab's Online Virtual Circuits Lab.....	12
2.6 TINA Schematic Editor and 3D Breadboard Simulator.....	13
2.7 TINA 3D Viewer with PCB and an Oscilloscope Virtual Instrument.....	14
2.8 Amatrol Virtual Measurement Instruments – Caliper interface/tutorial.....	15
2.9 iCircuit iPhone/iPad Real-time Circuit Simulation Application.....	17
2.10 “Every Circuit” Android Real-time Circuit Simulation Application.....	18
2.11 Falstad Open-Source Java Real-time SPICE Simulator.....	19
2.12 Sifteo Cubes.....	20
2.13 Sifteo Platform Sorting Concept.....	21
2.14 Sifteo Cubes in action.....	22
2.15 Data Tiles – Graspable UI.....	23
3.1 Original High-Level Hardware and GUI Concept.....	26
4.1 Original TouchSPICE High-Level Hardware Concept.....	32
4.2 FriendlyARM Mini2440 Development Board (Android OS displayed)	33
4.3 MicroBuilder LPC1343 Development Board.....	34
4.4 OMAP Zoom eXperimenter Development Board	35
4.5 Digi International's Xbee module with UART Breakout Board.....	43
4.6 Bluegiga's WT-12 Class 2 Bluetooth module with UART Breakout Board.....	44

4.7	Node-initiated Communication (Scheduled Nodes).....	49
4.8	Host-Initiated Communication.....	50
4.9	Time Sync Communication.....	51
4.10	Modified TouchSPICE Widescreen High-Level Hardware and GUI Concept.....	52
4.11	Sample SPICE Netlist.....	53
4.12	I/O Number Designation on a Node with tied physical inputs/outputs.....	54
4.13	I/O Number Designation on a Node with unique physical inputs/outputs.....	54
4.14	Operation of Hardware Flow Diagram.....	56
4.15	High-level node hardware block diagram.....	57
5.1	External Interrupts on CPU3 mapped to GPIO header.....	60
5.2	The 8 (EINT0-5, 8-9) Pins Used on the GPIO Header.....	61
5.3	Two-Wire Communication Block Diagram.....	67
5.4	Timing Diagram of a single Transaction between TX/RX0 and TX/RX2.....	69
5.5	GPIO Send Byte Function Flow Diagram.....	70
5.6	GPIO Receive Byte Function Flow Diagram.....	71
5.7	SparkFun Logic Level Converter.....	72
6.1	High-level Software Flow Diagram.....	74
6.2	GPIO/Communication Thread Flow Diagram.....	76
6.3	Main GUI Thread Flow Diagram.....	77
6.4	Neighbor Identification Flow Diagram (from GPIO thread).....	78
6.5	GUI Hit-box Diagram for a Capacitor.....	79
6.6	Select a Component GUI.....	81
6.7	Op-Amp Component GUI (no additional value controls).....	82
6.8	Host GUI High-level Software Flow Diagram.....	84
6.9	Host GUI Main Window.....	85

7.1	ReActive-SPICE v1.0 GUI with 4 “virtual” nodes (SAC view).....	87
7.2	ReActive-SPICE v1.0 GUI with 4 “virtual” nodes (Circuit view)	88
8.1	Single node (A) before being powered on.....	93
8.2	Single node (A) powered on.....	94
8.3	Component selection screen.....	94
8.4	“V Source” component selected.....	95
8.5	Another node is turned on and plugged into the AC source.....	95
8.6	Node Connection Success.....	96
8.7	An additional node is turned on and a low pass filter is constructed.....	96
8.8	User adjusting the resistor value via the slider bar.....	97
8.9	User again adjusting the resistor value via the slider bar.....	97
8.10	User again adjusting the resistor value via the slider bar.....	98
8.11	User adjusting the resistor value via the spin-box buttons for fine-tuning.....	98
8.12	Host GUI showing the user-created low-pass filter.....	99
8.13	User increased the resistor value in the circuit.....	99
8.14	ReActive-SPICE simulator v1.0 – initial startup.....	100
8.15	ReActive-SPICE simulator v1.0 – Inverting amplifier design/simulation.....	101
8.16	ReActive-SPICE simulator v1.0 – Inverting NMOS design/simulation.....	101
8.17	ReActive-SPICE simulator v1.0 – Inverting NMOS design adjusted.....	102
9.1	ReActive-SPICE simulator v2.0 – New Schematic Design.....	114
9.2	ReActive-SPICE simulator v2.0 – Components with controls.....	115
9.3	ReActive-SPICE simulator v2.0 – Simulating, No controls visible.....	116
9.4	ReActive-SPICE simulator v2.0 – Probing Voltages, No controls visible.....	117
9.5	ReActive-SPICE simulator v2.0 – Inverting Op-Amp Design.....	118
9.6	ReActive-SPICE simulator v2.0 – Voltage and Current Plots.....	119

11.1	TouchSPICE/ReActive-SPICE v1.0 User Interface – Select a Component.....	145
11.2	TouchSPICE/ReActive-SPICE v1.0 User Interface – Resistor.....	145
11.3	TouchSPICE/ReActive-SPICE v1.0 User Interface – Capacitor.....	146
11.4	TouchSPICE/ReActive-SPICE v1.0 User Interface – Inductor.....	146
11.5	TouchSPICE/ReActive-SPICE v1.0 User Interface – Op-amp.....	147
11.6	TouchSPICE/ReActive-SPICE v1.0 User Interface – NMOS.....	147
11.7	TouchSPICE/ReActive-SPICE v1.0 User Interface – BJT.....	148
11.8	TouchSPICE/ReActive-SPICE v1.0 User Interface – Current Source.....	148
11.9	TouchSPICE/ReActive-SPICE v1.0 User Interface – DC Voltage Source.....	149
11.10	TouchSPICE/ReActive-SPICE v1.0 User Interface – AC Voltage Source.....	149
11.11	Qt Creator Screenshot – GUI Editor.....	150
11.12	Qt Creator Screenshot – C++ Editor.....	150

Chapter 1 Introduction

The overall objective and scope of this thesis were to create two real-time interactive SPICE simulators: TouchSPICE and ReActive-SPICE, to assist with learning circuit theory. TouchSPICE is a hardware-based solution using multiple embedded processors (touchscreen hardware blocks) and ReActive-SPICE is an integrated software-based solution. In addition, a user study was designed and performed to assist in determining the overall effectiveness of the human-computer interaction (HCI) elements, principally the user interface design of the new simulation tools.

The development of the new simulators was motivated by the fact that traditional SPICE simulation tools such as LTspice™ and PSpice™ lack real-time simulation and feedback and often times overcomplicate the user interface design, causing a steep learning curve for the inexperienced user. The end goal was to yield a refined real-time SPICE simulation solution to make learning circuit theory easy, fun and effective.

After the design and implementation of the new simulation tools, a user study was performed where 20 students were asked to create circuits utilized in undergraduate-level labs using the real-time interactive SPICE simulator using embedded virtual blocks (TouchSPICE) and the exclusively software-based real-time interactive SPICE solution (ReActive-SPICE) for the sole purpose of providing feedback on the two user interfaces. Students were asked to complete a survey before, during and after circuit creation to examine the overall HCI experience and effectiveness of the two real-time interactive SPICE simulators.

1.1 Statement of the Problem

All incoming Electrical and Computer engineering students must take a circuit analysis lecture and lab at some point during their first or second year of college. Many of the existing lab practices involve solving circuits on paper, constructing simple circuits on a breadboard and drawing schematics on a computer and simulating them using conventional SPICE simulation tools such as PSpice™ and LTspice™. Although all of these enhance lecture and lab correlation and comprehension, they do not encourage further exploration and curiosity. This is because these traditional practices not only take time to construct, but making any minor/major modifications can be tedious and troublesome, affecting the overall student experience.

To demonstrate the typical circuit learning process, the evolution of circuit modeling and simulation is exhibited in Figure 1.1. First, students begin learning circuit theory by performing hand calculations and circuit analysis to characterize and determine the response of the circuit. Eventually the student will breadboard the components to create the desired circuit and analyze and gather data from it by using bench-tools such as oscilloscopes, logic analyzers and multimeters. Finally the student will be introduced to SPICE, allowing the virtual creation and simulation of circuits possible on a computer.

Each of these circuit analysis tools has its benefits and drawbacks. Hand calculation-based modeling tends to be quicker for displaying and analyzing ideal simple circuits, while generally the physical aspects of breadboard-based circuit analysis tends to be more exciting, accurate and closer to real-world imperfections compared to the virtual simulation of the circuit via software; however software simulation is cost-effective.

Figure 1.1 – Evolution of the modeling and simulation of circuits

Despite the benefits, there are many drawbacks to these types of circuit analysis—primarily the difficulty and time-consuming process of making circuit design modifications such as topology, parameter and value adjustments. These drawbacks are particularly noticeable in hand calculation-based and breadboard-based circuit analysis. Modern SPICE simulation tools have been able to overcome many of the aforementioned drawbacks; however as mentioned earlier, these tools lack real-time interaction and feedback, which can dislocate the student from the actual functionality of the components and the circuit as a whole. To see the effects of changing component values and how they affect the simulation, one must stop the SPICE simulation, change the component value and then re-simulate. A value sweep can be performed on up to three components per simulation (in LTspice™); however this takes time and program familiarity to setup and does not give the user full control over the varying component parameters.

1.2 Possible Solutions

In order to solve many of the said shortcomings while maximizing the benefits, an ideal solution would combine minimalistic and simple physical HCI with a virtual real-time simulator. The minimalistic and simple physical HCI would make the learning experience fast, easy, fun and also reduce the amount of initial SPICE knowledge necessary to perform simulations, while the virtual real-time simulator would be accurate and cost-effective.

The proposed solutions: TouchSPICE and ReActive-SPICE, needed to be created to not only encourage play and further circuit theory exploration, but to also give the student or user the ability to manipulate any component value seamlessly while simulating. This type of HCI and near-real-time simulation feedback would allow for the instant realization of how the parameters of each discrete component or hardware module affect the overall simulation and response of the circuit. This type of simulation tool would also allow the student to quickly and easily change the configuration and topology of the circuit itself.

Two types of real-time interactive SPICE simulators are explored in this thesis. One possible solution is to design a set of touchscreen virtual blocks (TouchSPICE) that would each display a component—such as a resistor, capacitor, inductor, diode, opamp, BJT, NMOS, voltage or current source. These virtual blocks would communicate with a host computer, which would assemble a netlist and run the simulation—a form of centralized processing. A second possible solution, which is more cost-effective but lacks physical play, is to create a real-time simulator interface based exclusively in software on

the host computer (ReActive-SPICE). This type of solution would also be more portable and widely distributable with the least amount of initial investment. Both options will be explored in depth and conclusions will be drawn based on the results of the user study.

1.3 Thesis Organization

Now that the proposed solution has been laid out, the next chapter will cover any previous studies and related works that have been done on the subjects of using multiple embedded processors (virtual blocks/nodes) to create/simulate circuits and exclusively software-based real-time SPICE simulators.

Since there are two distinct real-time interactive SPICE simulators being created and analyzed in this thesis, terms are established to distinguish them from each other. The TouchSPICE solution uses multiple embedded processors or “nodes” (virtual blocks) and a single “host” computer. The exclusively software-based ReActive-SPICE solution uses a single computer.

The TouchSPICE node design criteria is first laid out, followed by a chapter that discusses the node system design—including node component selection, node firmware design and node software and user interface design. The host software and user interface design will then be covered, followed by the ReActive-SPICE simulator software and user interface design. The user study results and feedback will then be discussed, followed by an analysis on the HCI components of the various user interfaces. Conclusions will be made based off these discussions and feedback. In-depth software procedures and processes will be covered in the Appendices for reference.

Chapter 2 Previous Studies and Related Works

One of the first steps before designing the TouchSPICE and ReActive-SPICE simulators was to research any previous studies that had experimented with interactive virtual blocks or real-time SPICE simulators.

2.1 SPICE with Virtual Hardware Blocks

No previous studies were found to have explored a real-time SPICE simulation tool using virtual hardware blocks. A summary of all the search terms used and results that were returned by the ISI Web of Knowledge academic citation indexing search system provided by Thomson Reuters is summarized in Table 2.1.

Table 2.1 – Literary Search terms used to find Previous Studies

Search Terms	Refined Search Terms	Number of Results
real-time		678,412
	interaction	36,422 (0 relevant)
	HCI	243 (0 relevant)
interaction		4,515,204
	SPICE	767 (0 relevant)
SPICE		57,269
human-computer interaction		19,919
	SPICE	1 (0 relevant)
	user interface	14,313 (0 relevant)
HCI		10,724
	SPICE	16 (0 relevant)
LabView		7,084
	real-time	1,045 (0 relevant)
	HCI	0

2.2 Virtual Lab Benches

There was a 2011 paper found on HCI with a virtual “lab-bench” environment called “Virtual Electronic Laboratory” that was developed by a team of students in China which demonstrated to drastically improve the circuit creation and modeling experience. The paper focused on the research and design of an intelligent virtual system embedded in a multimedia interface with ActiveX components written purely in software. This multimedia applet included a real-time SPICE simulator and consisted of hardware components such as oscilloscopes, function generators and power supplies realized in software as can be seen in Figure 2.1.

Figure 2.1 – The virtual environment created in pure software [1]

The motivation for the design of this virtual environment was purely economic and ergonomic. The graduate students at the Naval Aeronautical and Astronautical University of China wanted to improve the “effect and efficiency of the teaching of electronic circuit experiments” [1] while also saving educational funds and precious lab space for other important equipment and experiments.

The operation of the virtual experiment software is straightforward; first, components are dragged onto the experiment board where an input module reads the circuit topology and converts this data into a netlist to be simulated by SPICE 3F5 which checks for any circuit building errors that may have occurred. The SPICE simulation is performed and the data is filtered through an output module which converts the raw data back into a graphical display of waveforms on the virtual measurement equipment using a dynamic drawing technique. This process is modeled in Figure 2.2.

Figure 2.2 – The simulation process in virtual experiment system [1]

A standard AC amplifier experiment was used for testing on the virtual lab bench. The same circuit was constructed alongside and data was collected and compared throughout the experiment. The comparison results showed that the virtual lab bench simulation results agreed with the real experimental results.

Currently, the virtual lab bench product is used in many colleges and universities across China, which has received positive feedback and is slowly breaking the traditional way experiments are executed. “Test results and application show that the system achieves excellent performance in design ability, arbitrariness of operation and

interaction with sense of reality, which makes users operate it with great interest just like doing real experiments and gains good effect in experimental teaching.” [1]

Aside from this virtual lab-bench interface, there are a few other physical/virtual interfaces that offer similar functionality, such as National Instrument’s LabView and MultiSim. At Oklahoma State, Thomas W. Gedra of the Electrical and Computer Engineering Department wrote a paper on “Virtual Instrumentation in an Undergraduate Electrical Machines Lab” and how it substantially improved how the students viewed their learning environment and how they interacted positively with the virtual interfaces/instruments [2].

Figure 2.3 – LabView Virtual Instrument (VI) HCI

LabView’s virtual instrument HCI can be seen in Figure 2.3 and MultiSim’s simulated Oscilloscope can be seen in Figure 2.4. Gedra describes how the lab moved away from traditional instrumentation to virtual instrumentation and noted that the lab

equipment upgrade had made significant improvements on the quality of the lab experience for all of the undergraduate students. “Many measurements are now possible which were impossible or extremely tedious with the old equipment. Students can see, in real time, the effect of an experimental variable on real power, reactive power, and the phasor and spectral representation of AC signals.” [2] These results align directly with the goals of TouchSPICE and ReActive-SPICE, to simplify and provide a clean and friendly user interface to improve the student’s overall grasp on circuit theory.

Figure 2.4 – MultiSim Simulated Tektronix Oscilloscope Interface

Figure 2.5 – DC/AC Lab’s Online Virtual Circuits Lab [3]

Due to the wide-variety of web tools and online development software available, even simple online virtual lab experiments are springing up, including the DC/AC Lab virtual simulator seen in Figure 2.5. This tool allows the user to drag and drop rudimentary circuit elements, such as batteries, fuses, switches and light bulbs onto a grid and then connect them together with wires. As soon as the switch is flipped, current is visualized in the wires as moving dots and much like LabView, a simplified oscilloscope virtual instrument can be used to “probe” different parts of the circuit.

Similar to LabView, a program called TINA also allows real-time simulation along with 3D modeling of lab setups. The TINA Design Suite costs about \$90 and “is a powerful yet affordable circuit simulation and PCB design software package for analyzing, designing, and real time testing of analog, digital, VHDL, MCU, and mixed electronic circuits and their PCB layouts.” [4] TINA’s interface and 3D viewer can be seen in Figures 2.6 and 2.7.

Figure 2.6 – TINA Schematic Editor and 3D Breadboard Simulator

Figure 2.7 – TINA 3D Viewer with PCB and an Oscilloscope Virtual Instrument

Virtual lab HCIs are not limited to circuit analysis and characterization, Joe Fuehne, an Assistant Professor of Mechanical Engineering Technology for Purdue University integrated virtual lab simulation software into his program, allowing everyone in the class to “perform the same experiments, as available on the hardware, in a virtual environment on their computer monitors.” [5] This virtual lab software by Amatrol was for hydraulics and consisted of a simple user interface for connecting a network of hoses together to create a path for fluid flow and then perform the simulation on it. The software also featured virtualized instruments similar to LabView, such as a caliper instrument interface that can be seen in Figure 2.8.

Figure 2.8 – Amatrol Virtual Measurement Instruments – Caliper interface/tutorial

Although these virtual lab-benches achieved the goal of creating virtual instruments and real-time SPICE simulation tools in software, the human-computer interface was not optimized to create a smooth transition for the user from current SPICE simulation environments. This thesis will mainly focus on the creation and implementation of a hardware-based real-time SPICE simulation tool (TouchSPICE) along with an improved software-based real-time SPICE simulation tool (ReActive-SPICE) from an optimized HCI perspective.

2.3 Existing Real-time SPICE Simulators

Since there is a wide variety of mobile application development platforms and tools today, very clean yet complex applications are able to be developed within a matter of weeks. One of the most popular real-time SPICE simulators on the mobile market today is iCircuit (\$9.99 USD), which is for the iPhone and iPad. “iCircuit is the easy to use electronic circuit simulator and editor... But iCircuit is unlike other CAD programs because it is always simulating. It's just like working with the real circuit. You do not stop to take a measurement or spend a lot of time configuring reports. Instead, you just play with the circuit as you normally would, with the power on!” [6] The easy-to-use interface can be seen in Figure 2.9. In a sense, this is what we wanted the hardware blocks of TouchSPICE to be modeled after, only instead of having all of the elements on a single screen, they would be spread out across multiple touchscreens and allow the student to connect them together to preserve the feeling of actually building the circuit.

Figure 2.9 – iCircuit iPhone/iPad Real-time Circuit Simulation Application (\$9.99 USD)

A rival or complement to iCircuit is “Every Circuit”, which is a similar mobile application for Android; this interface can be seen in Figure 2.10. “Build any circuit, tap [the] play button, and watch dynamic voltage, current, and charge animations. This gives you insight into circuit operation like no equation does. While [the] simulation is running, adjust circuit parameters with analog knob, and the circuit responds to your actions in real-time. You can even generate an arbitrary input signal with your finger!” [7] This application is very similar to iCircuit and costs about the same amount. It too has a well-crafted and easy-to-use UI.

Figure 2.10 – “Every Circuit” Android Real-time Circuit Simulation Application (\$10.00 USD)

The pitfall of both of these mobile applications is that they are not open source, free or scalable. The user is constrained to the physical size of the phone/tablet touchscreen, which can cause frustration when constructing larger circuit networks.

Figure 2.11 –Falstad Open-Source Java Real-time SPICE Simulator [8]

A final related piece of work is the Falstad Circuit Simulator, which is a free open-source Java real-time SPICE simulator. This interface can be observed in Figure 2.11. The Falstad Circuit Simulator actually created the foundation for the iCircuit application previously mentioned [6]. With this application you can also simulate circuits in real-time and view waveforms of multiple components as you change parameters in real-time. The user interface on this application was limited and required some time to become accustomed to.

2.4 Existing Physical-Virtual Interfaces

All of the previously discussed related works have utilized a central interface on a single computer or touchscreen. However, TouchSPICE is to use multiple physical-virtual interfaces to construct a circuit, allowing the user to manipulate each of the interfaces simultaneously to cause a change in the real-time SPICE simulation.

Using physical-virtual interfaces is not new to the HCI realm; in 2007, David Merrill and Jeevan Kalanithi invented Siftables, “a novel platform that applies technology and methodology from wireless sensor networks to tangible user interfaces in order to yield new possibilities for human-computer interaction.” [9] The Sifteo platform was created by David and Jeevan in 2009 and was based off the Siftables prototype. Sifteo cubes are essentially compact 1.5 inch devices with a touchscreen LCD display, 3-axis accelerometer, near field communication and a single button under the touchscreen. The devices can be “physically manipulated as a group to interact with digital information and media” and can also be used for game development [9]. The Sifteo cubes can be observed in Figure 2.12 and Figure 2.14.

Figure 2.12 –Sifteo Cubes [10]

The Sifteo platform was motivated by the theory that humans have a skill for “sifting, sorting, and otherwise manipulating large numbers of small physical objects” [9] and that we as humans use all “our fingers and both hands actively and efficiently” [9] when performing certain tasks such as sorting photographs or organizing nuts and bolts.

Figure 2.13 –Sifteo Platform Sorting Concept [9]

The Sifteo platform takes advantage of these skills and enables us to manipulate all of the cubes either independently or simultaneously. This mode of operation uses multiple input devices and is *space-multiplexed*, meaning that “different devices can be attached to different functions, each independently (but possibly simultaneously) accessible. This then affords the capability to take advantage of the shape, size and position of the [cubes] to increase functionality and decrease complexity.” [11 & 12] However, when we sort digital information on a modern day computer, we are typically using only two input devices: a mouse and a keyboard, to perform multiple actions and GUI manipulations sequentially. This mode of operation is *time-multiplexed* and typically does not leverage our physical manipulation skills [11 & 12].

The Sifteo cubes are what are known as a graspable user interfaces (UIs), which “provides users concurrent access to multiple, specialized input devices which can serve as dedicated physical interface widgets, affording physical manipulation and spatial arrangements.” [11] These graspable UIs can be seen in action in Figure 2.14.

Figure 2.14 – Sifteo Cubes in action [10]

A similar graspable UI created in 2001 was DataTiles, which was designed to utilize the strengths of both graphical and physical user interfaces. DataTiles consist of multiple transparent acrylic square objects with embedded RFID tags and a single flat panel screen (tray) placed horizontally “to serve as both an information display and a support surface for the tiles.” [13] Each tile has an associated function with it (weather, maps) and when they are placed on the tray, the associated function is automatically triggered—meaning when a weather tile is placed on the tray, weather information from the internet is processed and then displayed in the region directly below the tile [13]. The

DataTiles graspable UIs can be seen in Figure 2.15. These tiles could be manipulated and customized to serve multiple different functions, similar to the Sifteo platform.

Physical objects offer stronger affordances than purely visual ones. People can use their sophisticated skills for manipulating objects –not just pointing and clicking, but also rotating, grasping, attaching, etc. Interactions may also involve two hands, or allow many people to interact cooperatively in the same physical interaction space. [13]

Also, since the physical graspable UIs cannot suddenly change states, this “leaves users with more confidence and a stronger sense of control over the status of the interaction.” [13] However, these physical graspable UIs are typically designed for a single task and are less flexible and harder to repurpose.

Figure 2.15 – Data Tiles – Graspable UI [13]

After researching many of these HCI devices and applications, the main objective of the thesis was revisited. Like some of the related works, the aim was to create a real-time SPICE simulator, however this thesis was to have two unique solutions: one was to create a graspable user interface (TouchSPICE) to allow the manipulation and simulation of discrete electrical components in near-real-time while optimizing and streamlining the HCI elements to make the user experience as positive and transparent as possible and the other was to create a pure software version of this (ReActive-SPICE), much like these other related works. To make it clear, one of the main roles of the software-based ReActive-SPICE solution was to validate the effectiveness of the physical play in the TouchSPICE solution. “By using physical objects, we not only allow users to employ a larger expressive range of gestures and grasping behaviors but also to leverage off of a user's innate spatial reasoning skills and everyday knowledge of object manipulations.” [11] Regardless, both solutions will be discussed in depth.

Chapter 3 Design Criteria

The general system requirements of TouchSPICE were: to be capable of creating basic undergraduate-level circuits with less than 10 components, have a simple touchscreen interface on each of the embedded blocks or a simple yet powerful interface on the host computer and to be capable of simulating the circuits in real-time.

The original design criterion for TouchSPICE was to consist of at least 10 embedded blocks, which would enable the use of up to 10 circuit elements (i.e. voltage source, current source, resistor, capacitor, inductor, diode, opamp, NMOS or BJT)—the original concept can be seen in Figure 3.1. Using 10 blocks would enable the creation of most undergraduate-level lab circuits [14]. Each of the blocks would have a microcontroller on-board and a means of wireless communication with the other blocks and the host computer. Each of the blocks would also have a touchscreen LCD on-board which would display and emulate a discrete circuit element. The virtual circuit would consist of up to 10 blocks connected together with wires representing the physical connections to the elements in the circuit. Each block would communicate wirelessly with one another to render a complete netlist of the virtual circuit. This netlist would be constantly sent to the host computer, which would update the simulation in real-time.

Figure 3.1 – Original High-Level Hardware and GUI Concept

The student would be able to adjust certain parameters of each discrete component (i.e., voltage, current, resistance, capacitance, or inductance) by moving the appropriate horizontal slider bar on the touchscreen and the computer simulation would update and respond according to the stimulated change. These blocks should all be self-sufficient and in the future be able to last on batteries for at least five hours. For the sake of cost-effectiveness, any open-source platforms and tools should be used over costly platforms such as Windows or Mac.

The entire software-based simulator (ReActive-SPICE) would essentially do all of this, except solely on the host computer. Much of the wireless and inter-node communication is obviously cutout which should enable the creation of a faster and more efficient real-time SPICE simulator mainly limited on the computer's ability to read and write files, which will be discussed further in the software design chapter.

Throughout the thesis project, the design criteria adapted due to certain limitations and advantages which lead to a more realistic design. These adaptations will be discussed in the node hardware chapter and again in the hardware and software design chapters.

3.1 Requirements

Since the simulator solutions of this thesis project could possibly and hopefully morph into a new wide-spread lab tool for universities across the globe, many design requirements must be imposed to keep the tools cost-effective, robust, limited in size and low-maintenance—all while providing scalability, interchangeability and modularity.

3.1.1 Budget

For TouchSPICE, each hardware block must be less than \$100 and preferably less than \$50 to allow an affordable tool suite for any university lab-bench. This includes the cost of the board, touchscreen and all of the external/internal communication components. If this requirement is strictly adhered to, a prototype system should be under \$1000 for the hardware blocks plus the cost of a mid-performance laptop/desktop to run the simulations. Since we will be exclusively using open-source platforms and tools, the cost of the operating system, environment and design tools should not be a factor.

3.1.2 Supported Circuit Elements

If TouchSPICE and ReActive-SPICE are supposed to be useful simulation tools in engineering labs, they must contain the basic components that make up any electrical circuit. Basic circuit networks were pulled from lower-division EE labs such as EE 241 and EE 242 and the maximum number of components needed to construct any of the circuits turned out to be 10 [14]. The circuit elements to be supported are: voltage source, current source, resistor, capacitor, inductor, diode, op-amp, NMOS and BJT. Having this number of graspable UIs should also avoid physical and cognitive clutter for the user.

3.1.3 Form Factor

These simulation tools should be ergonomic and low-profile so as to not intrude on a large portion of existing lab space. Portability is crucial. The entire TouchSPICE system needs to be as small as possible to preserve lab space yet still be able to maintain comfortable and relaxed HCI for the best user experience. The same goes for the host computer, or the software-based ReActive-SPICE solution, it needs to be small and portable, making a laptop the ultimate solution to meet the portability and size requirements.

However, the small output display restrictions of the virtual blocks hinder the usable area of the interface. Creating a tight user interface is much more difficult because clutter and simplicity need to be considered at all stages of the design process. Features such as timescale control and options between transient and DC analysis will either have to be delicately dealt with so as to not confuse or overwhelm the user, or be left out altogether.

3.1.4 Interactivity and Play

One of the critical design requirements to maintain is the importance of user interactivity—with interactivity, fun follows closely behind. The interface must be as active as possible in order to prevent the user from playing a passive role in the circuit assembly and simulation process. Since TouchSPICE is a graspable UI, it adopts a *space-multiplexed* input design and should make interactivity come naturally, as the user will be forced to use the touchscreens and wires to create the initial circuit. Once the simulation is running, the user must also interact with the touchscreen controls to change circuit element parameters.

3.1.5 Intuitive UI

Since one of the foremost goals of this thesis was to create both TouchSPICE and ReActive-SPICE while optimizing and streamlining the HCI elements to make the user experience as positive and transparent as possible, an intuitive UI is the key to giving the user a seamless learning experience with little to no prior knowledge of how SPICE or even circuits work. According to Scott Klemmer, an Associate Professor of Computer Science at Stanford University and HCI class instructor, bad user interface design is frustrating and hinders the purpose of learning. The best interfaces become invisible to us—when it becomes automatic, our attention shifts from manipulating an interface to accomplishing a task (such as learning). That’s when an interface becomes intuitive, when it becomes invisible [15].

3.1.6 Power Consumption

The TouchSPICE hardware blocks all need to be capable of operating off batteries for at least five hours; sustainability and leaving a small footprint are always on any project agenda of today. Both simulator solutions should use as minimal power as possible.

3.2 Scalability Analysis

Being HCI lab tools, both TouchSPICE and ReActive-SPICE simulators should be scalable and expandable; to accomplish this, the codebase needs to be very modular and portable (multi-platform). One of the design decisions was to make it so that each of the hardware blocks could be interchangeable and all have the same base source-code as the other. This would make software changes quick and manageable. However, this added a certain complexity to the software design of the blocks which will be discussed in the software design chapter.

Chapter 4 TouchSPICE System Design

4.1 TouchSPICE Hardware

A complete analysis on component selection was performed and tabulated in order to identify the most cost-effective and fitting solution to meet design criteria and requirements.

TouchSPICE needed to incorporate a development platform (board) with an integrated touchscreen, a form of wired inter-node communication and a form of wireless communication. The development board was needed to not only display a user interface and provide the touchscreen HCI, but also for all of the wired inter-node communication and wireless host communication to relay circuit layout and parameters to the host computer for netlist generation and simulation.

The original TouchSPICE high-level hardware concept can be seen in Figure 4.1. The concept was to use the development boards as touchscreen interfaces to display the discrete electrical components and use the host computer for control, simulation and displaying waveforms. The targets or nodes would be connected together using wires to enable the physical visualization of the circuit. The nodes would all communicate wirelessly with the host computer to send circuit parameters and topology. This model will slightly adapt throughout the design process as optimizations and modifications are made.

Figure 4.1 – Original TouchSPICE High-Level Hardware Concept

The host laptop selection will not be discussed because any mid-performance laptop that is available for less than \$700 should fit the design requirements appropriately.

4.1.1 Development Platform

Instead of heavily focusing on board layout and manufacturing for the thesis project, it was decided that it was best to utilize an existing development platform that could later be used to not only expand on the thesis project, but also be used for future generic application development. Three development boards were chosen to compare because they were the top three development boards available on the market with the form-factor and processing capabilities that were needed. At the bare minimum, each board needed to have a touchscreen LCD display, a processor fast enough to draw the screen and relay data quickly, a secure digital (SD) card for storing images and the root file system, universal asynchronous receiver/transmitter (UART) for a simple protocol to

facilitate wireless communication and a few general purpose input/outputs (GPIOs) for wired inter-node communication. Ideally, each node would be able to support an open-source operating system (OS) that could be used to expand upon in the future.

4.1.1.1 FriendlyARM Mini2440/Mini6410 (ARM9/ARM11)

Figure 4.2 – FriendlyARM Mini2440 Development Board (Android OS displayed)

The FriendlyARM 2440 and 6410 were complete development boards which utilized Samsung's S3C2440A 405MHz processor (ARM9) and Samsung's S3C6410 533MHz (ARM11) processor respectively. FriendlyARM is an organization founded in Germany which now has many global distributors and worldwide support. These were the smallest development boards out of the three—they fit inside a 4"/4.3" rectangle. The boards included a power supply, serial cable, USB device cable, Ethernet cable, JTAG connector and cable, software and documentation DVD, and a telescoping stylus.

Documentation is found all over the internet along with software and examples. The price for one of these development boards was \$120/\$190 respectively.

4.1.1.2 MicroBuilder LPC1343 (ARM Cortex M3)

Figure 4.3 – MicroBuilder LPC1343 Development Board

The MicroBuilder LPC1343 was a semi-complete development board which utilized NXP's (formerly Phillips) LPC1343 72MHz ARM Cortex-M3 processor. The MicroBuilder development board was designed by an open-source organization which can be found at MicroBuilder.eu. Included is the full LPC1343 development board with the 2.8" TFT Touch LCD+SD breakout board. The price for one of these development boards was \$80, which was by far the cheapest development board out of the three.

4.1.1.3 OMAP Zoom eXperimenter (TI OMAP)

Figure 4.4 – OMAP Zoom eXperimenter Development Board

The OMAP Zoom eXperimenter was a complete development board which utilized TI's OMAP-LPC1343 C6-Integra DSP+ARM processor which is a low-power applications processor based on an ARM926EJ-S and a C674x DSP core. The Zoom OMAP-L138 eXperimenter Kit included the OMAP-L138 SOM-M1, application baseboard, 4.3" LCD panel, accessories, and software required to immediately begin development work. The price for one of these development boards was \$495.

4.1.1.4 Overall Development Board Comparison

Table 4.1 – Development Board Selection Comparison

	Mini2440	Mini6410	MicroBuilder	OMAP Zoom
CPU/SOC	Samsung S3C2440A	Samsung S3C6410	NXP LPC1343	TI OMAP-L138
Core/Clock	ARM920T (405MHz)	ARM1176JZF-S (533 MHz)	ARM Cortex-M3 (72MHz)	ARM926EJ-S (456MHz)
RAM	64MB SDRAM	256MB DDR2	8KB SRAM	128MB DDR2
Flash	128MB NAND	1GB NAND	32KB On-Chip	N/A
LCD	Toppoly 3.5" 240x320	Sharp 4.3" 480x272	2.8" color TFT touchscreen	4.3" LCD Display
Touchscreen	Integrated 4 wire resistive touchscreen interface	Integrated 4 wire resistive touchscreen interface	Integrated 4 wire resistive touchscreen interface	Integrated 4 wire resistive touchscreen interface
Ethernet	RJ45 10/100M	RJ45 10/100M	N/A	RJ45 10/100M
Serial	1 DB9 RS232 COM0 2 TTL COM0, 1 with 4-wire sockets 1x4 2mm	1 DB9 RS232 COM0 4 TTL COM0, 1, 2, 3 with 4-wire sockets 1x4 2mm	UART/I2C/SPI	115.2kbps RS-232 debug serial port
USB	1 MiniUSB Device USB2.0 1 USB Host USB1.1	1 MiniUSB Device USB2.0 1 USB Host USB1.1	USB 2.0 HID and Mass Storage support built right into the ROM	1 USB SD card reader Serial cable (null-modem) 1 USB 2.0 high-speed On-the-Go interface 1 USB 1.1 full-speed host
Audio	Stereo in/out - 3.5mm Jack Built in Microphone	Stereo out - 3.5mm Jack Built in Microphone	N/A	Stereo in/out connectors TLV320AIC3106 audio codec
TV	N/A	RCA Jack Composite TV output	N/A	N/A
IR	N/A	Infrared Receiver	N/A	N/A
SD	SD Card standard	SD Card standard	SD Card standard	SD Card standard
SDIO	2 IIC channels 2 SPI channels	SDIO header for SDIO Wifi, etc. + SPI and IIC. 2x10 2mm	Full Speed USB, TTL UART, SPI and I2C interfaces	3 UART 2 I2C / 2 SPI 1 SATA
User Inputs	6 push buttons 1 Pot 1 Power switch for 5V power	8 push buttons/interrupts 1 Reset push button 1 Pot 1 Power switch for 5V power 1 IR Receiver	1 Reset button 1 Bootload button	2 buttons
Indicators	4 User LEDs, Green	4 User LEDs, Green 1 Power LED, Red	1 User LED, Green 1 Power LED, Red	N/A

	1 Power LED, Red		1 USB LED, Green	
JTAG	10 Pin JTAG Header 2x5 2mm	10 Pin JTAG Header 2x5 2mm	Available to be soldered	Connectors for JTAG interface
LCD	40-pin header, 2x20 2mm	40-pin header, 2x20 2mm 41 pin Mini/Micro2550 style for FFC	Custom	Custom
GPIO	34-Pin header 2mm	30-pin header 2x15 2mm	Up to 42 General Purpose I/O (GPIO) pins with configurable pull-up/pull-down resistors	Up to 9 banks of 16-pins each
ADC	N/A	N/A	8 10-bit Analog-to-Digital Converter pins	N/A
Timer/ Counters	N/A	N/A	4 general purpose counter/timers with 4 capture inputs and 13 match outputs Programmable WatchDog Timer	Three 64-Bit General-Purpose Timers
Camera	20 pin Camera Interface	CMOS CAM130 or similar. 2x10 2mm	N/A	N/A
Bus Expansion	N/A	2x20 2mm	20-pin expansion connector	N/A
External Int.	8-pin interrupt and user button connector	Buttons/Interrupts 1x10 2mm	N/A	Programmable
RTC	Battery Backed RTC	Battery Backed RTC	System tick timer for easy timekeeping	On-board, no battery backup
Power Input	5V	5V	6-12V	5V
Software Support	Superboot - can auto-program flash from an SD card	Superboot - can auto-program flash from an SD card	No ARM or JTAG programmer is required! The chip comes with a built in USB bootloader that appears as a very small disk drive	U-Boot (bootloader/monitor)
OS	<ul style="list-style-type: none"> Linux 2.6.x kernel supports all I/O with file system Linux 2.6.32 with Qtopia 2.2.0 includes source and tools Android 2.0 Windows CE Demo 5.0 (TS support) uC/OS-II 	<ul style="list-style-type: none"> Linux 2.6.x kernel supports all I/O with file system Qtopia 2.2.0 + Qt + Qt/E Android 2.0 Windows CE 6.0r3.NET uC/OS-II 	<ul style="list-style-type: none"> MicroBuilder has written a full software library The software library includes complete GCC-based startup code and details on setting up an ARM development environment using open source tools. Linux 2.6.x 	<ul style="list-style-type: none"> Open source Linux DVSDK and demos Win CE SDK Code Composer Studio (CCS) v4 DSP/BIOS v5 Board Support Library (BSL) sample programs
PCB/Size	6 layer 4"x4"	6 layer 4.3"x4.3"	3.75"x2.75"x0.5"	8"x6"x0.5"

4.1.1.4.1 Power Consumption Analysis

Since each of the development boards would need to last at least five hours on a full-charge, a power consumption analysis needed to be done to make sure the correct battery is selected to maximize the device longevity and performance. With a 5V input, the Mini2440/6410 + 3.5" LCD uses 600mA peak current (3 Watts), 440mA when Linux is idling and the backlight is on, 290 mA when Linux is idling and the backlight is off. When Linux has shutdown, the system is waiting for a reset at 250 mA (1.25 Watts). A Mini2440/6410 with no display runs between 200 mA and 300 mA. The LPC1311 consumes roughly 1.5W max without the LCD and 2.5W max with the LCD. The OMAP Zoom pulls about 220mA of current at 5V without the LCD and roughly 410mA of current with the 4.3" LCD. Adding SD cards has a small effect. USB devices like Wi-Fi can increase the load quite a bit. Table 4.2 summarizes the power consumption of each development platform.

Table 4.2 – Development Board Power Consumption Summary

	Mini2440	Mini6410	MicroBuilder	OMAP Zoom
Peak Power	3W	3W	2.5W	2.05W

In conclusion, each node will consume a maximum of 3 watts, meaning a standard cellphone battery or 4-pack of rechargeable Li-Ion AAA's (~1200mAh) will keep the nodes powered for 20 hours on a full charge. The preliminary design will use wall-power; however power consumption still needed to be taken into consideration for future developments where portability will be more critical.

4.1.1.4.2 Development Platform Evaluation

An evaluation of the overall criteria was performed for the three development boards and tabulated in Table 4.3. Lower point scores are better. The ideal choice would be the lowest total value. The value in parentheses represents the weight of the parameter. The price, form factor and processor speed were the most heavily weighted (x2) criteria because those were the criteria most critical to the overall design.

Table 4.3 – Development Board Evaluation
(Lower point scores are better, number in parenthesis represents the weight of the parameter)

	Mini2440	Mini6410	MicroBuilder	OMAP Zoom
Price (2)	1	2	1	2
Form factor (2)	1	1	2	2
OS Flexibility	1	1	2	1
Processor Speed (2)	2	1	2	1
Assembly	1	1	2	1
Documentation	1	1	2	1
Power Consumption	2	2	1	1
Appearance	1	1	2	2
Total	14	14	19	16

As can be seen in Table 4.3, the design choice was fairly obvious, either the FriendlyARM Mini2440 or Mini6410 were the ideal candidates for the design. Due to budget restrictions and practicality, price was deemed more important than processor speed, thus the Mini2440 was the development board selected.

4.1.2 Inter-node Communication

A “standard” wire and connector needed to be determined to enable the TouchSPICE nodes to be connected together to form the virtual circuit. Two wire and connector solutions were proposed for simplicity and durability; much of the wire selection decision was highly dependent on the actual design of the inter-node communication protocol, which will be further discussed in the firmware design chapter.

4.1.2.1 Banana-to-Banana Connectors (1-wire)

The main advantage of using a banana-to-banana wiring solution is the abundant supply and availability of them in the labs along with the fact that they are already standardized and have connectors. The size of the wires simultaneously poses advantages and disadvantages. Since the wires are large in diameter and have thick insulation, they would be robust and easy for the user to plug and unplug; the connectors could also be stacked to increase the fan-in and fan-out of each of the devices. However, since the wires are bulky, they use additional space and would require larger connectors on each of the blocks, further increasing the form factor and cost of the overall design.

4.1.2.2 Twisted Pair Connectors (2-wire)

Twisted pair wire, which can be taken out of CAT5/6 cable, would provide a cost-effective, small form-factor solution for inter-node communication and have the advantage of reducing noise and cross-talk between wires. This communication method allows the nodes to both send and receive data on separate lines. The disadvantage of using twisted pair or any wire pair for that matter is that connectors would need to be

retrofitted and would require more soldering and materials. A solution to this would be to solder 2-pin male/female headers to the wires to create a simple male to female header connection.

4.1.2.3 Wired Connections Evaluation

A major disadvantage of using a form of 1-wire or 2-wire communication protocol is the lack of a clock signal for communication synchronization and the lack of the ability to send and receive acknowledgements without major timing considerations. Nonetheless, the asynchronous protocol could be developed by having all the devices controlled by a coordinator (the host computer) that would signal when a node is supposed to transmit and then receive the node identification information in a round-robin fashion. All nodes that are not signaled by the coordinator will strictly be listening for node identification information and when it receives the information, it will send its node identifier back and the coordinator could move on to the next node and all that follow.

An evaluation of the overall criteria was performed for the two types of wired connections and tabulated in Table 4.4. Lower point scores are better. The ideal choice would be the lowest total value. The value in parentheses represents the weight of the parameter. The connector size and reliability of the 2-wire protocol, which will be discussed further in the firmware design chapter, was the most heavily weighted (x2) criteria because those were the criteria most critical to the overall design.

Table 4.4 – Wired Connections Evaluation
 (Lower point scores are better, number in parenthesis represents the weight of the parameter)

	Banana-to-banana	Twisted Pair
2-wire vs. 1-wire Easy Connection	1	2
2-wire vs. 1-wire Reliable Protocol (2)	2	1
Connector Size (2)	2	1
Stackable	1	2
Durability	1	2
Noise	2	1
Total	13	11

One of the major pitfalls of the 1-wire protocol was the protocol's complexity and delicacy. 1-wire setups typically require additional external circuitry (such as capacitors and Schottky diodes) to protect against signal undershoot, reflections and voltage reversals [16 & 17]; thus a 2-wire communication protocol was chosen to simplify the communication protocol and to also enable data to be exchanged much faster, improving the real-time response of the system.

4.1.3 Wireless Connections

While all of the nodes will be using wired communication to talk with each other, a wireless solution is required for the nodes to communicate with the associated host computer, which will perform the node coordination and SPICE simulations. Having a wireless means of communication on-board each node also would allow for future expansion. For example, if a full wireless solution was desired, it would be possible to

have a wire-free demonstration of the same HCI tool. Some important factors in deciding the wireless communication method include size, power consumption and bandwidth.

4.1.3.1 Zigbee Wireless

The Zigbee protocol is an IEEE/802.15.4 RF solution for wireless communication. Digi International's Xbee modules are popular devices that use the Zigbee protocol to provide a simple solution for wireless end-point connectivity. Xbee modules come with a simple setup tool that allows for easy configuration and operation. The modules use simple UART communication which can be emulated as a serial port on both the computer and the nodes.

Figure 4.5 – Digi International's Xbee module with UART Breakout Board

Tutorials for configuring Xbee modules are readily available online, which results in quick setup. Xbee modules can be configured as either a coordinator or as a router and must have the same baud rate, personal area network (PAN) and channel number in order to communicate with each other. For example, all of the nodes would be configured as routers and the host computer would be configured as a coordinator.

The downside to using Zigbee transceivers tends to be the form factor. Many modules come as an oddly shaped pentagon and require a breakout board for proper connections.

4.1.3.2 Bluetooth Wireless

Bluetooth, an IEEE/802.15.1 RF solution for wireless communication, is used in a wide variety of consumer electronics such as cell phones, personal computers and audio devices. This would make host integration much easier because most laptops already come with an internal Bluetooth card. Even if the host computer didn't have an internal Bluetooth card, a low-cost dongle could be purchased for less than \$25 to enable Bluetooth communication on the host.

Similar to Zigbee, a Bluetooth solution would require a module on every node and would utilize UART communication by emulating a serial port. A Bluetooth module however, has a more convenient form factor and typically utilizes less space than Zigbee alternatives.

Figure 4.6 – Bluegiga’s WT-12 Class 2 Bluetooth module with UART Breakout Board

Bluegiga’s WT-12 class 2 Bluetooth modules are fairly inexpensive (~\$40) and have a very small form-factor. The WT-12 modules would also require a breakout board to utilize only its UART capabilities.

4.1.3.3 Wireless Evaluation

Table 4.5 – Bluetooth vs. Zigbee Features

	Bluetooth	Zigbee
Frequency	2.4 GHz	2.4 GHz
Antenna	Internal	External/Internal
Data Rate	3000 kbps	250 kbps
Interface	UART	UART
Range	40 m	1 km
Power	4 dBm	10 dBm
Sensitivity	-84 dBm	-100 dBm
Size	14mm x 25.6mm x 2.4mm	27mm x 24mm x 9mm

Table 4.5 shows that the WT-12 Bluetooth module provides a faster transfer rate with a much smaller form factor; the range of Bluetooth modules is only a fraction of the X-bee modules, however this is not an issue as each of the 10 nodes will be easily within arm's reach of each other.

Overall, as seen in evaluation Table 4.6, the Bluetooth modules will provide the same wireless requirements (UART) as the X-Bee modules but with a smaller form factor and will have the capability to expand on all the profiles Bluetooth offers, such as PAN, SPP, DUN, HID and HCRP for later project development. Lower point scores are better. The ideal choice would be the lowest total value. The value in parentheses represents the weight of the parameter, thus Bluetooth was selected as the wireless communication protocol.

Table 4.6 – Wireless Connections Evaluation
(Lower point scores are better, number in parenthesis represents the weight of the parameter)

	Bluetooth	Zigbee
Data Rate (2)	1	2
Range	2	1
Power	1	2
Form Factor (2)	1	2
Total	7	11

4.2 TouchSPICE Software

Since the design was to be completely open-source, expandable and scalable, the operating system and programming environment needed to support robust and simple multi-application development. Hardware selections dictated the majority of the software decisions. Since the FriendlyARM Mini2440 was selected as the development board, there was the option to use Linux 2.6, Android, Windows CE or no operating system. Either Java or C/C++ could be used in any of these operating environments; however one that would optimize code development and provide an easy platform for future development of a wide-variety of applications needed to be chosen.

4.2.1 Operating System and Programming Environment

Each node could either have the HCI application running on top of an OS or the HCI application could run directly off of the microprocessor acting as its own OS. To exponentially reduce the amount of OS programming needed, an existing OS will be utilized to save time rather than having to implement things such as custom serial port

emulation and native screen rendering/display drivers. The design decision to choose an established platform/OS would also make future development easier and more modular.

A common and easily scalable OS choice would be a Linux distribution such as 2.6 which comes with the FriendlyARM development board. Qtopia, an older GUI development application and engine by Trolltech (which was bought by Nokia in 2008, and sold to Digia in 2012 [18]) comes with the FriendlyARM as well, which would make developing the human-computer interface even simpler and more powerful. Linux was also compatible with multiple languages such as Java or C++.

Another solution would be running an Android distribution. This would require writing the node program in an Android App format which would make node software deployment possible on cell-phones as well. Conversely, there are many unwanted and unneeded features of the Android OS itself; a language such as Java or C++ would provide a more precise and tailored solution to the problem.

Windows CE is a graphical interface for Windows that can be navigated using touch. This would be nice because the GUI would be taken care of; however, much of the features of this operating system would go unused for this project and would result in unwanted processor consumption.

The final OS decision was to use Linux 2.6 with Qt 4.6, a new version of Qtopia, running on top of it, which supports running C/C++ along with the GUI. Java, an open-source object-oriented language by Sun Microsystems, runs on a virtual machine and was not chosen as the final language because although it is cross-platform compatible, running a virtual machine is not as efficient when it comes down to using

processor/memory resources and may result in higher power consumption and unwanted processor time spent on running the virtual machine itself.

4.2.2 Wireless Communication Protocol

A few of the different wireless communication protocols were taken into consideration. Node-initiated coordination, host-initiated coordination and time-sync communication were all taken into account and discussed to determine the most efficient means of wireless communication.

4.2.2.1 Node-initiated Coordination (Scheduled Nodes)

When the nodes are powered on, they could attempt to discover all available nodes. After a total number is determined, the nodes could be assigned a turn order to transmit within. Once the initial netlist is generated, the nodes are free to send updates as needed. In order to ensure that updates are correctly processed, a handshake system could be implemented. The component node will send its update. Once the host has received this new netlist component, it will send a response to the node, acknowledging that it received its update. If no acknowledgement is received, the node will continue to transmit its update in a random interval until the host sends its ack. See Figure 4.7 for an example of node-initiated coordination.

Figure 4.7 – Node-initiated Communication (Scheduled Nodes) – Each of the nodes are free to send updates as needed, if the host receives the update, it sends an ACK (handshake), if not, the node randomly broadcasts until the ACK is received.

4.2.2.2 Host-initiated Coordination

All communication could be initiated by the host. It could ask each node if it has new information for it, and then process the information accordingly. This would result in the slowest updates, as a node could be ready with new data while host is asking other nodes for data when they have nothing new to transmit. The benefit to this is that there will be no collisions/noise over the network, as every node will be waiting for contact to be initiated by the host. Only then will the nodes be allowed to respond. This causes a lot of excess traffic on the network because even if a node does not have an update, the host will still ask it for one, wasting time and bandwidth. See Figure 4.8 for an example of host-initiated coordination.

Figure 4.8 – Host-Initiated Communication – Each of the nodes waits for the host to request an update from it in order. This method is slow, but sure; however it also causes a lot of excess traffic on the network.

4.2.2.3 Time Synchronization

In order to avoid conflicts when sending and receiving data, all nodes would first sync up with the host's real-time clock and then keep track of time/counts locally. As each node comes online, the host assigns them a time-slot to broadcast/communicate within (with a slight grace/buffer period to avoid any slight chance of error). Since the time slots are very small, communication should be able to occur seamlessly and as if all the nodes could communicate at the same time. In reality, each node would need to wait its turn in order to relay its component update to the computer. If the sync is performed each time the nodes are started up, there should be no timing conflicts between nodes trying to transmit/receive. This method would get rid of the excess traffic on the network. See Figure 4.9 for an example of time sync communication.

Figure 4.9 – Time Sync Communication – Each of the nodes waits for its timeslot and then only sends an update when it needs to. This method gets rid of the excess traffic problem

4.2.2.4 Wireless Communication Protocol Evaluation

Although the time sync communication protocol would get rid of excess transmission traffic, the timing considerations are not feasible using this many devices. The chances of the on-board RTCs becoming unsynchronized are too high and risk everything becoming completely out of sync with each other. The simplest wireless communication protocol to implement would be host-initiated because it has the minimum number of race-conditions. However, this design decision will affect the overall performance of the real-time simulation because the host will constantly be flooding the nodes with requests. For the prototype, host-initiated will be used and can be fine-tuned later.

4.3 TouchSPICE High-level Concept

After all of the hardware components and software platforms/environments were selected, a modified high-level concept diagram needed to be recreated and is shown in Figure 4.10. Using the Mini2440 development board meant that a widescreen LCD would be used, which slightly modified the design of the user interface.

Figure 4.10 – Modified TouchSPICE Widescreen High-Level Hardware and GUI Concept

The overall system design flow is fairly simple at a high level of abstraction. After the nodes are turned on and the host computers application is started, the TouchSPICE simulator will be functional.

The main purpose of connecting wires to the nodes is to create an illusion of creating a physical circuit, like you would on a breadboard. The wires will also serve as a communication medium to transfer neighbor identification information to the nodes connected. After this information is collected and aggregated on the host, a netlist is able to be generated. For example, in a SPICE netlist, there typically is a component reference designator, an input net, an output net and a value. This can be seen in Figure 4.11.

```
|-----> Component reference designator
|  -----> Input net
|  | -----> Output net
|  | | -----> Value (options)
|  | | |
v1 1 0 dc 24
v2 3 0 dc 15
r1 1 2 10k
r2 2 3 8.1k
c1 2 0 4.7u
```

Figure 4.11 – Sample SPICE Netlist

In order to determine the input and output nets of the component, circuit topology information is needed. The host computer, or coordinator, will first assign a node identification letter (NIL) to each node (i.e. A, B, C) in a pre-specified order based on Bluetooth MAC addresses. After this process is completed and all active nodes have been assigned a NIL, the host will ask each node, in order, who its neighbors are. This is where the inter-node communication comes in. Once the node has been signaled to identify its neighbors, it will send out its own identification letter to all of its connected inputs/outputs (I/O) in order (i.e. 0, 1, 2, 3). The I/O number designation is labeled in

Figure 4.12 and Figure 4.13. The node attached to the other end of the current sending I/O will reply back with its identification letter and store the one that was sent to it to speed up the neighbor identification process and limit redundancy. The initial sending node will then send out the respective I/O number to all of its I/O. The node attached to the current sending I/O will reply back with its I/O number and store the one that was sent to it.

Figure 4.12 – I/O Number Designation on a Node with tied physical inputs/outputs

Figure 4.13 – I/O Number Designation on a Node with unique physical inputs/outputs

For example, suppose node A I/O 3 and node B I/O 1 are connected; node A sends its identity letter out of I/O 3 to node B I/O 1, node B stores that I/O 1 is connected to node A and then sends node A its identity letter. Node A I/O 3 sends its I/O number (3) to node B I/O 1, node B stores that I/O 1 is connected to I/O number 3 on node A and then sends node A its current I/O number back. The transaction is complete for that single connection—A has identified B and B has identified A. A will now move on to any other I/O that are connected. After node A has identified all of its neighbors successfully, it will send its neighbor identification to the host and the host will move onto node B, which has already identified at least one of its connections (to node A) and does not need to repeat the transaction. As you can see, this will exponentially speed up the neighbor identification process as the host progresses through each of the nodes because by the time that it gets to the last node, all of the other nodes will have identified its connections and it will have no wired transactions to make, it will simply send the host what it has stored. This protocol is illustrated in the SW flowcharts in Figure 5.5 and Figure 5.6.

Once the neighbor identification process is complete, the host has the complete topology of the virtual circuit. Next, the host goes around again and asks each of the nodes for their component name, value and what the user would like measured (current or voltage) on each I/O. After the host receives all of that information, a complete netlist can be generated and a SPICE simulation can be run. The host computer will create a file with all of the plot data necessary and display it to the user. This process is repeated, and if any component values or measurements change, the waveforms will be updated appropriately. If any connections (wires) are removed while running the real-time simulation, the simulation will stop and the neighbor identification process will need to

be repeated. The operation of the hardware is shown in Figure 4.14. The high-level block diagram of the node hardware is shown in Figure 4.15.

Figure 4.14 – Operation of Hardware Flow Diagram

Figure 4.15 –High-level node hardware block diagram

Chapter 5 TouchSPICE Firmware Design

This chapter will cover the main driver and application programming interface (API) design for the TouchSPICE hardware (firmware). This includes researching and designing a two-wire communication protocol, driver and API and developing an API for the Bluetooth communication modules.

5.1 Minimum Fan-in and Fan-out Considerations

Since all nodes were to be physically identical, consideration of the minimum fan-in and fan-out that each node must be able to accommodate, and methods for exceeding it was necessary. There needed to be enough input and output connections available on each node to accommodate for all of the common circuit components and configurations. For example, ground and power supplies tend to accumulate the most nets in a circuit schematic or netlist, this needed to be accommodated for. The first idea, which turned out to be the final idea, was to have 4 input/output connections, 2 on opposite sides of the board, to accommodate for FETS and OPAMPS. If it was a single input and/or output component, the 2 inputs would essentially be tied together and/or the 2 outputs would be tied together. This should result in the user being able to complete most of the basic circuit configurations by taking advantage of the tied inputs and using them like a wire bus. An alternative would have been a dedicated wire-net component, which would essentially tie all the connected inputs and outputs together to form a single net.

5.2 Two-Wire Communication Protocol Considerations

To improve the effectiveness of HCI, a simple two-wire communication protocol needed to be designed that was specifically tailored for the project. A means for sending bytes quickly and reliably between the nodes was needed to minimize the delay before simulation. Also, since these wires were to be plugged and unplugged seemingly randomly at times, the protocol needed to know how to deal with this and how to flag a status change when something was plugged in or unplugged in near-real-time.

The 2-wire communication protocol turned out to be a vital non-trivial part of the design and research. Since each of the nodes were to be identical physically and were to have the same source code on them, a dedicated clock signal would not be possible. For example, if any “input/output” pair could be plugged into any other “input/output” pair, who would be the one to provide the clock signal? This would not work unless there was a clock for each unidirectional wire, which would require two clocks for each inter-node connection, resulting in a total of 8 separate clock inputs/outputs needed just for a single node. For simplicity no dedicated clock signal was used and an asynchronous approach to a UART-like protocol was taken.

5.3 Two-Wire Communication Driver Design

Since there were to be 4 I/O connections on each node (2 inputs, 2 outputs) and there needed to be 2 wires (RX, TX) per I/O, a total of 8 pins were needed. Typically microcontrollers and microprocessors such as the ARM9 used in this project have built-in UARTs and I2C protocols which can be utilized in applications on top of the kernel to make communication programming easier. Unfortunately, there were only three UARTs

on-board the Mini2440 and one I2C bus. One of the UARTs was connected to an RS232 logic converter to enable a DB9 connection to a computer for console access and another was going to be used for the Bluetooth wireless module, leaving a single UART left for application use. Thus a communication protocol needed to be developed using external I/O that was abundant—the GPIO header, which allowed access to several external interrupt pins (EINTx) that could be used as GPIO.

EINT0	N17	EINT0/GPF0	EXT INT
EINT1	M16	EINT1/GPF1	
EINT2	L13	EINT2/GPF2	
EINT3	M15	EINT3/GPF3	
EINT4	M17	EINT4/GPF4	
EINT5	L14	EINT5/GPF5	
EINT6	L15	EINT6/GPF6	
IRQ_LAN	L16	EINT7/GPF7	
EINT8	N9	EINT8/GPG0	
EINT9	T9	EINT9/GPG1	

Figure 5.1 – External Interrupts on CPU3 mapped to GPIO header

Figure 5.1 highlights the 8 external interrupt pins on CPU3 that were used as GPIO. The 8 pins spanned across 2 data/control registers ‘F’ and ‘G’. Traces on the Mini2440 were routed to expose these external interrupts on the GPIO header which allowed for easy connection using a modified GPIO header ribbon cable. The 8 pins used on the GPIO header can be seen in Figure 5.2.

Figure 5.2 – The 8 (EINT0-5, 8-9) Pins Used on the GPIO Header

EINT6 was avoided because this had a secondary function as IRQ_LAN on CPU3 and no possible contention between the application and the kernel was wanted. Instead EINT8 and EINT9 were used on the GPG data register to keep things symmetrical. The most logical grouping of the 4 I/O was as follows in Table 5.1.

Table 5.1 – Target Software I/O Pair Mapping and Functions

I/O Number	Node Function	Input (EINTx) (RX)	Output (EINTx) (TX)
0	Input (top-left)	0	1
1	Input (bottom-left)	2	3
2	Output (top-right)	4	5
3	Output (bottom-right)	8	9

After deciding which GPIO pins were to be used for communication, the I/O needed to be properly initialized at application startup and a simple driver needed to be developed to read each input and write a value to each output .

An `initDigIO()` function was created based-off a function in the Embedded Linux Learning Kit (ELLK) written by Doug Abbott [19] which took in 4 parameters: the `input_bits[]`, `num_inputs`, `output_bits[]` and `num_outputs`. The input and output bits were the ones previously specified in Table 5.1 (note that each I/O pair's output was conveniently 1 more than its input, this made math and programming loops much easier to manage). First, all of the GPIO registers needed to be memory mapped to the system peripheral space and a file descriptor (`/dev/mem`) needed to be opened, using C++'s `mmap()` function. Then, for each input, the `write_reg()` function needed to be used to setup the registers properly. For example, to setup a pin as an input, the function call would look like:

```
write_reg(&GPIOp->GPFCON, pow(2,input_bits[i]*2)+pow(2,input_bits[i]*2+1),
GP_BIT(input_bits[i], PIN));
```

`&GPIOp` is the pointer to the base address of all the registers; `GPFCON` is the offset of control register 'F' (`*regp`). The second parameter is the `mask`, it is needed so that the rest of the bits in the register aren't clobbered when using a bitwise `&` or `|`. Notice the `mask` is 2 high-bits that are shifted to the corresponding `input_bits[]` places to create a `mask`—only those two bits in the register will be modified and changed to a particular value, this is where the third parameter comes in. The third parameter is used to specify the function of the GPIO (input: 00, output: 01, special: 10) or the `pattern`. This means 2 control bits are needed. `PIN` is simply a `#define` of 0 which will be left-

shifted 2*the number of input bits to create the pattern. For example, to setup pin 0 (I/O 0's input pin) as an input, a mask of 0x3 (0b0011) is created and PIN (0) is left-shifted by 0. The write_reg() function performs this operation:

```
*regp = (*regp & ~mask) | pattern;
```

This takes whatever regp is pointing to, bit-wise ANDs it with the complement of the mask and then “sets” or ORs it with the pattern. To setup pin 3 (I/O 1's output pin) as an output, a mask of 0xC0 (0b11000000) is created and POUT (1) is left-shifted by 6 (0b1000000 = 0x40) and the resulting operation is:

```
*regp = (*regp & ~0xC0) | 0x40;
Say *regp = 0b1101111111110011 = 0xDFF3
*regp & 0xFF3F = 0b1101111111110011 & 0b1111111100111111 = 0b1101111100110011
0b1101111100110011 | 0x40 = 0b1101111100110011 | 0b10000000 = 0b1101111101110011
```

Pin 3 was set as an output (01) while not affecting the rest of the control register bits. As proved, this can also be done to configure the GPIO pins as outputs. By default, pull-up resistors are enabled on all pins configured as inputs, however just to be sure, the dedicated pull-up control register (GPFUP) register was configured similarly to enable pull-up resistors on the inputs.

Similarly, two core driver functions writeDigOut() and getDigIn() were developed to use later as part of an API for sending and receiving bytes. These two functions can be seen here:

```

// Write the specified value to the GPIOF data register
void writeDigOut (char data_register, unsigned int bit, unsigned int value)
{
    if(data_register == 'f')
    {
        // Write to the GPFDAT register (register, bitmask, value)
        write_reg (&GPIO->GPFDAT, pow(2, bit), value << bit);
    }
    else if(data_register == 'g')
    {
        // Since pins 8 and 9 are actually pins 0 and 1 (respectively) subtract 8
        bit -= 8;

        // Write to the GPGDAT register (register, bitmask, value)
        write_reg (&GPIO->GPGDAT, pow(2, bit), value << bit);
    }
}

// Reads all bits in the specified GPIO data register
unsigned int getDigIn (char data_register)
{
    if(data_register == 'g')
    {
        return GPIO->GPGDAT;
    }
    else
    {
        return GPIO->GPFDAT;
    }
}

```

Additional APIs that utilize these core driver functions will be discussed later in the chapter.

5.4 Two-Wire Communication API Design

Once more, since this 2-wire communication scheme lacks a dedicated clock signal for synchronous communication, the design would need to take further measures to ensure data transfer integrity and would require a coordinator to provide the round-robin signaling of which node should initialize communication to prevent contention.

Standard UART communication begins with a start bit, followed by 8 data bits or 1 byte, a parity bit and a stop bit. This scheme was modified to meet the needs of the protocol. At a high-level overview, the only type of data that would be transferred was a single character (node identification letter [A-Z]) and a single positive number less than 4

(node I/O number [0-3]). Hence the UART scheme of a single byte transfer would work perfectly. However, data integrity needed to be the highest priority, so instead of using a simple single parity bit or a complicated cyclic redundancy check (CRC), a compromise was made with a 4-bit parity. In summary, the proposed transfer scheme was to consist of a start bit, followed by 8 data bits, 4 parity bits and a stop bit.

Since a “bit-banging” method of communication was resorted to, the luxuries of having the kernel or a predefined API deal with errors, timing and reliability issues were not enjoyed. These problems needed to be manually addressed by a custom driver and API that would abstract these low-level operations.

The first design decision: How to detect if there is no connection? The simple solution was to use the internal pull-ups on all the input pins and to make all the output pins drive low when not executing a transaction. This would mean that if there wasn't a wire connecting two nodes together, the I/O inputs would be pulled high. If there was a wire connecting two nodes together, the outputs would drive the opposing inputs low. Essentially, if each I/O were connected, all logic levels would be 0 as long as no transactions were taking place.

The second design decision: How to initiate a transfer? Instead of using a single start bit, it was more reliable to send a “hello?” bit, wait for an acknowledgment (ACK) that the connected node was ready to receive and then send a start transfer bit if the ACK was received before a specified timeout period. Thus, since when the wires were connected both lines were pulled low, the initiating node would pull the TX line high (logic 1) which flagged the intent to send a byte. The receiving node would ACK back

with a “101” because it would initially be at logic 0, and a change to 1 would be easily picked up by the initiating node, followed by a change in logic to 0 and then back to 1.

The third design decision: What to use for timing? Linux is not a real-time operating system (RTOS); it has multiple threads and sub-threads that are executing along with having to deal with external hardware such as NAND flash, SD card reading/writing, screen rendering and touch translation. This made the timing situation very difficult and hard to predict. The best that could be done was use the Linux `usleep()` function, which would sleep for the specified amount of microseconds. This however proved to be microseconds plus or minus a few 100 microseconds, depending on the OS-load and the current executing application section. After some experimenting and time shaving, a standard sleep rate value was selected that completed the most successful transfers in the least amount of time. The sweet spot was a 6000 microsecond period (6ms), which is the minimum amount of wiggle room required to accurately complete simultaneous transfers between nodes.

The fourth design decision: What type of 4-bit parity to use? Again to keep things simple and quick, the two adjacent bits in each byte were exclusively OR'd (XOR) together to create the 4-bit parity. For example, if the byte was 0b01001011, the 4-bit parity would simply be 0 XOR 1, 0 XOR 0, 1 XOR 0, 1 XOR 1. Which would be 1010; this operation could easily be done in a single statement as demonstrated:

```
// Calculate 4-bit sent_parity
for(i = 0 ; i < 4 ; i++)
{
    //          bit i          XOR          bit i + 1
    sent_parity[i] = !(byte & (1 << (i*2))) ^ !(byte & (1 << ((i*2)+1)));
}
```

The double exclamation points are a clever way to return the single bit value in a mask. For example, `!!0b000001000` would return 1 and `!!0b000000000` would return 0.

Finally, after most of the design decisions were made, an API was developed to enable easy 2-wire byte transfers. Two functions were created that encapsulated both core GPIO drivers: `gpioSend()` and `gpioReceive()`. These functions are thoroughly commented in the code in Appendix E and the function concept can be observed from the flow diagrams in Figure 5.5 and Figure 5.6. Figure 5.3 shows a block diagram of a simple 2-wire connection; notice that a crossover cable is used to connect TX to RX and RX to TX.

Figure 5.3 – Two-Wire Communication Block Diagram

It was found that for short distances, noise and cross-talk were not a problem, which was one of the main reasons for choosing twisted-pair. In addition, the CAT5/6 cable turned out to be very brittle and did not take well to having headers soldered to them. So instead

of using twisted-pair as originally decided, two attached female-to-female jumper wires (like the ones used for bread boarding) turned out to be an easier alternative because they already had a connection mechanism, were clean, small and readily available. A timing diagram of a single two-wire transaction can be seen in Figure 5.4.

Each 12-bit transaction ended up taking roughly 100ms to complete, so the transmission frequency was around 10Hz. This relatively slow transmission frequency definitely needs to be improved upon in the future to enable faster neighbor identification.

Figure 5.4 – Timing Diagram of a single Transaction between TX/RX0 and TX/RX2

Figure 5.5 – GPIO Send Byte Function Flow Diagram

Figure 5.6 – GPIO Receive Byte Function Flow Diagram

5.5 Bluetooth API

The WT-12 was a standalone surface-mount Bluetooth module that was mounted on a simple UART breakout board (Figure 4.6). This board allowed access to the RX, TX, 3.3V and GND lines needed for UART communication. The only problem that was encountered in physically integrating the device was the logic levels. The Mini2440 UART connectors operated on TTL 5V logic levels whereas the WT-12 operated on 3.3V logic levels. Although a simple resistor divider was used in the beginning, for appearance purposes, a logic level converter was assimilated into the mix to correct for the level difference. This logic level converter can be seen in Figure 5.7. It simply shifts the 5V logic levels (HV) down to 3.3V logic levels (LV) using surface-mount resistors and an N-Channel Logic Level Enhancement Mode FET (BSS138).

Figure 5.7 – SparkFun Logic Level Converter

Once the logic levels were taken care of, the module was plugged into one of the 2 on-board UART headers and direct device access (`/dev/ttySAC2`) was available in Linux using any C++ reader/writer or file stream. APIs for opening, reading and writing to the port were developed using standard streams and file I/O as can be seen below:

```

// Create the output file stream using ofstream (bt_stream)
std::ofstream bt_stream("/dev/ttySAC2");

// termios is the newer Unix API for terminal I/O (included in termios.h)
struct termios options;

// Open the device using fstream (bt)
bt = open("/dev/ttySAC2", O_RDWR | O_NOCTTY | O_NDELAY);
if(bt == -1) {
    printf("ERROR Opening Serial Port!");
}

// Serial Configuration
tcgetattr(bt, &options);
cfsetispeed(&options, B115200);
cfsetospeed(&options, B115200);
options.c_cflag = (options.c_cflag & ~CSIZE) | CS8;
options.c_iflag = IGNBRK;
options.c_lflag = 0;
options.c_oflag = 0;
options.c_cflag |= CLOCAL | CREAD;
options.c_cc[VMIN] = 1;
options.c_cc[VTIME] = 5;
options.c_iflag &= ~(IXON|IXOFF|IXANY);
options.c_cflag &= ~(PARENB | PARODD);

// Get Current Config
// Set input Baud Rate
// Set output Baud Rate
// Set cflag (data bits)
// Set iflag
// Set lflag
// Set oflag
// Set cflag
// Set cc VMIN
// Set cc VTIME
// Set iflag
// Set cflag

// Save Configuration
tcsetattr(bt, TCSANOW, &options);

// Flush the input (read) buffer
tcflush(bt, TCIOFLUSH);

// API to write a character using fstream
void btWriteChar(char output_char){
    write(bt, &output_char, 1);
    write(bt, "!", 1);
}

// API to write a string using ofstream
void btWriteString(const char * output_string){
    //write(bt, output_string, strlen(output_string));
    bt_stream.write(output_string, strlen(output_string));
    bt_stream.flush();
    bt_stream.write("!", 1);
    bt_stream.flush();
    //printf("Actually sent: %s\n", output_string);
    //write(bt, "!", 1);
}

// API to read a char using fstream
char btReadChar (void)
{
    char buffer[2];
    memset(buffer, 0, 2);
    read(bt, buffer, 1);
    return buffer[0];
}

// API to read a string using fstream
void btReadString (char buffer[])
{
    memset(buffer, 0, 255);
    read(bt, buffer, 255);
}

```

Chapter 6 TouchSPICE Software/UI Design

This chapter will cover both the software and user interface designs of the TouchSPICE targets (nodes) and the host computer. The high-level software flow diagram can be seen in Figure 6.1.

Figure 6.1 – High-level Software Flow Diagram

6.1 Target

Each target node needed to be responsive to inter-node communication, Bluetooth communication and touchscreen stimulation and feedback. To emulate this type of responsiveness, multi-threading needed to be utilized on the Qt C++ application running on top of the Linux 2.6 kernel on the FriendlyARM Mini2440s. Multithreading would allow the almost parallel execution of two threads, thus enabling the application to respond quickly to user stimuli while still being able to communicate between nodes and the host. Two threads were created: the main GUI thread, which handles all of the drawing of the display and user interaction, and a GPIO or communication thread, which handles all of the inter-node and host communication. High-level flow diagrams of the two threads can be observed in Figure 6.2 (GPIO/Communication thread) and Figure 6.3 (Main GUI thread).

The GPIO/Communication thread continuously is waiting to receive a command over Bluetooth from the host computer. If it receives a valid command, it executes the command and returns data if applicable. If nothing is currently being sent to the node, then the thread constantly checks its GPIOs to make sure that connections and disconnections are made when the user plugs or unplugs nodes. One of the Bluetooth commands is to identify node neighbors (to enable circuit topology rendering), this algorithm is displayed in Figure 6.4 (Neighbor Identification Algorithm). The GUI thread is solely in charge of reacting to touchscreen stimuli and emitting signals from the GPIO thread in order to update the touchscreen display to reflect changes.

Figure 6.2 – GPIO/Communication Thread Flow Diagram

Figure 6.3 – Main GUI Thread Flow Diagram

Figure 6.4 – Neighbor Identification Flow Diagram (from GPIO thread)

6.1.1 Graphical User Interface

The graphical user interface (GUI) on the nodes was one of the key HCI design challenges. The intuitiveness and quality of the GUI would ultimately determine whether a user would be able to build and simulate circuits in a matter of seconds, or a matter of hours. With limited display size (approximately 3.5" diagonally), the amount of screen clutter needed to be minimized and the elements on the touchscreen would need to be big enough for a user to easily use either the tip of their finger or a stylus—all this, while still enabling broad functionality and controllability. The GUI hit-box diagram for a component (capacitor) can be observed in Figure 6.5.

Figure 6.5 – GUI Hit-box Diagram for a Capacitor

The main element on the GUI was the component symbol, this needed to be maximized so as to enable quick and easy identification of which node was what component; again, the main goal is to build, simulate and interact with circuits in real-

time, so it is very important that identifying components and setting up the circuit is as trivial as possible. To reinforce the component symbol, the component name was also drawn on the top of the node's GUI to assist someone who may be learning what each of the component symbols look like.

In addition to the component symbol and name, there needed to be a way to display and adjust the value of the component (if applicable). For this, a scrollbar was used to make major value changes (quickly) and a spin-box was used to display the current value as well as to make any minor value changes (slowly) with the up/down arrows on the side. All of the components have their own parameters and their own values, meaning that appropriate units (ohms, farads, henrys...) needed to be displayed for each symbol as well as a way to adjust the scaling factor (k, M, m...) of each value. For this, a scale button was put on the side of the value display to not only display the current unit scaling factor, but also enable the user to change it by clicking on it, which would cycle through all of the scaling factors until the desired one was reached.

One of the next important things that would enhance HCI was the creation of connection statuses. The connection statuses are the little circles next to each physical input/output (I/O). To enforce user feedback, the circles would turn green if a successful connection to another node was made, or red if the I/O was not connected. Next to the connection statuses, checkboxes were created to enable the user to check/uncheck whether they wanted the simulation to probe/display the current or voltage on the corresponding wire.

After all of the main GUI elements were established, each node needed a way to switch gracefully between components. The easiest solution to this was to create a

component menu (see Figure 6.6) that would appear during startup; then, once a component was clicked, the component would appear and a back button would be drawn in the top-left corner of the display. This back button would take the user back to the component selection menu to make a new selection if desired.

Figure 6.6 – Select a Component GUI

Finally, in order for node identification, neighbor discovery and netlist creation, each node needed to be assigned a node letter [A-Z] as previously discussed. For convenience and visualization, once the node was assigned a node letter by the host computer, it would display the assigned letter next to the back button if a component was selected. This would not only let the user know that the host was successfully connected to the node, but also help them to topologically visualize how the netlist was being created.

For components that do not have any value parameters that can be adjusted easily in SPICE such as diodes, op-amps, BJTs and NMOSs, some controls are left off so as to not confuse the user. An example is shown in the Op-Amp component GUI in Figure 6.7. All component GUIs not shown in this chapter can be found in Appendix D.

Figure 6.7 – Op-Amp Component GUI (no additional value controls)

6.2 Host

The software on the host was responsible for coordinating all of the nodes, gathering circuit topology and simulating and updating the circuit by displaying the simulated waveform(s). The host had a single GUI thread which consisted of an event handler that monitored four buttons: connect, disconnect, simulate and stop. The high-level software flow diagram for the GUI thread can be observed in Figure 6.8.

The connect button would bind all of the Bluetooth MAC addresses of the node's Bluetooth modules (hard-coded) to the host's rfcomm channels, which is a Linux kernel API that provides a simple way to establish UART transmissions to and from Bluetooth modules. The only limitation to this protocol was the cap on the number of devices that could be binded, which was a total of 7 concurrent Bluetooth devices, meaning that the overall system would now only be able to consist of 7 nodes or circuit components unless a parking algorithm was created to temporarily unbind and park a device and then unpark another device and rebind it. This would require keeping track of all the

open/closed/parked devices and also would impact the overall latency of the system, so only 7 nodes were used during testing. After the Bluetooth devices on the nodes were binded to the host, input and output connection streams were established to allow data transmission.

Once wireless communication has been established between the host and all of the active (powered-on) nodes, the host proceeds to assign node identity letters to nodes in the same sequential order they were binded. The disconnect button on the GUI disconnects from all Bluetooth devices, unbinds the MAC addresses, flushes the input/output stream buffers and then closes the rfcomm port associated with that particular connection.

The simulate button removes an old plot if there is one, creates and formats the netlist (circuit topology) by asking all of the nodes for their neighbor information, runs the SPICE simulation, formats the output file to be used by gnuplot, creates the gnuplot script and then executes the gnuplot script to display the simulated waveforms. After the simulate button is clicked, this process repeats until the stop button is clicked—constantly polling the nodes for any value changes. With this method of simulation and plotting, the maximum update rate that can be achieved is proportional to the time it takes to read and write files from the hard drive. This is the most time consuming process on the host, next is the SPICE simulation itself followed by Bluetooth communication latency. Overall, to receive and display an update from the nodes onto the host computer, it can take anywhere from approximately 100ms to 1 second, as long as there are no hanging processes.

Figure 6.8 – Host GUI High-level Software Flow Diagram

6.2.1 Graphical User Interface

The host GUI was to be kept as simple, minimalistic and transparent as possible in order to keep the focus on circuit building and circuit manipulation on the nodes. So the host GUI only contained 4 buttons as mentioned earlier. While simulating, the host would display the current and/or voltage waveform(s) at the locations selected for probing on the nodes. The user would first need to connect to all of the nodes, and then once a complete circuit has been built, click the simulate button and continue to interact with the nodes in the circuit while the simulated waveforms are being updated in near-real-time. The host GUI can be seen in Figure 6.9.

Figure 6.9 – Host GUI MainWindow

Chapter 7 ReActive-SPICE Design

The next step was to create ReActive-SPICE, an integrated simulator design, to compare it to the functionality and intuitiveness of TouchSPICE. To simplify the design flow process and to keep the two simulation tools on comparable levels with each other interface-wise, instead of creating an entirely new look for the ReActive-SPICE simulator, the preliminary version (v1.0) of the integrated simulator simply emulated the TouchSPICE hardware blocks. This exponentially reduced the implementation time while keeping a comfortable environment that TouchSPICE users would be accustomed to. This would lead to more accurate comparison results in the user study.

7.1 Porting TouchSPICE to ReActive-SPICE

Essentially, the node software design and GUI from TouchSPICE were duplicated 4 times in a single window. Since there was no longer a need for two distinct threads for communication and simulation, the adaption process involved drastic source code changes and architecture changes to the way the signals and slots communicated. The same simulation and plotting algorithms were modified to work with the ReActive-SPICE integrated simulator. Again, this is one of the reasons why the Qt framework was chosen as a development and design tool, because code-reuse and modularity made it possible for a fairly smooth transition between the two tools as opposed to having to start over.

7.2 Graphical User Interface

Due to resolution and screen size limitations on the host computer, ReActive-SPICE v1.0's user interface consisted only of 4 “virtual” nodes. This would only allow for simple circuit building and experimentation (max 4 components) for comparison to TouchSPICE. The ReActive-SPICE v1.0 GUI can be seen in Figure 7.1 and Figure 7.2.

Figure 7.1 – ReActive-SPICE v1.0 GUI with 4 “virtual” nodes (SAC view)

Figure 7.2 – ReActive-SPICE v1.0 GUI with 4 “virtual” nodes (Circuit view)

Each “virtual” node could be set to emulate any circuit component, just as in TouchSPICE. Each “virtual” node had a component symbol, name, back button and voltage/current measurement check boxes on each input and output. Components where the value parameter could be adjusted (voltage/current source, resistor, capacitor and inductor) contained a horizontal slider (scrollbar) and spin-box that could adjust and display the current value of the component in the respective units with the selected scale (p,n,u,m,k,M). The code-block below represents one of the many Qt slots (functions) that perform a scale switch when the scale button is pressed:

```

void MainWindow::on_scale_button_4_clicked()
{
    switch(scale_factor_4)
    {
        case ' ':
            scale_factor_4 = 'k';
            break;
        case 'k':
            scale_factor_4 = 'M';
            break;
        case 'M':
            scale_factor_4 = 'p';
            break;
        case 'p':
            scale_factor_4 = 'n';
            break;
        case 'n':
            scale_factor_4 = 'u';
            break;
        case 'u':
            scale_factor_4 = 'm';
            break;
        case 'm':
            scale_factor_4 = ' ';
            break;
    }
    sprintf(buffer, "%c", scale_factor_4);
    ui->scale_button_4->setText(buffer);
}

```

This simple function changes the scale factor variable depending on the current scale factor variable. This enables the user to “scroll” through all of the scale values using a single button until the desired scale factor is reached.

Finally, on either side of the “virtual” nodes are net buttons, which will be discussed further in the next section.

7.3 Creating Circuits via Net Matching

Since there were no more physical wire connections, none of the GPIO or even the Bluetooth drivers or API’s were needed. Instead, nodes were connected together by matching net numbers to each other, much like creating a netlist manually would entail. Below is another Qt slot (function) that creates the net matching logic for each of the nets:

```

void MainWindow::on_IO_1_clicked()
{
    if(IO_1_net < 9)
    {
        IO_1_net++;
    }
    else
    {
        IO_1_net = -1;
    }
    if(ui->stackedWidget->currentIndex() != 5 &&
       ui->stackedWidget->currentIndex() != 6 &&
       ui->stackedWidget->currentIndex() != 7 &&
       ui->stackedWidget->currentIndex() != 0)
    {
        IO_0_net = IO_1_net;
        if(IO_1_net < 0)
        {
            ui->IO_1->setText("N");
            ui->IO_0->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_1_net);
            ui->IO_1->setText(buffer);
            ui->IO_0->setText(buffer);
        }
    }
    else
    {
        if(IO_1_net < 0)
        {
            ui->IO_1->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_1_net);
            ui->IO_1->setText(buffer);
        }
    }
}

```

This function firstly updates each “IO net” button when it is clicked to display the proper net number. Each “IO net” button will rotate through net numbers 0-9 and display the character ‘N’ when there is no connection. Most of the components are single input and single output, so the two “IO nets” on the input side need to be tied together and the two “IO nets” on the output side need to be tied together. In the source code above, if the current component is not 5, 6, 7 or 0 (opamp, BJT, NMOS or SAC), then IO_0_net is set equal to IO_1_net and the proper net number is displayed on both of them, effectively tying them together. If they happen to be components 5, 6, 7 or 0, then the “IO net”

buttons will act independently from its adjacent net. For more information, all source code with comments can be seen in Appendix G.

Even though the user may need a little more time and instruction as to how to get started and build the circuit using nets, this GUI will prove to be much faster at constructing circuits. This is because it is faster to click and create net connections than it is to connect wires. The circuit creation process is also faster because all of the intermediate communication steps and setup are no longer necessary. The simulation part behaved the same way as in the previous design and is fairly easy to understand. Screen shots of ReActive-SPICE v1.0 in action can be seen in Figures 8.14-8.17 in the next section.

Chapter 8 User Study

In order to receive comparative feedback on TouchSPICE and ReActive-SPICE, a user study was conducted consisting of 20 Cal Poly students, ranging from freshman-level to graduate-level Electrical/Computer Engineering majors. See Table 8.1 for a breakdown of user grade level.

Table 8.1 – User-base Grade Level

Grades					
Grade Level	Freshman	Sophomore	Junior	Senior	Graduate
Number of Participants	4	2	11	2	1

The user study comprised of a simple online (Google Docs) pre-survey and post-survey, which was anonymous. This user study included mostly questions that would allow comparative conclusions to be drawn about traditional SPICE simulation tools, TouchSPICE and ReActive-SPICE. Since a learning effectiveness study was being concurrently performed, the pre-survey consisted mainly of beginner-level circuit analysis questions but also asked the users what SPICE simulation software they have used before. Table 8.2 below summarizes these results. As can be seen from the survey results, LTspiceTM was the most widely used and favorite SPICE simulation tool—this will act as our basis for HCI comparison in the next chapter.

Table 8.2 – User-base SPICE Software Experience

SPICE Software Used				
LTspice	PSpice	Microcap	None	lcap
19	14	7	1	1

After the users filled out the pre-test survey, they proceeded to interact and build circuits on both the TouchSPICE and ReActive-SPICE simulators. The users were specifically given no instructions or hints on how to use the simulators; this was to enable gathering accurate time durations on how long circuit creation/simulation took from start to finish. Most users picked up the concept within a matter of seconds—others took upwards to two minutes. Tabulated data collected after the user study can be found later in this chapter. A typical user went through the following process to build and simulate a circuit using the TouchSPICE simulator:

Figure 8.1 – Single node (A) before being powered on. The Bluetooth module is sticking out at the bottom for reception purposes. The SD card seen sticking out of the top hosts the root file system for the Linux OS, this is where the main application is stored. The four 2-pin male headers are the custom GPIO made easily accessible for connecting nodes together.

Figure 8.2 – Single node (A) powered on. After the Linux Kernel is loaded, the standard UNIX penguin is displayed while the main application starts up.

Figure 8.3 – After the application starts up, the component selection screen is the first screen visible to the user; any component can be selected with either the user's fingertip or the provided stylus.

Figure 8.4 – After the “V Source” is clicked from the select a component screen and the AC waveform is enabled by pushing the AC/DC toggle button located at the center-left of the component symbol.

Figure 8.5 – Another node is turned on and plugged into the AC source. Notice the white dots on the wires are facing up on both nodes and wires; this is to indicate a correct crossover connection for communication. The connection status light has turned green for the plugged-in wire, meaning the connection is good. If the wire is not plugged in correctly, the respective connection status light will not illuminate to green, it will stay red, informing the user that no connection has been detected.

Figure 8.6 – Notice the connection status light is green only for the connection with a wire attached.

Figure 8.7 – An additional node is turned on and a low pass filter is constructed.

Figure 8.8 – User adjusting the resistor value via the slider bar (scrollbar)

Figure 8.9 – User adjusting the resistor value via the slider bar (scrollbar)

Figure 8.10 – User adjusting the resistor value via the slider bar (scrollbar)

Figure 8.11 – User adjusting the resistor value via the spin-box buttons for fine-tuning.

Figure 8.12 – Host GUI showing the user-created low-pass filter. The green waveform is the AC input; the red waveform is the AC output of the LPF.

Figure 8.13 – User increased the resistor value in the circuit, so the red output waveform of the LPF decreased, demonstrating the inversely proportional nature of the resistor value and the output.

After the users simulated circuits using TouchSPICE, they were transitioned over to the integrated ReActive-SPICE simulator on a different computer where they were to start the process over again. Some of the users were building specific circuits, such as low-pass filters, inverting amplifiers, non-inverting amplifiers, voltage dividers and current dividers and others were just making whatever they felt like.

During the integrated ReActive-SPICE simulator HCI, users would often times stare at the 4 “virtual” nodes and wonder how they could connect them together to form a circuit. After they clicked around for a little bit, they were able to successfully locate the connection net buttons on either side of the nodes and create a circuit by net matching. Nets were connected if they had the same number.

Figure 8.14 – ReActive-SPICE simulator v1.0 – initial startup

Figure 8.15 – ReActive-SPICE simulator v1.0 – Inverting amplifier design/simulation

Figure 8.16 – ReActive-SPICE simulator v1.0 – Inverting NMOS design/simulation

Figure 8.17 – ReActive-SPICE simulator v1.0 – Inverting NMOS design adjusted

After the user had experienced HCI with both TouchSPICE and ReActive-SPICE simulators, a post-survey was conducted that asked the following questions (Scale: 1 low/bad, 5 high/good):

1. Was there anything confusing or unclear as to what needed to be done to build and simulate the circuit using the TouchSPICE simulator?
2. On average, how long did it take you to figure out how to choose a component, set the value, connect the wires and simulate the circuit using the TouchSPICE simulator? [seconds/minutes]
3. How easy was the new TouchSPICE simulator to use? [1-5] Explain
4. Would you recommend either of these real-time interactive SPICE simulator tools to your friends as a way to learn circuit concepts? Why or why not?
5. Did the TouchSPICE nodes make it easier to understand the results? [1-5] Explain
6. How intuitive was the TouchSPICE node user interface? [1-5] Explain
7. How intuitive was the TouchSPICE host user interface? [1-5] Explain
8. How intuitive was the ReActive-SPICE v1.0 user interface? [1-5] Explain

9. Was there anything confusing or unclear as to what needed to be done to build and simulate the circuit using ReActive-SPICE v1.0?
10. How easy was ReActive-SPICE v1.0 to use? [1-5] Explain
11. On average, how long did it take you to figure out how to choose a component, set the value, connect the wires and simulate the circuit using ReActive-SPICE v1.0? [seconds/minutes]
12. Did ReActive-SPICE v1.0 make it easier to understand the results? [1-5] Explain

These questions were very comprehensive and a large amount of positive and negative feedback was collected from the users. All of the feedback is summarized and analyzed in the next chapter.

Chapter 9 HCI Analysis

Since this thesis involves creating TouchSPICE and ReActive-SPICE from an HCI standpoint, many of the HCI aspects, such as user interface design, usability, intuitiveness and task accomplishment will be discussed in this chapter. As mentioned earlier, bad HCI design is frustrating and hinders the purpose of learning. The best interfaces become invisible to us—when the interaction becomes automatic, user attention shifts from manipulating an interface to accomplishing a task, in this case, real-time SPICE simulation and learning circuit concepts. In order to evaluate these new real-time interactive SPICE simulator prototypes, usability studies, surveys and comparative experiments needed to be performed according to Scott Klemmer from Stanford’s HCI Class [15]. This was the reason why we designed our user study to encapsulate all of these evaluation elements. We were able to watch someone use our interactive tools, have them give us feedback and then redesign based-off the received feedback, this is why ReActive-SPICE v2.0 was created. In addition, we also “ate our own dog food”, which provided a lot of internal feedback that we were able to channel toward the interfaces before released to the users.

9.1 User Study Summary

Again, the user study was designed to assist in determining the overall effectiveness of the HCI elements, principally the user interface design of the new simulation tools. Many of the user study responses to each of the questions varied, however only the trends and majority alignments on feedback were summarized in Table 9.1.

Table 9.1 – User Study Tabulated Results Summary
Scale: 1 low/bad, 5 high/good

Time to initially build a circuit using TouchSPICE	<2.6 minutes
How easy was TouchSPICE to use?	3.7 / 5.0
How easy was TouchSPICE to understand?	3.6 / 5.0
Would you recommend either of the real-time SPICE simulators?	19 Yes, 1 Maybe
How intuitive was the TouchSPICE node UI?	4.1 / 5.0
How intuitive was the TouchSPICE host UI?	3.9 / 5.0
How intuitive was ReActive-SPICE v1.0's UI ?	4 / 5.0
Time to initially build a circuit using ReActive-SPICE v1.0	<2.2 minutes
How easy was ReActive-SPICE v1.0 to use?	3.2 / 5.0
How easy was ReActive-SPICE v1.0 to understand?	3.9 / 5.0

Overall the user feedback was positive and every feedback question received a 3.2 out of 5 and above, which isn't horrible for prototypes looking to receive feedback. When asked if there was "anything confusing or unclear as to what needed to be done to build and simulate the circuit with the touchscreen SPICE simulator," 13 out of 17 people rated the intuitiveness of TouchSPICE as a 4 or higher and thought the TouchSPICE simulator had a very short learning curve. TouchSPICE was rated by the users as easier to use than ReActive-SPICE v1.0 and ReActive-SPICE v1.0 was rated easier to understand than TouchSPICE within approximately .4 stars. After using both of the real-time interactive simulators, the students were overall excited and would recommend the new prototype

simulation tools to other students who may be having trouble understanding circuit analysis concepts and even to other students who may want to just play around with the simulators or use them to optimize lab or personal projects requiring SPICE analysis.

9.2 Learning Curves

As can be seen in Table 9.1, the overall average time to build a circuit using TouchSPICE was less than 2.6 minutes. However, after the users became accustomed to the interface and how it operated, circuit creation was quick, simple and occurred in less than 1 minute for 17 out of 20 users. Initially, 5 out of 20 users were able to build and simulate a circuit in less than a minute. This 1.6 minute disparity between the first time using TouchSPICE to build and simulate a circuit and all successive times can be attributed to initially getting accustomed to the TouchSPICE interfaces, on both the nodes and the host computer. At first, roughly 6 out of 20 users did not understand how the wire connection process worked or if they could and couldn't do certain actions with the nodes. Eventually, with the on-screen connection status indicators, the user was able to figure out how to construct a simple circuit and then simulate it. Table 9.2 shows a breakdown of the user's time to build a circuit using TouchSPICE:

Table 9.2 – Time to build a circuit using TouchSPICE Summary

Question	Student Response in minutes (number of students)						
	< 1	1	2	3	4	5	15
How long did it take to build and simulate a circuit using TouchSPICE?	5	7	1	2	1	3	1

As can be seen in Table 9.1, the overall average time to build a circuit using ReActive-SPICE v1.0 was less than 2.2 minutes. However, after the users became accustomed to the interface and how it operated, circuit creation was quick, simple and occurred in approximately less than 1 minute for 18 out of 20 users. Initially, 4 out of 20 users were able to build and simulate a circuit in less than a minute. This 1.2 minute disparity between the first time using ReActive-SPICE v1.0 to build and simulate a circuit and all successive times can be attributed to initially getting accustomed to the ReActive-SPICE interface. At first, roughly 13 out of 20 users did not understand how the net matching process worked. Eventually, the user was able to figure out how to construct a simple circuit and then simulate it. Table 9.3 shows a breakdown of the user's time to build a circuit using ReActive-SPICE v1.0:

Table 9.3 – Time to build a circuit using ReActive-SPICE v1.0 Summary

Question	Student Response in minutes (number of students)				
	< 1	1	2	4	10
How long did it take to build and simulate a circuit using ReActive-SPICE v1.0?	4	8	5	1	2

Since the students were using both real-time interactive SPICE simulators sequentially after the other, once using one of the simulators, the use of the next simulator would result in a better understanding of what needed to be done to build and simulate a circuit, meaning that the build times drastically decreased after using one of the simulators. This goes to show how not much of a manual needed to be written or read

to start simulating a circuit, unlike LTspice™ or PSpice™, which requires a little more understanding to achieve similar results.

9.3 Intuitive Interfaces and Real-time Features

Feedback from the user study indicated very positive interest and appreciation of the intuitive interfaces and real-time features of both TouchSPICE and ReActive-SPICE v1.0. For all students, it was their first time interacting with a real-time SPICE simulator and most of them had a lot of fun while participating in the user study. Approximately 14 out of 20 users felt that the real-time interactive simulators were much easier and more intuitive compared to building the circuit on a bread board or in a traditional SPICE simulation program. The user interface was the critical component in determining the acceptance of the new real-time interactive SPICE simulators. The prototype user interfaces served their purpose of providing a real-time interactive solution, however there was a lot of feedback on how these interfaces and interaction in general could be improved for future revisions.

Some users offered the feedback that pressing the stop/simulate button on the host machine after a circuit topology change was not clear and that layout-wise, the buttons (connect, start simulation, stop simulation, disconnect or reset nets, start simulation and stop simulation) should be moved further apart from each other to avoid accidental clicks, especially since when the “Reset Nets” button was accidentally clicked in ReActive-SPICE v1.0, all of the net matching would be cleared without the ability to undo.

Some users were confused with the V/I check boxes to measure voltage and current, but eventually figured how they functioned after clicking on them while simulating. A few responses offered that the check boxes should be “flashier or have a hint that pops up saying that something needs to be measured for a waveform to show up.” Also, the horizontal slider (scrollbar) was not obvious that it could change the component value in large increments. Another user suggested that instead of having to scroll through all of the scaling factors (magnitude prefixes), that a simple drop-down box be used. Another big trend in the user interface feedback for TouchSPICE was that many users wish they had known the meaning of the white dots on the wires beforehand. A good response was:

“The whole "dot" system with the wiring is something that needed to be specified before any connections are made. However, the red and green dots on the touchscreens were good for troubleshooting in case of "unhappy" connections. I would like that instead of error messages yelling at me.”

Some additional suggestions for user interface improvements included a button to reset all of the components and/or connections and the ability to adjust the scales on the plots as well as to add additional plots with different scales. The main reason why these features were avoided in the prototypes was to keep the interface as simple as possible so that the user didn’t need to worry about scaling and could just focus on circuit analysis concepts. However this would be a nice advanced feature for future development.

ReActive-SPICE v1.0 tended to be more intuitive and efficient for users with some existing SPICE knowledge in terms of text-based netlists, others had some

difficulty understanding the concepts of net matching at first. Many users commented that since they were on a fully functional computer, they would prefer to use the keyboard as an input method for values. Also, since the prototype design of ReActive-SPICE v1.0 simply contained the same “virtual” node design as TouchSPICE, some features, such as the connection status indicators were left on there, so there were a cluster of red dots on the “virtual” nodes with absolutely no functionality, which caused confusion to the users. The users suggested that the interface provide a way for net numbering to go in the opposite direction with a right mouse click, instead of having to click through nets 1-9 to get to the desired net number again.

One of the biggest interface complaints of ReActive-SPICE v1.0 was that the components (“virtual” nodes) were locked down on the screen and immovable, making it visually unintuitive and confusing as to which components were connected and how they were connected. This was an anticipated reaction and the main reason why a second ReActive-SPICE prototype would be created. ReActive-SPICE v2.0 would enable the relocation and rotation of components and traditional wiring of circuits using lines, much like how LTspiceTM or PSpiceTM would create circuit schematics.

9.4 Importance of the Physical-Virtual Interfaces

The physical wiring of the TouchSPICE nodes allowed for a pseudo-schematic view, and feel, thus providing the user with a physical representation of their circuit layout. ReActive-SPICE v1.0 on the other hand was of course much less interactive than TouchSPICE and much harder to visualize the layout of circuits. A user commented:

“Overall I found both of these simulation tools to be helpful and very useful to younger EE/CPE students who have not learned how to use LTspiceTM or PSpiceTM. The physical representation of the nodes and components with wires helped tie the simulation to the real world.”

TouchSPICE’s physical-virtual interfaces essentially gives the user the feel of breadboarding, with the power of modern real-time simulation—allowing the user to waste nothing, simplify anything, learn something and play everything.

Many students indicated that because the TouchSPICE nodes were so fun to play with, TouchSPICE morphed into a great learn-by-doing experience. One user commented: “The touchscreens were fun to use and moving things around to build circuits was really interesting and sparked interested to build more complex circuits and test new things.” In general, both interfaces were widely accepted as another user commented:

“They are the most fun I’ve had with a simulator. The visible aspect made everything much easier to implement, and the novel nature of the set up was very cool. The technology itself was impressive, and the simulations were surprisingly easy to start once the first circuit was built. With about five minutes of explanation, anyone with basic computer knowledge could begin using this interface to build circuits much quicker and easier than with a breadboard.”

9.5 Shortcomings of TouchSPICE and ReActive-SPICE v1.0

Despite much of the positive feedback on the real-time interactive SPICE simulators, there was an alarmingly high amount of negative feedback that revolved around slow simulation response times and bugs. The slow simulation response times were attributed to the type of wireless communication protocol chosen; since the host computer was constantly polling all of the TouchSPICE nodes, even if they didn't have an update, the GPIO thread on the touchscreens would consume most of the processor time, making node redraw rates and updates disturbingly slower as time progressed. This constant polling from the host also caused the host computer's processor to heat up and often times throttle because of the large amount of utilization, which would also cause the simulation to slow down to much less than real-time. Since this was a result of the communication protocol, ReActive-SPICE did not face this problem; instead, it faced the problem of having a few bugs that originated from porting TouchSPICE over to ReActive-SPICE. Some of the net matching conventions did not align with the wiring connection conventions, and without a major source code rework, this would continue to be a problem. However, most of the functionality was there, and as long as the user knew what configurations not to try, such as parallel resistor configurations, the hanging could be avoided. Fixes to these bugs would be addressed in ReActive-SPICE v2.0, which is discussed in the next section.

In addition to the slow simulation updates and bug hangs, there was also a problem with the functionality and versatility of the touchscreens. Since the physical touchscreens used were resistive touch as opposed to capacitive touch, a lot was taken

away from the interface because even if it was intuitive, sometimes the users didn't know if they were clicking in the right spot or if they needed to press harder or move slower. This touchscreen type also made it hard to manipulate the displays using your fingertips, so a stylus was used to improve accuracy and to hopefully alleviate frustration with the nodes. The ability to use the mouse in ReActive-SPICE drastically improved pointer precision and speed.

9.6 ReActive-SPICE v2.0

Based off the user feedback, a new revision of the ReActive-SPICE integrated simulator was created to improve usability, intuitiveness, and efficiency. An entirely separate program was created from the ground up to allow the creation of circuits in a simple schematic editor that is similar to LTspiceTM and PSpiceTM. The below figures are the newly designed ReActive-SPICE v2.0 integrated circuit simulator. Some of the new features included drag and drop components that snap to a grid, the ability to move and rotate components after placed, connecting circuit components with wires by drawing lines and improved interface controls to either simplify visualizing the circuit or to reveal advanced circuit control features for customizability.

Figure 9.1 – Integrated ReActive-SPICE simulator v2.0 – New Schematic Design

Figure 9.2 – Integrated ReActive-SPICE simulator v2.0 – Components with controls (are able to toggle controls visible/invisible by double clicking), standard drag-and-drop components, can move, rotate, delete and create wires between nodes.

Figure 9.3 – Integrated ReActive-SPICE simulator v2.0 – Simulating, No controls visible

Figure 9.4 – Integrated ReActive-SPICE simulator v2.0 – Probing Voltages, No controls visible

Figure 9.5 – Integrated ReActive-SPICE simulator v2.0 – Inverting Op-Amp Design

Figure 9.6 – Integrated ReActive-SPICE simulator v2.0 – Voltage and Current Plots

Chapter 10 Conclusion

In conclusion, TouchSPICE bridges the gap between simulation and the real world—harnessing the power of real-time virtual simulation while still keeping the physical touch of building a circuit on a breadboard. ReActive-SPICE creates a much more cost-effective solution that is driven by the same motivations of TouchSPICE: to assist with learning circuit analysis and to enforce how the characteristics of each component affect the overall simulation and functionality of the circuit. By designing and implementing each of these simulators from an HCI standpoint, the simulators were tailored and optimized to be as simple and intuitive as possible to enable transparent circuit building and manipulation; this allowed the users to focus on the effects that each discrete component had on the overall circuit.

Conclusions based on the user study surveys support the hypothesis that both TouchSPICE and ReActive-SPICE simulation tools were more intuitive and overall simpler than traditional SPICE simulation tools. This consensus was held by 14 out of the 20 users. Feedback collected also showed the TouchSPICE solution to have a more intuitive user interface (4.1/5.0) while the ReActive-SPICE v1.0 solution proved to be more efficient (2.2 minute average initial build & simulation time) when it came to constructing the circuit after the concept of net matching was established. With the feedback from the user study, a new revision of the ReActive-SPICE simulator was able to be created to maximize and optimize the overall HCI experience, making the user interface more intuitive and easier to understand. ReActive-SPICE v2.0 had an average learning curve of less than 30 seconds for a small trial run of 4 users and enabled circuit

creation and simulation with minimal SPICE and circuit analysis knowledge. The interface was designed loosely to mimic the schematic capture view employed by both LTspice™ and PSpice™ while adding the critical key feature of being able to run simulations in real-time.

Both TouchSPICE and ReActive-SPICE solutions foster a learn-by-doing experience that simultaneously educates and entertains. With additional software work, interface development, refinement, evaluation and hardware optimization, these simulators, primarily ReActive-SPICE 2.0, could become the new cost-effective real-time SPICE alternative for those learning circuit theory.

FUTURE WORK

One of the main factors in the scalability of TouchSPICE is the cost. In order to reduce the cost in future revisions, a custom PCB could be created to interface with a touchscreen. This would make the most expensive item, the node itself, cheaper. With an inexpensive solution and an expandable fan-in and fan-out, these nodes could simulate more advanced circuits fairly easily.

The easiest way to expand this project in the future is to create more components to choose from, such as PNP, PMOS, or even full circuit stages such as current mirrors and Darlington pair amplifiers. As long as a netlist can be generated for the component/stage, it can be integrated into the node. Capacitor polarity and varying diode voltage drops, VGS, VBE, opamp supply and gain could all be added parameters that could be adjusted during simulation time. Additional features would be to add the ability to measure voltage differences between nodes/nets and to add the ability to rotate the screen for larger circuits.

Another possibility is to add more physical nodes. This would allow for more complex circuits. More components create the new problem of using up all available wire connections. A solution would be to have a dedicated wire node that would emulate a bus that can have multiple connections. Using a distributed processing method could represent a much larger advancement in the method used to simulate the circuits. By creating these in a Linux based environment, new programs can be added to the nodes to allow for interfacing with other simulation tools as well. Other features such as power calculations could be added into the TouchSPICE and ReActive-SPICE simulators.

Games could also be developed that ask the user to “make this signal out of this signal” or to “make an integrator (and the nodes will flash and change to the correct components that need to be connected together)” to further encourage playing with the devices. Something to be investigated would be to see if a simulation tool such as TouchSPICE could be implemented on Sifteo’s Siftables, which have only a single physical button, display, and an accelerometer. If that is possible, the benefits or pitfalls compared to the touchscreen capabilities of TouchSPICE should be explored from an HCI standpoint.

BIBLIOGRAPHY

- [1] Y. Yang and S. Zhou , "A circuit simulation experimental system based on re-development SPICE," in *2011 4th International Congress on Image and Signal Processing (CISP)*, vol. 5, pp. 2531-2535, 15-17 Oct. 2011.
- [2] T. Gedra, "Virtual Instrumentation in an Undergraduate Electrical Machines Lab," *1997 Midwest Section American Society for Engineering Education (ASEE) Conference*, Columbia, MO. 1997.
- [3] (2012, May 2). *DC/AC Virtual Lab – Build Circuits Online* [Online]. Available: <http://www.dcacalab.com/en/lab/>
- [4] M. Koltai. (2012, Mar. 24). *TINA - The Complete Electronics Lab* [Online]. Available: <http://www.tina.com>
- [5] J. Fuehne, "Integrating Virtual Lab Simulation Software into an MET Program. Purdue University," in *Proceedings of the 2008 International Association of Journals & Conferences - International Journal of Medical Education International Conference (IAJC-IJME)*, Nashville, TN. 2008.
- [6] F. A. Krueger and P. Falstad. (2012, Jun. 3). *iCircuit – The Realtime Circuit Simulator and Editor* [Online]. Available: <http://icircuitapp.com/>
- [7] I. Vytyaz. (2012, Jun. 3). *EveryCircuit Android Application* [Online]. Available: <https://play.google.com/store/apps/details?id=com.everycircuit.free>
- [8] P. Falstad. (2012, Jun. 3). *Falstad Java Circuit Simulator* [Online]. Available: <http://www.falstad.com/circuit>
- [9] D. Merrill et al., "Siftables: Towards Sensor Network User Interfaces," in *Proceedings of the 1st international conference on Tangible and Embedded Interaction (TEI)*, ACM pp. 75-78, New York, NY. 2007.
- [10] M. Shaivitz. (2012, Jun. 6). *Big Tech in a Little Package – Sifteo’s Hands-On Game Cubes* [Online]. Available: <http://techcocktail.com/sifteo-2011-03>
- [11] G. W. Fitzmaurice. (2012, Jun. 6). *Graspable User Interfaces* [Online]. Graduate Department of Computer Science, University of Toronto. 1996. Available: <http://www.dgp.toronto.edu/~gf/papers/PhD%20-%20Graspable%20UIs/Thesis.gf.html>
- [12] G. W. Fitzmaurice et al., "Bricks: Laying the Foundations for Graspable User Interfaces," in *Association for Computing Machinery Proceedings of Computer-Human Interaction (ACMCHI)*, ACM pp. 442-449, New York, NY. 1995.

- [13] J. Rekimoto et al., “DataTiles: A Modular Platform for Mixed Physical and Graphical Interactions,” in *Special Interest Group on Computer Human Interaction (SIGCHI)*, Seattle, WA. 2001.
- [14] J. Breitenbach. (2011, Dec. 1). *EE 241 Materials, California Polytechnic University, San Luis Obispo* [Online]. Available: <http://courseware.ee.calpoly.edu/~jbreiten/EE241>
- [15] S. Klemmer. (2012, Jun. 28). *A Course on Human-Computer Interaction* [Online]. Stanford University, June 2012. Offered by Coursera. Available: <https://class.coursera.org/hci/class/index>
- [16] D. Awtrey, (2011, Dec. 3). *Introduction to I-Wire Network Issues* [Online]. Springbok Digitronics. Available: <http://www.1wire.org/Files/Articles/1-Wire-Design%20Guide%20v1.0.pdf>
- [17] L. Wong. (2011, Dec. 10). *I-Wire ® (Protocol)* [Online]. Application Note, Dallas Semiconductor/Maxim. Available: http://coecsl.ece.illinois.edu/ge423/sensorprojects/1-wire_full.doc
- [18] G. Sandstrom. (2012, Aug. 5). “Nokia App Unit to Focus on Windows-Based Phones,” *Wall Street Journal* [Online]. Available: http://online.wsj.com/article/SB10000872396390443537404577578763539348328.html?mod=googlenews_wsj
- [19] D. Abbott, *The Embedded Linux Learning Kit Version 2 User’s Guide (ELLK II)*. Silver City, NM: Intellimetrix, 2011.

Appendix A: TouchSPICE Bill of Materials

Component	Purchase Location	Quantity	Cost per Unit
Mini2440	Andahammer.com	10	\$120
Mini2440 Wire Kit	Andahammer.com	10	\$12
WT12-A	Inmojo.com	10	\$40
Logic Level Converter	Sparkfun.com	10	\$2
F/F Jumpers	IEEE-SB Lounge	Pack of 40	\$8

Appendix B: Mini2440 Bring-up / Kernel building / Environment Setup

Mini2440 Bring-Up Online References used (** most important):

Step by Step - Booting and running the Android OS from the SD card

<http://mini2440.spaceopera.org/site/developmentenviroment/stepbystep/index.html>

Bill's Mini2440 Forum

<http://billforums.station51.net/viewforum.php?f=4>

Deploying u-boot, Linux kernel and GPE onto Mini2440, powered by SAMSUNG S3C2440

<http://narnia.cs.ttu.edu/drupal/node/131>

Manual for compiling Android for mini2440

<http://my.opera.com/vinodmunikoti/blog/manual-for-compiling-android-for-mini2440>

***How to cross compile a working kernel for Mini2440*

http://wiki.iface.ch/index.php/Main_Page

USB-Upload-Tool (German)

<http://programmers-projects.de/node/16>

Install Linux onto the Mini2440 Board

<http://mini2440vietnam.blogspot.com/2011/01/install-linux-into-mini2440-board.html>

Android W35 zImage

<http://www.mediafire.com/?dh7iw3dw0nvqpac>

Main FriendlyArm Downloads Page

<http://www.friendlyarm.net/downloads>

5.9.6. Environment Variables Commands (U-Boot)

<http://www.denx.de/wiki/DULG/UBootCmdGroupEnvironment>

MiniBringup Summary

<http://code.google.com/p/mini2440/wiki/MiniBringup>

Creating a Hello world app (Qtopia 2.2)/Qt 4.6 installation

<http://code.google.com/p/friendlyarm/wiki/Qt>

Installation of Qtopia for mini2440 Linux board

<http://intresting2440.blogspot.com/>

Qt 4.6.2 installation procedure for FriendlyARM mini 2440 Linux

<http://www.sereno-labs.com/qt-4-6-2-installation-procedure-for-friendlyarm-mini-2440-board>

How to install the cross-copiler for mini2440 on host PC (Video 1)

How to compile QtEmbedded 4.6.3 for mini2440 (Video 2)

Install QtCreator and set it up for QtEmbedded. (Video 3)

<http://farm2440.blogspot.com/>

Qtopia everywhere 4.6.1

<http://www.friendlyarm.net/forum/topic/439>

***Qt 4.5 on mini2440 (framebuffer/tslib config)*

<http://cor-net.org/2009/03/qt-45-on-mini2440/>

***tslib download and cross compile for arm*

<http://linux-dvbh.blogspot.com/2006/12/tslib-download-and-cross-compile-for.html>

***Upgrade Qt4.6.2 in mini2440*

<http://mini2440vietnam.blogspot.com/2011/04/upgrade-qt462-in-mini2440.html>

Cross compiling Qt-embedded 4.5.3

<http://www.electronics.diycinema.co.uk/embedded/mini2440/cross-compiling-qt-embedded-4-5-3/>

Error Message is: selected device is not a touchscreen

<http://www.friendlyarm.net/forum/topic/2786>

*How to write Applications in C++ for Friendly ARM mini 2440 using Qt 4.6.2 library:
part 1*

<http://www.tuxfeed.it/2010/05/14/how-to-write-applications-in-c-for-friendly-arm-mini-2440-using-qt-462-library-part-1/>

Mini2440 Bring-Up Summarized Procedure:

Taken from: <http://code.google.com/p/mini2440/wiki/MiniBringup>

install the libusb-dev to enable the use of USBPUSH

Formatted Linux

Updated everything
Installed wicd
Started using/customizing root user
Downloaded arm-linux-gcc-4.4.3.tgz from FA.com
Copied files in opt to /usr/local/arm/4.4.3/*
apt-get install git
Follow tutorial on tslib compilation:
<http://mini2440vietnam.blogspot.com/2011/04/upgrade-gt462-in-mini2440.html>

Couldn't get Qte to compile...big surprise...

Installed g++ from the Software Manager
Made ./Qt_SDK_Lin32_offline_v1_1_2_en.run executable and ran it

TFTP BOOT

<http://esdw.wordpress.com/2010/03/22/setup-a-tftp-server/>

```
sudo apt-get install xinetd tftpd tftp
sudo gedit /etc/xinetd.conf, add the following,
service tftp
{
protocol          = udp
port              = 69
socket_type       = dgram
wait              = yes
user              = nobody
server            = /usr/sbin/in.tftpd
server_args       = /tftpboot
disable           = no
}
```

```
sudo mkdir /tftpboot
sudo chmod -R 777 /tftpboot
sudo chown -R nobody /tftpboot
sudo /etc/init.d/xinetd restart
OR sudo service xinetd restart
```

Test the TFTP server:

```
cd
touch /tftpboot/testfile
tftp 127.0.0.1
tftp > get testfile
```

INSTALLING U-BOOT / RESET TO FACTORY DEFAULT ####**Das U-Boot:**

<http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Introduction-to-Das-UBoot-the-universal-open-source-bootloader/>

Steps taken from ELLK Guide (pg 82-84) and the below online tutorial:

http://wiki.iface.ch/index.php/Installing_u-boot_on_Mini2440

Summarized Steps (FA = FriendlyArm):

```
FA: Boot FA from NOR
FA: Download and Run (d)
HOST: arm-dev@arm-dev ~/Desktop/arm-dev $ sudo ./usbpush
/tftpboot/u-boot.bin (v1.3.2 128M version) // NOTE IF YOU WANT
MMCINIT TO WORK YOU NEED TO USE THE u-boot.bin FILE THAT CAME IN
THE FACTORY IMAGES FOLDER ON ELLKCD!!!
```

```
Stop autoboot
```

```
FA: MINI2440# nand scrub
FA: MINI2440 # nand createbbt
FA: MINI2440 # setenv ipaddr 192.168.1.50
FA: MINI2440 # setenv serverip 192.168.1.2
FA: MINI2440 # setenv netmask 255.255.255.0
```

```
// Create mini_boot script (can be skipped if the .scr file in
/tftpboot hasn't changed):
```

```
arm-dev u-boot # ./mk_uboot.sh mini_boot
arm-dev u-boot # ls -lart /tftpboot/
```

```
FA: MINI2440 # tftp 32000000 mini_boot.scr
FA: MINI2440 # autoscr 32000000
```

Contents of mini_boot.scr:**// For TFTP**

```
//setenv bootargs console=ttySAC0,115200 noinitrd init=/sbin/init
mini2440=0tb root=/dev/nfs rw nfsroot=192.168.1.2:/home/target_fs
ip=static
```

// For SD

```
setenv sd_image_name uImage
setenv bootcmd mmcinit\; fatload mmc 1 0x32000000
\${sd_image_name}\; bootm 0x32000000
```

// For NAND

```
setenv bootcmd nand read 32000000 kernel 267000\; bootm 32000000
setenv set_bootargs_nand setenv bootargs \${bootargs_base}
\${root_nand} ip=\${ipaddr}
```

```
setenv hostname MINI2440
setenv root_nand root=/dev/mtdblock3 rootfstype=yaffs
setenv root_nfs /root/Desktop/arm-dev/target_fs/
dynenv set 40000
setenv ifconfig
ip=\${ipaddr}:\${serverip}::\${netmask}:mini2440:eth0
run set_bootargs_mmc
saveenv
tftp 32000000 u-boot-128M.bin
nand write.e 32000000 u-boot \${filesize}
tftp 32000000 uImage
nand write.e 32000000 kernel \${filesize}
tftp 30008000 target_fs.yaffs
nand write.yaffs 30008000 root \${filesize}
```

To boot from nfs in U-Boot type:

```
run set_bootargs_nfs
run bootcmd
```

```
setenv bootargs console=ttySAC0,115200 noinitrd init=/sbin/init
mini2440=0tb root=/dev/nfs rw nfsroot=192.168.1.2:root=/dev/nfs
rw nfsroot=192.168.1.2:/root/Desktop/arm-dev/target_fs
ip=192.168.1.50:192.168.1.2::255.255.255.0:mini2440:eth0
```

To boot from NAND in U-Boot type:

```
run set_bootargs_nand
run bootcmd
```

To boot from TFTP in U-Boot type:

```
tftpboot 32000000 uImage; bootm 32000000
```

To boot from SDHC in U-Boot type:

```
(sd_image_name env var must be set first! Default that I set was
uImage)
mmcinit; fatload mmc 1 0x32000000 \${sd_image_name}; bootm
0x32000000
```


To save it:

```
setenv bootcmd mmcinit\; fatload mmc 1 0x32000000  
\${sd_image_name}\; bootm 0x32000000  
run set_bootargs_nfs  
OR run set_bootargs_mmc
```

Found in: http://wiki.openmoko.org/wiki/Booting_from_SD

Setup the ENV

```
sudo apt-get install nfs-kernel-server  
apt-get install rpm
```

```
Add this: /root/Desktop/arm-dev/target_fs/  
*(rw, sync, no_root_squash)  
to /etc/exports file
```

```
sudo service nfs-kernel-server restart
```

CROSSCOMPILE AND INSTALL LINUX KERNEL

http://wiki.iface.ch/index.php/How_to_cross_compile_a_working_kernel_for_Mini2440

```
sudo apt-get install libncurses5-dev  
sudo apt-get install uboot-mkimage
```

```
setenv bootcmd nfs 0x31000000 192.168.1.2:/home/arm-  
dev/fadev/target/uImage \; bootm
```

Make sure you set EXT2/3 support in .config file before building the kernel!!!

```
CONFIG_EXT3_FS=y
```

```
setenv bootargs console=ttySAC0,115200  
ip=192.168.1.50:192.168.1.2::255.255.255.0 root=/dev/nfs  
nfsroot=192.168.1.2:/home/arm-dev/fadev/target init=helloworld
```

Fresh out of the box how-to's

<http://www.friendlyarm.net/forum/topic/885>

To get the kernel info:

```
#uname -a
```

To get the cpu info:

```
#cat /proc/cpuinfo
```

To get the memory info:

```
#cat /proc/meminfo
```

To get info on usb devices:

```
#cat /proc/bus/usb/devices
```

To get the messages:

```
#dmesg | less
```

COMPILING QTOPIA

<http://farm2440.blogspot.com/>

```
su
```

```
apt-get install build-essential
```

Possibly need this if no gcc compiler installed:

```
apt-get install gcc-4.6-arm-linux-gnueabi
```

Also need tslib...:

<https://github.com/kergoth/tslib/downloads>

cd to the tslib extracted directory and as root do this:

```
./autogen.sh
./configure
make install
cp -v -R * /home/arm-dev/target_fs/
```

ON FA TO CONFIG TS:

<http://cor-net.org/2009/03/qt-45-on-mini2440/>

Create the ts node:

```
mknod /dev/input/ts c 13 64
```

Test the ts node:

```
cat /dev/input/ts
```

Calibrate the ts node:

```
ts_calibrate /etc/ts.calib
```

Edit:

```
vi /home/arm-dev/Desktop/arm-dev/qtopia/arm-qte-4.6.3/qt-
everywhere-opensource-src-4.6.3/mkspecs/qws/linux-arm-
g++/qmake.conf
```

Add these lines:

```
QMAKE_INCDIR
QMAKE_LIBDIR
```

Configure command: (didn't work, never was able to CC)

```
./configure -prefix /qtbuild/ -release -shared -fast -pch -no-qt3support -qt-sql-sqlite -no-libtiff -no-libmng -qt-libjpeg -qt-zlib -qt-libpng -qt-freetype -no-openssl -nomake examples -nomake demos -nomake tools -optimized-qmake -no-phonon -no-nis -no-opengl -no-cups -no-xcursor -no-xfixes -no-xrandr -no-xrender -no-xkb -no-sm -no-xinerama -no-xshape -no-separate-debug-info -xplatform qws/linux-arm-g++ -embedded arm -depths 16 -no-qvfb -qt-gfx-linuxfb -no-gfx-qvfb -no-kbd-qvfb -no-mouse-qvfb-qt-kbd-usb -confirm-license -qt-mouse-tslib
```

DEFAULT U-BOOT ENV VARS

MINI2440 # printenv

```
bootargs=root=/dev/mtdblock3 rootfstype=jffs2
console=ttySAC0,115200
bootcmd=
bootdelay=3
baudrate=115200
ethaddr=08:08:11:18:12:27
ipaddr=10.0.0.111
serverip=10.0.0.4
netmask=255.255.255.0
usbttty=cdc_acm
mtdparts=mtdparts=mini2440-nand:256k@0(u-
boot),128k(env),5m(kernel),-(root)
mini2440=mini2440=0tb
bootargs_base=console=ttySAC0,115200 noinitrd
bootargs_init=init=/sbin/init
root_nand=root=/dev/mtdblock3 rootfstype=jffs2
root_mmc=root=/dev/mmcblk0p2 rootdelay=2
root_nfs=/mnt/nfs
set_root_nfs=setenv root_nfs root=/dev/nfs rw
nfsroot=${serverip}:${root_nfs}
ifconfig_static=run setenv ifconfig
ip=${ipaddr}:${serverip}::${netmask}:mini2440:eth0
ifconfig_dhcp=run setenv ifconfig ip=dhcp
ifconfig=ip=dhcp
set_bootargs_mmc=setenv bootargs ${bootargs_base}
${bootargs_init} ${mini2440} ${root_mmc}
set_bootargs_nand=setenv bootargs ${bootargs_base}
${bootargs_init} ${mini2440} ${root_nand}
set_bootargs_nfs=run set_root_nfs; setenv bootargs
${bootargs_base} ${bootargs_init} ${mini2440} ${root_nfs}
${ifconfig}
mtdids=nand0=mini2440-nand
partition=nand0,0
mtddevnum=0
mtddevname=u-boot
Environment size: 1089/131068 bytes
```

******* FINAL QT install *******

Modified /etc/ts.conf in RFS

```
rm /dev/input/mice
mknod /dev/input/ts c 13 64
mknod /dev/input/mice c 13 63
```

ENV Vars:

```
export TSLIB_TSEVENTTYPE=INPUT
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/ts
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_CONFFILE=/etc/ts.conf
export TSLIB_PLUGINDIR=/lib/ts
```

```
export TSLIB_TSEVENTTYPE=INPUT
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/ts
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_CONFFILE=/usr/local/etc/ts.conf
export TSLIB_PLUGINDIR=/usr/local/tslib/lib/ts
export QTDIR=/usr/local/Qt
export QWS_MOUSE_PROTO=tslib:/dev/input/event0
```

```
cd /usr/local/Qt/demos/embedded/fluidlauncher/
./fluidlauncher -qws
```

Demo should work now!

Formatting the SD cards

Use gParted to create 1GB fat16 and 3GB ext2 partitions
killall udisks

```
rfcomm connect 0 00:07:80:45:f0:7d 1
```

```
setWindowFlags(Qt::FramelessWindowHint);
```

OUR NODE BRINGUP

Summarized Steps:

```
FA: Boot FA from NOR
FA: Download and Run (d)
HOST: cd Desktop/arm-dev
HOST: arm-dev # ./usbpush /tftpboot/u-boot.bin (v1.3.2 128M
version) // NOTE IF YOU WANT MMCINIT TO WORK YOU NEED TO USE THE
u-boot.bin FILE THAT CAME IN THE FACTORY IMAGES FOLDER ON
ELLKCD!!!
Stop autoboot
FA: MINI2440# nand scrub
FA: MINI2440 # nand createbbt
FA: MINI2440 # setenv ipaddr 192.168.1.50
FA: MINI2440 # setenv serverip 192.168.1.2
FA: MINI2440 # setenv netmask 255.255.255.0
FA: MINI2440 # setenv sd_image_name uImage
FA: MINI2440 # setenv bootcmd mmcinit\; fatload mmc 1 0x32000000
\${sd_image_name}\; bootm 0x32000000

FA: MINI2440 # setenv hostname MINI2440
FA: MINI2440 # dynenv set 40000
FA: MINI2440 # setenv ifconfig
ip=${ipaddr}:${serverip}::${netmask}:mini2440:eth0

FA: MINI2440 # run set_bootargs_mmc
FA: MINI2440 # saveenv
FA: MINI2440 # tftp 32000000 u-boot.bin
FA: MINI2440 # nand write.e 32000000 u-boot ${filesize}
FA: Turn off, switch to NAND...done

HOST: Open Gparted and delete all partitions on the 4GB flash
drive, create a new 1000 MiB fat16 partition and the rest ext2

cp -avr ~/Desktop/arm-dev/sd_release/3GB\ ext2/* /media/ddb9f982-
5279-4ab4-84fb-bc205de9bd7e/

cp -avr ~/Desktop/arm-dev/sd_release/1GB\ fat16/* /media/85AE-
74AC/
```

Appendix C: User Surveys

Master's Thesis Pre-Survey

If you don't know an answer, simply write, "I don't know". Please do not make a blind guess.

* Required

User Number *We gave you this number

Grade Level *

- ☐ Freshman
- ☐ Sophomore
- ☐ Junior
- ☐ Senior
- ☐ Graduate

SPICE Simulators Used *

- ☐ LTspice IV
- ☐ HSPICE
- ☐ PSpice
- ☐ Multisim
- ☐ Micro-Cap
- ☐ None
- ☐ Other:

Preferred SPICE Simulator and Why

Lab Equipment Used *

- ☐ Breadboard
- ☐ Decade Resistor/Capacitor
- ☐ Oscilloscope
- ☐ Multimeter
- ☐ Function Generator
- ☐ DC Power Supply
- ☐ Sweep and Go

Other: Any other lab equipment used?

As the resistance is increased in a low-pass filter, what happens to the cutoff frequency? *

How does increasing the value of a feedback resistor on an opamp affect its gain? *

You have a circuit with a DC source and three resistors set up with the source and one resistor in series, and the other two resistors in parallel. How would decreasing the value of one of the resistors in parallel affect current in the branches? *

With a 5V source, what value resistor is required to get 20mA of current through a diode in series? *

Master's Thesis Post-Survey

* Required

User Number *Same as before

As the resistance is increased in a low-pass filter, what happens to the cutoff frequency? *

How does increasing the value of a feedback resistor on an opamp affect its gain? *

You have a circuit with a DC source and three resistors set up with the source and one resistor in series, and the other two resistors in parallel. How would decreasing the value of one of the resistors in parallel affect current in the branches? *

With a 5V source, what value resistor is required to get 20mA of current through a diode in series? *

How well did the touchscreen nodes make it easier to understand the results? *1 is the worst, 5 is the best

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

Did the touchscreen nodes make it easier to understand the results? Please Explain. *

How well did the PURE software real-time SPICE simulator make it easier to understand the results? *1 is the worst, 5 is the best

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

Did the PURE software real-time SPICE simulator make it easier to understand the results?
Please Explain. *

How intuitive was the user interface on the touchscreens? *1 is the worst, 5 is the best

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

How intuitive was the user interface on the touchscreens? Explain *Things you liked and ways to improve them.

How intuitive was the user interface on the touchscreen SPICE simulator host computer? *1 is the worst, 5 is the best

- 1
- 2
- 3
- 4
- 5

How intuitive was the user interface on the touchscreen SPICE simulator host computer?
Explain *Things you liked and ways to improve them.

How intuitive was the user interface on the PURE software real-time SPICE simulator? *1 is the worst, 5 is the best

- 1
- 2
- 3
- 4
- 5

How intuitive was the user interface on the PURE software real-time SPICE simulator?
Explain *Things you liked and ways to improve them.

Was there anything confusing or unclear as to what needed to be done to build and simulate the circuit with the touchscreen SPICE simulator? *

Was there anything confusing or unclear as to what needed to be done to build and simulate the circuit with the PURE software real-time SPICE simulator? *

On average, how long did it take you to figure out how to choose a component, set the value, connect the wires and simulate the circuit on the touchscreen SPICE simulator? *In minutes

On average, how long did it take you to figure out how to choose a component, set the value, connect the wires and simulate the circuit on the PURE software real-time SPICE simulator? *In minutes

How easy was the new touchscreen SPICE simulator to use? *1 being easy and 5 being hard.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

Why did you select that answer for the above question? *(touchscreen SPICE simulator)

How easy was the new PURE software real-time SPICE simulator to use? *1 being easy and 5 being hard.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

Why did you select that answer for the above question? *(PURE software real-time SPICE simulator)

Would you recommend either of these real-time SPICE simulator tools to your friends as a way to learn circuit concepts? Why or why not? *

Appendix D: UI Screenshots

Figure 11.1 – TouchSPICE/ReActive-SPICE v1.0 User Interface – Select a Component

Figure 11.2 – TouchSPICE/ReActive-SPICE v1.0 User Interface – Resistor

Figure 11.3 – TouchSPICE/ReActive-SPICE v1.0 User Interface – Capacitor

Figure 11.4 – TouchSPICE/ReActive-SPICE v1.0 User Interface – Inductor

Figure 11.5 – TouchSPICE/ReActive-SPICE v1.0 User Interface – Op-amp

Figure 11.6 – TouchSPICE/ReActive-SPICE v1.0 User Interface – NMOS

Figure 11.7 – TouchSPICE/ReActive-SPICE v1.0 User Interface – BJT

Figure 11.8 – TouchSPICE/ReActive-SPICE v1.0 User Interface – Current Source

Figure 11.9 – TouchSPICE/ReActive-SPICE v1.0 User Interface – DC Voltage Source

Figure 11.10 – TouchSPICE/ReActive-SPICE v1.0 User Interface – AC Voltage Source

Figure 11.11 – Qt Creator Screenshot – GUI Editor

Figure 11.12 – Qt Creator Screenshot – C++ Editor

Appendix E: Target Source Code

apps/main/mainwindow.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the TouchSPICE Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**
** * Redistributions of source code must retain the above copyright
**   notice, this list of conditions and the following disclaimer.
** * Redistributions in binary form must reproduce the above copyright
**   notice, this list of conditions and the following disclaimer in
**   the documentation and/or other materials provided with the
**   distribution.
** * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**   the names of its contributors may be used to endorse or promote
**   products derived from this software without specific prior written
**   permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QtGui/QMainWindow>
#include <QCustomEvent>
#include <stdio.h>
namespace Ui {
    class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT
signals:
    void sendComponentName(const char * component_name);
    void sendComponentValue(double component_value, char scale_factor);
    void sendfComponentValue(double fcomponent_value, char fscale_factor);
    void sendMeasureStatus(int index, int value);
public:

```

```

enum ScreenOrientation {
    ScreenOrientationLockPortrait,
    ScreenOrientationLockLandscape,
    ScreenOrientationAuto
};
explicit MainWindow(QWidget *parent = 0);
virtual ~MainWindow();
// Note that this will only have an effect on Symbian and Fremantle.
void setOrientation(ScreenOrientation orientation);
void showExpanded();
private slots:
    void hideControls();
    void showControls(char);
    void on_value_slider_valueChanged(int value);
    void on_doubleSpinBox_valueChanged(double value);
    void on_fvalue_slider_valueChanged(int value);
    void on_fdoubleSpinBox_valueChanged(double value);
    void on_sac_resistor_clicked();
    void on_sac_capacitor_clicked();
    void on_sac_inductor_clicked();
    void on_sac_diode_clicked();
    void on_sac_opamp_clicked();
    void on_sac_bjt_clicked();
    void on_sac_nmos_clicked();
    void on_sac_vsource_clicked();
    void on_sac_isource_clicked();
    void on_res_back_button_clicked();
    void on_cap_back_button_clicked();
    void on_inductor_back_button_clicked();
    void on_diode_back_button_clicked();
    void on_opamp_back_button_clicked();
    void on_bjt_back_button_clicked();
    void on_nmos_back_button_clicked();
    void on_isource_back_button_clicked();
    void on_vsource_back_button_clicked();
    void on_v0_stateChanged(int arg1);
    void on_v1_stateChanged(int arg1);
    void on_v2_stateChanged(int arg1);
    void on_v3_stateChanged(int arg1);
    void on_i0_stateChanged(int arg1);
    void on_i1_stateChanged(int arg1);
    void on_i2_stateChanged(int arg1);
    void on_i3_stateChanged(int arg1);
    void on_debug_button_clicked();
    void on_scale_button_clicked();
    void on_fscale_button_clicked();
    void connected_io_change(int io, bool status);
    void debug_append(const char * message);
    void on_dc_sine_button_clicked();
    void node_label_change(char text);
private:
    Ui::MainWindow *ui;
    char scale_factor;
    char fscale_factor;
    char buffer[255];
    char acdc;
};
#endif // MAINWINDOW_H

```

apps/main/gpiothread.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the TouchSPICE Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**     the names of its contributors may be used to endorse or promote
**     products derived from this software without specific prior written
**     permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#ifndef GPIOTHREAD_H
#define GPIOTHREAD_H
#include <QThread>
class GPIOThread : public QThread
{
    Q_OBJECT
signals:
    void sendIOStatus(int io, bool status);
    void sendDebugMessage(const char * message);
    void sendNodeLabel(char text);
private:
    const char * component_name;
    double component_value;
    char scale_factor;
    double fcomponent_value;
    char fscale_factor;
    char measure_status[9];
    void run();
private slots:
    void on_component_name_change(const char * component_name);

```

```

void on_component_value_change(double component_value, char scale_factor);
void on_fcomponent_value_change(double fcomponent_value, char fscale_factor);
void on_measure_status_change(int index, int value);
};
#endif // GPIOTHRREAD_H

```

apps/gpioAPI/include/s3c2410-regs.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the TouchSPICE Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**     the names of its contributors may be used to endorse or promote
**     products derived from this software without specific prior written
**     permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#ifndef GPIOTHRREAD_H
#define GPIOTHRREAD_H
#include <QThread>
class GPIOThread : public QThread
{
    Q_OBJECT
signals:
    void sendIOStatus(int io, bool status);
    void sendDebugMessage(const char * message);
    void sendNodeLabel(char text);
private:
    const char * component_name;

```

```

    double component_value;
    char scale_factor;
    double fcomponent_value;
    char fscale_factor;
    char measure_status[9];
    void run();
private slots:
    void on_component_name_change(const char * component_name);
    void on_component_value_change(double component_value, char scale_factor);
    void on_fcomponent_value_change(double fcomponent_value, char fscale_factor);
    void on_measure_status_change(int index, int value);
};
#endif // GPIOTHREAD_H

```

apps/gpioAPI/gpioAPI.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the TouchSPICE Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**   notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**   notice, this list of conditions and the following disclaimer in
**   the documentation and/or other materials provided with the
**   distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**   the names of its contributors may be used to endorse or promote
**   products derived from this software without specific prior written
**   permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#ifndef GPIOAPI_H
#define GPIOAPI_H
#include <stdio.h>
#include <stdlib.h>

```



```

#include <signal.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <termios.h>
#include <math.h>
#include <string.h>
#include <iostream>
#include <fstream>
int initDigIO (unsigned int input_bits[], unsigned int num_inputs, unsigned int
output_bits[], unsigned int num_outputs);
void writeDigOut (char data_register, unsigned int bit, unsigned int value);
unsigned int getDigIn (char data_register);
void closeDigIO (void);
bool gpioSend (int byte, int output, int * running);
int gpioReceive (int input, int * running);
void btWriteChar (char output_char);
void btWriteString (const char * output_string);
char btReadChar (void);
void btReadString (char buffer[]);
#endif // GPIOAPI_H

```

apps/main.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the TouchSPICE Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**     the names of its contributors may be used to endorse or promote
**     products derived from this software without specific prior written
**     permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

```

```

** $QT_END_LICENSE$
**
*****/

#include "mainwindow.h"
#include "gpiothread.h"
#include <QtGui/QApplication>
#include <QObject>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    GPIOThread gpioThread;
    MainWindow mainWindow;
    mainWindow.setOrientation(MainWindow::ScreenOrientationAuto);
    mainWindow.showExpanded();
    QObject::connect(&gpioThread,
        SIGNAL(sendIOStatus(int, bool)), &mainWindow, SLOT(connected_io_change(int, bool)));
    QObject::connect(&gpioThread, SIGNAL(sendDebugMessage(const
char*)), &mainWindow, SLOT(debug_append(const char*)));
    QObject::connect(&mainWindow, SIGNAL(sendComponentName(const char
*)), &gpioThread, SLOT(on_component_name_change(const char *)));
    QObject::connect(&mainWindow,
        SIGNAL(sendComponentValue(double, char)), &gpioThread, SLOT(on_component_value_change(double
, char)));
    QObject::connect(&mainWindow,
        SIGNAL(sendfComponentValue(double, char)), &gpioThread, SLOT(on_fcomponent_value_change(doub
le, char)));
    QObject::connect(&mainWindow, SIGNAL(sendMeasureStatus(int,
int)), &gpioThread, SLOT(on_measure_status_change(int, int)));
    QObject::connect(&gpioThread,
        SIGNAL(sendNodeLabel(char)), &mainWindow, SLOT(node_label_change(char)));
    gpioThread.start();
    return app.exec();
}

```

apps/gpioAPI/gpioAPI.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the TouchSPICE Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**
** * Redistributions of source code must retain the above copyright
**   notice, this list of conditions and the following disclaimer.
** * Redistributions in binary form must reproduce the above copyright
**   notice, this list of conditions and the following disclaimer in
**   the documentation and/or other materials provided with the
**   distribution.
** * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**   the names of its contributors may be used to endorse or promote
**   products derived from this software without specific prior written

```

```

**      permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#include "gpioAPI.h"
#include "include/s3c2410-regs.h"
#define TIMEOUT_PERIOD 5000000
std::ofstream bt_stream("/dev/ttySAC2");
static S3C2410P_GPIO GPIOp;
unsigned int rate = 6000;
static int fd;
int bt;
int initDigIO (unsigned int input_bits[], unsigned int num_inputs, unsigned int
output_bits[], unsigned int num_outputs)
{
    unsigned int i;
    struct termios options;
    if ((fd = open ("/dev/mem", O_RDWR | O_SYNC)) < 0)
        return -1;
    GPIOp = (_S3C2410_GPIO*) mmap (0, sizeof (S3C2410_GPIO), PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0x56000000);
    if (GPIOp == (void *) -1)
    {
        close (fd);
        return -1;
    }
    // Setup inputs
    for(i = 0; i < num_inputs ; i++)
    {
        //printf("Currently setting up input: %d\n",input_bits[i]);
        // If input bit is > 4, then I/O is in G register, else F
        if(input_bits[i] > 4)
        {
            // Setup pin as input
            write_reg (&GPIOp->GPGCON, pow(2,(input_bits[i]-8)*2)+pow(2,(input_bits[i]-
8)*2+1), GP_BIT((input_bits[i]-8), PIN));
            // Enable pull-up function (Set GPGUP to 0 for the specified pin)
            write_reg (&GPIOp->GPGUP, pow(2,(input_bits[i]-8)), 0 << (input_bits[i]-8));
        }
        else
        {
            // Setup pin as input
            write_reg (&GPIOp->GPFCON, pow(2,input_bits[i]*2)+pow(2,input_bits[i]*2+1),
GP_BIT(input_bits[i], PIN));
            // Enable pull-up function (Set GPFUP to 0 for the specified pin)
            write_reg (&GPIOp->GPFUP, pow(2,(input_bits[i])), 0 << (input_bits[i]));
        }
    }
    // Setup outputs
    for(i = 0 ; i < num_outputs ; i++)
    {
        //("Currently setting up output: %d\n",output_bits[i]);
        // If output bit is > 5, then I/O is in G register, else F
        if(output_bits[i] > 5)
        {
            write_reg (&GPIOp->GPGCON, pow(2,(output_bits[i]-8)*2)+pow(2,(output_bits[i]-
8)*2+1), GP_BIT((output_bits[i]-8), POUT));

```

```

        // Disable pull-up function (Set GPGUP to 0 for the specified pin)
        write_reg (&GPIOp->GPGUP, pow(2,(input_bits[i]-8)), 1 << (input_bits[i]-8));
        // Set digital out to 0 to make sure it is low before starting program
        (normally is low on setup)
        // This will make it so that we can detect when something is plugged in or
        not.
        //(When plugged in, the input of the other node will be pulled down
        (changed))
        writeDigOut ('g',output_bits[i],0);
    }
    else
    {
        write_reg (&GPIOp->GPFCON, pow(2,output_bits[i]*2)+pow(2,output_bits[i]*2+1),
        GP_BIT(output_bits[i], POUT));
        // Disable pull-up function (Set GPFUP to 0 for the specified pin)
        write_reg (&GPIOp->GPFUP, pow(2,(input_bits[i])), 1 << (input_bits[i]));
        // Set digital out to 0 to make sure it is low before starting program
        (normally is low on setup)
        // This will make it so that we can detect when something is plugged in or
        not.
        //(When plugged in, the input of the other node will be pulled down
        (changed))
        writeDigOut ('f',output_bits[i],0);
    }
}
/* Open Port */
bt = open("/dev/ttySAC2", O_RDWR | O_NOCTTY | O_NDELAY);
if(bt == -1) {
    printf("ERROR Open Serial Port!");
}
/* Serial Configuration */
tcgetattr(bt, &options); // Get Current Config
cfsetispeed(&options, B115200); // Set Baud Rate
cfsetospeed(&options, B115200);
options.c_cflag = (options.c_cflag & ~CSIZE) | CS8;
options.c_iflag = IGNBRK;
options.c_lflag = 0;
options.c_oflag = 0;
options.c_cflag |= CLOCAL | CREAD;
options.c_cc[VMIN] = 1;
options.c_cc[VTIME] = 5;
options.c_iflag &= ~(IXON|IXOFF|IXANY);
options.c_cflag &= ~(PARENB | PARODD);
/* Save the configuration */
tcsetattr(bt, TCSANOW, &options);
/* Flush the input (read) buffer */
tcflush(bt,TCIOFLUSH);
// set single EINT1->GPF1 as input (Mask 0xC = 0b1100)
//write_reg (&GPIOp->GPFCON, 0xC, GP_BIT(1, PIN));
// set single EINT3->GPF3 as output (Mask 0xC0 = 0b11000000)
//write_reg (&GPIOp->GPFCON, 0xC0, GP_BIT(3, POUT));
return 0;
}
// Unmap the system peripheral space and close the /dev/mem file descriptor
void closeDigIO (void)
{
    munmap (GPIOp, sizeof (S3C2410_GPIO));
}
// Write the specified value to the GPIOF data register
void writeDigOut (char data_register, unsigned int bit, unsigned int value)
{
    if(data_register == 'f')
    {
        // Write to the GPFDAT register (register, bitmask, value)
        write_reg (&GPIOp->GPFDAT, pow(2, bit), value << bit);
    }
    else if(data_register == 'g')
    {
        bit -= 8;
        // Write to the GPGDAT register (register, bitmask, value)
    }
}

```

```

        write_reg (&GPIOp->GPGDAT, pow(2, bit), value << bit);
    }
}
// Reads all bits in the GPIOF data register
unsigned int getDigIn (char data_register)
{
    if(data_register == 'g')
        return GPIOp->GPGDAT;
    else
        return GPIOp->GPFDAT;
}
bool gpioSend(int byte, int output, int * running)
{
    unsigned int sent_parity[4];
    unsigned int i;
    unsigned int timeout;
    unsigned int input_mask;
    char data_register;
    // I/O 0-5 are in the F Data Register
    // I/O 8-9 are in the G Data Register
    if(output > 5)
    {
        data_register = 'g';
        input_mask = pow(2,output - 9);
    }
    else
    {
        data_register = 'f';
        input_mask = pow(2,output - 1);
    }
    // Calculate 4-bit sent_parity
    for(i = 0 ; i < 4 ; i++)
    {
        sent_parity[i] = !(byte & (1 << (i*2))) ^ !(byte & (1 << ((i*2)+1)));
        //printf("Parity[%d] = %d\n",i, sent_parity[i]);
    }
    // Shift it into the MSB of the byte
    for(i = 0 ; i < 4 ; i++)
    {
        if(sent_parity[i] == 0)
            byte += pow(2,sent_parity[i]*(i+8))-1;
        else
            byte += pow(2,sent_parity[i]*(i+8));
    }
    // Send a 1 to flag intent to send a byte
    writeDigOut (data_register,output,1);
    // Wait for acknowledgment
    //printf("W1\n");
    usleep(rate-5000);
    // Wait for a 1
    timeout = 0;
    while(!(getDigIn(data_register) & input_mask) && running)
    {
        if(timeout > TIMEOUT_PERIOD)
            break;
        else
            timeout++;
    }
    if(timeout > TIMEOUT_PERIOD)
    {
        writeDigOut (data_register,output,0);
        return false;
    }
    //printf("G1\n");
    //usleep(rate);
    // Wait for a 0
    timeout = 0;
    while((getDigIn(data_register) & input_mask) && running)
    {
        if(timeout > TIMEOUT_PERIOD)

```

```

        break;
    else
        timeout++;
    }
    if(timeout > TIMEOUT_PERIOD)
    {
        writeDigOut (data_register,output,0);
        return false;
    }
    //printf("G0\n");
    //usleep(rate);
    // Wait for a 1
    timeout = 0;
    while(!(getDigIn(data_register) & input_mask) && running)
    {
        if(timeout > TIMEOUT_PERIOD)
            break;
        else
            timeout++;
    }
    if(timeout > TIMEOUT_PERIOD)
    {
        writeDigOut (data_register,output,0);
        return false;
    }
    //printf("G1\n");
    usleep(rate+4000);
    // Ok to send data
    // Send a 0 to signal transmission begin
    writeDigOut (data_register,output,0);
    //usleep(rate-5);
    //printf("Sending: %d\n",byte);
    // Begin transmission
    // LSB -> MSB
    for(i = 0 ; i < 12 ; i++){
        if(byte & (1 << i)){
            writeDigOut (data_register,output,1);
            //printf("Sending 1\n");
        }else{
            writeDigOut (data_register,output,0);
            //printf("Sending 0\n");
        }
        usleep(rate);
    }
    // Set digital out to 0 to conclude transfer
    writeDigOut (data_register,output,0);
    return true;
}

int gpioReceive (int input, int * running)
{
    unsigned int bits_received[12];
    unsigned int received_parity[4];
    unsigned int i;
    unsigned int output = input + 1;
    int received_byte = 0;
    char data_register;
    unsigned int input_mask;
    unsigned int timeout;
    // I/O 0-5 are in the F Data Register
    // I/O 8-9 are in the G Data Register
    if(output > 5)
    {
        data_register = 'g';
        input_mask = pow(2,input - 8);
    }
    else
    {
        data_register = 'f';
        input_mask = pow(2,input);
    }
}

```

```

// My turn to receive a byte
// Since we have a 1 on our input, we need to send an ack
// Send a 1
writeDigOut (data_register,output,1);
usleep(rate+2000);
// Still have a 1 on our input pin? If not...fuck it
if(!(getDigIn(data_register) & input_mask) && running)
{
    writeDigOut (data_register,output,0);
    return -1;
}
// Then send a 0
writeDigOut (data_register,output,0);
usleep(rate+2000);
// Then send a 1
writeDigOut (data_register,output,1);
usleep(rate+2000);
// Set output back to 0 just in case something fails
writeDigOut (data_register,output,0);
// Wait for a 0 to start reception
// pow(2,input) is the mask to grab the specified input from the GPF/G register
timeout = 0;
while(!((getDigIn(data_register) & input_mask) && running))
{
    if(timeout > TIMEOUT_PERIOD)
        break;
    else
        timeout++;
}
if(timeout > TIMEOUT_PERIOD)
    return -1;
usleep(rate-2000);
// Begin reception
for(i = 0 ; i < 12 ; i++){
    if(getDigIn(data_register) & input_mask)
    {
        bits_received[i] = 1;
        //printf("Receive 1\n");
    }
    else
    {
        bits_received[i] = 0;
        //printf("Receive 0\n");
    }
    usleep(rate);
}
// Processes 0 LSB -> 7 MSB
received_byte = 0;
for(i = 0 ; i < 8 ; i++){
    if(bits_received[i] == 0)
        received_byte += pow(2,bits_received[i]*i)-1;
    else
        received_byte += pow(2,bits_received[i]*i);
}
//printf("Received: %d\n",received_byte);
// Calculate 4-bit received_parity
for(i = 0 ; i < 4 ; i++)
{
    received_parity[i] = !!(received_byte & (1 << (i*2))) ^ !!(received_byte & (1 << ((i*2)+1)));
    //printf("Parity[%d] = %d\n",i, received_parity[i]);
}
if(received_parity[0] == bits_received[8] && received_parity[1] == bits_received[9]
&& received_parity[2] == bits_received[10] && received_parity[3] == bits_received[11])
{
    return received_byte;
}
else
{

```

```

        //printf("Parity error! %d %d %d
%d\n",bits_received[8],bits_received[9],bits_received[10],bits_received[11]);
        //printf("          %d %d %d
%d\n",received_parity[0],received_parity[1],received_parity[2],received_parity[3]);
        return -2;
    }
}

void btWriteChar(char output_char){
    write(bt,&output_char,1);
    write(bt,"!",1);
}

void btWriteString(const char * output_string){
    //write(bt,output_string,strlen(output_string));
    bt_stream.write(output_string,strlen(output_string));
    bt_stream.flush();
    bt_stream.write("!",1);
    bt_stream.flush();
    //printf("Actually sent: %s\n",output_string);
    //write(bt,"!",1);
}

char btReadChar (void)
{
    char buffer[2];
    memset(buffer,0,2);
    read(bt,buffer,1);
    return buffer[0];
}

void btReadString (char buffer[])
{
    memset(buffer,0,255);
    read(bt,buffer,255);
}

```

apps/gpiothread.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the TouchSPICE Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**
** * Redistributions of source code must retain the above copyright
** notice, this list of conditions and the following disclaimer.
** * Redistributions in binary form must reproduce the above copyright
** notice, this list of conditions and the following disclaimer in
** the documentation and/or other materials provided with the
** distribution.
** * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
** the names of its contributors may be used to endorse or promote
** products derived from this software without specific prior written
** permission.
**
**

```



```

** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#include <QtGui/QApplication>
#include "gpiothread.h"
#include "../gpioAPI/gpioAPI.h"
#define TIMEOUT_PERIOD 5000000
extern int bt;
int running = 50;
using namespace std;
/*
    Signal handler to stop the program gracefully
*/
void done (int sig)
{
    running = 0;
}
void GPIOThread::on_component_name_change(const char * component_name)
{
    this->component_name = component_name;
}
void GPIOThread::on_component_value_change(double component_value, char scale_factor)
{
    this->component_value = component_value;
    this->scale_factor = scale_factor;
}
void GPIOThread::on_fcomponent_value_change(double fcomponent_value, char fscale_factor)
{
    this->fcomponent_value = fcomponent_value;
    this->fscale_factor = fscale_factor;
}
// measure_status is of form:      type: "VVVVIIII"
//                                io: "01230123"
//                                value: "02020202" (0 = false, 2 = true)
void GPIOThread::on_measure_status_change(int index, int value)
{
    measure_status[index] = '0'+value;
}
void GPIOThread::run()
{
    int received_byte = 0; // Needs to be int because -1 is the return value if fail
    int connected_identity_pins[4] = {4,4,4,4};
    unsigned int i, j, io_flag, node_identity_flag,
neighbor_flag, done_getting_neighbors;
    unsigned int input_bits[4] = {0,2,4,8};
    unsigned int output_bits[4] = {1,3,5,9};
    unsigned int num_inputs = sizeof(input_bits)/sizeof(int);
    unsigned int num_outputs = sizeof(output_bits)/sizeof(int);
    unsigned int input_mask[4];
    unsigned int timeout;
    bool connected_io[] = {false,false,false,false};
    char connected_identities[5] = {'N','N','N','N',0};
    char temp[9] = {'N','N','N','N','N','N','N','N',0};
    char data_register[4] = {0};
    char bt_buffer;
    char node_identity = '0';
    char temp_value[20] = {0};
    char debug_buffer[255] = {0};

```

```

component_name = "none";
component_value = 0;
scale_factor = ' ';
for(j = 0 ; j < 8 ; j++)
    measure_status[j] = '0';
measure_status[8] = 0;
//signal (SIGINT, done); // set up signal handler
//emit sendDebugMessage("Initializing digital I/O...\n");
if (initDigIO(input_bits,num_inputs,output_bits,num_outputs) < 0)
{
    printf ("Couldn't initialize digital I/O!\n");
    exit (1);
}
//emit sendDebugMessage("Done Initializing digital I/O!\n");
// Create input masks used for checking input pins
for(i = 0; i < num_inputs ; i++)
{
    if(input_bits[i] > 4)
    {
        data_register[i] = 'g';
        input_mask[i] = pow(2,input_bits[i]-8);
    }
    else
    {
        data_register[i] = 'f';
        input_mask[i] = pow(2,input_bits[i]);
    }
}
// Begin main GPIOThread execution loop
while (running)
{
    //QCoreApplication::instance()->processEvents();
    bt_buffer = btReadChar();
    while(bt_buffer == 0 && running)
    {
        bt_buffer = btReadChar();
        for(i = 0; i < num_inputs ; i++)
        {
            // If current input is high (either internal pull-up or actually
receiving something)
            if((getDigIn(data_register[i]) & input_mask[i]))
            {
                if(connected_io[i] == true)
                {
                    received_byte = gpioReceive(input_bits[i],&running);
                    // If we successfully received a byte, send a byte back
                    if(received_byte > -1)
                    {
                        usleep(8000);
                        ///printf("Successfully received byte %c from pin
%d\n",received_byte,input_bits[i]);
                        if(node_identity == '0')
                        {
                            //emit sendDebugMessage("ERROR: NODE LETTER HAS NOT BEEN
ASSIGNED!\n");
                        }
                        else if(received_byte >= 48 && received_byte <= 51)
                        {
                            // Store the newly received IO # into our array to speed
things up
                            connected_identity_pins[i] = received_byte;
                            // Send IO number
                            if(!gpioSend(i+'0', input_bits[i]+1, &running))
                            {
                                sprintf(debug_buffer,"ERROR SENDING IO NUMBER FROM
PIN %d!\n",input_bits[i]+1);
                                //emit sendDebugMessage(debug_buffer);
                                //connected_io[i] = false;
                                //emit sendIOStatus(i,connected_io[i]);
                                //connected_identities[i] = 'N';

```

```

        //connected_identity_pins[i] = 4;
    }
    else
    {
        ///printf("Successfully sent IO number from pin
%d!\n",input_bits[i]+1);
    }
}
else if(received_byte >= 65 && received_byte <= 90)
{
    // Store the newly received identity into our array to
speed things up
    connected_identities[i] = received_byte;
    // Send node identity
    if(!gpioSend(node_identity, input_bits[i]+1, &running))
    {
        //sprintf(debug_buffer,"ERROR SENDING IDENTITY FROM
PIN %d!\n",input_bits[i]+1);
        //emit sendDebugMessage(debug_buffer);
        //connected_io[i] = false;
        //emit sendIOStatus(i,connected_io[i]);
        //connected_identities[i] = 'N';
        //connected_identity_pins[i] = 4;
    }
    else
    {
        ///printf("Successfully sent identity from pin
%d!\n",input_bits[i]+1);
    }
}
else
{
    sprintf(debug_buffer,"ERROR RECEIVED NON-A-Z IDENTITY OR
NON-0-3 IO NUMBER FROM PIN %d!\n",input_bits[i]+1);
    //emit sendDebugMessage(debug_buffer);
}
}
else
{
    if(received_byte == -1)
    {
        sprintf(debug_buffer,"ERROR RECEIVING DATA FROM PIN %d!
TIMED OUT! IO %d DISCONNECTED!\n",input_bits[i],i);
        //emit sendDebugMessage(debug_buffer);
    }
    else
    {
        sprintf(debug_buffer,"ERROR RECEIVING DATA FROM PIN %d!
PARITY MISMATCH! IO %d DISCONNECTED!\n",input_bits[i],i);
        //emit sendDebugMessage(debug_buffer);
    }
    connected_io[i] = false;
    emit sendIOStatus(i,connected_io[i]);
    connected_identities[i] = 'N';
    connected_identity_pins[i] = 4;
}
}
}
else if(connected_io[i] == false)
{
    usleep(100000);
    if(!(getDigIn(data_register[i]) & input_mask[i]))
    {
        sprintf(debug_buffer,"IO %d Connected!\n",i);
        printf("IO %d Connected!\n",i);
        //emit sendDebugMessage(debug_buffer);
        connected_io[i] = true;
        emit sendIOStatus(i,connected_io[i]);
    }
}
}
}

```

```

    }
    switch (bt_buffer)
    {
        case 'g' :
            //emit sendDebugMessage("Listening for node #...\n");
            node_identity = btReadChar();
            timeout = 0;
            while(node_identity == 0)
            {
                if(timeout > 100000)
                    break;
                else
                    timeout++;
                node_identity = btReadChar();
            }
            if(timeout > 100000)
            {
                //emit sendDebugMessage("Timed-out Listening...\n");
            }
            else
            {
                if(node_identity >=65 && node_identity <= 90)
                {
                    btWriteChar(node_identity);
                    QCoreApplication::processEvents();
                    sprintf(debug_buffer, "%c\n", node_identity);
                    emit sendNodeLabel(node_identity);
                    //emit sendDebugMessage(debug_buffer);
                }
                //else
                //emit sendDebugMessage("ERROR: NODE IDENTITY IS NOT A-Z!\n");
            }
            break;
        case 'i' :
            btWriteString(component_name);
            QCoreApplication::processEvents();
            sprintf(debug_buffer, "Asked for component name, gave:
%s\n", component_name);
            //emit sendDebugMessage(debug_buffer);
            break;
        case 'j' :
            if(fscale_factor == '0')
            {
                if(scale_factor == 'M')
                    sprintf(temp_value, "%3.1fmeg", component_value);
                else
                    sprintf(temp_value, "%3.1f%c", component_value, scale_factor);
            }
            else
            {
                if(fscale_factor == 'M')
                {
                    if(scale_factor == 'M')
                        sprintf(temp_value, "%3.1fmeg
%3.1fmeg", component_value, fcomponent_value);
                    else
                        sprintf(temp_value, "%3.1f%c
%3.1fmeg", component_value, scale_factor, fcomponent_value);
                }
                else
                {
                    if(scale_factor == 'M')
                        sprintf(temp_value, "%3.1fmeg
%3.1f%c", component_value, fcomponent_value, fscale_factor);
                    else
                        sprintf(temp_value, "%3.1f%c
%3.1f%c", component_value, scale_factor, fcomponent_value, fscale_factor);
                }
            }
        }
    }
}

```

```

        btWriteString(temp_value);
        QCoreApplication::processEvents();
        sprintf(debug_buffer, "Asked for component value, gave: %s\n", temp_value);
        //emit sendDebugMessage(debug_buffer);
        break;
    case 'k' :
        btWriteString(measure_status);
        QCoreApplication::processEvents();
        sprintf(debug_buffer, "Asked for measure_status, gave:
%s\n", measure_status);
        //emit sendDebugMessage(debug_buffer);
        break;
    case 'z' :
        btWriteString("Simulate Disconnect\n");
        for(i=0; i<4; i++)
        {
            connected_io[i] = false;
            connected_identities[i] = 'N';
            connected_identity_pins[i] = 4;
        }
        break;
    case 'h' :
        // Go through each output and ask for the identity of what they are
        connected to
        if(node_identity == '0')
        {
            //emit sendDebugMessage("ERROR: NODE LETTER HAS NOT BEEN
ASSIGNED!\n");
            break;
        }
        neighbor_flag = 0;
        done_getting_neighbors = 0;
        while(!done_getting_neighbors && neighbor_flag < 3)
        {
            for(i = 0; i < num_outputs; i++)
            {
                // If the IO reports being connected and it is not the first try
                // for neighbor identification, the IO might have been
                disconnected
                // mid-command, recheck to make sure it's ACTUALLY connected
                if(connected_io[i] && neighbor_flag != 0)
                {
                    // If the IO is high (either internal pull-up = disconnected,
                    or trying to receive = BAD (can't send/receive at same time))
                    if((getDigIn(data_register[i]) & input_mask[i]))
                    {
                        // The node isn't actually connected...
                        sprintf(debug_buffer, "PIN %d IS'T CONNECTED ANYMORE,
                        UPDATING...\n", i);
                        //emit sendDebugMessage(debug_buffer);
                        //btWriteString("PIN %d IS'T CONNECTED ANYMORE,
                        UPDATING...\n", i);

                        connected_io[i] = false;
                        connected_identities[i] = 'N';
                        connected_identity_pins[i] = 4;
                        emit sendIOStatus(i, connected_io[i]);
                    }
                }
            }
            // If the IO reports it is connected and has not established it's
            neighbor:
            if(connected_io[i] && connected_identities[i] == 'N')
            {
                // Wait until we receive a good byte back
                node_identity_flag = 0;
                while(node_identity_flag < 3)
                {
                    if(node_identity_flag != 0)
                    {
                        //emit sendDebugMessage("ERROR! RETRYING IDENTITY
                        EXCHANGE!\n");

```

```

        usleep(800000);
    }
    //usleep(8000);
    // Send a byte with a 4-bit parity and the address of
running
    if(!gpioSend(node_identity, output_bits[i], &running))
    {
        sprintf(debug_buffer,"ERROR SENDING IDENTITY FROM PIN
%d!\n",input_bits[i]+1);

        //emit sendDebugMessage(debug_buffer);
        node_identity_flag++;
    }
    else
    {
        ///printf("Successfully sent node identity %c from
pin %d!\n",node_identity,output_bits[i]);
        // Wait to receive from the expected pin
        timeout = 0;
        while(!(getDigIn(data_register[i]) & input_mask[i]))
        {
            if(timeout > TIMEOUT_PERIOD)
                break;
            else
                timeout++;
        }
        if(timeout > TIMEOUT_PERIOD)
        {
            //emit sendDebugMessage("FAILED TO RECEIVE
IDENTITY BYTE BACK!\n");
            node_identity_flag++;
        }
        else
        {
            // Receive from the expected pin
            received_byte = gpioReceive(output_bits[i]-
1,&running);
            received a byte

            if(received_byte > -1)
            {
                if(received_byte >=65 && received_byte <= 90)
                {
                    connected_identities[i] = received_byte;
                    node_identity_flag = 3;
                    ///printf("IO %d is connected to node
identity %c\n",i,received_byte);

                    // Wait until we receive a good byte back
                    io_flag = 0;
                    while(io_flag < 3)
                    {
                        if(io_flag != 0)
                        {
                            //emit sendDebugMessage("ERROR!
RETRYING IO EXCHANGE!\n");

                            usleep(800000);
                        }
                        //usleep(8000);
                        // Send IO number
                        if(!gpioSend(i+'0', output_bits[i],
&running))
                        {
                            sprintf(debug_buffer,"ERROR
SENDING IO NUMBER FROM PIN %d!\n",input_bits[i]+1);

                            //emit
                            sendDebugMessage(debug_buffer);

                            io_flag++;
                        }
                        else
                        {

```

```

number %d from pin %d!\n",i,output_bits[i]);
expected pin

while(!(getDigIn(data_register[i]) & input_mask[i]))

sendDebugMessage("FAILED TO RECEIVE IO BYTE BACK!\n");

pin
gpioReceive(output_bits[i]-1,&running);
if we successfully received a byte

received_byte <= 51)

connected_identity_pins[i] = received_byte;

connected to IO %d\n",i,received_byte);

sprintf(debug_buffer,"ERROR: NODE IO RECEIVED BACK IS NOT 0-3! RECEIVED:
%c\n",received_byte);

sendDebugMessage(debug_buffer);

sprintf(debug_buffer,"ERROR RECEIVING IO NUMBER FROM PIN %d! TIMED
OUT!\n",input_bits[i]);

sendDebugMessage(debug_buffer);

sprintf(debug_buffer,"ERROR RECEIVING IO NUMBER FROM PIN %d! PARITY
MISMATCH!\n",input_bits[i]);

sendDebugMessage(debug_buffer);

//printf("Successfully sent IO
// Wait to receive from the
timeout = 0;

{
    if(timeout > TIMEOUT_PERIOD)
        break;
    else
        timeout++;
}
if(timeout > TIMEOUT_PERIOD)
{
    //emit
    io_flag++;
}
else
{
    // Receive from the expected
    received_byte =
    // Set the received flag to 1
    if(received_byte > -1)
    {
        if(received_byte >= 48 &&
        {
            io_flag = 3;
            //printf("IO %d is
        }
        else
        {
            //emit
            io_flag++;
        }
    }
    else
    {
        if(received_byte == -1)
        {
            //emit
            io_flag += 2;
        }
        else
        {
            //emit
            io_flag++;
        }
    }
}
}
}

```

```

        }
    }
    else
    {
        //emit sendDebugMessage("ERROR: NODE
IDENTITY RECEIVED BACK IS NOT A-Z!\n");
        node_identity_flag++;
    }
    else
    {
        if(received_byte == -1)
        {
            sprintf(debug_buffer,"ERROR RECEIVING
IDENTITY FROM PIN %d! TIMED OUT!\n",input_bits[i]);
            //emit sendDebugMessage(debug_buffer);
            node_identity_flag += 2;
        }
        else
        {
            sprintf(debug_buffer,"ERROR RECEIVING
IDENTITY FROM PIN %d! PARITY MISMATCH!\n",input_bits[i]);
            //emit sendDebugMessage(debug_buffer);
            node_identity_flag++;
        }
    }
}
}
}
else if(connected_io[i])
{
    ///printf("IO %d is connected to pin %d on node identity
%c\n",i,connected_identity_pins[i],connected_identities[i]);
}
else
{
    ///printf("IO %d is not connected\n",i);
}
}
// Go through and make sure all IO that reported as connected, have
neighbor identities and IO
for(i = 0; i < num_outputs; i++)
{
    // If the IO reports it is connected and has not established it's
neighbor:
    if((connected_io[i] && connected_identities[i] == 'N') ||
(connected_io[i] && connected_identity_pins[i] == 4))
    {
        neighbor_flag++;
        if(neighbor_flag == 3)
        {
            sprintf(debug_buffer,"CRITICAL ERROR: SIMULATE DISCONNECT
FOR PIN %d!\n",i);
            //emit sendDebugMessage(debug_buffer);
            //btWriteString("CRITICAL ERROR: SIMULATE DISCONNECT FOR
PIN %d!\n",i);

            connected_io[i] = false;
            connected_identities[i] = 'N';
            connected_identity_pins[i] = 4;
            //emit sendIOStatus(i,connected_io[i]);
        }
        else
        {
            done_getting_neighbors = 0;
            break;
        }
    }
}
done_getting_neighbors = 1;

```



```

    }
}
// Send the host all of the node information
j = 0;
for(i = 0; i < num_outputs * 2; i+=2)
{
    temp[i] = connected_identities[j];
    if(connected_identity_pins[j] == 4)
        temp[i+1] = 'N';
    else
        temp[i+1] = (char)connected_identity_pins[j];
    j++;
}
sprintf(debug_buffer,"Just sent %s\n",temp);
//emit sendDebugMessage(debug_buffer);
btWriteString(temp);
QCoreApplication::processEvents();
break;
default :
    sprintf(debug_buffer,"Invalid Command %c!\n", bt_buffer);
    //emit sendDebugMessage(debug_buffer);
    break;
}
}
printf ("Goodbye\n");
closeDigIO();
}

```

apps/mainwindow.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the TouchSPICE Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**
** * Redistributions of source code must retain the above copyright
** notice, this list of conditions and the following disclaimer.
** * Redistributions in binary form must reproduce the above copyright
** notice, this list of conditions and the following disclaimer in
** the documentation and/or other materials provided with the
** distribution.
** * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
** the names of its contributors may be used to endorse or promote
** products derived from this software without specific prior written
** permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT

```

```

** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtCore/QCoreApplication>
extern int bt;
extern char buff[255];
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow)
{
    // Setup the UI
    ui->setupUi(this);
    setWindowFlags(Qt::FramelessWindowHint);
    // Set the current index to 0 and hide the controls overlay
    ui->stackedWidget->setCurrentIndex(0);
    hideControls();
    update();
}
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::setOrientation(ScreenOrientation orientation)
{
    #if defined(Q_OS_SYMBIAN)
        // If the version of Qt on the device is < 4.7.2, that attribute won't work
        if (orientation != ScreenOrientationAuto) {
            const QStringList v = QString::fromAscii(qVersion()).split(QLatin1Char('.'));
            if (v.count() == 3 && (v.at(0).toInt() < 16 | v.at(1).toInt() < 8 |
v.at(2).toInt() < 0x040702) {
                qWarning("Screen orientation locking only supported with Qt 4.7.2 and
above");
                return;
            }
        }
    #endif // Q_OS_SYMBIAN
    Qt::WidgetAttribute attribute;
    switch (orientation) {
    #if QT_VERSION < 0x040702
        // Qt < 4.7.2 does not yet have the Qt::WA_*Orientation attributes
        case ScreenOrientationLockPortrait:
            attribute = static_cast<Qt::WidgetAttribute>(128);
            break;
        case ScreenOrientationLockLandscape:
            attribute = static_cast<Qt::WidgetAttribute>(129);
            break;
        default:
            case ScreenOrientationAuto:
                attribute = static_cast<Qt::WidgetAttribute>(130);
                break;
    #else // QT_VERSION < 0x040702
        case ScreenOrientationLockPortrait:
            attribute = Qt::WA_LockPortraitOrientation;
            break;
        case ScreenOrientationLockLandscape:
            attribute = Qt::WA_LockLandscapeOrientation;
            break;
        default:
            case ScreenOrientationAuto:
                attribute = Qt::WA_AutoOrientation;
                break;
    #endif
    }
}

```

```

#endif // QT_VERSION < 0x040702
};
setAttribute(attribute, true);
}
void MainWindow::showExpanded()
{
#ifdef Q_OS_SYMBIAN
    showFullScreen();
#elif defined(Q_WS_MAEMO_5)
    showMaximized();
#else
    show();
#endif
}
void MainWindow::node_label_change(char text)
{
    char temp[2] = {0};
    sprintf(temp, "%c", text);
    ui->node_label->setText(temp);
}
void MainWindow::debug_append(const char * message)
{
    ui->debug_box->appendPlainText(message);
}
void MainWindow::connected_io_change(int io, bool status)
{
    switch (io)
    {
        case 0 :
            if(status)
            {
                ui->status_0->setPixmap(QPixmap(":/images/images/green.png"));
            }
            else
            {
                ui->status_0->setPixmap(QPixmap(":/images/images/red.png"));
            }
            update();
            break;
        case 1 :
            if(status)
            {
                ui->status_1->setPixmap(QPixmap(":/images/images/green.png"));
            }
            else
            {
                ui->status_1->setPixmap(QPixmap(":/images/images/red.png"));
            }
            update();
            break;
        case 2 :
            if(status)
            {
                ui->status_2->setPixmap(QPixmap(":/images/images/green.png"));
            }
            else
            {
                ui->status_2->setPixmap(QPixmap(":/images/images/red.png"));
            }
            update();
            break;
        case 3 :
            if(status)
            {
                ui->status_3->setPixmap(QPixmap(":/images/images/green.png"));
            }
            else
            {
                ui->status_3->setPixmap(QPixmap(":/images/images/red.png"));
            }
    }
}

```

```

        update();
        break;
    default:
        break;
    }
}

void MainWindow::on_value_slider_valueChanged(int value)
{
    ui->doubleSpinBox->setValue(value);
}

void MainWindow::on_fvalue_slider_valueChanged(int value)
{
    ui->fdoubleSpinBox->setValue(value);
}

void MainWindow::on_sac_resistor_clicked()
{
    emit sendComponentName("resistor");
    ui->stackedWidget->setCurrentIndex(1);
    ui->value_label->setText("Ohms");
    showControls('a');
}

void MainWindow::on_sac_capacitor_clicked()
{
    emit sendComponentName("capacitor");
    ui->stackedWidget->setCurrentIndex(2);
    ui->value_label->setText("Farads");
    showControls('a');
}

void MainWindow::on_sac_inductor_clicked()
{
    emit sendComponentName("inductor");
    ui->stackedWidget->setCurrentIndex(3);
    ui->value_label->setText("Henrys");
    showControls('a');
}

void MainWindow::on_sac_diode_clicked()
{
    emit sendComponentName("diode");
    ui->stackedWidget->setCurrentIndex(4);
    showControls('m');
}

void MainWindow::on_sac_opamp_clicked()
{
    emit sendComponentName("opamp");
    ui->stackedWidget->setCurrentIndex(5);
    showControls('m');
}

void MainWindow::on_sac_bjt_clicked()
{
    emit sendComponentName("bjt");
    ui->stackedWidget->setCurrentIndex(6);
    showControls('m');
}

void MainWindow::on_sac_nmos_clicked()
{
    emit sendComponentName("nmos");
    ui->stackedWidget->setCurrentIndex(7);
    showControls('m');
}

void MainWindow::on_sac_vsource_clicked()
{
    emit sendComponentName("vsource");
    ui->stackedWidget->setCurrentIndex(8);
    ui->dc_sine_button->setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
    ui->vsource_label->setPixmap(QPixmap(":/images/images/vsource2.png"));
    acdc = 'd';
    ui->value_label->setText("Volts");
    ui->fvalue_label->setText("Hz");
    showControls('a');
}

```

```

void MainWindow::on_sac_isource_clicked()
{
    emit sendComponentName("isource");
    ui->stackedWidget->setCurrentIndex(9);
    ui->value_label->setText("Amps");
    showControls('a');
}
void MainWindow::on_res_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_cap_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_inductor_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_diode_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_opamp_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_bjt_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_nmos_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_vsource_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_isource_back_button_clicked()
{
    hideControls();
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::hideControls()
{
    ui->v0->hide();
    ui->v1->hide();
    ui->v2->hide();
    ui->v3->hide();
    ui->i0->hide();
    ui->i1->hide();
    ui->i2->hide();
    ui->i3->hide();
    ui->status_0->hide();
    ui->status_1->hide();
    ui->status_2->hide();
    ui->status_3->hide();
    ui->fdoubleSpinBox->hide();
    ui->fvalue_label->hide();
    ui->fvalue_slider->hide();
    ui->fscale_button->hide();
}

```

```

    ui->doubleSpinBox->hide();
    ui->value_label->hide();
    ui->value_slider->hide();
    ui->scale_button->hide();
    emit sendComponentName("none");
    fscale_factor = '0';
    emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
}
void MainWindow::showControls(char options)
{
    // Move everything back to their default positions and default checked states
    ui->status_0->move(QPoint(11,50));
    ui->status_1->move(QPoint(11,125));
    ui->status_2->move(QPoint(296,50));
    ui->status_3->move(QPoint(296,125));
    ui->v0->move(QPoint(10,70));
    ui->i0->move(QPoint(10,104));
    ui->v2->move(QPoint(281,70));
    ui->i2->move(QPoint(281,104));
    ui->doubleSpinBox->move(QPoint(91,155));
    ui->scale_button->move(QPoint(175,155));
    ui->value_label->move(QPoint(200,155));
    ui->value_slider->move(QPoint(10,210));
    ui->v0->setCheckState(Qt::CheckState(0));
    ui->i0->setCheckState(Qt::CheckState(0));
    ui->v1->setCheckState(Qt::CheckState(0));
    ui->i1->setCheckState(Qt::CheckState(0));
    ui->v2->setCheckState(Qt::CheckState(0));
    ui->i2->setCheckState(Qt::CheckState(0));
    ui->v3->setCheckState(Qt::CheckState(0));
    ui->i3->setCheckState(Qt::CheckState(0));
    // Show "a"ll measurement and value controls
    // Resistor, Capacitor, Inductor, Isource, Vsource
    if(options == 'a')
    {
        // Display V/I measurements 0 and 2
        ui->v0->show();
        ui->i0->show();
        ui->v2->show();
        ui->i2->show();
        // Enable controls based off componenet
        switch (ui->stackedWidget->currentIndex())
        {
            case 1 : // Resistor
                ui->value_slider->setRange(1,999);
                ui->doubleSpinBox->setRange(0,999.9);
                ui->value_slider->setValue(50);
                scale_factor = 'k';
                break;
            case 2 : // Capacitor
                ui->value_slider->setRange(1,999);
                ui->doubleSpinBox->setRange(0,999.9);
                ui->value_slider->setValue(5);
                scale_factor = 'u';
                break;
            case 3 : // Inductor
                ui->value_slider->setRange(1,999);
                ui->doubleSpinBox->setRange(0,999.9);
                ui->value_slider->setValue(3);
                scale_factor = 'm';
                break;
            case 8 : // Voltage Source
                ui->value_slider->setRange(1,99);
                ui->doubleSpinBox->setRange(0,99.9);
                ui->value_slider->setValue(5);
                scale_factor = ' ';
                break;
            case 9 : // Current Source
                ui->value_slider->setRange(1,999);
                ui->doubleSpinBox->setRange(0,999.9);

```

```

        ui->value_slider->setValue(80);
        scale_factor = 'm';
        break;
    default :
        ui->value_slider->setValue(1);
        scale_factor = ' ';
        break;
    }
    sprintf(buffer,"%c",scale_factor);
    ui->scale_button->setText(buffer);
    emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
    ui->doubleSpinBox->show();
    ui->value_label->show();
    ui->value_slider->show();
    ui->scale_button->show();
}
// Show only "m"easurement controls
// Diode, Opamp, NMOS, BJT
else if(options == 'm')
{
    // Only show measurements based off component
    switch (ui->stackedWidget->currentIndex())
    {
        case 4 : // Diode
            ui->v0->show();
            ui->i0->show();
            ui->v2->show();
            ui->i2->show();
            break;
        case 5 : // Opamp
            ui->v0->move(QPoint(30,50));
            ui->i0->move(QPoint(70,50));
            ui->v1->move(QPoint(30,125));
            ui->i1->move(QPoint(70,125));
            ui->v0->show();
            ui->i0->show();
            ui->v1->show();
            ui->i1->show();
            ui->v2->show();
            ui->i2->show();
            break;
        case 6 : case 7: // NMOS/BJT
            ui->status_2->move(QPoint(296,65));
            ui->status_3->move(QPoint(296,110));
            ui->v2->move(QPoint(220,65));
            ui->i2->move(QPoint(260,65));
            ui->v3->move(QPoint(220,110));
            ui->i3->move(QPoint(260,110));
            ui->v0->show();
            ui->i0->show();
            ui->v2->show();
            ui->i2->show();
            ui->v3->show();
            ui->i3->show();
            break;
    }
    ui->status_0->show();
    ui->status_1->show();
    ui->status_2->show();
    ui->status_3->show();
    fscale_factor = '0';
    emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
}
void MainWindow::on_doubleSpinBox_valueChanged(double value)
{
    ui->value_slider->setValue(value);
    emit sendComponentValue(value,scale_factor);
}
void MainWindow::on_fdoubleSpinBox_valueChanged(double value)

```

```

{
    ui->fvalue_slider->setValue(value);
    emit sendfComponentValue(value, fscale_factor);
}
void MainWindow::on_v0_stateChanged(int state)
{
    sendMeasureStatus(0, state);
}
void MainWindow::on_v1_stateChanged(int state)
{
    sendMeasureStatus(1, state);
}
void MainWindow::on_v2_stateChanged(int state)
{
    sendMeasureStatus(2, state);
}
void MainWindow::on_v3_stateChanged(int state)
{
    sendMeasureStatus(3, state);
}
void MainWindow::on_i0_stateChanged(int state)
{
    sendMeasureStatus(4, state);
}
void MainWindow::on_i1_stateChanged(int state)
{
    sendMeasureStatus(5, state);
}
void MainWindow::on_i2_stateChanged(int state)
{
    sendMeasureStatus(6, state);
}
void MainWindow::on_i3_stateChanged(int state)
{
    sendMeasureStatus(7, state);
}
// Toggle the debug message box
void MainWindow::on_debug_button_clicked()
{
    if(ui->debug_box->isHidden())
        ui->debug_box->show();
    else
        ui->debug_box->hide();
}
void MainWindow::on_scale_button_clicked()
{
    switch(scale_factor)
    {
        case ' ':
            scale_factor = 'k';
            break;
        case 'k':
            scale_factor = 'M';
            break;
        case 'M':
            scale_factor = 'p';
            break;
        case 'p':
            scale_factor = 'n';
            break;
        case 'n':
            scale_factor = 'u';
            break;
        case 'u':
            scale_factor = 'm';
            break;
        case 'm':
            scale_factor = ' ';
            break;
    }
}

```



```

    sprintf(buffer, "%c", scale_factor);
    ui->scale_button->setText(buffer);
    emit sendComponentValue(ui->doubleSpinBox->value(), scale_factor);
}
void MainWindow::on_fscale_button_clicked()
{
    switch(fscale_factor)
    {
        case ' ':
            fscale_factor = 'k';
            break;
        case 'k':
            fscale_factor = 'M';
            break;
        case 'M':
            fscale_factor = ' ';
            break;
    }
    sprintf(buffer, "%c", fscale_factor);
    ui->fscale_button->setText(buffer);
    emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
}
// Toggle DC or AC source
void MainWindow::on_dc_sine_button_clicked()
{
    if(acdc == 'd')
    {
        ui->doubleSpinBox->move(QPoint(10,155));
        ui->scale_button->move(QPoint(94,155));
        ui->value_label->move(QPoint(118,155));
        ui->fdoubleSpinBox->move(QPoint(171,155));
        ui->fscale_button->move(QPoint(255,155));
        ui->fvalue_label->move(QPoint(280,155));
        ui->value_slider->move(QPoint(10,190));
        ui->fvalue_slider->move(QPoint(10,210));
        ui->dc_sine_button->setIcon(QIcon(QPixmap(":/images/images/vdc_symbol.png")));
        ui->vsource_label->setPixmap(QPixmap(":/images/images/vsource3.png"));
        ui->fvalue_slider->setValue(10);
        fscale_factor = 'k';
        emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
        acdc = 'a';
        ui->fdoubleSpinBox->show();
        ui->fvalue_label->show();
        ui->fvalue_slider->show();
        ui->fscale_button->show();
    }
    else
    {
        ui->fdoubleSpinBox->hide();
        ui->fvalue_label->hide();
        ui->fvalue_slider->hide();
        ui->fscale_button->hide();
        ui->value_slider->move(QPoint(10,210));
        ui->doubleSpinBox->move(QPoint(91,155));
        ui->scale_button->move(QPoint(175,155));
        ui->value_label->move(QPoint(200,155));
        ui->dc_sine_button->setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
        ui->vsource_label->setPixmap(QPixmap(":/images/images/vsource2.png"));
        fscale_factor = '0';
        emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
        acdc = 'd';
    }
}
}

```

Appendix F: Host Source Code

Communication.h

```
/* Kevin Peters
 * This is the header file for the communication based functions
 * used on the host computer for the touchscreen real time spice
 * simulator.
 *
 * May 2012
 */
#ifndef COMMUNICATION_H
#define COMMUNICATION_H
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include "globals.h"
#define MAX_ATTEMPTS 3
using namespace std;
extern ofstream rfcomms_out[10];
extern ifstream rfcomms_in[10];
extern char inputNeighborA[10][5];
extern char inputNeighborB[10][5];
extern char outputNeighborA[10][5];
extern char outputNeighborB[10][5];
extern int measureVIn[2][10];
extern int measureVOut[2][10];
extern int measureIIn[2][10];
extern int measureIOut[2][10];
extern int component[10];
extern char componentValue[10][15];
extern int updateDetected;
void btConnect();
void btdisconnect();
void detectNeighbors();
void updateNeighbors(int);
void getPartName(int);
void getPartValue(int);
void getMeasureSettings(int);
void checkIfValueChanged(int,int);
string readInput(int);
#endif // COMMUNICATION_H
```

Communication.cpp

```
/* Kevin Peters
 * This is the file with the communication based functions
 * used on the host computer for the touchscreen real time spice
 * simulator.
 *
 * May 2012
 */
#include "communication.h"
#include <fstream>
#include <string>
using namespace std;
extern int deviceConnected[10];
extern int simulating;
// Used to open new connections to the touchscreen nodes
void btConnect()
{
```

```

char buffer[50] = {0};
char input[5] = {0};
int attempts;
// Bind rfcomms to device MAC Addresses
system("rfcomm bind 0 00:07:80:4D:25:47"); // WT12-A
system("rfcomm bind 1 00:07:80:4D:22:FC"); // WT12-B
system("rfcomm bind 2 00:07:80:4D:25:48"); // WT12-C
system("rfcomm bind 3 00:07:80:4D:22:F8"); // WT12-D
system("rfcomm bind 4 00:07:80:4D:22:FA"); // WT12-E
system("rfcomm bind 5 00:07:80:45:F0:7D"); // WT12-F
system("rfcomm bind 6 00:07:80:4D:22:F6"); // WT12-G
system("rfcomm bind 7 00:07:80:4D:22:F9"); // WT12-H
system("rfcomm bind 8 00:07:80:45:F1:55"); // WT12-I
system("rfcomm bind 9 00:07:80:4D:25:4D"); // WT12-J
qDebug("Opening Connection");
char comm_channel[15];
// Attempt to open the output channel on each rfcomm
for(int i=0; i < NUM_DEVICES; i++)
{
    // Only attempt to connect devices that aren't already
    // connected
    if(!deviceConnected[i])
    {
        // Close old connections
        rfcomms_out[i].close();
        rfcomms_in[i].close();
        attempts = 0;
        sprintf(comm_channel, "/dev/rfcomm%d", i);
        qDebug(comm_channel);
        rfcomms_out[i].open(comm_channel);
        // If the connection failed to open, try again
        if(!rfcomms_out[i].is_open())
        {
            while(!rfcomms_out[i].is_open() && attempts < MAX_ATTEMPTS)
            {
                qDebug("RFCOMM OUT DID NOT OPEN FOR: %s", comm_channel);
                rfcomms_out[i].open(comm_channel);
                attempts++;
            }
        }
    }
}
// Allow for Bluetooth devices to discover capabilities
sleep(5);
// Open input connections
for(int i = 0; i < NUM_DEVICES; i++)
{
    // Only attempt if the output connection succeeded
    if(rfcomms_out[i].is_open())
    {
        sprintf(comm_channel, "/dev/rfcomm%d", i);
        if(!deviceConnected[i])
        {
            rfcomms_in[i].open(comm_channel);
            // Retry if the connection fails
            if(!rfcomms_in[i].is_open())
            {
                attempts = 0;
                while(!rfcomms_in[i].is_open() && attempts < MAX_ATTEMPTS)
                {
                    qDebug("RFCOMM IN DID NOT OPEN FOR: %s", comm_channel);
                    rfcomms_in[i].open(comm_channel);
                    attempts++;
                }
            }
        }
        qDebug("Sending g and Char");
        sprintf(buffer, "g%c", 'A' + i);
        rfcomms_out[i].write(buffer, strlen(buffer));
        rfcomms_out[i].flush();
    }
}

```

```

        sprintf(input,"%s",readInput(i).c_str());
        if(input[0] != 0)
        {
            qDebug("Successful connection to node: %c", input[0]);
            deviceConnected[i] = 1;
        }
        else
        {
            qDebug("Connection not established to node: %c",'A'+i);
            deviceConnected[i] = 0;
        }
    }
    else
    {
        qDebug("Connection not established to node: %c",'A' + i);
        deviceConnected[i] = 0;
    }
}

// Used to disconnect the connected nodes
void btdisconnect()
{
    char releaseCommand[20];
    // Close all output and input channels then clear bindings
    for(int i = 0; i < 10; i++)
    {
        sprintf(releaseCommand,"rfcomm release %d", i);
        rfcomms_out[i].close();
        rfcomms_in[i].close();
        system(releaseCommand);
        qDebug(releaseCommand);
        deviceConnected[i] = 0;
    }
}

// Called to go to all nodes and gather the neighbor information
void detectNeighbors()
{
    char input[255] = {0};
    for(int i=0; i < NUM_DEVICES; i++)
    {
        // Only check if the device is connected
        if(deviceConnected[i])
        {
            // h is the command to get neighbor information
            rfcomms_out[i].write("h",1);
            rfcomms_out[i].flush();
            memset(input,0,255);
            sprintf(input,"%s", (readInput(i).c_str()));
            qDebug("Input read as: %s for node %c",input,i+'A');
            // Store the returned information
            inputNeighborA[i][0] = input[0];
            inputNeighborA[i][1] = input[1];
            inputNeighborB[i][0] = input[2];
            inputNeighborB[i][1] = input[3];
            outputNeighborA[i][0] = input[4];
            outputNeighborA[i][1] = input[5];
            outputNeighborB[i][0] = input[6];
            outputNeighborB[i][1] = input[7];
        }
    }
    return;
}

// Gets the part name from the node on connection i
void getPartName(int i)
{
    char part_name[15] = {0};
    // i is the command to get part name
    rfcomms_out[i].write("i",1);
    rfcomms_out[i].flush();
    sprintf(part_name,"%s",readInput(i).c_str());
}

```

```

qDebug("Partname is %s",part_name);
// Compare the returned name with the supported components
// and set to the value of the defined part name. Check
// the previously stored name to see if the part changed.
if(!strcmp(part_name, "resistor"))
{
    checkIfValueChanged(component[i],RESISTOR);
    component[i] = RESISTOR;
    return;
} else if(!strcmp(part_name, "capacitor"))
{
    checkIfValueChanged(component[i],CAPACITOR);
    component[i] = CAPACITOR;
    return;
} else if(!strcmp(part_name, "inductor"))
{
    checkIfValueChanged(component[i],INDUCTOR);
    component[i] = INDUCTOR;
    return;
} else if(!strcmp(part_name, "diode"))
{
    checkIfValueChanged(component[i],DIODE);
    component[i] = DIODE;
    return;
} else if(!strcmp(part_name, "nmos"))
{
    checkIfValueChanged(component[i],NMOS);
    component[i] = NMOS;
    return;
} else if(!strcmp(part_name, "bjt"))
{
    checkIfValueChanged(component[i],BJT);
    component[i] = BJT;
    return;
} else if(!strcmp(part_name, "opamp"))
{
    checkIfValueChanged(component[i],OPAMP);
    component[i] = OPAMP;
    return;
} else if(!strcmp(part_name, "vsource"))
{
    checkIfValueChanged(component[i],VSOURCE);
    component[i] = VSOURCE;
    return;
} else if(!strcmp(part_name, "isource"))
{
    checkIfValueChanged(component[i],ISOURCE);
    component[i] = ISOURCE;
    return;
} else
{
    qDebug("NOT A COMPONENT!");
}
}
// Gets the part value from the node on connection i
void getPartValue(int i)
{
    char part_value[25] = {0};
    // j is the command to receive part value
    rfcomms_out[i].write("j",1);
    rfcomms_out[i].flush();
    sprintf(part_value,"%s",readInput(i).c_str());
    qDebug("part value is %s",part_value);
    // Check for an update in the value
    if(strcmp(part_value,componentValue[i]))
    {
        updateDetected = 1;
    }
    sprintf(componentValue[i],"%s",part_value);
    qDebug(componentValue[i]);
}

```

```

}
// Gets the measurement information from the node on connection i
void getMeasureSettings(int i)
{
    char input[10] = {0};
    // Send a 'k' to initiate Measurement query
    rfcomms_out[i].write("k",1);
    rfcomms_out[i].flush();
    sprintf(input,"%s",readInput(i).c_str());
    qDebug("Read in Measurement settings: ");
    // Convert the char to an int and check for changes
    checkIfValueChanged(measureVIN[0][i],input[0] - '0');
    checkIfValueChanged(measureVIN[1][i],input[1] - '0');
    checkIfValueChanged(measureVOut[0][i],input[2] - '0');
    checkIfValueChanged(measureVOut[1][i],input[3] - '0');
    measureVIN[0][i] = input[0] - '0';
    measureVIN[1][i] = input[1] - '0';
    measureVOut[0][i] = input[2] - '0';
    measureVOut[1][i] = input[3] - '0';
    checkIfValueChanged(measureIIn[0][i],input[4] - '0');
    checkIfValueChanged(measureIIn[1][i],input[5] - '0');
    checkIfValueChanged(measureIOut[0][i],input[6] - '0');
    checkIfValueChanged(measureIOut[1][i],input[7] - '0');
    measureIIn[0][i] = input[4] - '0';
    measureIIn[1][i] = input[5] - '0';
    measureIOut[0][i] = input[6] - '0';
    measureIOut[1][i] = input[7] - '0';
    return;
}
// Updates the neighbor information for the node on connection i
// CURRENTLY UNUSED DUE TO INSTABILITY RESULTING IN SIMULATIONS
void updateNeighbors(int i)
{
    char input[10] = {0};
    char temp[10] = {0};
    // Send 'h' to receive neighbor information
    rfcomms_out[i].write("h",1);
    rfcomms_out[i].flush();
    sprintf(input,"%s",readInput(i).c_str());
    inputNeighborA[i][0] = input[0];
    inputNeighborA[i][1] = input[1];
    inputNeighborB[i][0] = input[2];
    inputNeighborB[i][1] = input[3];
    outputNeighborA[i][0] = input[4];
    outputNeighborA[i][1] = input[5];
    outputNeighborB[i][0] = input[6];
    outputNeighborB[i][1] = input[7];
    return;
}
// Used to compare the previously stored value with the
// newly received value so updates can be detected
void checkIfValueChanged(int oldValue, int newValue)
{
    if(oldValue != newValue)
    {
        updateDetected = 1;
        qDebug("UPDATE DETECTED between %d and %d!",oldValue,newValue);
    }
    return;
}
// Reads the input buffer and returns a string of the data
string readInput(int i)
{
    char input[255] = {0};
    int j = 0;
    char temp[2] = {0};
    int timeout = 0;
    string result;
    qDebug("Attempting to read from node %c",i+'A');
    // Read the input until an '!' is received

```

```

while(temp[0] != '!' && timeout < 100000000)
{
    temp[0] = 0;
    rfcomms_in[i].readsome(temp,1);
    if(temp[0] && temp[0] != '!')
    {
        qDebug("%c",temp[0]);
        input[j] = temp[0];
        j++;
        timeout = 0;
    }
    if(temp[0]=='!')
    {
        qDebug("%c",temp[0]);
    }
    timeout++;
}
if(timeout == 100000000)
{
    qDebug("Read timed out");
}
qDebug("Returning %s",input);
result = input;
return result;
}

```

Globals.h

```

/* Kevin Peters
 * This is the header file for the globally defined variables
 * used on the host computer for the touchscreen real time spice
 * simulator.
 *
 * May 2012
 */
#ifndef GLOBALS_H
#define GLOBALS_H
#include <QDebug>
#define NUM_DEVICES 7
#define NONE -1
#define RESISTOR 0
#define CAPACITOR 1
#define INDUCTOR 2
#define DIODE 3
#define NMOS 4
#define BJT 5
#define OPAMP 6
#define VSOURCE 7
#define ISOURCE 8
#endif // GLOBALS_H

```

Main.cpp

```
#include <QtGui/QApplication>
#include "mainwindow.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

MainWindow.h

```
/* Kevin Peters
 * This is the header file for the main window functions
 * used on the host computer for the touchscreen real time spice
 * simulator.
 *
 * May 2012
 */
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include "communication.h"
#include "simulate_and_plot.h"
#include "globals.h"
namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_btConnect_clicked();
    void on_btDisconnect_clicked();
    void on_simulate_clicked();
    void on_stopSim_clicked();
private:
    Ui::MainWindow *ui;
    void updateSimulation();
    void toggleEnabled(bool);
};
#endif // MAINWINDOW_H
```


Mainwindow.cpp

```
/* Kevin Peters
 * This is the file with the functions related to the user interface
 * used on the host computer for the touchscreen real time spice
 * simulator.
 *
 * May 2012
 */
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <fcntl.h>
#include <iostream>
#include <fstream>
using namespace std;
// Define global variables used in all files
ofstream rfcomms_out[10];
ifstream rfcomms_in[10];
int deviceConnected[10] = {0};
int component[10];
char componentValue[10][15] = {{0}};
char inputNeighborA[10][5];
char inputNeighborB[10][5];
char outputNeighborA[10][5];
char outputNeighborB[10][5];
int measureVIN[2][10] = {{0}};
int measureVOut[2][10] = {{0}};
int measureIIn[2][10] = {{0}};
int measureIOut[2][10] = {{0}};
int updateDetected = 0;
int nodeVoltageMeasured[10] = {0};
int nodeCurrentMeasured[10] = {0};
int numOfVMeasured;
int numOfIMeasured;
int maxNodeValue = 0;
int dummyInVoltageNodeValue[10] = {0};
int maxInDummyNodeCount = 0;
int dummyOutVoltageNodeValue[10] = {0};
int maxOutDummyNodeCount = 0;
int inputNode[2][10];
int outputNode[2][10];
int simulating = 1;
// Constructor
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    btdisconnect();
}
// Exit function
MainWindow::~MainWindow()
{
    // Disconnect all nodes before closing
    btdisconnect();
    delete ui;
}
// When Connect is clicked, call the connect function
void MainWindow::on_btConnect_clicked()
{
    btConnect();
}
```

```

// When Disconnect is clicked, call the disconnect function
void MainWindow::on_btDisconnect_clicked()
{
    btdisconnect();
}
// When Simulate is clicked run the simulation loop
void MainWindow::on_simulate_clicked()
{
    // Delete the old plot to prevent displaying an old image
    remove("/root/plot.jpg");
    // Disable all buttons except 'stop'
    toggleEnabled(false);
    // Enable simulating and initial update
    simulating = 1;
    updateDetected = 1;
    qDebug("Go for Neighbor Detection");
    // Detect neighbor information
    detectNeighbors();
    qDebug("Go for Update Loop");
    // Enter simulation loop
    updateSimulation();
    return;
}
// Used to update the simulation information until stopped
void MainWindow::updateSimulation()
{
    while(simulating)
    {
        for(int i = 0; i < NUM_DEVICES;i++)
        {
            if(deviceConnected[i])
            {
                qDebug("Go for Part Name");
                // Get part name
                getPartName(i);
                // If a legitimate component...
                if(component[i] != NONE)
                {
                    qDebug("Go for Part value");
                    // Get part value
                    getPartValue(i);
                    qDebug("Go for measures");
                    // Get measure settings
                    getMeasureSettings(i);
                    qDebug("Go for updated neighbors");
                    // UNUSED DUE TO INSTABILITY
                    //updateNeighbors(i);
                }
            }
        }
        // Check to see if 'stop' has been clicked before simulating
        QApplication::processEvents();
        // Exit if stopped
        if(!simulating)
        {
            break;
        }
        // Only simulate when a measurement option is selected,
        // and update detected
        if(updateDetected && measureBoxChecked())
        {
            qDebug("Writing Netlist");
            // Generate the netlist
            formatNetlist();
            qDebug("Run Spice");
            // Execute a SPICE simulation in batch mode
            system("ngspice -b -o /root/results.txt /root/netlist.cir");
            qDebug("format the output file");
            // Format the output so it can be used by gnuplot
            formatSpiceOutput();
        }
    }
}

```

```

        qDebug("GNUPlot stuff");
        // Plot the results
        gnuPlot();
        // Reset the update flag
        updateDetected = 0;
    }
    // Check to see if stop has been clicked
    QApplication::processEvents();
    // Display the image
    ui->label->setPixmap(QPixmap("/root/plot.jpg"));
}
return;
}
// When 'stop' is clicked
void MainWindow::on_stopSim_clicked()
{
    // Set the flag to leave the infinite loop
    // and enable all buttons except 'stop'
    simulating = 0;
    toggleEnabled(true);
    qDebug("STOPPING SIMULATION!");
}
// Function that toggles enabled and disabled buttons
void MainWindow::toggleEnabled(bool logic)
{
    qDebug("Enabling/Disabling GUI buttons");
    ui->stopSim->setEnabled(!logic);
    ui->simulate->setEnabled(logic);
    ui->btConnect->setEnabled(logic);
    ui->btDisconnect->setEnabled(logic);
    update();
    QApplication::processEvents();
}
}

```

Simulate_and_plot.h

```

/* Kevin Peters
 * This is the header file for the SPICE and plotting based functions
 * used on the host computer for the touchscreen real time spice
 * simulator.
 *
 * May 2012
 */
#ifndef SIMULATE_AND_PLOT_H
#define SIMULATE_AND_PLOT_H
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include "globals.h"
using namespace std;
extern int nodeVoltageMeasured[10];
extern int nodeCurrentMeasured[10];
extern int numOfVMeasured;
extern int numOfIMeasured;
extern int maxNodeValue;
extern int dummyInVoltageNodeValue[10];
extern int maxInDummyNodeCount;
extern int dummyOutVoltageNodeValue[10];
extern int maxOutDummyNodeCount;
extern int inputNode[2][10];
extern int outputNode[2][10];
extern char inputNeighborA[10][5];
extern char inputNeighborB[10][5];
extern char outputNeighborA[10][5];

```

```

extern char outputNeighborB[10][5];
extern int measureVIn[2][10];
extern int measureVOut[2][10];
extern int measureIIn[2][10];
extern int measureIOut[2][10];
extern int component[10];
extern char componentValue[10][15];
extern int deviceConnected[10];
void formatSpiceOutput();
void gnuPlot();
string determineFile(int);
int voltageAndCurrentPlots();
void formatNetlist();
int tiedOutput(int);
int tiedInput(int);
void assignNodeValues();
void buildWires(char *, int);
int hasNeighbors(int);
int nodeVoltageAlreadyMeasured(int);
int nodeCurrentAlreadyMeasured(int);
void adjustForCurrentMeasurements();
int measureBoxChecked();
#endif // SIMULATE_AND_PLOT_H

```

Simulate_and_plot.cpp

```

/* Kevin Peters
 * This is the file with the functions related to the simulating
 * and plotting SPICE netlists used on the host computer for the
 * touchscreen real time spice simulator.
 *
 * May 2012
 */
#include "simulate_and_plot.h"
#include <fstream>
#include <string>
// Used to create the wire numbers used in a spice netlist
void assignNodeValues()
{
    int i, j = 0;
    char temp[5] = {0};
    int nodeCount = 1;
    // Initiate the node values to -1
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
        {
            inputNode[j][i] = -1;
            outputNode[j][i] = -1;
        }
    }
    // Create Node 0 to start numbering all other devices.
    // Assign it to the first connected Voltage Source ground
    for(i = 0; i < NUM_DEVICES; i++)
    {
        if(deviceConnected[i] && component[i] == VSOURCE && hasNeighbors(i))
        {
            // Establish ground
            inputNode[0][i] = 0;
            temp[0] = i + 'A';
            temp[1] = '0';
            // Build the wiremap for this value
            buildWires(temp, inputNode[0][i]);
            // Ground on sources are tied
            inputNode[1][i] = inputNode[0][i];
        }
    }
}

```

```

        temp[0] = i + 'A';
        temp[1] = '1';
        buildWires(temp, inputNode[1][i]);
        break;
    }
}
// Assign Output Node Values
for(i = 0; i < NUM_DEVICES; i++)
{
    if(deviceConnected[i] && component[i] != NONE && hasNeighbors(i))
    {
        for(j = 0; j < 2; j++)
        {
            // Check to see if the node is unassigned
            if(outputNode[j][i] == -1)
            {
                // Assign the next value and build wire map
                outputNode[j][i] = nodeCount;
                temp[0] = i + 'A';
                temp[1] = '2'+j;
                buildWires(temp, outputNode[j][i]);
                // If the outputs are tied, assign and build wire
                if(tiedOutput(i))
                {
                    qDebug("Outputs are tied for node %c", 'A'+i);
                    outputNode[j+1][i] = outputNode[j][i];
                    temp[0] = i + 'A';
                    temp[1] = '3';
                    buildWires(temp, outputNode[j+1][i]);
                    nodeCount++;
                    break;
                }
                nodeCount++;
            }
            else
            {
                // If already assigned, build wire map to value
                temp[0] = i + 'A';
                temp[1] = '2'+j;
                buildWires(temp, outputNode[j][i]);
                if(tiedOutput(i))
                {
                    temp[0] = i + 'A';
                    temp[1] = '3';
                    buildWires(temp, outputNode[j+1][i]);
                    break;
                }
            }
        }
    }
}
// Assign Input Node Values, same as above
for(i = 0; i < NUM_DEVICES; i++)
{
    if(deviceConnected[i] && component[i] != NONE && hasNeighbors(i))
    {
        for(j = 0; j < 2; j++)
        {
            if(inputNode[j][i] == -1)
            {
                inputNode[j][i] = nodeCount;
                temp[0] = i + 'A';
                temp[1] = '0'+j;
                buildWires(temp, inputNode[j][i]);
                if(tiedInput(i))
                {
                    inputNode[j+1][i] = inputNode[j][i];
                    temp[0] = i + 'A';
                    temp[1] = '1';
                    buildWires(temp, inputNode[j+1][i]);
                }
            }
        }
    }
}

```

```

        nodeCount++;
        break;
    }
    nodeCount++;
}
else
{
    temp[0] = i + 'A';
    temp[1] = '0' + j;
    buildWires(temp, inputNode[j][i]);
    if(tiedInput(i))
    {
        temp[0] = i + 'A';
        temp[1] = '1';
        buildWires(temp, inputNode[j+1][i]);
        break;
    }
}
}
}
}
maxNodeValue = nodeCount;
// Create spacing in wires for dummy nodes used in current
// measurements
adjustForCurrentMeasurements();
}
// Checks all nodes to see if they match the
// neighbor code that just had its value assigned
void buildWires(char * match, int nodeValue)
{
    for(int i = 0; i < NUM_DEVICES; i++)
    {
        // Compare neighbor information to node connection
        // being set. Assign other connection if tied
        if(!strcmp(inputNeighborA[i], match))
        {
            inputNode[0][i] = nodeValue;
            if(tiedInput(i))
            {
                inputNode[1][i] = nodeValue;
            }
        }
        if(!strcmp(inputNeighborB[i], match))
        {
            inputNode[1][i] = nodeValue;
            if(tiedInput(i))
            {
                inputNode[0][i] = nodeValue;
            }
        }
        if(!strcmp(outputNeighborA[i], match))
        {
            outputNode[0][i] = nodeValue;
            if(tiedOutput(i))
            {
                outputNode[1][i] = nodeValue;
            }
        }
        if(!strcmp(outputNeighborB[i], match))
        {
            outputNode[1][i] = nodeValue;
            if(tiedOutput(i))
            {
                outputNode[0][i] = nodeValue;
            }
        }
    }
    return;
}
// Use the information collected to build the netlist

```

```

void formatNetlist()
{
    // delete the old file
    remove("/root/netlist.cir");
    ofstream spiceFile("/root/netlist.cir");
    char buffer [50] = {0};
    assignNodeValues();
    int opamp = 0;
    int i, j;
    int dummyInCount = 0;
    int dummyOutCount = 0;
    memset(nodeVoltageMeasured,0,10);
    memset(nodeCurrentMeasured,0,10);
    sprintf(buffer,"*Generated Netlist\n\n");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
    for(i = 0; i < NUM_DEVICES; i++)
    {
        memset(buffer, 0, 50);
        if(deviceConnected[i] && component[i] != NONE && hasNeighbors(i))
        {
            // Determine component and insert its netlist format
            switch(component[i])
            {
                case 0:
                    sprintf(buffer,"R%d %d %d %s\n", i, inputNode[0][i], outputNode[0][i],
                                componentValue[i]);
                    qDebug("Component value sent to netlist %s",componentValue[i]);
                    break;
                case 1:
                    sprintf(buffer,"C%d %d %d %s\n", i, inputNode[0][i], outputNode[0][i],
                                componentValue[i]);
                    break;
                case 2:
                    sprintf(buffer,"L%d %d %d %s\n", i, inputNode[0][i], outputNode[0][i],
                                componentValue[i]);
                    break;
                case 3:
                    sprintf(buffer,"D%d %d %d \n", i, inputNode[0][i], outputNode[0][i]);
                    break;
                case 4:
                    sprintf(buffer,"M%d %d %d %d nmos\n", i, outputNode[0][i],
                                inputNode[0][i], outputNode[1][i]);
                    break;
                case 5:
                    sprintf(buffer,"Q%d %d %d %d npn\n", i, outputNode[0][i],
                                inputNode[0][i], outputNode[1][i]);
                    break;
                case 6:
                    sprintf(buffer,"X%d %d %d %d opamp741\n", i, inputNode[0][i],
                                inputNode[1][i], outputNode[1][i]);
                    opamp = 1;
                    break;
                case 7:
                    // If true, it is an AC source, else it is DC
                    if(strlen(componentValue[i]) > 6)
                    {
                        sprintf(buffer,"V%d %d %d sin(0 %s 0 0 0)\n", i, outputNode[0][i],
                                    inputNode[0][i],componentValue[i]);
                    }
                    else
                    {
                        sprintf(buffer,"V%d %d %d %s\n", i, outputNode[0][i],
                                    inputNode[0][i],componentValue[i]);
                    }
                    break;
                case 8:
                    sprintf(buffer,"I%d %d %d DC %s\n", i, outputNode[0][i],
                                inputNode[0][i],componentValue[i]);
                    break;
            }
        }
    }
}

```

```

    }
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
}
}
// Create the LM741 Subcircuit
if(opamp)
{
    memset(buffer, 0, 50);
    sprintf(buffer, ".subckt opamp741 1 2 3\nrin 1 2 2meg\nrout 6 3 75\n");
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
    memset(buffer, 0, 50);
    sprintf(buffer, "e 4 0 1 2 100k\nrbw 4 5 0.5meg\nncbw 5 0 31.85nf\n");
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
    memset(buffer, 0, 50);
    sprintf(buffer, "eout 6 0 5 0 1\n.ends opamp741\n");
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
}
// Build dummy voltage sources to measure current
for(j = 0; j < 2; j++)
{
    for(i = 0; i < NUM_DEVICES; i++)
    {
        if(measureIIn[j][i] && component[i] != VSOURCE)
        {
            memset(buffer, 0, 50);
            sprintf(buffer, "V%c %d %d DC 0\n", 'a'+dummyInCount+dummyOutCount,
                dummyInVoltageNodeValue[dummyInCount], inputNode[j][i]);
            spiceFile.write(buffer, strlen(buffer));
            spiceFile.flush();
            dummyInCount++;
        }
        if(measureIOut[j][i] && component[i] != VSOURCE)
        {
            memset(buffer, 0, 50);
            sprintf(buffer, "V%c %d %d DC 0\n", 'a'+dummyInCount+dummyOutCount,
                outputNode[j][i], dummyOutVoltageNodeValue[dummyOutCount]);
            spiceFile.write(buffer, strlen(buffer));
            spiceFile.flush();
            dummyOutCount++;
        }
    }
}
memset(buffer, 0, 50);
sprintf(buffer, ".control\ntran 10n 100u\nprint ");
spiceFile.write(buffer, strlen(buffer));
spiceFile.flush();
numOfVMeasured = 0;
numOfIMeasured = 0;
dummyInCount = 0;
dummyOutCount = 0;
// Insert measurement statements, but limited to 7 total
for(j = 0; j < 2; j++)
{
    for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured + numOfIMeasured); i++)
    {
        if(measureVIN[j][i] && !nodeVoltageAlreadyMeasured(inputNode[j][i]))
        {
            memset(buffer, 0, 50);
            sprintf(buffer, "V(%d) ", inputNode[j][i]);
            spiceFile.write(buffer, strlen(buffer));
            spiceFile.flush();
            nodeVoltageMeasured[numOfVMeasured] = inputNode[j][i];
            numOfVMeasured++;
        }
    }
}
}

```



```

// Insert measurement statements, but limited to 7 total
for(j = 0; j < 2; j++)
{
    for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured + numOfIMeasured); i++)
    {
        if(measureVOut[j][i] && !nodeVoltageAlreadyMeasured(outputNode[j][i]))
        {
            memset(buffer,0,50);
            sprintf(buffer,"V(%d) ",outputNode[j][i]);
            spiceFile.write(buffer,strlen(buffer));
            spiceFile.flush();
            nodeVoltageMeasured[numOfVMeasured] = outputNode[j][i];
            numOfVMeasured++;
        }
    }
}
// Insert measurement statements, but limited to 7 total
for(j = 0; j < 2; j++)
{
    for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured + numOfIMeasured); i++)
    {
        if(measureIIn[j][i] && !nodeCurrentAlreadyMeasured(inputNode[j][i]))
        {
            if(component[i] == VSOURCE)
            {
                memset(buffer,0,50);
                sprintf(buffer,"I(V%d) ",i);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeCurrentMeasured[numOfIMeasured] = inputNode[j][i];
                numOfIMeasured++;
            }
            else
            {
                memset(buffer,0,50);
                sprintf(buffer,"I(V%c) ",'a'+dummyInCount+dummyOutCount);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeCurrentMeasured[numOfIMeasured] = inputNode[j][i];
                numOfIMeasured++;
                dummyInCount++;
            }
        }
    }
}
// Insert measurement statements, but limited to 7 total
for(j = 0; j < 2; j++)
{
    for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured + numOfIMeasured); i++)
    {
        if(measureIOut[j][i] && !nodeCurrentAlreadyMeasured(outputNode[j][i]))
        {
            if(component[i] == VSOURCE)
            {
                memset(buffer,0,50);
                sprintf(buffer,"I(V%d) ",i);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeCurrentMeasured[numOfIMeasured] = outputNode[j][i];
                numOfIMeasured++;
            }
            else
            {
                memset(buffer,0,50);
                sprintf(buffer,"I(V%c) ",'a'+dummyInCount+dummyOutCount);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeCurrentMeasured[numOfIMeasured] = outputNode[j][i];
                numOfIMeasured++;
                dummyOutCount++;
            }
        }
    }
}

```

```

    }
    }
}
memset(buffer,0,50);
sprintf(buffer, "\n.endc\n.end");
spiceFile.write(buffer,strlen(buffer));
spiceFile.flush();
return;
}
// The output from spice needs to be formatted
// before gnuplot can use it.
void formatSpiceOutput()
{
    remove("/root/formattedPart1.txt");
    remove("/root/formattedPart2.txt");
    remove("/root/formattedPart3.txt");
    ifstream inFile("/root/results.txt");
    ofstream outFile1("/root/formattedPart1.txt");
    ofstream outFile2("/root/formattedPart2.txt");
    ofstream outFile3("/root/formattedPart3.txt");
    string line;
    int indexFound = 0;
    int i = -2;
    int j = 1;
    int goodLine = 0;
    char indexCount[10];
    while(inFile.good())
    {
        // Read the next line and add a newline to the end of it
        getline(inFile,line);
        line.append("\n");
        // If the word index hasn't been found yet, look for it
        if(!indexFound)
        {
            if(line.find("Index") == 0)
            {
                indexFound = 1;
            }
        }
        // If the line count is increasing, still good
        if(i >= 0)
        {
            sprintf(indexCount,"%d",i);
            if(line.find(indexCount) == 0)
            {
                goodLine = 1;
                i++;
            }
            else
            {
                // If it resets to 0, its a new measurement setting
                if(line.find("0") == 0)
                {
                    i = 1;
                    goodLine = 1;
                    j++;
                }
            }
        }
        else
        {
            if(indexFound)
            {
                i++;
            }
        }
        if (i >= 0 && goodLine)
        {
            switch(j)

```



```

        sprintf(buffer, "\n\n");
        script.write(buffer, strlen(buffer));
    }
}
if (column > 5)
{
    column = 3;
}
}
}
// Voltage Outputs
for(j = 0; j < 2; j++)
{
    for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
    {
        if(measureVOut[j][i] && !nodeVoltageAlreadyMeasured(outputNode[j][i]))
        {
            fileName = determineFile(k);
            sprintf(buffer, "%s\" using 2:%d with linespoints axis xly1 title 'V(%c)
                        out'", fileName.c_str(), column, i+'A');
            script.write(buffer, strlen(buffer));
            column++;
            k++;
            nodeVoltageMeasured[voltageCount] = outputNode[j][i];
            voltageCount++;
            if(voltageCount + currentCount < numOfVMeasured + numOfIMeasured)
            {
                sprintf(buffer, "\n\n");
                script.write(buffer, strlen(buffer));
            }
        }
        if (column > 5)
        {
            column = 3;
        }
    }
}
// Current Inputs
for(j = 0; j < 2; j++)
{
    for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
    {
        if(measureIIn[j][i] && !nodeCurrentAlreadyMeasured(inputNode[j][i]))
        {
            fileName = determineFile(k);
            sprintf(buffer, "%s\" using 2:%d with linespoints axis xly2 title 'I(%c)
                        in'", fileName.c_str(), column, i+'A');
            script.write(buffer, strlen(buffer));
            column++;
            k++;
            nodeCurrentMeasured[currentCount] = inputNode[j][i];
            currentCount++;
            if(voltageCount + currentCount < numOfVMeasured + numOfIMeasured)
            {
                sprintf(buffer, "\n\n");
                script.write(buffer, strlen(buffer));
            }
        }
        if (column > 5)
        {
            column = 3;
        }
    }
}
}
// Current Outputs
for(j = 0; j < 2; j++)
{
    for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
    {
        if(measureIOut[j][i] && !nodeCurrentAlreadyMeasured(outputNode[j][i]))

```

```

        {
            fileName = determineFile(k);
            sprintf(buffer, "\\\"%s\\\" using 2:%d with linespoints axis xly2 title 'I(%c)
                        out'", fileName.c_str(), column, i+'A');
            script.write(buffer, strlen(buffer));
            column++;
            k++;
            nodeCurrentMeasured[currentCount] = outputNode[j][i];
            currentCount++;
            if(voltageCount + currentCount < numOfVMeasured + numOfIMeasured)
            {
                sprintf(buffer, ", \\\"\\n");
                script.write(buffer, strlen(buffer));
            }
        }
        if (column > 5)
        {
            column = 3;
        }
    }
}
script.close();
system("gnuplot /root/gnuscrypt");
return;
}
// Determine which file the measurement values are
// stored in and return the file name in a string
string determineFile(int i)
{
    string fileName = "";
    switch(i)
    {
        case 0:
        case 1:
        case 2:
            fileName = "/root/formattedPart1.txt";
            break;
        case 3:
        case 4:
        case 5:
            fileName = "/root/formattedPart2.txt";
            break;
        case 6:
        case 7:
        case 8:
            fileName = "/root/formattedPart3.txt";
            break;
    }
    return fileName;
}
// Check to see if both voltage and current is measured
int voltageAndCurrentPlots()
{
    int i, j = 0;
    int voltage, current = 0;
    for(i=0; i < 2; i++)
    {
        for(j=0; j<10; j++)
        {
            if(measureVIn[i][j] || measureVOut[i][j])
            {
                voltage = 1;
                break;
            }
        }
        if(voltage)
        {
            break;
        }
    }
}

```

```

    for(i=0; i < 2; i++)
    {
        for(j=0; j<10; j++)
        {
            if(measureIIn[i][j] || measureIOut[i][j])
            {
                current = 1;
                break;
            }
        }
        if(current)
        {
            break;
        }
    }
    return (current & voltage);
}
// If the outputs are tied together, return 1
int tiedOutput(int i)
{
    switch(component[i])
    {
        case 0:
        case 1:
        case 2:
        case 3:
        case 6:
        case 7:
        case 8:
            return 1;
            break;
        case 4:
        case 5:
            return 0;
            break;
    }
    return 0;
}
// If the inputs are tied together, return 1
int tiedInput(int i)
{
    switch(component[i])
    {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 7:
        case 8:
            return 1;
            break;
        case 6:
            return 0;
            break;
    }
    return 0;
}
// Find out if a node is connected to other nodes (not isolated)
int hasNeighbors(int i)
{
    if(inputNeighborA[i][0] == 'N' && inputNeighborB[i][0] == 'N' &&
        outputNeighborA[i][0] == 'N' && outputNeighborB[i][0] == 'N')
    {
        return 0;
    }
    else
    {
        return 1;
    }
}

```

```

    }
}
// Check if a node current has already been measured, to not
// repeat measurements
int nodeCurrentAlreadyMeasured(int nodeNumber)
{
    qDebug("Node values that have been measured for Current:");
    for(int i = 0; i < 10; i++)
    {
        qDebug("%d", nodeCurrentMeasured[i]);
        if(nodeNumber == nodeCurrentMeasured[i])
        {
            return 1;
        }
    }
    return 0;
}
// Check if a node voltage has already been measured, to not
// repeat measurements
int nodeVoltageAlreadyMeasured(int nodeNumber)
{
    qDebug("Node values that have been measured for Voltage:");
    for(int i = 0; i < 10; i++)
    {
        qDebug("%d", nodeVoltageMeasured[i]);
        if(nodeNumber == nodeVoltageMeasured[i])
        {
            return 1;
        }
    }
    return 0;
}
// Adjust node values for dummy voltage nodes
void adjustForCurrentMeasurements()
{
    int i, j;
    maxInDummyNodeCount = 0;
    maxOutDummyNodeCount = 0;
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
        {
            if(measureIIn[j][i])
            {
                if(component[i] != VSOURCE)
                {
                    dummyInVoltageNodeValue[maxInDummyNodeCount] = inputNode[j][i];
                    inputNode[j][i] = maxNodeValue;
                    maxInDummyNodeCount++;
                    maxNodeValue++;
                }
            }
        }
    }
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
        {
            if(measureIOut[j][i])
            {
                if(component[i] != VSOURCE)
                {
                    dummyOutVoltageNodeValue[maxOutDummyNodeCount] = outputNode[j][i];
                    outputNode[j][i] = maxNodeValue;
                    maxOutDummyNodeCount++;
                    maxNodeValue++;
                }
            }
        }
    }
}

```

```

}
// Check for a measurement option
int measureBoxChecked()
{
    int i;
    for(i = 0; i < NUM_DEVICES; i++)
    {
        if(measureVIn[0][i] || measureVIn[1][i])
        {
            return 1;
        }
        if(measureVOut[0][i] || measureVOut[1][i])
        {
            return 1;
        }
        if(measureIIn[0][i] || measureIIn[1][i])
        {
            return 1;
        }
        if(measureIOut[0][i] || measureIOut[1][i])
        {
            return 1;
        }
    }
    return 0;
}

```


Appendix G: ReActive-SPICE v1.0 Source Code

rs1/mainwindow.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v1.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**   notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**   notice, this list of conditions and the following disclaimer in
**   the documentation and/or other materials provided with the
**   distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**   the names of its contributors may be used to endorse or promote
**   products derived from this software without specific prior written
**   permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QMenu>
#include <QMenuBar>
#include <QMessageBox>
#include <QStatusBar>
#include <QMouseEvent>
#include <QPainter>
#include <QGraphicsView>
#include <QGraphicsLineItem>
#include "communication.h"
#include "simulate_and_plot.h"
#include "globals.h"
namespace Ui {
class MainWindow;

```

```

}
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget* parent=0);
    ~MainWindow();

private slots:
    void on_simulate_clicked();
    void on_stopSim_clicked();
    void help();
    // My slots
    void on_IO_0_clicked();
    void on_IO_1_clicked();
    void on_IO_2_clicked();
    void on_IO_3_clicked();
    void hideControls(int);
    void showControls(int, char);
    void on_value_slider_valueChanged(int value);
    void on_doubleSpinBox_valueChanged(double value);
    void on_fvalue_slider_valueChanged(int value);
    void on_fdoubleSpinBox_valueChanged(double value);
    void on_sac_resistor_clicked();
    void on_sac_capacitor_clicked();
    void on_sac_inductor_clicked();
    void on_sac_diode_clicked();
    void on_sac_opamp_clicked();
    void on_sac_bjt_clicked();
    void on_sac_nmos_clicked();
    void on_sac_vsource_clicked();
    void on_sac_isource_clicked();
    void on_res_back_button_clicked();
    void on_cap_back_button_clicked();
    void on_inductor_back_button_clicked();
    void on_diode_back_button_clicked();
    void on_opamp_back_button_clicked();
    void on_bjt_back_button_clicked();
    void on_nmos_back_button_clicked();
    void on_isource_back_button_clicked();
    void on_vsource_back_button_clicked();
    void on_v0_stateChanged(int arg1);
    void on_v1_stateChanged(int arg1);
    void on_v2_stateChanged(int arg1);
    void on_v3_stateChanged(int arg1);
    void on_i0_stateChanged(int arg1);
    void on_i1_stateChanged(int arg1);
    void on_i2_stateChanged(int arg1);
    void on_i3_stateChanged(int arg1);
    void on_scale_button_clicked();
    void on_fscale_button_clicked();
    void on_dc_sine_button_clicked();
    void on_IO_0_2_clicked();
    void on_IO_1_2_clicked();
    void on_IO_2_2_clicked();
    void on_IO_3_2_clicked();
    void on_value_slider_2_valueChanged(int value);
    void on_doubleSpinBox_2_valueChanged(double value);
    void on_fvalue_slider_2_valueChanged(int value);
    void on_fdoubleSpinBox_2_valueChanged(double value);
    void on_sac_resistor_2_clicked();
    void on_sac_capacitor_2_clicked();
    void on_sac_inductor_2_clicked();
    void on_sac_diode_2_clicked();
    void on_sac_opamp_2_clicked();
    void on_sac_bjt_2_clicked();
    void on_sac_nmos_2_clicked();
    void on_sac_vsource_2_clicked();
    void on_sac_isource_2_clicked();

```

```

void on_res_back_button_2_clicked();
void on_cap_back_button_2_clicked();
void on_inductor_back_button_2_clicked();
void on_diode_back_button_2_clicked();
void on_opamp_back_button_2_clicked();
void on_bjt_back_button_2_clicked();
void on_nmos_back_button_2_clicked();
void on_isource_back_button_2_clicked();
void on_vsource_back_button_2_clicked();
void on_scale_button_2_clicked();
void on_fscale_button_2_clicked();
void on_dc_sine_button_2_clicked();
void on_IO_0_3_clicked();
void on_IO_1_3_clicked();
void on_IO_2_3_clicked();
void on_IO_3_3_clicked();
void on_value_slider_3_valueChanged(int value);
void on_doubleSpinBox_3_valueChanged(double value);
void on_fvalue_slider_3_valueChanged(int value);
void on_fdoubleSpinBox_3_valueChanged(double value);
void on_sac_resistor_3_clicked();
void on_sac_capacitor_3_clicked();
void on_sac_inductor_3_clicked();
void on_sac_diode_3_clicked();
void on_sac_opamp_3_clicked();
void on_sac_bjt_3_clicked();
void on_sac_nmos_3_clicked();
void on_sac_vsource_3_clicked();
void on_sac_isource_3_clicked();
void on_res_back_button_3_clicked();
void on_cap_back_button_3_clicked();
void on_inductor_back_button_3_clicked();
void on_diode_back_button_3_clicked();
void on_opamp_back_button_3_clicked();
void on_bjt_back_button_3_clicked();
void on_nmos_back_button_3_clicked();
void on_isource_back_button_3_clicked();
void on_vsource_back_button_3_clicked();
void on_scale_button_3_clicked();
void on_fscale_button_3_clicked();
void on_dc_sine_button_3_clicked();
void on_IO_0_4_clicked();
void on_IO_1_4_clicked();
void on_IO_2_4_clicked();
void on_IO_3_4_clicked();
void on_value_slider_4_valueChanged(int value);
void on_doubleSpinBox_4_valueChanged(double value);
void on_fvalue_slider_4_valueChanged(int value);
void on_fdoubleSpinBox_4_valueChanged(double value);
void on_sac_resistor_4_clicked();
void on_sac_capacitor_4_clicked();
void on_sac_inductor_4_clicked();
void on_sac_diode_4_clicked();
void on_sac_opamp_4_clicked();
void on_sac_bjt_4_clicked();
void on_sac_nmos_4_clicked();
void on_sac_vsource_4_clicked();
void on_sac_isource_4_clicked();
void on_res_back_button_4_clicked();
void on_cap_back_button_4_clicked();
void on_inductor_back_button_4_clicked();
void on_diode_back_button_4_clicked();
void on_opamp_back_button_4_clicked();
void on_bjt_back_button_4_clicked();
void on_nmos_back_button_4_clicked();
void on_isource_back_button_4_clicked();
void on_vsource_back_button_4_clicked();
void on_scale_button_4_clicked();
void on_fscale_button_4_clicked();
void on_dc_sine_button_4_clicked();

```

```

    void formatSpiceOutput();
    void gnuPlot();
    string determineFile(int);
    int voltageAndCurrentPlots();
    void formatNetlist();
    int nodeVoltageAlreadyMeasured(int);
    int nodeCurrentAlreadyMeasured(int);
    void adjustForCurrentMeasurements();
    int measureBoxChecked();
    void on_reset_nets_clicked();
protected:
    void paintEvent(QPaintEvent *event);
private:
    Ui::MainWindow *ui;
    char scale_factor;
    char scale_factor_2;
    char scale_factor_3;
    char scale_factor_4;
    char fscale_factor;
    char fscale_factor_2;
    char fscale_factor_3;
    char fscale_factor_4;
    char acdc;
    char acdc_2;
    char acdc_3;
    char acdc_4;
    char buffer[255];
    int node_x_offset[5];
    int node_y_offset[5];
    int start_line_x;
    int start_line_y;
    int end_line_x;
    int end_line_y;
    bool eventFilter(QObject *obj, QEvent *event);
    void updateSimulation();
    void toggleEnabled(bool);
};
#endif // MAINWINDOW_H

```

rs1/communication.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v1.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the

```

```

**      distribution.
**      * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**        the names of its contributors may be used to endorse or promote
**        products derived from this software without specific prior written
**        permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#ifndef COMMUNICATION_H
#define COMMUNICATION_H
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include "globals.h"
#define MAX_ATTEMPTS 3
using namespace std;
extern ofstream rfcomms_out[10];
extern ifstream rfcomms_in[10];
extern char inputNeighborA[10][5];
extern char inputNeighborB[10][5];
extern char outputNeighborA[10][5];
extern char outputNeighborB[10][5];
extern int  measureVIn[2][10];
extern int  measureVOut[2][10];
extern int  measureIIn[2][10];
extern int  measureIOut[2][10];
extern int  component[10];
extern char componentValue[10][15];
extern int  updateDetected;
void checkIfValueChanged(int,int);
#endif // COMMUNICATION_H

```

rs1/globals.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v1.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without

```

```

** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**     the names of its contributors may be used to endorse or promote
**     products derived from this software without specific prior written
**     permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
***** /

#ifndef GLOBALS_H
#define GLOBALS_H
#include <QDebug>
#define NUM_DEVICES 4
#define NONE -1
#define RESISTOR 0
#define CAPACITOR 1
#define INDUCTOR 2
#define DIODE 3
#define NMOS 4
#define BJT 5
#define OPAMP 6
#define VSOURCE 7
#define ISOURCE 8
#endif // GLOBALS_H

```

rs1/simulate_and_plot.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v1.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:

```

```

**      * Redistributions of source code must retain the above copyright
**      notice, this list of conditions and the following disclaimer.
**      * Redistributions in binary form must reproduce the above copyright
**      notice, this list of conditions and the following disclaimer in
**      the documentation and/or other materials provided with the
**      distribution.
**      * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**      the names of its contributors may be used to endorse or promote
**      products derived from this software without specific prior written
**      permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#ifndef SIMULATE_AND_PLOT_H
#define SIMULATE_AND_PLOT_H
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include "globals.h"
using namespace std;
extern int  nodeVoltageMeasured[10];
extern int  nodeCurrentMeasured[10];
extern int  numOfVMeasured;
extern int  numOfIMeasured;
extern int  maxNodeValue;
extern int  dummyInVoltageNodeValue[10];
extern int  maxInDummyNodeCount;
extern int  dummyOutVoltageNodeValue[10];
extern int  maxOutDummyNodeCount;
extern int  inputNode[2][10];
extern int  outputNode[2][10];
extern char inputNeighborA[10][5];
extern char inputNeighborB[10][5];
extern char outputNeighborA[10][5];
extern char outputNeighborB[10][5];
extern int  measureVIn[2][10];
extern int  measureVOut[2][10];
extern int  measureIIn[2][10];
extern int  measureIOut[2][10];
extern int  component[10];
extern char componentValue[10][15];
extern int  deviceConnected[10];
void formatSpiceOutput();
void gnuPlot();
string determineFile(int);
int voltageAndCurrentPlots();
void formatNetlist();

```

```

int tiedOutput(int);
int tiedInput(int);
int hasNeighbors(int);
int nodeVoltageAlreadyMeasured(int);
int nodeCurrentAlreadyMeasured(int);
void adjustForCurrentMeasurements();
int measureBoxChecked();
#endif // SIMULATE_AND_PLOT_H

```

rs1/main.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v1.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**     the names of its contributors may be used to endorse or promote
**     products derived from this software without specific prior written
**     permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#include <QtGui/QApplication>
#include "mainwindow.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;

```



```

        w.show();

        return a.exec();
    }

```

rs1/mainwindow.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v1.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**     the names of its contributors may be used to endorse or promote
**     products derived from this software without specific prior written
**     permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <signal.h>
#include <fcntl.h>
#include <iostream>
#include <fstream>
using namespace std;
ofstream rfcomms_out[10];

```

```

ifstream rfcomms_in[10];
int deviceConnected[10] = {0};
int component[10];
char componentValue[10][15] = {{0}};
char inputNeighborA[10][5];
char inputNeighborB[10][5];
char outputNeighborA[10][5];
char outputNeighborB[10][5];
int measureVIn[2][10] = {{0}};
int measureVOut[2][10] = {{0}};
int measureIIn[2][10] = {{0}};
int measureIOut[2][10] = {{0}};
int updateDetected = 0;
int nodeVoltageMeasured[10] = {0};
int nodeCurrentMeasured[10] = {0};
int numOfVMeasured;
int numOfIMeasured;
int maxNodeValue = 0;
int dummyInVoltageNodeValue[10] = {0};
int maxInDummyNodeCount = 0;
int dummyOutVoltageNodeValue[10] = {0};
int maxOutDummyNodeCount = 0;
int inputNode[2][10];
int outputNode[2][10];
int simulating = 1;
int IO_0_net = -1;
int IO_1_net = -1;
int IO_2_net = -1;
int IO_3_net = -1;
int IO_0_net_2 = -1;
int IO_1_net_2 = -1;
int IO_2_net_2 = -1;
int IO_3_net_2 = -1;
int IO_0_net_3 = -1;
int IO_1_net_3 = -1;
int IO_2_net_3 = -1;
int IO_3_net_3 = -1;
int IO_0_net_4 = -1;
int IO_1_net_4 = -1;
int IO_2_net_4 = -1;
int IO_3_net_4 = -1;
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    statusBar();
    //menuBar();
    qApp->installEventFilter(this);
    QMenuBar* menu = menuBar();
    QMenu* file = new QMenu("&File", menu );
    file->addSeparator();
    file->addAction("E&xit", qApp, SLOT(quit()), Qt::CTRL+Qt::Key_Q);
    menu->addMenu(file);
    QMenu* edit = new QMenu("&Edit", menu );
    edit->addAction("Add &Text", this, SLOT(help()), Qt::ALT+Qt::Key_T);
    edit->addAction("Add &Line", this, SLOT(help()), Qt::ALT+Qt::Key_L);
    menu->addMenu(edit);
    QMenu* view = new QMenu("&View", menu );
    view->addAction("&Enlarge", this, SLOT(help()), Qt::SHIFT+Qt::CTRL+Qt::Key_Plus);
    view->addAction("Shr&ink", this, SLOT(help()), Qt::SHIFT+Qt::CTRL+Qt::Key_Minus);
    view->addSeparator();
    view->addAction("&Rotate clockwise", this, SLOT(help()), Qt::CTRL+Qt::Key_PageDown);
    view->addAction("Rotate &counterclockwise", this, SLOT(help()),
Qt::CTRL+Qt::Key_PageUp);
    view->addAction("&Zoom in", this, SLOT(help()), Qt::CTRL+Qt::Key_Plus);
    view->addAction("Zoom &out", this, SLOT(help()), Qt::CTRL+Qt::Key_Minus);
    view->addAction("Translate left", this, SLOT(help()), Qt::CTRL+Qt::Key_Left);
    view->addAction("Translate right", this, SLOT(help()), Qt::CTRL+Qt::Key_Right);
    view->addAction("Translate up", this, SLOT(help()), Qt::CTRL+Qt::Key_Up);
    view->addAction("Translate down", this, SLOT(help()), Qt::CTRL+Qt::Key_Down);

```

```

view->addAction("&Mirror", this, SLOT(help()), Qt::CTRL+Qt::Key_Home);
menu->addMenu(view);
menu->addSeparator();
QMenu* help = new QMenu("&Help", menu );
help->addAction("&About", this, SLOT(help()), Qt::Key_F1);
menu->addMenu(help);
ui->setupUi(this);
setWindowTitle("ISpice - Real-Time Interactive SPICE Simulator v1.0");
// Set the base coordinates
node_x_offset[1] = ui->stackedWidget->geometry().x();
node_y_offset[1] = ui->stackedWidget->geometry().y();
node_x_offset[2] = ui->stackedWidget_2->geometry().x();
node_y_offset[2] = ui->stackedWidget_2->geometry().y();
node_x_offset[3] = ui->stackedWidget_3->geometry().x();
node_y_offset[3] = ui->stackedWidget_3->geometry().y();
node_x_offset[4] = ui->stackedWidget_4->geometry().x();
node_y_offset[4] = ui->stackedWidget_4->geometry().y();
// Set the current index to 0 and hide the controls overlay
ui->IO_0->move(node_x_offset[1]-20,node_y_offset[1]+60);
ui->IO_1->move(node_x_offset[1]-20,node_y_offset[1]+100);
ui->IO_2->move(node_x_offset[1]+320,node_y_offset[1]+40);
ui->IO_3->move(node_x_offset[1]+320,node_y_offset[1]+120);
ui->IO_0_2->move(node_x_offset[2]-20,node_y_offset[2]+60);
ui->IO_1_2->move(node_x_offset[2]-20,node_y_offset[2]+100);
ui->IO_2_2->move(node_x_offset[2]+320,node_y_offset[2]+40);
ui->IO_3_2->move(node_x_offset[2]+320,node_y_offset[2]+120);
ui->IO_0_3->move(node_x_offset[3]-20,node_y_offset[3]+60);
ui->IO_1_3->move(node_x_offset[3]-20,node_y_offset[3]+100);
ui->IO_2_3->move(node_x_offset[3]+320,node_y_offset[3]+40);
ui->IO_3_3->move(node_x_offset[3]+320,node_y_offset[3]+120);
ui->IO_0_4->move(node_x_offset[4]-20,node_y_offset[4]+60);
ui->IO_1_4->move(node_x_offset[4]-20,node_y_offset[4]+100);
ui->IO_2_4->move(node_x_offset[4]+320,node_y_offset[4]+40);
ui->IO_3_4->move(node_x_offset[4]+320,node_y_offset[4]+120);
ui->IO_0->setText("N");
ui->IO_1->setText("N");
ui->IO_2->setText("N");
ui->IO_3->setText("N");
ui->IO_0_2->setText("N");
ui->IO_1_2->setText("N");
ui->IO_2_2->setText("N");
ui->IO_3_2->setText("N");
ui->IO_0_3->setText("N");
ui->IO_1_3->setText("N");
ui->IO_2_3->setText("N");
ui->IO_3_3->setText("N");
ui->IO_0_4->setText("N");
ui->IO_1_4->setText("N");
ui->IO_2_4->setText("N");
ui->IO_3_4->setText("N");
ui->stackedWidget->setCurrentIndex(0);
ui->stackedWidget_2->setCurrentIndex(0);
ui->stackedWidget_3->setCurrentIndex(0);
ui->stackedWidget_4->setCurrentIndex(0);
hideControls(1);
hideControls(2);
hideControls(3);
hideControls(4);
update();
}
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::on_simulate_clicked()
{
    remove("/root/plot.jpg");
    toggleEnabled(false);
    simulating = 1;
    updateDetected = 1;
}

```

```

        updateSimulation();
        return;
    }
    void MainWindow::updateSimulation()
    {
        while(simulating)
        {
            qDebug("Go for Part Name");
            component[0] = ui->stackedWidget->currentIndex();
            component[1] = ui->stackedWidget_2->currentIndex();
            component[2] = ui->stackedWidget_3->currentIndex();
            component[3] = ui->stackedWidget_4->currentIndex();
            for(int i = 0; i < 4; i++)
            {
                if(component[i] != 0)
                {
                    qDebug("Go for Part value");
                    char part_value[25] = {0};
                    char input[10] = {0};
                    switch(i)
                    {
                        case 0:
                            input[0] = ui->v0->checkState();
                            input[1] = ui->v1->checkState();
                            input[2] = ui->v2->checkState();
                            input[3] = ui->v3->checkState();
                            input[4] = ui->i0->checkState();
                            input[5] = ui->i1->checkState();
                            input[6] = ui->i2->checkState();
                            input[7] = ui->i3->checkState();
                            if(fscale_factor == '0')
                            {
                                if(scale_factor == 'M')
                                    sprintf(part_value, "%3.1fmeg", ui->doubleSpinBox-
>value());
                                else
                                    sprintf(part_value, "%3.1f%c", ui->doubleSpinBox-
>value(), scale_factor);
                            }
                            else
                            {
                                if(fscale_factor == 'M')
                                {
                                    if(scale_factor == 'M')
                                        sprintf(part_value, "%3.1fmeg %3.1fmeg", ui-
>doubleSpinBox->value(), ui->fdoubleSpinBox->value());
                                    else
                                        sprintf(part_value, "%3.1f%c %3.1fmeg", ui-
>doubleSpinBox->value(), scale_factor, ui->fdoubleSpinBox->value());
                                }
                                else
                                {
                                    if(scale_factor == 'M')
                                        sprintf(part_value, "%3.1fmeg %3.1f%c", ui-
>doubleSpinBox->value(), ui->fdoubleSpinBox->value(), fscale_factor);
                                    else
                                        sprintf(part_value, "%3.1f%c %3.1f%c", ui-
>doubleSpinBox->value(), scale_factor, ui->fdoubleSpinBox->value(), fscale_factor);
                                }
                            }
                            break;
                        case 1:
                            input[0] = ui->v0_2->checkState();
                            input[1] = ui->v1_2->checkState();
                            input[2] = ui->v2_2->checkState();
                            input[3] = ui->v3_2->checkState();
                            input[4] = ui->i0_2->checkState();
                            input[5] = ui->i1_2->checkState();
                            input[6] = ui->i2_2->checkState();
                            input[7] = ui->i3_2->checkState();

```

```

        if(fscale_factor_2 == '0')
        {
            if(scale_factor_2 == 'M')
                sprintf(part_value, "%3.1fmeg", ui->doubleSpinBox_2-
>value());
            else
                sprintf(part_value, "%3.1f%c", ui->doubleSpinBox_2-
>value(), scale_factor_2);
        }
        else
        {
            if(fscale_factor_2 == 'M')
            {
                if(scale_factor_2 == 'M')
                    sprintf(part_value, "%3.1fmeg %3.1fmeg", ui-
>doubleSpinBox_2->value(), ui->fdoubleSpinBox_2->value());
                else
                    sprintf(part_value, "%3.1f%c %3.1fmeg", ui-
>doubleSpinBox_2->value(), scale_factor_2, ui->fdoubleSpinBox_2->value());
            }
            else
            {
                if(scale_factor_2 == 'M')
                    sprintf(part_value, "%3.1fmeg %3.1f%c", ui-
>doubleSpinBox_2->value(), ui->fdoubleSpinBox_2->value(), fscale_factor_2);
                else
                    sprintf(part_value, "%3.1f%c %3.1f%c", ui-
>doubleSpinBox_2->value(), scale_factor_2, ui->fdoubleSpinBox_2->value(), fscale_factor_2);
            }
        }
        break;
    case 2:
        input[0] = ui->v0_3->checkState();
        input[1] = ui->v1_3->checkState();
        input[2] = ui->v2_3->checkState();
        input[3] = ui->v3_3->checkState();
        input[4] = ui->i0_3->checkState();
        input[5] = ui->i1_3->checkState();
        input[6] = ui->i2_3->checkState();
        input[7] = ui->i3_3->checkState();
        if(fscale_factor_3 == '0')
        {
            if(scale_factor_3 == 'M')
                sprintf(part_value, "%3.1fmeg", ui->doubleSpinBox_3-
>value());
            else
                sprintf(part_value, "%3.1f%c", ui->doubleSpinBox_3-
>value(), scale_factor_3);
        }
        else
        {
            if(fscale_factor_3 == 'M')
            {
                if(scale_factor_3 == 'M')
                    sprintf(part_value, "%3.1fmeg %3.1fmeg", ui-
>doubleSpinBox_3->value(), ui->fdoubleSpinBox_3->value());
                else
                    sprintf(part_value, "%3.1f%c %3.1fmeg", ui-
>doubleSpinBox_3->value(), scale_factor_3, ui->fdoubleSpinBox_3->value());
            }
            else
            {
                if(scale_factor_3 == 'M')
                    sprintf(part_value, "%3.1fmeg %3.1f%c", ui-
>doubleSpinBox_3->value(), ui->fdoubleSpinBox_3->value(), fscale_factor_3);
                else
                    sprintf(part_value, "%3.1f%c %3.1f%c", ui-
>doubleSpinBox_3->value(), scale_factor_3, ui->fdoubleSpinBox_3->value(), fscale_factor_3);
            }
        }
    }
}

```

```

        break;
    case 3:
        input[0] = ui->v0_4->checkState();
        input[1] = ui->v1_4->checkState();
        input[2] = ui->v2_4->checkState();
        input[3] = ui->v3_4->checkState();
        input[4] = ui->i0_4->checkState();
        input[5] = ui->i1_4->checkState();
        input[6] = ui->i2_4->checkState();
        input[7] = ui->i3_4->checkState();
        if(fscale_factor_4 == '0')
        {
            if(scale_factor_4 == 'M')
                sprintf(part_value, "%3.1fmeg", ui->doubleSpinBox_4-
>value());
            else
                sprintf(part_value, "%3.1f%c", ui->doubleSpinBox_4-
>value(), scale_factor_4);
        }
        else
        {
            if(fscale_factor_4 == 'M')
            {
                if(scale_factor_4 == 'M')
                    sprintf(part_value, "%3.1fmeg %3.1fmeg", ui-
>doubleSpinBox_4->value(), ui->fdoubleSpinBox_4->value());
                else
                    sprintf(part_value, "%3.1f%c %3.1fmeg", ui-
>doubleSpinBox_4->value(), scale_factor_4, ui->fdoubleSpinBox_4->value());
            }
            else
            {
                if(scale_factor_4 == 'M')
                    sprintf(part_value, "%3.1fmeg %3.1f%c", ui-
>doubleSpinBox_4->value(), ui->fdoubleSpinBox_4->value(), fscale_factor_4);
                else
                    sprintf(part_value, "%3.1f%c %3.1f%c", ui-
>doubleSpinBox_4->value(), scale_factor_4, ui->fdoubleSpinBox_4->value(), fscale_factor_4);
            }
        }
        break;
    }
    if(strcmp(part_value, componentValue[i]))
    {
        updateDetected = 1;
    }
    sprintf(componentValue[i], "%s", part_value);
    //qDebug("Go for measures");
    //qDebug("Read in Measurement settings: ");
    // Convert the char to an int
    checkIfValueChanged(measureVIN[0][i], input[0]);
    checkIfValueChanged(measureVIN[1][i], input[1]);
    checkIfValueChanged(measureVOut[0][i], input[2]);
    checkIfValueChanged(measureVOut[1][i], input[3]);
    measureVIN[0][i] = input[0];
    measureVIN[1][i] = input[1];
    measureVOut[0][i] = input[2];
    measureVOut[1][i] = input[3];
    checkIfValueChanged(measureIIn[0][i], input[4]);
    checkIfValueChanged(measureIIn[1][i], input[5]);
    checkIfValueChanged(measureIOut[0][i], input[6]);
    checkIfValueChanged(measureIOut[1][i], input[7]);
    measureIIn[0][i] = input[4];
    measureIIn[1][i] = input[5];
    measureIOut[0][i] = input[6];
    measureIOut[1][i] = input[7];
}
//qDebug("First process events");
QCoreApplication::processEvents();

```

```

        if(!simulating)
        {
            break;
        }
    }
    if(updateDetected && measureBoxChecked())
    {
        //qDebug("Writing Netlist");
        formatNetlist();
        //qDebug("Run Spice");
        system("ngspice -b -o /root/results.txt /root/netlist.cir");
        //qDebug("Format the output file");
        formatSpiceOutput();
        //qDebug("GNUPlot stuff");
        gnuPlot();
        updateDetected = 0;
    }
    //qDebug("Second process events");
    QCoreApplication::processEvents();
    ui->label->setPixmap(QPixmap("/root/plot.jpg"));
}
return;
}
void MainWindow::on_stopSim_clicked()
{
    simulating = 0;
    toggleEnabled(true);
    qDebug("STOPPING SIMULATION!");
}
void MainWindow::toggleEnabled(bool logic)
{
    qDebug("Enabling/Disabling GUI buttons");
    ui->stopSim->setEnabled(!logic);
    ui->simulate->setEnabled(logic);
    update();
    QCoreApplication::processEvents();
}
void MainWindow::help()
{
    static QMessageBox* about = new QMessageBox( "ISpice",
        "<h3>ISpice - Real-time Interactive SPICE Simulator</h3>"
        "<ul>"
        "<li> Unfortunately none of these menus work yet..."
        "<li> Sorry."
        "</ul>", QMessageBox::Information, 1, 0, 0, this, 0);
    about->setButtonText( 1, "Dismiss" );
    about->show();
    qDebug("HELP!");
}
int button_pressed = 0;
bool MainWindow::eventFilter(QObject *obj, QEvent *event)
{
    if (event->type() == QEvent::MouseMove)
    {
        QMouseEvent *mouseEvent = static_cast<QMouseEvent*>(event);
        statusBar()->showMessage(QString("Mouse move (%1,%2)").arg(mouseEvent->pos().x()).arg(mouseEvent->pos().y()));
        if(button_pressed == 0)
        {
            start_line_x = mouseEvent->pos().x();
            start_line_y = mouseEvent->pos().y();
            button_pressed = 1;
        }
        else {
            end_line_x = mouseEvent->pos().x();
            end_line_y = mouseEvent->pos().y();
            update();
            button_pressed = 0;
        }
    }
    return false;
}

```

```

void MainWindow::paintEvent(QPaintEvent *event)
{
    /*QPainter painter(this);
    painter.setPen(QPen(QColor(Qt::blue), 6, Qt::SolidLine,
Qt::RoundCap,Qt::RoundJoin));
    painter.drawLine( start_line_x , start_line_y , end_line_x, end_line_y ); // 100x100
    ellipse at position (10, 20)*/
}
// My stuff
void MainWindow::on_value_slider_valueChanged(int value)
{
    ui->doubleSpinBox->setValue(value);
}
void MainWindow::on_value_slider_2_valueChanged(int value)
{
    ui->doubleSpinBox_2->setValue(value);
}
void MainWindow::on_value_slider_3_valueChanged(int value)
{
    ui->doubleSpinBox_3->setValue(value);
}
void MainWindow::on_value_slider_4_valueChanged(int value)
{
    ui->doubleSpinBox_4->setValue(value);
}
void MainWindow::on_fvalue_slider_valueChanged(int value)
{
    ui->fdoubleSpinBox->setValue(value);
}
void MainWindow::on_fvalue_slider_2_valueChanged(int value)
{
    ui->fdoubleSpinBox_2->setValue(value);
}
void MainWindow::on_fvalue_slider_3_valueChanged(int value)
{
    ui->fdoubleSpinBox_3->setValue(value);
}
void MainWindow::on_fvalue_slider_4_valueChanged(int value)
{
    ui->fdoubleSpinBox_4->setValue(value);
}
void MainWindow::on_sac_resistor_clicked()
{
    //emit sendComponentName("resistor");
    ui->stackedWidget->setCurrentIndex(1);
    ui->value_label->setText("Ohms");
    showControls(1, 'a');
}
void MainWindow::on_sac_resistor_2_clicked()
{
    //emit sendComponentName("resistor");
    ui->stackedWidget_2->setCurrentIndex(1);
    ui->value_label_2->setText("Ohms");
    showControls(2, 'a');
}
void MainWindow::on_sac_resistor_3_clicked()
{
    //emit sendComponentName("resistor");
    ui->stackedWidget_3->setCurrentIndex(1);
    ui->value_label_3->setText("Ohms");
    showControls(3, 'a');
}
void MainWindow::on_sac_resistor_4_clicked()
{
    //emit sendComponentName("resistor");
    ui->stackedWidget_4->setCurrentIndex(1);
    ui->value_label_4->setText("Ohms");
    showControls(4, 'a');
}
void MainWindow::on_sac_capacitor_clicked()

```



```

{
    //emit sendComponentName("capacitor");
    ui->stackedWidget->setCurrentIndex(2);
    ui->value_label->setText("Farads");
    showControls(1, 'a');
}
void MainWindow::on_sac_capacitor_2_clicked()
{
    //emit sendComponentName("capacitor");
    ui->stackedWidget_2->setCurrentIndex(2);
    ui->value_label_2->setText("Farads");
    showControls(2, 'a');
}
void MainWindow::on_sac_capacitor_3_clicked()
{
    //emit sendComponentName("capacitor");
    ui->stackedWidget_3->setCurrentIndex(2);
    ui->value_label_3->setText("Farads");
    showControls(3, 'a');
}
void MainWindow::on_sac_capacitor_4_clicked()
{
    //emit sendComponentName("capacitor");
    ui->stackedWidget_4->setCurrentIndex(2);
    ui->value_label_4->setText("Farads");
    showControls(4, 'a');
}
void MainWindow::on_sac_inductor_clicked()
{
    //emit sendComponentName("inductor");
    ui->stackedWidget->setCurrentIndex(3);
    ui->value_label->setText("Henrys");
    showControls(1, 'a');
}
void MainWindow::on_sac_inductor_2_clicked()
{
    //emit sendComponentName("inductor");
    ui->stackedWidget_2->setCurrentIndex(3);
    ui->value_label_2->setText("Henrys");
    showControls(2, 'a');
}
void MainWindow::on_sac_inductor_3_clicked()
{
    //emit sendComponentName("inductor");
    ui->stackedWidget_3->setCurrentIndex(3);
    ui->value_label_3->setText("Henrys");
    showControls(3, 'a');
}
void MainWindow::on_sac_inductor_4_clicked()
{
    //emit sendComponentName("inductor");
    ui->stackedWidget_4->setCurrentIndex(3);
    ui->value_label_4->setText("Henrys");
    showControls(4, 'a');
}
void MainWindow::on_sac_diode_clicked()
{
    //emit sendComponentName("diode");
    ui->stackedWidget->setCurrentIndex(4);
    showControls(1, 'm');
}
void MainWindow::on_sac_diode_2_clicked()
{
    //emit sendComponentName("diode");
    ui->stackedWidget_2->setCurrentIndex(4);
    showControls(2, 'm');
}
void MainWindow::on_sac_diode_3_clicked()
{
    //emit sendComponentName("diode");

```

```

        ui->stackedWidget_3->setCurrentIndex(4);
        showControls(3, 'm');
    }
    void MainWindow::on_sac_diode_4_clicked()
    {
        //emit sendComponentName("diode");
        ui->stackedWidget_4->setCurrentIndex(4);
        showControls(4, 'm');
    }
    void MainWindow::on_sac_opamp_clicked()
    {
        //emit sendComponentName("opamp");
        ui->stackedWidget->setCurrentIndex(5);
        showControls(1, 'm');
    }
    void MainWindow::on_sac_opamp_2_clicked()
    {
        //emit sendComponentName("opamp");
        ui->stackedWidget_2->setCurrentIndex(5);
        showControls(2, 'm');
    }
    void MainWindow::on_sac_opamp_3_clicked()
    {
        //emit sendComponentName("opamp");
        ui->stackedWidget_3->setCurrentIndex(5);
        showControls(3, 'm');
    }
    void MainWindow::on_sac_opamp_4_clicked()
    {
        //emit sendComponentName("opamp");
        ui->stackedWidget_4->setCurrentIndex(5);
        showControls(4, 'm');
    }
    void MainWindow::on_sac_bjt_clicked()
    {
        //emit sendComponentName("bjt");
        ui->stackedWidget->setCurrentIndex(6);
        showControls(1, 'm');
    }
    void MainWindow::on_sac_bjt_2_clicked()
    {
        //emit sendComponentName("bjt");
        ui->stackedWidget_2->setCurrentIndex(6);
        showControls(2, 'm');
    }
    void MainWindow::on_sac_bjt_3_clicked()
    {
        //emit sendComponentName("bjt");
        ui->stackedWidget_3->setCurrentIndex(6);
        showControls(3, 'm');
    }
    void MainWindow::on_sac_bjt_4_clicked()
    {
        //emit sendComponentName("bjt");
        ui->stackedWidget_4->setCurrentIndex(6);
        showControls(4, 'm');
    }
    void MainWindow::on_sac_nmos_clicked()
    {
        //emit sendComponentName("nmos");
        ui->stackedWidget->setCurrentIndex(7);
        showControls(1, 'm');
    }
    void MainWindow::on_sac_nmos_2_clicked()
    {
        //emit sendComponentName("nmos");
        ui->stackedWidget_2->setCurrentIndex(7);
        showControls(2, 'm');
    }
    void MainWindow::on_sac_nmos_3_clicked()

```

```

{
    //emit sendComponentName("nmos");
    ui->stackedWidget_3->setCurrentIndex(7);
    showControls(3, 'm');
}
void MainWindow::on_sac_nmos_4_clicked()
{
    //emit sendComponentName("nmos");
    ui->stackedWidget_4->setCurrentIndex(7);
    showControls(4, 'm');
}
void MainWindow::on_sac_vsource_clicked()
{
    //emit sendComponentName("vsource");
    ui->stackedWidget->setCurrentIndex(8);
    ui->dc_sine_button->setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
    ui->vsource_label->setPixmap(QPixmap(":/images/images/vsource2.png"));
    acdc = 'd';
    ui->value_label->setText("Volts");
    ui->fvalue_label->setText("Hz");
    showControls(1, 'a');
}
void MainWindow::on_sac_vsource_2_clicked()
{
    //emit sendComponentName("vsource");
    ui->stackedWidget_2->setCurrentIndex(8);
    ui->dc_sine_button_2->setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
    ui->vsource_label_2->setPixmap(QPixmap(":/images/images/vsource2.png"));
    acdc_2 = 'd';
    ui->value_label_2->setText("Volts");
    ui->fvalue_label_2->setText("Hz");
    showControls(2, 'a');
}
void MainWindow::on_sac_vsource_3_clicked()
{
    //emit sendComponentName("vsource");
    ui->stackedWidget_3->setCurrentIndex(8);
    ui->dc_sine_button_3->setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
    ui->vsource_label_3->setPixmap(QPixmap(":/images/images/vsource2.png"));
    acdc_3 = 'd';
    ui->value_label_3->setText("Volts");
    ui->fvalue_label_3->setText("Hz");
    showControls(3, 'a');
}
void MainWindow::on_sac_vsource_4_clicked()
{
    //emit sendComponentName("vsource");
    ui->stackedWidget_4->setCurrentIndex(8);
    ui->dc_sine_button_4->setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
    ui->vsource_label_4->setPixmap(QPixmap(":/images/images/vsource2.png"));
    acdc_4 = 'd';
    ui->value_label_4->setText("Volts");
    ui->fvalue_label_4->setText("Hz");
    showControls(4, 'a');
}
void MainWindow::on_sac_isource_clicked()
{
    //emit sendComponentName("isource");
    ui->stackedWidget->setCurrentIndex(9);
    ui->value_label->setText("Amps");
    showControls(1, 'a');
}
void MainWindow::on_sac_isource_2_clicked()
{
    //emit sendComponentName("isource");
    ui->stackedWidget_2->setCurrentIndex(9);
    ui->value_label_2->setText("Amps");
    showControls(2, 'a');
}
void MainWindow::on_sac_isource_3_clicked()

```

```

{
    //emit sendComponentName("isource");
    ui->stackedWidget_3->setCurrentIndex(9);
    ui->value_label_3->setText("Amps");
    showControls(3, 'a');
}
void MainWindow::on_sac_isource_4_clicked()
{
    //emit sendComponentName("isource");
    ui->stackedWidget_4->setCurrentIndex(9);
    ui->value_label_4->setText("Amps");
    showControls(4, 'a');
}
void MainWindow::on_res_back_button_clicked()
{
    hideControls(1);
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_res_back_button_2_clicked()
{
    hideControls(2);
    ui->stackedWidget_2->setCurrentIndex(0);
}
void MainWindow::on_res_back_button_3_clicked()
{
    hideControls(3);
    ui->stackedWidget_3->setCurrentIndex(0);
}
void MainWindow::on_res_back_button_4_clicked()
{
    hideControls(4);
    ui->stackedWidget_4->setCurrentIndex(0);
}
void MainWindow::on_cap_back_button_clicked()
{
    hideControls(1);
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_cap_back_button_2_clicked()
{
    hideControls(2);
    ui->stackedWidget_2->setCurrentIndex(0);
}
void MainWindow::on_cap_back_button_3_clicked()
{
    hideControls(3);
    ui->stackedWidget_3->setCurrentIndex(0);
}
void MainWindow::on_cap_back_button_4_clicked()
{
    hideControls(4);
    ui->stackedWidget_4->setCurrentIndex(0);
}
void MainWindow::on_inductor_back_button_clicked()
{
    hideControls(1);
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_inductor_back_button_2_clicked()
{
    hideControls(2);
    ui->stackedWidget_2->setCurrentIndex(0);
}
void MainWindow::on_inductor_back_button_3_clicked()
{
    hideControls(3);
    ui->stackedWidget_3->setCurrentIndex(0);
}
void MainWindow::on_inductor_back_button_4_clicked()
{

```

```

        hideControls(4);
        ui->stackedWidget_4->setCurrentIndex(0);
    }
void MainWindow::on_diode_back_button_clicked()
{
    hideControls(1);
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_diode_back_button_2_clicked()
{
    hideControls(2);
    ui->stackedWidget_2->setCurrentIndex(0);
}
void MainWindow::on_diode_back_button_3_clicked()
{
    hideControls(3);
    ui->stackedWidget_3->setCurrentIndex(0);
}
void MainWindow::on_diode_back_button_4_clicked()
{
    hideControls(4);
    ui->stackedWidget_4->setCurrentIndex(0);
}
void MainWindow::on_opamp_back_button_clicked()
{
    hideControls(1);
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_opamp_back_button_2_clicked()
{
    hideControls(2);
    ui->stackedWidget_2->setCurrentIndex(0);
}
void MainWindow::on_opamp_back_button_3_clicked()
{
    hideControls(3);
    ui->stackedWidget_3->setCurrentIndex(0);
}
void MainWindow::on_opamp_back_button_4_clicked()
{
    hideControls(4);
    ui->stackedWidget_4->setCurrentIndex(0);
}
void MainWindow::on_bjt_back_button_clicked()
{
    hideControls(1);
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_bjt_back_button_2_clicked()
{
    hideControls(2);
    ui->stackedWidget_2->setCurrentIndex(0);
}
void MainWindow::on_bjt_back_button_3_clicked()
{
    hideControls(3);
    ui->stackedWidget_3->setCurrentIndex(0);
}
void MainWindow::on_bjt_back_button_4_clicked()
{
    hideControls(4);
    ui->stackedWidget_4->setCurrentIndex(0);
}
void MainWindow::on_nmos_back_button_clicked()
{
    hideControls(1);
    ui->stackedWidget->setCurrentIndex(0);
}
void MainWindow::on_nmos_back_button_2_clicked()
{

```

```

        hideControls(2);
        ui->stackedWidget_2->setCurrentIndex(0);
    }
    void MainWindow::on_nmos_back_button_3_clicked()
    {
        hideControls(3);
        ui->stackedWidget_3->setCurrentIndex(0);
    }
    void MainWindow::on_nmos_back_button_4_clicked()
    {
        hideControls(4);
        ui->stackedWidget_4->setCurrentIndex(0);
    }
    void MainWindow::on_vsource_back_button_clicked()
    {
        hideControls(1);
        ui->stackedWidget->setCurrentIndex(0);
    }
    void MainWindow::on_vsource_back_button_2_clicked()
    {
        hideControls(2);
        ui->stackedWidget_2->setCurrentIndex(0);
    }
    void MainWindow::on_vsource_back_button_3_clicked()
    {
        hideControls(3);
        ui->stackedWidget_3->setCurrentIndex(0);
    }
    void MainWindow::on_vsource_back_button_4_clicked()
    {
        hideControls(4);
        ui->stackedWidget_4->setCurrentIndex(0);
    }
    void MainWindow::on_isource_back_button_clicked()
    {
        hideControls(1);
        ui->stackedWidget->setCurrentIndex(0);
    }
    void MainWindow::on_isource_back_button_2_clicked()
    {
        hideControls(2);
        ui->stackedWidget_2->setCurrentIndex(0);
    }
    void MainWindow::on_isource_back_button_3_clicked()
    {
        hideControls(3);
        ui->stackedWidget_3->setCurrentIndex(0);
    }
    void MainWindow::on_isource_back_button_4_clicked()
    {
        hideControls(4);
        ui->stackedWidget_4->setCurrentIndex(0);
    }
    void MainWindow::hideControls(int number)
    {
        switch (number)
        {
            case 1 :
                ui->v0->hide();
                ui->v1->hide();
                ui->v2->hide();
                ui->v3->hide();
                ui->i0->hide();
                ui->i1->hide();
                ui->i2->hide();
                ui->i3->hide();
                ui->fdoubleSpinBox->hide();
                ui->fvalue_label->hide();
                ui->fvalue_slider->hide();
                ui->fscale_button->hide();

```

```

        ui->doubleSpinBox->hide();
        ui->value_label->hide();
        ui->value_slider->hide();
        ui->scale_button->hide();
        //emit sendComponentName("none");
        fscale_factor = '0';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
        break;
    case 2 :
        ui->v0_2->hide();
        ui->v1_2->hide();
        ui->v2_2->hide();
        ui->v3_2->hide();
        ui->i0_2->hide();
        ui->i1_2->hide();
        ui->i2_2->hide();
        ui->i3_2->hide();
        ui->fdoubleSpinBox_2->hide();
        ui->fvalue_label_2->hide();
        ui->fvalue_slider_2->hide();
        ui->fscale_button_2->hide();
        ui->doubleSpinBox_2->hide();
        ui->value_label_2->hide();
        ui->value_slider_2->hide();
        ui->scale_button_2->hide();
        //emit sendComponentName("none");
        fscale_factor_2 = '0';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
        break;
    case 3 :
        ui->v0_3->hide();
        ui->v1_3->hide();
        ui->v2_3->hide();
        ui->v3_3->hide();
        ui->i0_3->hide();
        ui->i1_3->hide();
        ui->i2_3->hide();
        ui->i3_3->hide();
        ui->fdoubleSpinBox_3->hide();
        ui->fvalue_label_3->hide();
        ui->fvalue_slider_3->hide();
        ui->fscale_button_3->hide();
        ui->doubleSpinBox_3->hide();
        ui->value_label_3->hide();
        ui->value_slider_3->hide();
        ui->scale_button_3->hide();
        //emit sendComponentName("none");
        fscale_factor_3 = '0';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
        break;
    case 4 :
        ui->v0_4->hide();
        ui->v1_4->hide();
        ui->v2_4->hide();
        ui->v3_4->hide();
        ui->i0_4->hide();
        ui->i1_4->hide();
        ui->i2_4->hide();
        ui->i3_4->hide();
        ui->fdoubleSpinBox_4->hide();
        ui->fvalue_label_4->hide();
        ui->fvalue_slider_4->hide();
        ui->fscale_button_4->hide();
        ui->doubleSpinBox_4->hide();
        ui->value_label_4->hide();
        ui->value_slider_4->hide();
        ui->scale_button_4->hide();
        //emit sendComponentName("none");
        fscale_factor_4 = '0';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);

```

```

        break;
    }
}
void MainWindow::showControls(int number, char options)
{
    switch (number)
    {
        case 1 :
            // Move everything back to their default positions and default checked
states
            ui->v0->move(QPoint(10+node_x_offset[number],70+node_y_offset[number]));
            ui->i0->move(QPoint(10+node_x_offset[number],104+node_y_offset[number]));
            ui->v2->move(QPoint(281+node_x_offset[number],70+node_y_offset[number]));
            ui->i2->move(QPoint(281+node_x_offset[number],104+node_y_offset[number]));
            ui->doubleSpinBox->move(QPoint(91+node_x_offset[number],155+node_y_offset[number]));
            ui->scale_button->move(QPoint(175+node_x_offset[number],155+node_y_offset[number]));
            ui->value_label->move(QPoint(200+node_x_offset[number],155+node_y_offset[number]));
            ui->value_slider->move(QPoint(10+node_x_offset[number],210+node_y_offset[number]));
            ui->v0->setCheckState(Qt::CheckState(0));
            ui->i0->setCheckState(Qt::CheckState(0));
            ui->v1->setCheckState(Qt::CheckState(0));
            ui->i1->setCheckState(Qt::CheckState(0));
            ui->v2->setCheckState(Qt::CheckState(0));
            ui->i2->setCheckState(Qt::CheckState(0));
            ui->v3->setCheckState(Qt::CheckState(0));
            ui->i3->setCheckState(Qt::CheckState(0));
            // Show "a"ll measurement and value controls
            // Resistor, Capacitor, Inductor, Isource, Vsource
            if(options == 'a')
            {
                // Display V/I measurements 0 and 2
                ui->v0->show();
                ui->i0->show();
                ui->v2->show();
                ui->i2->show();
                // Enable controls based off componenet
                switch (ui->stackedWidget->currentIndex())
                {
                    case 1 : // Resistor
                        ui->value_slider->setRange(1,999);
                        ui->doubleSpinBox->setRange(0,999.9);
                        ui->value_slider->setValue(50);
                        scale_factor = 'k';
                        break;
                    case 2 : // Capacitor
                        ui->value_slider->setRange(1,999);
                        ui->doubleSpinBox->setRange(0,999.9);
                        ui->value_slider->setValue(5);
                        scale_factor = 'u';
                        break;
                    case 3 : // Inductor
                        ui->value_slider->setRange(1,999);
                        ui->doubleSpinBox->setRange(0,999.9);
                        ui->value_slider->setValue(3);
                        scale_factor = 'm';
                        break;
                    case 8 : // Voltage Source
                        ui->value_slider->setRange(1,99);
                        ui->doubleSpinBox->setRange(0,99.9);
                        ui->value_slider->setValue(5);
                        scale_factor = ' ';
                        break;
                    case 9 : // Current Source
                        ui->value_slider->setRange(1,999);
                        ui->doubleSpinBox->setRange(0,999.9);
                        ui->value_slider->setValue(80);

```



```

        scale_factor = 'm';
        break;
    default :
        ui->value_slider->setValue(1);
        scale_factor = ' ';
        break;
    }
    sprintf(buffer,"%c",scale_factor);
    ui->scale_button->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
    ui->doubleSpinBox->show();
    ui->value_label->show();
    ui->value_slider->show();
    ui->scale_button->show();
}
// Show only "m" measurement controls
// Diode, Opamp, NMOS, BJT
else if(options == 'm')
{
    // Only show measurements based off component
    switch (ui->stackedWidget->currentIndex())
    {
        case 4 : // Diode
            ui->v0->show();
            ui->i0->show();
            ui->v2->show();
            ui->i2->show();
            break;
        case 5 : // Opamp
            ui->v0->
>move(QPoint(30+node_x_offset[number],50+node_y_offset[number]));
            ui->i0->
>move(QPoint(70+node_x_offset[number],50+node_y_offset[number]));
            ui->v1->
>move(QPoint(30+node_x_offset[number],125+node_y_offset[number]));
            ui->i1->
>move(QPoint(70+node_x_offset[number],125+node_y_offset[number]));
            ui->v0->show();
            ui->i0->show();
            ui->v1->show();
            ui->i1->show();
            ui->v2->show();
            ui->i2->show();
            break;
        case 6 : case 7: // NMOS/BJT
            ui->v2->
>move(QPoint(220+node_x_offset[number],65+node_y_offset[number]));
            ui->i2->
>move(QPoint(260+node_x_offset[number],65+node_y_offset[number]));
            ui->v3->
>move(QPoint(220+node_x_offset[number],110+node_y_offset[number]));
            ui->i3->
>move(QPoint(260+node_x_offset[number],110+node_y_offset[number]));
            ui->v0->show();
            ui->i0->show();
            ui->v2->show();
            ui->i2->show();
            ui->v3->show();
            ui->i3->show();
            break;
    }
}
fscale_factor = '0';
//emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
break;
case 2 :
    // Move everything back to their default positions and default checked
states
    ui->v0_2->move(QPoint(10+node_x_offset[number],70+node_y_offset[number]));
    ui->i0_2->move(QPoint(10+node_x_offset[number],104+node_y_offset[number]));

```

```

        ui->v2_2->move(QPoint(281+node_x_offset[number],70+node_y_offset[number]));
        ui->i2_2->move(QPoint(281+node_x_offset[number],104+node_y_offset[number]));
        ui->doubleSpinBox_2-
>move(QPoint(91+node_x_offset[number],155+node_y_offset[number]));
        ui->scale_button_2-
>move(QPoint(175+node_x_offset[number],155+node_y_offset[number]));
        ui->value_label_2-
>move(QPoint(200+node_x_offset[number],155+node_y_offset[number]));
        ui->value_slider_2-
>move(QPoint(10+node_x_offset[number],210+node_y_offset[number]));
        ui->v0_2->setCheckState(Qt::CheckState(0));
        ui->i0_2->setCheckState(Qt::CheckState(0));
        ui->v1_2->setCheckState(Qt::CheckState(0));
        ui->i1_2->setCheckState(Qt::CheckState(0));
        ui->v2_2->setCheckState(Qt::CheckState(0));
        ui->i2_2->setCheckState(Qt::CheckState(0));
        ui->v3_2->setCheckState(Qt::CheckState(0));
        ui->i3_2->setCheckState(Qt::CheckState(0));
        // Show "a"ll measurement and value controls
        // Resistor, Capacitor, Inductor, Isource, Vsource
        if(options == 'a')
        {
            // Display V/I measurements 0 and 2
            ui->v0_2->show();
            ui->i0_2->show();
            ui->v2_2->show();
            ui->i2_2->show();
            // Enable controls based off componenet
            switch (ui->stackedWidget_2->currentIndex())
            {
                case 1 : // Resistor
                    ui->value_slider_2->setRange(1,999);
                    ui->doubleSpinBox_2->setRange(0,999.9);
                    ui->value_slider_2->setValue(50);
                    scale_factor_2 = 'k';
                    break;
                case 2 : // Capacitor
                    ui->value_slider_2->setRange(1,999);
                    ui->doubleSpinBox_2->setRange(0,999.9);
                    ui->value_slider_2->setValue(5);
                    scale_factor_2 = 'u';
                    break;
                case 3 : // Inductor
                    ui->value_slider_2->setRange(1,999);
                    ui->doubleSpinBox_2->setRange(0,999.9);
                    ui->value_slider_2->setValue(3);
                    scale_factor_2 = 'm';
                    break;
                case 8 : // Voltage Source
                    ui->value_slider_2->setRange(1,99);
                    ui->doubleSpinBox_2->setRange(0,99.9);
                    ui->value_slider_2->setValue(5);
                    scale_factor_2 = ' ';
                    break;
                case 9 : // Current Source
                    ui->value_slider_2->setRange(1,999);
                    ui->doubleSpinBox_2->setRange(0,999.9);
                    ui->value_slider_2->setValue(80);
                    scale_factor_2 = 'm';
                    break;
                default :
                    ui->value_slider_2->setValue(1);
                    scale_factor_2 = ' ';
                    break;
            }
            sprintf(buffer,"%c",scale_factor_2);
            ui->scale_button_2->setText(buffer);
            //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
            ui->doubleSpinBox_2->show();
            ui->value_label_2->show();

```

```

        ui->value_slider_2->show();
        ui->scale_button_2->show();
    }
    // Show only "m" measurement controls
    // Diode, Opamp, NMOS, BJT
    else if(options == 'm')
    {
        // Only show measurements based off component
        switch (ui->stackedWidget_2->currentIndex())
        {
            case 4 : // Diode
                ui->v0_2->show();
                ui->i0_2->show();
                ui->v2_2->show();
                ui->i2_2->show();
                break;
            case 5 : // Opamp
                ui->v0_2-
>move(QPoint(30+node_x_offset[number],50+node_y_offset[number]));
                ui->i0_2-
>move(QPoint(70+node_x_offset[number],50+node_y_offset[number]));
                ui->v1_2-
>move(QPoint(30+node_x_offset[number],125+node_y_offset[number]));
                ui->i1_2-
>move(QPoint(70+node_x_offset[number],125+node_y_offset[number]));
                ui->v0_2->show();
                ui->i0_2->show();
                ui->v1_2->show();
                ui->i1_2->show();
                ui->v2_2->show();
                ui->i2_2->show();
                break;
            case 6 : case 7: // NMOS/BJT
                ui->v2_2-
>move(QPoint(220+node_x_offset[number],65+node_y_offset[number]));
                ui->i2_2-
>move(QPoint(260+node_x_offset[number],65+node_y_offset[number]));
                ui->v3_2-
>move(QPoint(220+node_x_offset[number],110+node_y_offset[number]));
                ui->i3_2-
>move(QPoint(260+node_x_offset[number],110+node_y_offset[number]));
                ui->v0_2->show();
                ui->i0_2->show();
                ui->v2_2->show();
                ui->i2_2->show();
                ui->v3_2->show();
                ui->i3_2->show();
                break;
        }
    }
    fscale_factor_2 = '0';
    //emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
    break;
case 3 :
    // Move everything back to their default positions and default checked
states
    ui->v0_3->move(QPoint(10+node_x_offset[number],70+node_y_offset[number]));
    ui->i0_3->move(QPoint(10+node_x_offset[number],104+node_y_offset[number]));
    ui->v2_3->move(QPoint(281+node_x_offset[number],70+node_y_offset[number]));
    ui->i2_3->move(QPoint(281+node_x_offset[number],104+node_y_offset[number]));
    ui->doubleSpinBox_3-
>move(QPoint(91+node_x_offset[number],155+node_y_offset[number]));
    ui->scale_button_3-
>move(QPoint(175+node_x_offset[number],155+node_y_offset[number]));
    ui->value_label_3-
>move(QPoint(200+node_x_offset[number],155+node_y_offset[number]));
    ui->value_slider_3-
>move(QPoint(10+node_x_offset[number],210+node_y_offset[number]));
    ui->v0_3->setCheckState(Qt::CheckState(0));
    ui->i0_3->setCheckState(Qt::CheckState(0));

```

```

ui->v1_3->setCheckState(Qt::CheckState(0));
ui->i1_3->setCheckState(Qt::CheckState(0));
ui->v2_3->setCheckState(Qt::CheckState(0));
ui->i2_3->setCheckState(Qt::CheckState(0));
ui->v3_3->setCheckState(Qt::CheckState(0));
ui->i3_3->setCheckState(Qt::CheckState(0));
// Show "a"ll measurement and value controls
// Resistor, Capacitor, Inductor, Isource, Vsource
if(options == 'a')
{
    // Display V/I measurements 0 and 2
    ui->v0_3->show();
    ui->i0_3->show();
    ui->v2_3->show();
    ui->i2_3->show();
    // Enable controls based off componenet
    switch (ui->stackedWidget_3->currentIndex())
    {
        case 1 : // Resistor
            ui->value_slider_3->setRange(1,999);
            ui->doubleSpinBox_3->setRange(0,999.9);
            ui->value_slider_3->setValue(50);
            scale_factor_3 = 'k';
            break;
        case 2 : // Capacitor
            ui->value_slider_3->setRange(1,999);
            ui->doubleSpinBox_3->setRange(0,999.9);
            ui->value_slider_3->setValue(5);
            scale_factor_3 = 'u';
            break;
        case 3 : // Inductor
            ui->value_slider_3->setRange(1,999);
            ui->doubleSpinBox_3->setRange(0,999.9);
            ui->value_slider_3->setValue(3);
            scale_factor_3 = 'm';
            break;
        case 8 : // Voltage Source
            ui->value_slider_3->setRange(1,99);
            ui->doubleSpinBox_3->setRange(0,99.9);
            ui->value_slider_3->setValue(5);
            scale_factor_3 = ' ';
            break;
        case 9 : // Current Source
            ui->value_slider_3->setRange(1,999);
            ui->doubleSpinBox_3->setRange(0,999.9);
            ui->value_slider_3->setValue(80);
            scale_factor_3 = 'm';
            break;
        default :
            ui->value_slider_3->setValue(1);
            scale_factor_3 = ' ';
            break;
    }
    sprintf(buffer,"%c",scale_factor_3);
    ui->scale_button_3->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
    ui->doubleSpinBox_3->show();
    ui->value_label_3->show();
    ui->value_slider_3->show();
    ui->scale_button_3->show();
}
// Show only "m"easurement controls
// Diode, Opamp, NMOS, BJT
else if(options == 'm')
{
    // Only show measurements based off component
    switch (ui->stackedWidget_3->currentIndex())
    {
        case 4 : // Diode
            ui->v0_3->show();

```

```

        ui->i0_3->show();
        ui->v2_3->show();
        ui->i2_3->show();
        break;
    case 5 : // Opamp
        ui->v0_3->
>move(QPoint(30+node_x_offset[number],50+node_y_offset[number]));
        ui->i0_3->
>move(QPoint(70+node_x_offset[number],50+node_y_offset[number]));
        ui->v1_3->
>move(QPoint(30+node_x_offset[number],125+node_y_offset[number]));
        ui->i1_3->
>move(QPoint(70+node_x_offset[number],125+node_y_offset[number]));
        ui->v0_3->show();
        ui->i0_3->show();
        ui->v1_3->show();
        ui->i1_3->show();
        ui->v2_3->show();
        ui->i2_3->show();
        break;
    case 6 : case 7: // NMOS/BJT
        ui->v2_3->
>move(QPoint(220+node_x_offset[number],65+node_y_offset[number]));
        ui->i2_3->
>move(QPoint(260+node_x_offset[number],65+node_y_offset[number]));
        ui->v3_3->
>move(QPoint(220+node_x_offset[number],110+node_y_offset[number]));
        ui->i3_3->
>move(QPoint(260+node_x_offset[number],110+node_y_offset[number]));
        ui->v0_3->show();
        ui->i0_3->show();
        ui->v2_3->show();
        ui->i2_3->show();
        ui->v3_3->show();
        ui->i3_3->show();
        break;
    }
}
fscale_factor_3 = '0';
//emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
break;
case 4 :
    // Move everything back to their default positions and default checked
states
    ui->v0_4->move(QPoint(10+node_x_offset[number],70+node_y_offset[number]));
    ui->i0_4->move(QPoint(10+node_x_offset[number],104+node_y_offset[number]));
    ui->v2_4->move(QPoint(281+node_x_offset[number],70+node_y_offset[number]));
    ui->i2_4->move(QPoint(281+node_x_offset[number],104+node_y_offset[number]));
    ui->doubleSpinBox_4->
>move(QPoint(91+node_x_offset[number],155+node_y_offset[number]));
    ui->scale_button_4->
>move(QPoint(175+node_x_offset[number],155+node_y_offset[number]));
    ui->value_label_4->
>move(QPoint(200+node_x_offset[number],155+node_y_offset[number]));
    ui->value_slider_4->
>move(QPoint(10+node_x_offset[number],210+node_y_offset[number]));
    ui->v0_4->setCheckState(Qt::CheckState(0));
    ui->i0_4->setCheckState(Qt::CheckState(0));
    ui->v1_4->setCheckState(Qt::CheckState(0));
    ui->i1_4->setCheckState(Qt::CheckState(0));
    ui->v2_4->setCheckState(Qt::CheckState(0));
    ui->i2_4->setCheckState(Qt::CheckState(0));
    ui->v3_4->setCheckState(Qt::CheckState(0));
    ui->i3_4->setCheckState(Qt::CheckState(0));
    // Show "a"ll measurement and value controls
    // Resistor, Capacitor, Inductor, Isource, Vsource
    if(options == 'a')
    {
        // Display V/I measurements 0 and 2
        ui->v0_4->show();

```

```

    ui->i0_4->show();
    ui->v2_4->show();
    ui->i2_4->show();
    // Enable controls based off componenet
    switch (ui->stackedWidget_4->currentIndex())
    {
        case 1 : // Resistor
            ui->value_slider_4->setRange(1,999);
            ui->doubleSpinBox_4->setRange(0,999.9);
            ui->value_slider_4->setValue(50);
            scale_factor_4 = 'k';
            break;
        case 2 : // Capacitor
            ui->value_slider_4->setRange(1,999);
            ui->doubleSpinBox_4->setRange(0,999.9);
            ui->value_slider_4->setValue(5);
            scale_factor_4 = 'u';
            break;
        case 3 : // Inductor
            ui->value_slider_4->setRange(1,999);
            ui->doubleSpinBox_4->setRange(0,999.9);
            ui->value_slider_4->setValue(3);
            scale_factor_4 = 'm';
            break;
        case 8 : // Voltage Source
            ui->value_slider_4->setRange(1,99);
            ui->doubleSpinBox_4->setRange(0,99.9);
            ui->value_slider_4->setValue(5);
            scale_factor_4 = ' ';
            break;
        case 9 : // Current Source
            ui->value_slider_4->setRange(1,999);
            ui->doubleSpinBox_4->setRange(0,999.9);
            ui->value_slider_4->setValue(80);
            scale_factor_4 = 'm';
            break;
        default :
            ui->value_slider_4->setValue(1);
            scale_factor_4 = ' ';
            break;
    }
    sprintf(buffer,"%c",scale_factor_4);
    ui->scale_button_4->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
    ui->doubleSpinBox_4->show();
    ui->value_label_4->show();
    ui->value_slider_4->show();
    ui->scale_button_4->show();
}
// Show only "m" measurement controls
// Diode, Opamp, NMOS, BJT
else if(options == 'm')
{
    // Only show measurements based off component
    switch (ui->stackedWidget_4->currentIndex())
    {
        case 4 : // Diode
            ui->v0_4->show();
            ui->i0_4->show();
            ui->v2_4->show();
            ui->i2_4->show();
            break;
        case 5 : // Opamp
            ui->v0_4->
>move(QPoint(30+node_x_offset[number],50+node_y_offset[number]));
            ui->i0_4->
>move(QPoint(70+node_x_offset[number],50+node_y_offset[number]));
            ui->v1_4->
>move(QPoint(30+node_x_offset[number],125+node_y_offset[number]));

```

```

        ui->i1_4-
>move(QPoint(70+node_x_offset[number],125+node_y_offset[number]));
        ui->v0_4->show();
        ui->i0_4->show();
        ui->v1_4->show();
        ui->i1_4->show();
        ui->v2_4->show();
        ui->i2_4->show();
        break;
        case 6 : case 7: // NMOS/BJT
        ui->v2_4-
>move(QPoint(220+node_x_offset[number],65+node_y_offset[number]));
        ui->i2_4-
>move(QPoint(260+node_x_offset[number],65+node_y_offset[number]));
        ui->v3_4-
>move(QPoint(220+node_x_offset[number],110+node_y_offset[number]));
        ui->i3_4-
>move(QPoint(260+node_x_offset[number],110+node_y_offset[number]));
        ui->v0_4->show();
        ui->i0_4->show();
        ui->v2_4->show();
        ui->i2_4->show();
        ui->v3_4->show();
        ui->i3_4->show();
        break;
    }
    }
    fscale_factor_4 = '0';
    //emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
    break;
}

void MainWindow::on_doubleSpinBox_valueChanged(double value)
{
    ui->value_slider->setValue(value);
    //emit sendComponentValue(value,scale_factor);
}
void MainWindow::on_doubleSpinBox_2_valueChanged(double value)
{
    ui->value_slider_2->setValue(value);
    //emit sendComponentValue(value,scale_factor);
}
void MainWindow::on_doubleSpinBox_3_valueChanged(double value)
{
    ui->value_slider_3->setValue(value);
    //emit sendComponentValue(value,scale_factor);
}
void MainWindow::on_doubleSpinBox_4_valueChanged(double value)
{
    ui->value_slider_4->setValue(value);
    //emit sendComponentValue(value,scale_factor);
}
void MainWindow::on_fdoubleSpinBox_valueChanged(double value)
{
    ui->fvalue_slider->setValue(value);
    //emit sendfComponentValue(value,fscale_factor);
}
void MainWindow::on_fdoubleSpinBox_2_valueChanged(double value)
{
    ui->fvalue_slider_2->setValue(value);
    //emit sendfComponentValue(value,fscale_factor);
}
void MainWindow::on_fdoubleSpinBox_3_valueChanged(double value)
{
    ui->fvalue_slider_3->setValue(value);
    //emit sendfComponentValue(value,fscale_factor);
}
void MainWindow::on_fdoubleSpinBox_4_valueChanged(double value)
{
    ui->fvalue_slider_4->setValue(value);
}

```

```

        //emit sendfComponentValue(value,fscale_factor);
    }
    void MainWindow::on_v0_stateChanged(int state)
    {
        //sendMeasureStatus(0,state);
    }
    void MainWindow::on_v1_stateChanged(int state)
    {
        //sendMeasureStatus(1,state);
    }
    void MainWindow::on_v2_stateChanged(int state)
    {
        //sendMeasureStatus(2,state);
    }
    void MainWindow::on_v3_stateChanged(int state)
    {
        //sendMeasureStatus(3,state);
    }
    void MainWindow::on_i0_stateChanged(int state)
    {
        //sendMeasureStatus(4,state);
    }
    void MainWindow::on_i1_stateChanged(int state)
    {
        //sendMeasureStatus(5,state);
    }
    void MainWindow::on_i2_stateChanged(int state)
    {
        //sendMeasureStatus(6,state);
    }
    void MainWindow::on_i3_stateChanged(int state)
    {
        ////sendMeasureStatus(7,state);
    }
    void MainWindow::on_scale_button_clicked()
    {
        switch(scale_factor)
        {
            case ' ':
                scale_factor = 'k';
                break;
            case 'k':
                scale_factor = 'M';
                break;
            case 'M':
                scale_factor = 'p';
                break;
            case 'p':
                scale_factor = 'n';
                break;
            case 'n':
                scale_factor = 'u';
                break;
            case 'u':
                scale_factor = 'm';
                break;
            case 'm':
                scale_factor = ' ';
                break;
        }
        sprintf(buffer,"%c",scale_factor);
        ui->scale_button->setText(buffer);
        //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
    }
    void MainWindow::on_scale_button_2_clicked()
    {
        switch(scale_factor_2)
        {
            case ' ':
                scale_factor_2 = 'k';

```



```

        break;
    case 'k':
        scale_factor_2 = 'M';
        break;
    case 'M':
        scale_factor_2 = 'p';
        break;
    case 'p':
        scale_factor_2 = 'n';
        break;
    case 'n':
        scale_factor_2 = 'u';
        break;
    case 'u':
        scale_factor_2 = 'm';
        break;
    case 'm':
        scale_factor_2 = ' ';
        break;
    }
    sprintf(buffer, "%c", scale_factor_2);
    ui->scale_button_2->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(), scale_factor);
}
void MainWindow::on_scale_button_3_clicked()
{
    switch(scale_factor_3)
    {
        case ' ':
            scale_factor_3 = 'k';
            break;
        case 'k':
            scale_factor_3 = 'M';
            break;
        case 'M':
            scale_factor_3 = 'p';
            break;
        case 'p':
            scale_factor_3 = 'n';
            break;
        case 'n':
            scale_factor_3 = 'u';
            break;
        case 'u':
            scale_factor_3 = 'm';
            break;
        case 'm':
            scale_factor_3 = ' ';
            break;
    }
    sprintf(buffer, "%c", scale_factor_3);
    ui->scale_button_3->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(), scale_factor);
}
void MainWindow::on_scale_button_4_clicked()
{
    switch(scale_factor_4)
    {
        case ' ':
            scale_factor_4 = 'k';
            break;
        case 'k':
            scale_factor_4 = 'M';
            break;
        case 'M':
            scale_factor_4 = 'p';
            break;
        case 'p':
            scale_factor_4 = 'n';
            break;
    }
}

```

```

        case 'n':
            scale_factor_4 = 'u';
            break;
        case 'u':
            scale_factor_4 = 'm';
            break;
        case 'm':
            scale_factor_4 = ' ';
            break;
    }
    sprintf(buffer, "%c", scale_factor_4);
    ui->scale_button_4->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(), scale_factor);
}
void MainWindow::on_fscale_button_clicked()
{
    switch(fscale_factor)
    {
        case ' ':
            fscale_factor = 'k';
            break;
        case 'k':
            fscale_factor = 'M';
            break;
        case 'M':
            fscale_factor = ' ';
            break;
    }
    sprintf(buffer, "%c", fscale_factor);
    ui->fscale_button->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(), fscale_factor);
}
void MainWindow::on_fscale_button_2_clicked()
{
    switch(fscale_factor_2)
    {
        case ' ':
            fscale_factor_2 = 'k';
            break;
        case 'k':
            fscale_factor_2 = 'M';
            break;
        case 'M':
            fscale_factor_2 = ' ';
            break;
    }
    sprintf(buffer, "%c", fscale_factor_2);
    ui->fscale_button_2->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(), fscale_factor);
}
void MainWindow::on_fscale_button_3_clicked()
{
    switch(fscale_factor_3)
    {
        case ' ':
            fscale_factor_3 = 'k';
            break;
        case 'k':
            fscale_factor_3 = 'M';
            break;
        case 'M':
            fscale_factor_3 = ' ';
            break;
    }
    sprintf(buffer, "%c", fscale_factor_3);
    ui->fscale_button_3->setText(buffer);
    //emit sendComponentValue(ui->doubleSpinBox->value(), fscale_factor);
}
void MainWindow::on_fscale_button_4_clicked()
{

```

```

switch(fscale_factor_4)
{
    case ' ':
        fscale_factor_4 = 'k';
        break;
    case 'k':
        fscale_factor_4 = 'M';
        break;
    case 'M':
        fscale_factor_4 = ' ';
        break;
}
sprintf(buffer, "%c", fscale_factor_4);
ui->fscale_button_4->setText(buffer);
//emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
}
// Toggle DC or AC source
void MainWindow::on_dc_sine_button_clicked()
{
    if(acdc == 'd')
    {
        ui->doubleSpinBox->move(QPoint(10+node_x_offset[1], 155+node_y_offset[1]));
        ui->scale_button->move(QPoint(94+node_x_offset[1], 155+node_y_offset[1]));
        ui->value_label->move(QPoint(118+node_x_offset[1], 155+node_y_offset[1]));
        ui->fdoubleSpinBox->move(QPoint(171+node_x_offset[1], 155+node_y_offset[1]));
        ui->fscale_button->move(QPoint(255+node_x_offset[1], 155+node_y_offset[1]));
        ui->fvalue_label->move(QPoint(280+node_x_offset[1], 155+node_y_offset[1]));
        ui->value_slider->move(QPoint(10+node_x_offset[1], 190+node_y_offset[1]));
        ui->fvalue_slider->move(QPoint(10+node_x_offset[1], 210+node_y_offset[1]));
        ui->dc_sine_button->setIcon(QIcon(QPixmap(":/images/images/vdc_symbol.png")));
        ui->vsource_label->setPixmap(QPixmap(":/images/images/vsource3.png"));
        ui->fvalue_slider->setValue(10);
        fscale_factor = 'k';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
        acdc = 'a';
        ui->fdoubleSpinBox->show();
        ui->fvalue_label->show();
        ui->fvalue_slider->show();
        ui->fscale_button->show();
    }
    else
    {
        ui->fdoubleSpinBox->hide();
        ui->fvalue_label->hide();
        ui->fvalue_slider->hide();
        ui->fscale_button->hide();
        ui->value_slider->move(QPoint(10+node_x_offset[1], 210+node_y_offset[1]));
        ui->doubleSpinBox->move(QPoint(91+node_x_offset[1], 155+node_y_offset[1]));
        ui->scale_button->move(QPoint(175+node_x_offset[1], 155+node_y_offset[1]));
        ui->value_label->move(QPoint(200+node_x_offset[1], 155+node_y_offset[1]));
        ui->dc_sine_button->setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
        ui->vsource_label->setPixmap(QPixmap(":/images/images/vsource2.png"));
        fscale_factor = '0';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(), fscale_factor);
        acdc = 'd';
    }
}
// Toggle DC or AC source
void MainWindow::on_dc_sine_button_2_clicked()
{
    int n = 2;
    if(acdc_2 == 'd')
    {
        ui->doubleSpinBox_2->move(QPoint(10+node_x_offset[n], 155+node_y_offset[n]));
        ui->scale_button_2->move(QPoint(94+node_x_offset[n], 155+node_y_offset[n]));
        ui->value_label_2->move(QPoint(118+node_x_offset[n], 155+node_y_offset[n]));
        ui->fdoubleSpinBox_2->move(QPoint(171+node_x_offset[n], 155+node_y_offset[n]));
        ui->fscale_button_2->move(QPoint(255+node_x_offset[n], 155+node_y_offset[n]));
        ui->fvalue_label_2->move(QPoint(280+node_x_offset[n], 155+node_y_offset[n]));
        ui->value_slider_2->move(QPoint(10+node_x_offset[n], 190+node_y_offset[n]));
    }
}

```

```

        ui->fvalue_slider_2->move(QPoint(10+node_x_offset[n],210+node_y_offset[n]));
        ui->dc_sine_button_2->setIcon(QIcon(QPixmap(":/images/images/vdc_symbol.png")));
        ui->vsource_label_2->setPixmap(QPixmap(":/images/images/vsource3.png"));
        ui->fvalue_slider_2->setValue(10);
        fscale_factor_2 = 'k';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
        acdc_2 = 'a';
        ui->fdoubleSpinBox_2->show();
        ui->fvalue_label_2->show();
        ui->fvalue_slider_2->show();
        ui->fscale_button_2->show();
    }
    else
    {
        ui->fdoubleSpinBox_2->hide();
        ui->fvalue_label_2->hide();
        ui->fvalue_slider_2->hide();
        ui->fscale_button_2->hide();
        ui->value_slider_2->move(QPoint(10+node_x_offset[n],210+node_y_offset[n]));
        ui->doubleSpinBox_2->move(QPoint(91+node_x_offset[n],155+node_y_offset[n]));
        ui->scale_button_2->move(QPoint(175+node_x_offset[n],155+node_y_offset[n]));
        ui->value_label_2->move(QPoint(200+node_x_offset[n],155+node_y_offset[n]));
        ui->dc_sine_button_2->
>setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
        ui->vsource_label_2->setPixmap(QPixmap(":/images/images/vsource2.png"));
        fscale_factor_2 = '0';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
        acdc_2 = 'd';
    }
}
// Toggle DC or AC source
void MainWindow::on_dc_sine_button_3_clicked()
{
    int n = 3;
    if(acdc_3 == 'd')
    {
        ui->doubleSpinBox_3->move(QPoint(10+node_x_offset[n],155+node_y_offset[n]));
        ui->scale_button_3->move(QPoint(94+node_x_offset[n],155+node_y_offset[n]));
        ui->value_label_3->move(QPoint(118+node_x_offset[n],155+node_y_offset[n]));
        ui->fdoubleSpinBox_3->move(QPoint(171+node_x_offset[n],155+node_y_offset[n]));
        ui->fscale_button_3->move(QPoint(255+node_x_offset[n],155+node_y_offset[n]));
        ui->fvalue_label_3->move(QPoint(280+node_x_offset[n],155+node_y_offset[n]));
        ui->value_slider_3->move(QPoint(10+node_x_offset[n],190+node_y_offset[n]));
        ui->fvalue_slider_3->move(QPoint(10+node_x_offset[n],210+node_y_offset[n]));
        ui->dc_sine_button_3->setIcon(QIcon(QPixmap(":/images/images/vdc_symbol.png")));
        ui->vsource_label_3->setPixmap(QPixmap(":/images/images/vsource3.png"));
        ui->fvalue_slider_3->setValue(10);
        fscale_factor_3 = 'k';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
        acdc_3 = 'a';
        ui->fdoubleSpinBox_3->show();
        ui->fvalue_label_3->show();
        ui->fvalue_slider_3->show();
        ui->fscale_button_3->show();
    }
    else
    {
        ui->fdoubleSpinBox_3->hide();
        ui->fvalue_label_3->hide();
        ui->fvalue_slider_3->hide();
        ui->fscale_button_3->hide();
        ui->value_slider_3->move(QPoint(10+node_x_offset[n],210+node_y_offset[n]));
        ui->doubleSpinBox_3->move(QPoint(91+node_x_offset[n],155+node_y_offset[n]));
        ui->scale_button_3->move(QPoint(175+node_x_offset[n],155+node_y_offset[n]));
        ui->value_label_3->move(QPoint(200+node_x_offset[n],155+node_y_offset[n]));
        ui->dc_sine_button_3->
>setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
        ui->vsource_label_3->setPixmap(QPixmap(":/images/images/vsource2.png"));
        fscale_factor_3 = '0';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);

```

```

        acdc_3 = 'd';
    }
}
// Toggle DC or AC source
void MainWindow::on_dc_sine_button_4_clicked()
{
    int n = 4;
    if(acdc_4 == 'd')
    {
        ui->doubleSpinBox_4->move(QPoint(10+node_x_offset[n],155+node_y_offset[n]));
        ui->scale_button_4->move(QPoint(94+node_x_offset[n],155+node_y_offset[n]));
        ui->value_label_4->move(QPoint(118+node_x_offset[n],155+node_y_offset[n]));
        ui->fdoubleSpinBox_4->move(QPoint(171+node_x_offset[n],155+node_y_offset[n]));
        ui->fscale_button_4->move(QPoint(255+node_x_offset[n],155+node_y_offset[n]));
        ui->fvalue_label_4->move(QPoint(280+node_x_offset[n],155+node_y_offset[n]));
        ui->value_slider_4->move(QPoint(10+node_x_offset[n],190+node_y_offset[n]));
        ui->fvalue_slider_4->move(QPoint(10+node_x_offset[n],210+node_y_offset[n]));
        ui->dc_sine_button_4->setIcon(QIcon(QPixmap(":/images/images/vdc_symbol.png")));
        ui->vsource_label_4->setPixmap(QPixmap(":/images/images/vsource3.png"));
        ui->fvalue_slider_4->setValue(10);
        fscale_factor_4 = 'k';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
        acdc_4 = 'a';
        ui->fdoubleSpinBox_4->show();
        ui->fvalue_label_4->show();
        ui->fvalue_slider_4->show();
        ui->fscale_button_4->show();
    }
    else
    {
        ui->fdoubleSpinBox_4->hide();
        ui->fvalue_label_4->hide();
        ui->fvalue_slider_4->hide();
        ui->fscale_button_4->hide();
        ui->value_slider_4->move(QPoint(10+node_x_offset[n],210+node_y_offset[n]));
        ui->doubleSpinBox_4->move(QPoint(91+node_x_offset[n],155+node_y_offset[n]));
        ui->scale_button_4->move(QPoint(175+node_x_offset[n],155+node_y_offset[n]));
        ui->value_label_4->move(QPoint(200+node_x_offset[n],155+node_y_offset[n]));
        ui->dc_sine_button_4->
>setIcon(QIcon(QPixmap(":/images/images/vsin_symbol.png")));
        ui->vsource_label_4->setPixmap(QPixmap(":/images/images/vsource2.png"));
        fscale_factor_4 = '0';
        //emit sendfComponentValue(ui->fdoubleSpinBox->value(),fscale_factor);
        acdc_4 = 'd';
    }
}
void MainWindow::on_IO_0_clicked()
{
    if(IO_0_net < 9)
    {
        IO_0_net++;
    }
    else
    {
        IO_0_net = -1;
    }
    if(ui->stackedWidget->currentIndex() != 5 &&
        ui->stackedWidget->currentIndex() != 6 &&
        ui->stackedWidget->currentIndex() != 7 &&
        ui->stackedWidget->currentIndex() != 0)
    {
        IO_1_net = IO_0_net;
        if(IO_0_net < 0)
        {
            ui->IO_0->setText("N");
            ui->IO_1->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_0_net);

```

```

        ui->IO_0->setText(buffer);
        ui->IO_1->setText(buffer);
    }
}
else
{
    if(IO_0_net < 0)
    {
        ui->IO_0->setText("N");
    }
    else
    {
        sprintf(buffer,"%d",IO_0_net);
        ui->IO_0->setText(buffer);
    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_0_2_clicked()
{
    if(IO_0_net_2 < 9)
    {
        IO_0_net_2++;
    }
    else
    {
        IO_0_net_2 = -1;
    }
    if(ui->stackedWidget_2->currentIndex() != 5 &&
        ui->stackedWidget_2->currentIndex() != 6 &&
        ui->stackedWidget_2->currentIndex() != 7 &&
        ui->stackedWidget_2->currentIndex() != 0)
    {
        IO_1_net_2 = IO_0_net_2;
        if(IO_0_net_2 < 0)
        {
            ui->IO_0_2->setText("N");
            ui->IO_1_2->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_0_net_2);
            ui->IO_0_2->setText(buffer);
            ui->IO_1_2->setText(buffer);
        }
    }
    else
    {
        if(IO_0_net_2 < 0)
        {
            ui->IO_0_2->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_0_net_2);
            ui->IO_0_2->setText(buffer);
        }
    }
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_0_3_clicked()
{
    if(IO_0_net_3 < 9)
    {
        IO_0_net_3++;
    }
    else
    {
        IO_0_net_3 = -1;
    }
}

```

```

if(ui->stackedWidget_3->currentIndex() != 5 &&
   ui->stackedWidget_3->currentIndex() != 6 &&
   ui->stackedWidget_3->currentIndex() != 7 &&
   ui->stackedWidget_3->currentIndex() != 0)
{
    IO_1_net_3 = IO_0_net_3;
    if(IO_0_net_3 < 0)
    {
        ui->IO_0_3->setText("N");
        ui->IO_1_3->setText("N");
    }
    else
    {
        sprintf(buffer, "%d", IO_0_net_3);
        ui->IO_0_3->setText(buffer);
        ui->IO_1_3->setText(buffer);
    }
}
else
{
    if(IO_0_net_3 < 0)
    {
        ui->IO_0_3->setText("N");
    }
    else
    {
        sprintf(buffer, "%d", IO_0_net_3);
        ui->IO_0_3->setText(buffer);
    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(), scale_factor);
}
void MainWindow::on_IO_0_4_clicked()
{
    if(IO_0_net_4 < 9)
    {
        IO_0_net_4++;
    }
    else
    {
        IO_0_net_4 = -1;
    }
    if(ui->stackedWidget_4->currentIndex() != 5 &&
       ui->stackedWidget_4->currentIndex() != 6 &&
       ui->stackedWidget_4->currentIndex() != 7 &&
       ui->stackedWidget_4->currentIndex() != 0)
    {
        IO_1_net_4 = IO_0_net_4;
        if(IO_0_net_4 < 0)
        {
            ui->IO_0_4->setText("N");
            ui->IO_1_4->setText("N");
        }
        else
        {
            sprintf(buffer, "%d", IO_0_net_4);
            ui->IO_0_4->setText(buffer);
            ui->IO_1_4->setText(buffer);
        }
    }
    else
    {
        if(IO_0_net_4 < 0)
        {
            ui->IO_0_4->setText("N");
        }
        else
        {
            sprintf(buffer, "%d", IO_0_net_4);
            ui->IO_0_4->setText(buffer);
        }
    }
}

```

```

    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_1_clicked()
{
    if(IO_1_net < 9)
    {
        IO_1_net++;
    }
    else
    {
        IO_1_net = -1;
    }
    if(ui->stackedWidget->currentIndex() != 5 &&
        ui->stackedWidget->currentIndex() != 6 &&
        ui->stackedWidget->currentIndex() != 7 &&
        ui->stackedWidget->currentIndex() != 0)
    {
        IO_0_net = IO_1_net;
        if(IO_1_net < 0)
        {
            ui->IO_1->setText("N");
            ui->IO_0->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_1_net);
            ui->IO_1->setText(buffer);
            ui->IO_0->setText(buffer);
        }
    }
    else
    {
        if(IO_1_net < 0)
        {
            ui->IO_1->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_1_net);
            ui->IO_1->setText(buffer);
        }
    }
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_1_2_clicked()
{
    if(IO_1_net_2 < 9)
    {
        IO_1_net_2++;
    }
    else
    {
        IO_1_net_2 = -1;
    }
    if(ui->stackedWidget_2->currentIndex() != 5 &&
        ui->stackedWidget_2->currentIndex() != 6 &&
        ui->stackedWidget_2->currentIndex() != 7 &&
        ui->stackedWidget_2->currentIndex() != 0)
    {
        IO_0_net_2 = IO_1_net_2;
        if(IO_1_net_2 < 0)
        {
            ui->IO_1_2->setText("N");
            ui->IO_0_2->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_1_net_2);

```



```

        ui->IO_1_2->setText(buffer);
        ui->IO_0_2->setText(buffer);
    }
}
else
{
    if(IO_1_net_2 < 0)
    {
        ui->IO_1_2->setText("N");
    }
    else
    {
        sprintf(buffer,"%d",IO_1_net_2);
        ui->IO_1_2->setText(buffer);
    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_1_3_clicked()
{
    if(IO_1_net_3 < 9)
    {
        IO_1_net_3++;
    }
    else
    {
        IO_1_net_3 = -1;
    }
    if(ui->stackedWidget_3->currentIndex() != 5 &&
        ui->stackedWidget_3->currentIndex() != 6 &&
        ui->stackedWidget_3->currentIndex() != 7 &&
        ui->stackedWidget_3->currentIndex() != 0)
    {
        IO_0_net_3 = IO_1_net_3;
        if(IO_1_net_3 < 0)
        {
            ui->IO_1_3->setText("N");
            ui->IO_0_3->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_1_net_3);
            ui->IO_1_3->setText(buffer);
            ui->IO_0_3->setText(buffer);
        }
    }
    else
    {
        if(IO_1_net_3 < 0)
        {
            ui->IO_1_3->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_1_net_3);
            ui->IO_1_3->setText(buffer);
        }
    }
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_1_4_clicked()
{
    if(IO_1_net_4 < 9)
    {
        IO_1_net_4++;
    }
    else
    {
        IO_1_net_4 = -1;
    }
}

```

```

if(ui->stackedWidget_4->currentIndex() != 5 &&
    ui->stackedWidget_4->currentIndex() != 6 &&
    ui->stackedWidget_4->currentIndex() != 7 &&
    ui->stackedWidget_4->currentIndex() != 0)
{
    IO_0_net_4 = IO_1_net_4;
    if(IO_1_net_4 < 0)
    {
        ui->IO_1_4->setText("N");
        ui->IO_0_4->setText("N");
    }
    else
    {
        sprintf(buffer,"%d",IO_1_net_4);
        ui->IO_1_4->setText(buffer);
        ui->IO_0_4->setText(buffer);
    }
}
else
{
    if(IO_1_net_4 < 0)
    {
        ui->IO_1_4->setText("N");
    }
    else
    {
        sprintf(buffer,"%d",IO_1_net_4);
        ui->IO_1_4->setText(buffer);
    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_2_clicked()
{
    if(IO_2_net < 9)
    {
        IO_2_net++;
    }
    else
    {
        IO_2_net = -1;
    }
    if(ui->stackedWidget->currentIndex() != 5 &&
        ui->stackedWidget->currentIndex() != 6 &&
        ui->stackedWidget->currentIndex() != 7 &&
        ui->stackedWidget->currentIndex() != 0)
    {
        IO_3_net = IO_2_net;
        if(IO_2_net < 0)
        {
            ui->IO_2->setText("N");
            ui->IO_3->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_2_net);
            ui->IO_2->setText(buffer);
            ui->IO_3->setText(buffer);
        }
    }
    else
    {
        if(IO_2_net < 0)
        {
            ui->IO_2->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_2_net);
            ui->IO_2->setText(buffer);
        }
    }
}

```

```

    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_2_2_clicked()
{
    if(IO_2_net_2 < 9)
    {
        IO_2_net_2++;
    }
    else
    {
        IO_2_net_2 = -1;
    }
    if(ui->stackedWidget_2->currentIndex() != 5 &&
        ui->stackedWidget_2->currentIndex() != 6 &&
        ui->stackedWidget_2->currentIndex() != 7 &&
        ui->stackedWidget_2->currentIndex() != 0)
    {
        IO_3_net_2 = IO_2_net_2;
        if(IO_2_net_2 < 0)
        {
            ui->IO_2_2->setText("N");
            ui->IO_3_2->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_2_net_2);
            ui->IO_2_2->setText(buffer);
            ui->IO_3_2->setText(buffer);
        }
    }
    else
    {
        if(IO_2_net_2 < 0)
        {
            ui->IO_2_2->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_2_net_2);
            ui->IO_2_2->setText(buffer);
        }
    }
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_2_3_clicked()
{
    if(IO_2_net_3 < 9)
    {
        IO_2_net_3++;
    }
    else
    {
        IO_2_net_3 = -1;
    }
    if(ui->stackedWidget_3->currentIndex() != 5 &&
        ui->stackedWidget_3->currentIndex() != 6 &&
        ui->stackedWidget_3->currentIndex() != 7 &&
        ui->stackedWidget_3->currentIndex() != 0)
    {
        IO_3_net_3 = IO_2_net_3;
        if(IO_2_net_3 < 0)
        {
            ui->IO_2_3->setText("N");
            ui->IO_3_3->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_2_net_3);

```

```

        ui->IO_2_3->setText(buffer);
        ui->IO_3_3->setText(buffer);
    }
}
else
{
    if(IO_2_net_3 < 0)
    {
        ui->IO_2_3->setText("N");
    }
    else
    {
        sprintf(buffer,"%d",IO_2_net_3);
        ui->IO_2_3->setText(buffer);
    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_2_4_clicked()
{
    if(IO_2_net_4 < 9)
    {
        IO_2_net_4++;
    }
    else
    {
        IO_2_net_4 = -1;
    }
    if(ui->stackedWidget_4->currentIndex() != 5 &&
        ui->stackedWidget_4->currentIndex() != 6 &&
        ui->stackedWidget_4->currentIndex() != 7 &&
        ui->stackedWidget_4->currentIndex() != 0)
    {
        IO_3_net_4 = IO_2_net_4;
        if(IO_2_net_4 < 0)
        {
            ui->IO_2_4->setText("N");
            ui->IO_3_4->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_2_net_4);
            ui->IO_2_4->setText(buffer);
            ui->IO_3_4->setText(buffer);
        }
    }
    else
    {
        if(IO_2_net_4 < 0)
        {
            ui->IO_2_4->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_2_net_4);
            ui->IO_2_4->setText(buffer);
        }
    }
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_3_clicked()
{
    if(IO_3_net < 9)
    {
        IO_3_net++;
    }
    else
    {
        IO_3_net = -1;
    }
}

```

```

if(ui->stackedWidget->currentIndex() != 5 &&
   ui->stackedWidget->currentIndex() != 6 &&
   ui->stackedWidget->currentIndex() != 7 &&
   ui->stackedWidget->currentIndex() != 0)
{
    IO_2_net = IO_3_net;
    if(IO_3_net < 0)
    {
        ui->IO_3->setText("N");
        ui->IO_2->setText("N");
    }
    else
    {
        sprintf(buffer,"%d",IO_3_net);
        ui->IO_3->setText(buffer);
        ui->IO_2->setText(buffer);
    }
}
else
{
    if(IO_3_net < 0)
    {
        ui->IO_3->setText("N");
    }
    else
    {
        sprintf(buffer,"%d",IO_3_net);
        ui->IO_3->setText(buffer);
    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_3_2_clicked()
{
    if(IO_3_net_2 < 9)
    {
        IO_3_net_2++;
    }
    else
    {
        IO_3_net_2 = -1;
    }
    if(ui->stackedWidget_2->currentIndex() != 5 &&
       ui->stackedWidget_2->currentIndex() != 6 &&
       ui->stackedWidget_2->currentIndex() != 7 &&
       ui->stackedWidget_2->currentIndex() != 0)
    {
        IO_2_net_2 = IO_3_net_2;
        if(IO_3_net_2 < 0)
        {
            ui->IO_3_2->setText("N");
            ui->IO_2_2->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_3_net_2);
            ui->IO_3_2->setText(buffer);
            ui->IO_2_2->setText(buffer);
        }
    }
    else
    {
        if(IO_3_net_2 < 0)
        {
            ui->IO_3_2->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_3_net_2);
            ui->IO_3_2->setText(buffer);
        }
    }
}

```

```

    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_3_3_clicked()
{
    if(IO_3_net_3 < 9)
    {
        IO_3_net_3++;
    }
    else
    {
        IO_3_net_3 = -1;
    }
    if(ui->stackedWidget_3->currentIndex() != 5 &&
        ui->stackedWidget_3->currentIndex() != 6 &&
        ui->stackedWidget_3->currentIndex() != 7 &&
        ui->stackedWidget_3->currentIndex() != 0)
    {
        IO_2_net_3 = IO_3_net_3;
        if(IO_3_net_3 < 0)
        {
            ui->IO_3_3->setText("N");
            ui->IO_2_3->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_3_net_3);
            ui->IO_3_3->setText(buffer);
            ui->IO_2_3->setText(buffer);
        }
    }
    else
    {
        if(IO_3_net_3 < 0)
        {
            ui->IO_3_3->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_3_net_3);
            ui->IO_3_3->setText(buffer);
        }
    }
    //emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}
void MainWindow::on_IO_3_4_clicked()
{
    if(IO_3_net_4 < 9)
    {
        IO_3_net_4++;
    }
    else
    {
        IO_3_net_4 = -1;
    }
    if(ui->stackedWidget_4->currentIndex() != 5 &&
        ui->stackedWidget_4->currentIndex() != 6 &&
        ui->stackedWidget_4->currentIndex() != 7 &&
        ui->stackedWidget_4->currentIndex() != 0)
    {
        IO_2_net_4 = IO_3_net_4;
        if(IO_3_net_4 < 0)
        {
            ui->IO_3_4->setText("N");
            ui->IO_2_4->setText("N");
        }
        else
        {
            sprintf(buffer,"%d",IO_3_net_4);

```

```

        ui->IO_3_4->setText(buffer);
        ui->IO_2_4->setText(buffer);
    }
}
else
{
    if(IO_3_net_4 < 0)
    {
        ui->IO_3_4->setText("N");
    }
    else
    {
        sprintf(buffer,"%d",IO_3_net_4);
        ui->IO_3_4->setText(buffer);
    }
}
//emit sendComponentValue(ui->doubleSpinBox->value(),scale_factor);
}

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
void MainWindow::formatNetlist()
{
    remove("/root/netlist.cir");
    ofstream spiceFile("/root/netlist.cir");
    char buffer [50] = {0};
    int opamp = 0;
    int i, j;
    int dummyInCount = 0;
    int dummyOutCount = 0;
    memset(nodeVoltageMeasured,0,10);
    memset(nodeCurrentMeasured,0,10);
    sprintf(buffer,"*Generated Netlist\n\n");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
    for(i = 0; i < NUM_DEVICES; i++)
    {
        memset(buffer, 0, 50);
        if(component[i] != 0)
        {
            switch(i)
            {
            case 0:
                inputNode[0][i] = IO_0_net;
                inputNode[1][i] = IO_1_net;
                outputNode[0][i] = IO_2_net;
                outputNode[1][i] = IO_3_net;
                break;
            case 1:
                inputNode[0][i] = IO_0_net_2;
                inputNode[1][i] = IO_1_net_2;
                outputNode[0][i] = IO_2_net_2;
                outputNode[1][i] = IO_3_net_2;
                break;
            case 2:
                inputNode[0][i] = IO_0_net_3;
                inputNode[1][i] = IO_1_net_3;
                outputNode[0][i] = IO_2_net_3;
                outputNode[1][i] = IO_3_net_3;
                break;
            case 3:
                inputNode[0][i] = IO_0_net_4;
                inputNode[1][i] = IO_1_net_4;
                outputNode[0][i] = IO_2_net_4;
                outputNode[1][i] = IO_3_net_4;
                break;
            }
            switch(component[i])
            {

```

```

        case 1:
            sprintf(buffer,"R%d %d %d %s\n", i, inputNode[0][i], outputNode[0][i],
componentValue[i]);
            qDebug("Component value sent to netlist %s",componentValue[i]);
            break;
        case 2:
            sprintf(buffer,"C%d %d %d %s\n", i, inputNode[0][i], outputNode[0][i],
componentValue[i]);
            break;
        case 3:
            sprintf(buffer,"L%d %d %d %s\n", i, inputNode[0][i], outputNode[0][i],
componentValue[i]);
            break;
        case 4:
            sprintf(buffer,"D%d %d %d \n", i, inputNode[0][i], outputNode[0][i]);
            break;
        case 7:
            sprintf(buffer,"M%d %d %d %d nmos\n", i, outputNode[0][i],
inputNode[0][i], outputNode[1][i]);
            //sprintf(modelBuffer,"");
            break;
        case 6:
            sprintf(buffer,"Q%d %d %d %d npn\n", i, outputNode[0][i],
inputNode[0][i], outputNode[1][i]);
            //sprintf(modelBuffer,"");
            break;
        case 5:
            sprintf(buffer,"X%d %d %d %d opamp741\n", i, inputNode[0][i],
inputNode[1][i], outputNode[1][i]);
            opamp = 1;
            break;
        case 8:
            if(strlen(componentValue[i]) > 6)
            {
                sprintf(buffer,"V%d %d %d sin(0 %s 0 0 0)\n", i, outputNode[0][i],
inputNode[0][i],componentValue[i]);
            }
            else
            {
                sprintf(buffer,"V%d %d %d %s\n", i, outputNode[0][i],
inputNode[0][i],componentValue[i]);
            }
            break;
        case 9:
            sprintf(buffer,"I%d %d %d DC %s\n", i, outputNode[0][i],
inputNode[0][i],componentValue[i]);
            break;
    }
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
}
}
if(opamp)
{
    memset(buffer,0,50);
    sprintf(buffer,".subckt opamp741 1 2 3\nrin 1 2 2meg\nrout 6 3 75\n");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
    memset(buffer,0,50);
    sprintf(buffer,"e 4 0 1 2 100k\nrbw 4 5 0.5meg\nncbw 5 0 31.85nf\n");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
    memset(buffer,0,50);
    sprintf(buffer,"eout 6 0 5 0 1\n.ends opamp741\n");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
}
for(j = 0; j < 2; j++)
{
    for(i = 0; i < NUM_DEVICES; i++)

```



```

        {
            if(measureIIn[j][i] && component[i] != 8)
            {
                memset(buffer,0,50);
                sprintf(buffer,"V%c %d %d DC
0\n",'a'+dummyInCount+dummyOutCount,dummyInVoltageNodeValue[dummyInCount],inputNode[j][i]
);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                dummyInCount++;
            }
            if(measureIOut[j][i] && component[i] != 8)
            {
                memset(buffer,0,50);
                sprintf(buffer,"V%c %d %d DC
0\n",'a'+dummyInCount+dummyOutCount,outputNode[j][i],dummyOutVoltageNodeValue[dummyOutCou
nt]);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                dummyOutCount++;
            }
        }
    }
    memset(buffer,0,50);
    //    sprintf(buffer,".control\ntran 1u 100u 30u 1u\nprint ");
    sprintf(buffer,".control\ntran 10n 100u\nprint ");
    spiceFile.write(buffer,strlen(buffer));
    spiceFile.flush();
    numOfVMeasured = 0;
    numOfIMeasured = 0;
    dummyInCount = 0;
    dummyOutCount = 0;
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured + numOfIMeasured); i++)
        {
            if(measureVIn[j][i] && !nodeVoltageAlreadyMeasured(inputNode[j][i]))
            {
                memset(buffer,0,50);
                sprintf(buffer,"V(%d) ",inputNode[j][i]);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeVoltageMeasured[numOfVMeasured] = inputNode[j][i];
                numOfVMeasured++;
            }
        }
    }
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured + numOfIMeasured); i++)
        {
            if(measureVOut[j][i] && !nodeVoltageAlreadyMeasured(outputNode[j][i]))
            {
                memset(buffer,0,50);
                sprintf(buffer,"V(%d) ",outputNode[j][i]);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeVoltageMeasured[numOfVMeasured] = outputNode[j][i];
                numOfVMeasured++;
            }
        }
    }
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured + numOfIMeasured); i++)
        {
            if(measureIIn[j][i] && !nodeCurrentAlreadyMeasured(inputNode[j][i]))
            {
                if(component[i] == 8)
                {

```

```

        memset(buffer,0,50);
        sprintf(buffer,"I(V%d) ",i);
        spiceFile.write(buffer,strlen(buffer));
        spiceFile.flush();
        nodeCurrentMeasured[numOfIMeasured] = inputNode[j][i];
        numOfIMeasured++;
    }
    else
    {
        memset(buffer,0,50);
        sprintf(buffer,"I(V%c) ",'a'+dummyInCount+dummyOutCount);
        spiceFile.write(buffer,strlen(buffer));
        spiceFile.flush();
        nodeCurrentMeasured[numOfIMeasured] = inputNode[j][i];
        numOfIMeasured++;
        dummyInCount++;
    }
}
}
}
for(j = 0; j < 2; j++)
{
    for(i = 0; i < NUM_DEVICES && 7 > (numOfVMeasured + numOfIMeasured); i++)
    {
        if(measureIOut[j][i] && !nodeCurrentAlreadyMeasured(outputNode[j][i]))
        {
            if(component[i] == 8)
            {
                memset(buffer,0,50);
                sprintf(buffer,"I(V%d) ",i);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeCurrentMeasured[numOfIMeasured] = outputNode[j][i];
                numOfIMeasured++;
            }
            else
            {
                memset(buffer,0,50);
                sprintf(buffer,"I(V%c) ",'a'+dummyInCount+dummyOutCount);
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
                nodeCurrentMeasured[numOfIMeasured] = outputNode[j][i];
                numOfIMeasured++;
                dummyOutCount++;
            }
        }
    }
}
memset(buffer,0,50);
sprintf(buffer,"\n.endc\n.end");
spiceFile.write(buffer,strlen(buffer));
spiceFile.flush();
return;
}
void MainWindow::formatSpiceOutput()
{
    remove("/root/formattedPart1.txt");
    remove("/root/formattedPart2.txt");
    remove("/root/formattedPart3.txt");
    ifstream inFile("/root/results.txt");
    ofstream outFile1("/root/formattedPart1.txt");
    ofstream outFile2("/root/formattedPart2.txt");
    ofstream outFile3("/root/formattedPart3.txt");
    string line;
    int indexFound = 0;
    int i = -2;
    int j = 1;
    int goodLine = 0;
    char indexCount[10];
    while(inFile.good())

```

```

{
    getline(inFile,line);
    line.append("\n");
    if(!indexFound)
    {
        if(line.find("Index") == 0)
        {
            indexFound = 1;
        }
    }
    if(i >= 0)
    {
        sprintf(indexCount,"%d",i);
        if(line.find(indexCount) == 0)
        {
            goodLine = 1;
            i++;
        }
        else
        {
            if(line.find("0") == 0)
            {
                i = 1;
                goodLine = 1;
                j++;
            }
        }
    }
    else
    {
        if(indexFound)
        {
            i++;
        }
    }
    if (i >= 0 && goodLine)
    {
        switch(j)
        {
            case 1:
                outFile1.write(line.c_str(),line.length());
                break;
            case 2:
                outFile2.write(line.c_str(),line.length());
                break;
            case 3:
                outFile3.write(line.c_str(),line.length());
                break;
        }
        goodLine = 0;
    }
}
inFile.close();
outFile1.close();
outFile2.close();
outFile3.close();
return;
}

void MainWindow::gnuPlot()
{
    remove("/root/gnuscript");
    ofstream script("/root/gnuscript");
    char buffer[100];
    char initiate[10] = "plot \\n";
    string fileName;
    int column = 3;
    int j, i, k = 0;
    int voltageCount = 0;
    int currentCount = 0;
    memset(nodeVoltageMeasured, 0, 10);

```

```

memset(nodeCurrentMeasured, 0, 10);
sprintf(buffer, "set term jpeg size 875,400\nset output \"/root/plot.jpg"\n");
script.write(buffer, strlen(buffer));
script.flush();
sprintf(buffer, "set size 1,1\nset origin 0,0\nset key outside\n");
script.write(buffer, strlen(buffer));
script.flush();
// Setup the plot command
if(voltageAndCurrentPlots())
{
    sprintf(buffer, "set y2label 'Current (A)'\nset ytics nomirror\nset y2tics
auto\n");
    script.write(buffer, strlen(buffer));
}
sprintf(buffer, "set xlabel 'Seconds'\nset ylabel 'Volts'\nset x2label 'Voltage
Plots'\n");
script.write(buffer, strlen(buffer));
script.write(initiate, strlen(initiate));
qDebug("%d", numOfVMeasured + numOfIMeasured);
// Add all traces to plot
// Voltage Inputs
for(j = 0; j < 2; j++)
{
    for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
    {
        if(measureVIN[j][i] && !nodeVoltageAlreadyMeasured(inputNode[j][i]))
        {
            fileName = determineFile(k);
            sprintf(buffer, "\"%s\" using 2:%d with linespoints axis x1y1 title
'V(%c) in'", fileName.c_str(), column, i+'A');
            script.write(buffer, strlen(buffer));
            column++;
            k++;
            nodeVoltageMeasured[voltageCount] = inputNode[j][i];
            voltageCount++;
            if(voltageCount + currentCount < numOfVMeasured + numOfIMeasured)
            {
                sprintf(buffer, ", \\n");
                script.write(buffer, strlen(buffer));
            }
        }
        if (column > 5)
        {
            column = 3;
        }
    }
}
// Voltage Outputs
for(j = 0; j < 2; j++)
{
    for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
    {
        if(measureVOut[j][i] && !nodeVoltageAlreadyMeasured(outputNode[j][i]))
        {
            fileName = determineFile(k);
            sprintf(buffer, "\"%s\" using 2:%d with linespoints axis x1y1 title
'V(%c) out'", fileName.c_str(), column, i+'A');
            script.write(buffer, strlen(buffer));
            column++;
            k++;
            nodeVoltageMeasured[voltageCount] = outputNode[j][i];
            voltageCount++;
            if(voltageCount + currentCount < numOfVMeasured + numOfIMeasured)
            {
                sprintf(buffer, ", \\n");
                script.write(buffer, strlen(buffer));
            }
        }
        if (column > 5)
        {

```

```

        column = 3;
    }
}
// Current Inputs
for(j = 0; j < 2; j++)
{
    for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
    {
        if(measureIIn[j][i] && !nodeCurrentAlreadyMeasured(inputNode[j][i]))
        {
            fileName = determineFile(k);
            sprintf(buffer, "\"%s\" using 2:%d with linespoints axis xly2 title 'I(%c) in'", fileName.c_str(), column, i+'A');
            script.write(buffer, strlen(buffer));
            column++;
            k++;
            nodeCurrentMeasured[currentCount] = inputNode[j][i];
            currentCount++;
            if(voltageCount + currentCount < numOfVMeasured + numOfIMeasured)
            {
                sprintf(buffer, ", \\n");
                script.write(buffer, strlen(buffer));
            }
            if (column > 5)
            {
                column = 3;
            }
        }
    }
}
// Current Outputs
for(j = 0; j < 2; j++)
{
    for(i = 0; i < 10 && 7 > voltageCount + currentCount; i++)
    {
        if(measureIOut[j][i] && !nodeCurrentAlreadyMeasured(outputNode[j][i]))
        {
            fileName = determineFile(k);
            sprintf(buffer, "\"%s\" using 2:%d with linespoints axis xly2 title 'I(%c) out'", fileName.c_str(), column, i+'A');
            script.write(buffer, strlen(buffer));
            column++;
            k++;
            nodeCurrentMeasured[currentCount] = outputNode[j][i];
            currentCount++;
            if(voltageCount + currentCount < numOfVMeasured + numOfIMeasured)
            {
                sprintf(buffer, ", \\n");
                script.write(buffer, strlen(buffer));
            }
            if (column > 5)
            {
                column = 3;
            }
        }
    }
}
script.close();
system("gnuplot /root/gnuscript");
return;
}
string MainWindow::determineFile(int i)
{
    string fileName = "";
    switch(i)
    {
        case 0:
        case 1:
        case 2:

```

```

        fileName = "/root/formattedPart1.txt";
        break;
    case 3:
    case 4:
    case 5:
        fileName = "/root/formattedPart2.txt";
        break;
    case 6:
    case 7:
    case 8:
        fileName = "/root/formattedPart3.txt";
        break;
    }
    return fileName;
}
int MainWindow::voltageAndCurrentPlots()
{
    int i, j = 0;
    int voltage, current = 0;
    for(i=0; i < 2; i++)
    {
        for(j=0; j<10; j++)
        {
            if(measureVIN[i][j] || measureVOut[i][j])
            {
                voltage = 1;
                break;
            }
        }
        if(voltage)
        {
            break;
        }
    }
    for(i=0; i < 2; i++)
    {
        for(j=0; j<10; j++)
        {
            if(measureIIn[i][j] || measureIOut[i][j])
            {
                current = 1;
                break;
            }
        }
        if(current)
        {
            break;
        }
    }
    return (current & voltage);
}
int MainWindow::nodeCurrentAlreadyMeasured(int nodeNumber)
{
    qDebug("Node values that have been measured for Current:");
    for(int i = 0; i < 10; i++)
    {
        qDebug("%d", nodeCurrentMeasured[i]);
        if(nodeNumber == nodeCurrentMeasured[i])
        {
            return 1;
        }
    }
    return 0;
}
int MainWindow::nodeVoltageAlreadyMeasured(int nodeNumber)
{
    qDebug("Node values that have been measured for Voltage:");
    for(int i = 0; i < 10; i++)
    {
        qDebug("%d", nodeVoltageMeasured[i]);
    }
}

```

```

        if(nodeNumber == nodeVoltageMeasured[i])
        {
            return 1;
        }
    }
    return 0;
}

void MainWindow::adjustForCurrentMeasurements()
{
    int i,j;
    maxInDummyNodeCount = 0;
    maxOutDummyNodeCount = 0;
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
        {
            if(measureIIn[j][i])
            {
                if(component[i] != VSOURCE)
                {
                    dummyInVoltageNodeValue[maxInDummyNodeCount] = inputNode[j][i];
                    inputNode[j][i] = maxNodeValue;
                    maxInDummyNodeCount++;
                    maxNodeValue++;
                }
            }
        }
    }
    for(j = 0; j < 2; j++)
    {
        for(i = 0; i < NUM_DEVICES; i++)
        {
            if(measureIOut[j][i])
            {
                if(component[i] != VSOURCE)
                {
                    dummyOutVoltageNodeValue[maxOutDummyNodeCount] = outputNode[j][i];
                    outputNode[j][i] = maxNodeValue;
                    maxOutDummyNodeCount++;
                    maxNodeValue++;
                }
            }
        }
    }
}

int MainWindow::measureBoxChecked()
{
    int i;
    for(i = 0; i < NUM_DEVICES; i++)
    {
        if(measureVIN[0][i] || measureVIN[1][i])
        {
            return 1;
        }
        if(measureVOut[0][i] || measureVOut[1][i])
        {
            return 1;
        }
        if(measureIIn[0][i] || measureIIn[1][i])
        {
            return 1;
        }
        if(measureIOut[0][i] || measureIOut[1][i])
        {
            return 1;
        }
    }
    return 0;
}

void MainWindow::on_reset_nets_clicked()

```

```

{
    remove("/root/plot.jpg");
    IO_0_net = -1;
    IO_1_net = -1;
    IO_2_net = -1;
    IO_3_net = -1;
    IO_0_net_2 = -1;
    IO_1_net_2 = -1;
    IO_2_net_2 = -1;
    IO_3_net_2 = -1;
    IO_0_net_3 = -1;
    IO_1_net_3 = -1;
    IO_2_net_3 = -1;
    IO_3_net_3 = -1;
    IO_0_net_4 = -1;
    IO_1_net_4 = -1;
    IO_2_net_4 = -1;
    IO_3_net_4 = -1;
    ui->IO_0->setText("N");
    ui->IO_1->setText("N");
    ui->IO_2->setText("N");
    ui->IO_3->setText("N");
    ui->IO_0_2->setText("N");
    ui->IO_1_2->setText("N");
    ui->IO_2_2->setText("N");
    ui->IO_3_2->setText("N");
    ui->IO_0_3->setText("N");
    ui->IO_1_3->setText("N");
    ui->IO_2_3->setText("N");
    ui->IO_3_3->setText("N");
    ui->IO_0_4->setText("N");
    ui->IO_1_4->setText("N");
    ui->IO_2_4->setText("N");
    ui->IO_3_4->setText("N");
}

```

rs1/communication.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v1.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**
** * Redistributions of source code must retain the above copyright
**   notice, this list of conditions and the following disclaimer.
** * Redistributions in binary form must reproduce the above copyright
**   notice, this list of conditions and the following disclaimer in
**   the documentation and/or other materials provided with the
**   distribution.
** * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**   the names of its contributors may be used to endorse or promote

```



```

**      products derived from this software without specific prior written
**      permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#include "communication.h"
#include <fstream>
#include <string>
using namespace std;
extern int deviceConnected[10];
extern int simulating;
void checkIfValueChanged(int oldValue, int newValue)
{
    if(oldValue != newValue)
    {
        updateDetected = 1;
        qDebug("UPDATE DETECTED between %d and %d!",oldValue,newValue);
    }
    return;
}

```

Appendix H: ReActive-SPICE v2.0 Source Code

rs2/canvas.h

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v2.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**   notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**   notice, this list of conditions and the following disclaimer in
**   the documentation and/or other materials provided with the
**   distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**   the names of its contributors may be used to endorse or promote
**   products derived from this software without specific prior written
**   permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#ifndef EXAMPLE_H
#define EXAMPLE_H
#include <QMenu>
#include <QMainWindow>
#include <QMouseEvent>
#include <QGraphicsView>
#include <QGraphicsScene>
#include <QGraphicsItem>
#include <QDoubleSpinBox>
#include <QSlider>
#include <QPushButton>
#include <QGridLayout>
#include <QLabel>
#include <string>
/*****
** Figure Editor Class Definition

```

```

*****/
class FigureEditor : public QGraphicsView {
    Q_OBJECT
public:
    FigureEditor(QGraphicsScene&, QWidget* parent=0, const char* name=0,
                Qt::WindowFlags f=0);

signals:
    void status(const QString&);
};
/*****
** Component Class Definition
** To make things more C++-like and to modularize, all variables should
** eventually be made private and accessor methods created. For simplicity
** and quick code development, public variables and methods were used liberally.
*****/
class Component : public QObject , public QGraphicsItemGroup {
    Q_OBJECT
public:
    Component(QGraphicsScene* canvas, const QPixmap &pixmap, QPointF poffset,
              QSizeF size, QString &name, QVector<QPointF> node_coords,
              QGraphicsItem *parentItem=0);
    ~Component();

signals:
    void valueSliderChanged(int new_value);
    void fvalueSliderChanged(int new_value);

public slots:
    void setValue(int value);
    void fsetValue(int value);
    void setSpinValue(double value);
    void fsetSpinValue(double value);
    void on_scale_button_clicked();
    void on_fscale_button_clicked();

public:
    // Methods
    QRectF boundingRect() const;
    void paint(QPainter *painter,
               const QStyleOptionGraphicsItem *option, QWidget *widget=0);
    QVariant itemChange(GraphicsItemChange change, const QVariant &value);
    void hoverEnterEvent(QGraphicsSceneHoverEvent * event);
    void hoverLeaveEvent(QGraphicsSceneHoverEvent * event);
    void mousePressEvent (QGraphicsSceneMouseEvent * event);
    void setNodeColor(int node_index, int r, int g, int b);
    void setSize(QSizeF size);
    void setPos(qreal x, qreal y);
    void setNodeConnected(int node_index);
    void removeConnection(int node_index, int index);
    void updateNodeLocations(bool move, QPointF newPos);
    void updateWireLocations();
    int width();
    int height();
    // Variables
    bool controls;
    int current_rotation;
    QString component_name;
    QList<bool> node_connections;
    // NOTE on wire_connections 2-D array (2-D QList):
    // wire_connections[Node index 0,1,2][Wire index 0,1,2...]
    // = <global_wire_connection index (wire #)>
    // The Node index is the node number on the component, for
    // example, Node indices for a resistor: 1,2 (2 terminal comp)
    // example, Node indices for a bjt: 1,2,3 (3 terminal comp)
    // The Wire index is however many wires are connected to the
    // specified node (0,1,2...# of connected wires to node)
    QList<QList<int>> wire_connections;
    QList<QPointF> node_locations;
    QList<QGraphicsRectItem*> node;
    QList<QPointF> rotated_delta;
    QList<int> vprobe_nodes;
    QList<int> iprobe_nodes;
    QList<bool> hover;

```

```

    QList<int> assigned_net;
    QSlider *value_slider;
    QSlider *fvalue_slider;
    QDoubleSpinBox *spin_box;
    QDoubleSpinBox *fspin_box;
    QPushButton *scaler_button;
    QPushButton *fscaler_button;
    QPointF fvalue_slider_offset;
    QPointF fspin_box_offset;
    QPointF fscaler_button_offset;
    QPointF value_slider_offset;
    QPointF spin_box_offset;
    QPointF scaler_offset;
    QPointF scaler_button_offset;
    QPointF identifier_offset;
    QGraphicsTextItem* identifier;
private:
    QSizeF size;
    QPixmap image;
    QPointF poffset;
    QList<QList<bool> > p1;
    QGraphicsScene* canvas;
};
/*****
** Main Class Definition
*****/
class Main : public QMainWindow {
    Q_OBJECT
public:
    Main(QGraphicsScene&, QWidget* parent=0, Qt::WindowFlags f=0);
    QGraphicsScene& canvas;
    void removeWire(int index);
    void removeComponent(int index);
    void rotateComponent(int component_index, int degrees);
    char type;
    QPushButton* simulate_button;
    QPushButton* vprobe_button;
    QPushButton* iprobe_button;
    ~Main();
public slots:
    void createComponent(char);
    void groundButtonClicked();
    void resistorButtonClicked();
    void capacitorButtonClicked();
    void inductorButtonClicked();
    void diodeButtonClicked();
    void opampButtonClicked();
    void bjtButtonClicked();
    void nmosButtonClicked();
    void isourceButtonClicked();
    void vsourceButtonClicked();
    void vsourceACButtonClicked();
    void simulateButtonClicked();
    void vprobeButtonClicked();
    void iprobeButtonClicked();
    void help();
private slots:
    std::string determineFile(int i);
    bool eventFilter(QObject *obj, QEvent *event);
    int formatNetlist();
    int getNet(QPointF node);
    int createNets();
    void aboutQt();
    void deleteComponent(void);
    void rotateComponent(void);
    void clear();
    void init();
    void zoomIn();
    void zoomOut();
    void print();

```

```

void gnuPlot();
void formatSpiceOutput();
void stopSimulating();
void findAllLinks(QList<QPointF> points, int net_index);
private:
    FigureEditor *editor;
    QPointF mStartPoint;
    QMenu* options;
#ifdef Q_OS_SYMBIAN
    QPrinter* printer;
    // NOTE on net 2-D array (2-D QList):
    // net[Net number (index) 0,1,2...][Point index 0,1,2...]
    //      = <QPointF's associated with the net>
    // The Net number is sequential and starts at 0 of course (ground)
    // The Point index also starts at 0 and is as long as the number
    // of points associated with a net
    QList<QList<QPointF> > net;
    int dummy_nets;
    char dummy_sources [2000];
    bool v_and_c;
    bool iflag;
    bool vflag;
    int dummy_total;
#endif
};
#endif

```

rs2/main.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v2.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$
** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**     the names of its contributors may be used to endorse or promote
**     products derived from this software without specific prior written
**     permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

```

```

** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#include <QStatusBar>
#include <QMessageBox>
#include <QMenuBar>
#include <QApplication>
#include <QDesktopWidget>
#include <QImage>
#include <QTimer>
#include <stdlib.h>
#include "canvas.h"
// Main method that starts the simulator
// portedcanvas was the initial Qt demo template chosen, because it already
// had a QGraphicsScene (canvas) created with multiple examples of QGraphicsItems
int main(int argc, char** argv)
{
    // Load all of the image resources (circuit symbols) from the portedcanvas.qrc file
    Q_INIT_RESOURCE(portedcanvas);
    // Create the Qt Application
    QApplication app(argc,argv);
    // Create the QGraphicsScene, this will be the main window of FigureEditor, which
    // is where all of the QGraphicsItems and Objects will be drawn/painted to
    QGraphicsScene canvas;
    // Set the circuit/schematic drawing area to 1020x900, this parameter is a bit
    // finicky and requires you to also set the maximum/minimum height of the central
    // widget in order to function properly
    canvas.setSceneRect(0, 0, 1020, 900);
    // Construct the canvas object
    Main m(canvas);
    // Resize, rename and show the main window
    m.resize(m.sizeHint());
    m.setWindowTitle("ReActive-SPICE v2.0 - Real-time InterActive SPICE Simulator");
    m.show();
    // Connect the lastWindowClosed() signal to the quit() method so that the program
    // can exit gracefully. The quit() method or the Main destructor needs to be
    // improved (implemented) to kill any rogue ngspice or gnuplot processes in case
    // the program needs to be terminated.
    QObject::connect( qApp, SIGNAL(lastWindowClosed()), qApp, SLOT(quit()) );
    // Start the Qt Application
    return app.exec();
}

```

rs2/canvas.cpp

```

/*****
**
** Copyright (C) 2012 Joshua M. O'Hara. Electrical Engineering Department.
** California Polytechnic State University, San Luis Obispo, CA.
** All rights reserved.
** Contact: Josh O'Hara (joshua@oharanetworks.com)
**
** This file is part of the ReActive-SPICE v2.0 Real-time SPICE Simulator.
** This is merely a prototype for a real-time interactive SPICE simulation
** tool and needs much improvement. It is not optimized nor modularized.
** This tool was designed and implemented as part of a Masters Thesis
** Project which was defended in Summer of 2012 at CPSU SLO.
**
** $QT_BEGIN_LICENSE:BSD$

```

```

** You may use this file under the terms of the BSD license as follows:
**
** "Redistribution and use in source and binary forms, with or without
** modification, are permitted provided that the following conditions are
** met:
**   * Redistributions of source code must retain the above copyright
**     notice, this list of conditions and the following disclaimer.
**   * Redistributions in binary form must reproduce the above copyright
**     notice, this list of conditions and the following disclaimer in
**     the documentation and/or other materials provided with the
**     distribution.
**   * Neither the name of CPSU SLO and its Subsidiary(-ies) nor
**     the names of its contributors may be used to endorse or promote
**     products derived from this software without specific prior written
**     permission.
**
** THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
** "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
** LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
** A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
** OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
** SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
** LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
** DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
** THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
** (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
** OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."
** $QT_END_LICENSE$
**
*****/

#include <QDateTime>
#include <QMainWindow>
#include <QStatusBar>
#include <QMessageBox>
#include <QMenuBar>
#include <QApplication>
#include <QPainter>
#include <QPrinter>
#include <QPrintDialog>
#include <QLabel>
#include <QImage>
#include <QProgressDialog>
#include <QPixmap>
#include <QMouseEvent>
#include <QStyleOptionGraphicsItem>
#include <QGraphicsPixmapItem>
#include <QDebug>
#include <QProgressBar>
#include <QtCore/qmath.h>
#include <QGraphicsSceneMouseEvent>
#include "canvas.h"
#include <fstream>
#include <string>
// Create global variables
// Temporary line variables for wire connections
QPointF temp_line_start;
QPointF temp_line_mid;
QPointF temp_line_end;
QGraphicsLineItem* temp_line;
QGraphicsLineItem* temp_line1;
// Create the jpg pixmap item to display the
// waveform outputs generated by ngspice/gnuplot
QGraphicsPixmapItem* plot;
// Create a global variables to control whether the
// simulation is currently running/updating or not
bool simulating = false;
bool updateDetected = false;
// Temporary variables to keep track of what component
// the temporary lines are starting and ending on

```

```

int line_start_comp;
int line_end_comp = -1;
// Global wire connections (lines)
// Example: global_wire_connections[Global wire index 0,1,2...] = [wire1, wire2,
wire3...]
//
// Global wire index = 0 , 1 , 2 ...
QList<QGraphicsLineItem*> global_wire_connections;
// There are 2 cosmetic lines for each single global wire connection (line)
// Example: global_wire_connection_lines[Global wire index 0,1,2...][Line index 0,1] =
Cosmetic line
QList<QList<QGraphicsLineItem*> > global_wire_connection_lines;
// Global component list (keeps track of all the components)
QList<Component*> comp;
// This message variable is used in the status bar of the application
// to not only indicate what current mode the user is in, but also
// if there are any circuit build/simulation errors present
QString message = "Start";
// Schematic modes
bool delete_mode = false;
bool rotate_mode = false;
bool place_mode = false;
bool vprobe_mode = false;
bool iprobe_mode = false;
// ComponentType class: used for assisting in the creation of multiple
// components systematically. Essentially just a structure, however Qt
// does not explicitly allow the use of structs the way we needed it to.
class ComponentType
{
public:
    ComponentType(QString name, QPixmap pixmap, double scale,
                  QVector<QPointF> node_coords, QPointF poffset);
public:
    // Name of the component (resistor, capacitor, inductor)
    QString name;
    // QPixmap/Image/Symbol of the component
    QPixmap pixmap;
    // Size of the pixmap/image/symbol
    QSizeF size;
    // Coordinates of the node hit boxes on the component
    QVector<QPointF> node_coords;
    // Offset of the pixmap/image/symbol
    QPointF poffset;
};
// Component Type constructor, saves all of the input parameters as private
// instance variables
ComponentType::ComponentType(QString name, QPixmap pixmap, double scale,
                              QVector<QPointF> node_coords, QPointF poffset)
{
    this->name = name;
    this->pixmap = pixmap;
    this->size = QSizeF(scale*pixmap.width(), scale*pixmap.height());
    this->node_coords = node_coords;
    this->poffset = poffset;
}
// Component class: Enables the creation of multiple component objects with
// their individual properties and info. The component class utilizes all of
// the ComponentType class's variables upon component object creation
Component::Component(QGraphicsScene* canvas, const QPixmap &pixmap,
                    QPointF poffset, QSizeF size, QString &name,
                    QVector<QPointF> node_coords, QGraphicsItem *parentItem)
: QGraphicsItemGroup(new QGraphicsItemGroup(new QGraphicsPixmapItem(pixmap,
                                                                    parentItem)))
{
    // The canvas object is given to each of the components
    // to enable items to be drawn on the canvas
    this->canvas = canvas;
    // Save instance variables
    image = pixmap;
    this->poffset = poffset;
    component_name = name;
}

```



```

// If the component has the ability to be controlled
controls = false;
// The identifier is the SPICE label that will go next to each component
// See below for individual initialization
// identifier = new QGraphicsTextItem(component_name);
// Go through and create each of the nodes on the component
// The last four coordinates in the node_coords array are
// the coordinate offsets of the scaler button, value slider
// spinbox and identifier. This allows these controls to be
// dynamically positioned, depending on the size and shape
// of the component
for(int i = 0 ; i < node_coords.size() - 4 ; i++)
{
    //QDebug() << "Creating node box for: " << node_coords[i];
    // Create the wire connection number index list (empty)
    QList<int> temp;
    wire_connections.append(temp);
    // Create p1 list for line/wire dynamics
    // If p1 = true, the node is starting
    // the connection, if p1 = false, another
    // component has already started the wire
    // connection and so it must be the final
    // or end connection
    QList<bool> p1_temp;
    p1_temp.append(true);
    p1.append(p1_temp);
    // A boolean list to keep track of if the
    // mouse pointer is hovering inside the
    // node box or not
    QList<bool> hover_temp;
    hover_temp.append(false);
    hover.append(hover_temp);
    // Create the actual rectangle that outlines
    // the node boundaries. The node is a 9x9 pixel
    // square located at the node coordinates specified
    // upon component object creation
    QGraphicsRectItem* rect_temp;
    rect_temp = new QGraphicsRectItem(QRectF(node_coords[i], QSizeF(9,9)));
    node.append(rect_temp);
    // Append the location of the center of the node rectangle
    // to the node_locations list. The node locations are based
    // off the scene location of the center of the node rectangle
    // as opposed to the location of the center of the node rectangle
    // with respect to the component object itself.
    // node_locations.append(node[i]->rect().center());
    node_locations.append(node[i]->sceneBoundingRect().center());
    // Set the node initially connected to no other components
    node_connections.append(false);
    // Allow the node to accept hover events and scene position
    // changes so that the main function can tell if the node
    // rectangles have moved or if the mouse pointer is hovering
    // over the node bounding rectangle
    node[i]->setAcceptsHoverEvents(true);
    node[i]->setFlag(QGraphicsItem::ItemSendsScenePositionChanges);
    // Add the newly created node rectangle to this component
    // (which is also a QGraphicsItem) so that they act as one
    // The node will be reparented to this component, but its
    // position and transformation relative to the scene will
    // stay intact.
    addToGroup(node[i]);
}
// Set the current rotation of the node to 0 degrees
// (the component is not initially rotated)
current_rotation = 0;
// Get the component controls offsets from the last 4
// indices of the node_coords array. This was a late
// add on and eventually should use its own array to
// store the controls offsets, but we are lazy...
scaler_button_offset = node_coords[node_coords.size()-4];
value_slider_offset = node_coords[node_coords.size()-3];

```

```

spin_box_offset = node_coords[node_coords.size()-2];
identifier_offset = node_coords[node_coords.size()-1];
// Create the component value slider
value_slider = new QSlider(Qt::Vertical);
value_slider->setRange(1,999);
// If the component is an AC voltage source, we need to add
// the frequency parameter controls to the component
if(component_name == QString("vacsource"))
{
    fvalue_slider_offset = value_slider_offset + QPointF(20,0);
    fspin_box_offset = spin_box_offset + QPointF(0,30);
    fscaler_button_offset = scaler_button_offset + QPointF(0,30);
    fvalue_slider = new QSlider(Qt::Vertical);
    fvalue_slider->setRange(1,999);
    fspin_box = new QDoubleSpinBox();
    fspin_box->setRange(0,999.9);
    fscaler_button = new QPushButton();
}
// Create the component value display (spinbox) controls
// and scaler button
spin_box = new QDoubleSpinBox();
spin_box->setRange(0,999.9);
scaler_button = new QPushButton();
// Create the component scalar button control depending on
// the type of component. The identifier will be what
// the SPICE netlist uses as the reference designator,
// so we will make it unique, yet intuitive.
if(component_name == QString("resistor"))
{
    identifier = new QGraphicsTextItem("R" + QString::number(comp.length()));
    value_slider->setValue(30);
    setValue(30);
    scaler_button->setText("k Ohms");
    controls = true;
}
else if(component_name == QString("capacitor"))
{
    identifier = new QGraphicsTextItem("C" + QString::number(comp.length()));
    value_slider->setValue(10);
    setValue(10);
    scaler_button->setText("u Farads");
    controls = true;
}
else if(component_name == QString("inductor"))
{
    identifier = new QGraphicsTextItem("L" + QString::number(comp.length()));
    value_slider->setValue(50);
    setValue(50);
    scaler_button->setText("m Henrys");
    controls = true;
}
else if(component_name == QString("diode"))
{
    identifier = new QGraphicsTextItem("D" + QString::number(comp.length()));
}
else if(component_name == QString("opamp"))
{
    identifier = new QGraphicsTextItem("X" + QString::number(comp.length()));
}
else if(component_name == QString("nmos"))
{
    identifier = new QGraphicsTextItem("M" + QString::number(comp.length()));
}
else if(component_name == QString("bjt"))
{
    identifier = new QGraphicsTextItem("Q" + QString::number(comp.length()));
}
else if(component_name == QString("vsource"))
{
    identifier = new QGraphicsTextItem("VDC" + QString::number(comp.length()));
}

```

```

        value_slider->setValue(80);
        setValue(80);
        scaler_button->setText("m Volts");
        controls = true;
    }
    else if(component_name == QString("vacsource"))
    {
        identifier = new QGraphicsTextItem("V" + QString::number(comp.length()));
        value_slider->setValue(80);
        setValue(80);
        scaler_button->setText("m Volts RMS");
        fvalue_slider->setValue(1);
        fsetValue(1);
        fscaler_button->setText("k Hertz");
        controls = true;
    }
    else if(component_name == QString("isource"))
    {
        identifier = new QGraphicsTextItem("I" + QString::number(comp.length()));
        value_slider->setValue(80);
        setValue(80);
        scaler_button->setText("m Amps");
        controls = true;
    }
    else
    {
        identifier = new QGraphicsTextItem("G" + QString::number(comp.length()));
    }
    // Set all the component controls to have a transparent
    // background and initially hidden on the canvas
    value_slider->setAttribute(Qt::WA_NoBackground);
    spin_box->setAttribute(Qt::WA_NoBackground);
    scaler_button->setAttribute(Qt::WA_NoBackground);
    value_slider->setVisible(false);
    spin_box->setVisible(false);
    scaler_button->setVisible(false);
    if(component_name == QString("vacsource"))
    {
        fvalue_slider->setAttribute(Qt::WA_NoBackground);
        fspin_box->setAttribute(Qt::WA_NoBackground);
        fscaler_button->setAttribute(Qt::WA_NoBackground);
        fvalue_slider->setVisible(false);
        fspin_box->setVisible(false);
        fscaler_button->setVisible(false);
        QGraphicsProxyWidget *fvalue_slider_proxy = canvas->addWidget(fvalue_slider);
        QGraphicsProxyWidget *fspin_box_proxy = canvas->addWidget(fspin_box);
        QGraphicsProxyWidget *fscaler_button_proxy = canvas->addWidget(fscaler_button);
    }
    // Create proxy widgets and have the component controls added to the canvas
    QGraphicsProxyWidget *value_slider_proxy = canvas->addWidget(value_slider);
    QGraphicsProxyWidget *spin_box_proxy = canvas->addWidget(spin_box);
    QGraphicsProxyWidget *scaler_button_proxy = canvas->addWidget(scaler_button);
    // Add the identifier (component name or ref designator) to the
    // component GraphicsItem group so that it moves with it
    addToGroup(identifier);
    identifier->setPos(identifier_offset);
    // All item caching is disabled. QGraphicsItem::paint() is
    // called every time the item needs redrawing. We don't need
    // to use this feature because there is not a large amount
    // of graphics data being drawn to the canvas
    setCacheMode(NoCache);
    // Make the component movable and enable the component
    // to send scene position changes when moved around (to
    // keep track of location.
    setFlag(QGraphicsItem::ItemIsMovable);
    setFlag(QGraphicsItem::ItemSendsScenePositionChanges);
    // Set the size of the component object
    setSize(size);
}
// Component deconstructor, doesn't need anything in it now...

```

```

Component::~Component()
{
}
// Store the size instance variable so that it can
// be manipulated outside this class
void Component::setSize(QSizeF size)
{
    this->size = size;
}
// Allow other classes to get the width of the pixmap
// Currently an unused function, but in case we need it...
int Component::width()
{
    return image.width();
}
// Allow other classes to get the height of the pixmap
// Currently an unused function, but in case we need it...
int Component::height()
{
    return image.height();
}
// Set the position of the component. Everytime this method
// is called, the position of the component is set, and all
// of the node and controls locations are updated to enable
// them to stay with the component.
void Component::setPos(qreal x, qreal y)
{
    // Call the parent method
    QGraphicsItemGroup::setPos(x,y);
    // As the component moves, move the node/wire locations as well
    updateNodeLocations(false, QPointF(0,0));
    // Move all the controls with the component
    value_slider->move(QPoint(x,y)+value_slider_offset.toPoint());
    spin_box->move(QPoint(x,y)+spin_box_offset.toPoint());
    scaler_button->move(QPoint(x,y)+scaler_button_offset.toPoint());
    if(component_name == QString("vacsource"))
    {
        fvalue_slider->move(QPoint(x,y)+fvalue_slider_offset.toPoint());
        fspin_box->move(QPoint(x,y)+fspin_box_offset.toPoint());
        fscaler_button->move(QPoint(x,y)+fscaler_button_offset.toPoint());
    }
}
// Update the position of all node locations on the component
void Component::updateNodeLocations(bool move, QPointF newPos)
{
    // For each node
    for(int i = 0 ; i < node_locations.length() ; i++)
    {
        // If move is true, then component is being dragged (moved) and the
        // new node locations need to be relative to the newPos and the
        // boundingRect() of the node (in local component coordinates)
        if(move)
        {
            // If the component is not rotated, just set the new location to the
            // newPos + the center of the node bounding rectangle in local coords
            if(rotated_delta.isEmpty())
            {
                node_locations[i] = newPos+node[i]->boundingRect().center();
            }
            // If the component is rotated, the difference between the original
            // node location and the current rotated node location (rotated_delta)
            // in local coordinates must be subtracted from the new position
            else
            {
                node_locations[i] = newPos+node[i]->boundingRect().center()-
rotated_delta[i];
                //qDebug() << "rotated node:" << node_locations[i] << "delta: " <<
rotated_delta[i];
            }
        }
    }
}

```

```

        // If the component is not being moved, then just set the node locations to the
center
        // coordinate of the node bounding rectangle in scene (global) coordinates
        else
        {
            node_locations[i] = node[i]->sceneBoundingRect().center();
        }
    }
    // If the component has wires attached to it, move them to the new node locations
    if(wire_connections.length() != 0)
    {
        updateWireLocations();
    }
}
// Update the position of all wire connections attached to the component
// (Move connected wires with component)
void Component::updateWireLocations()
{
    // For each node attached to the current component:
    for(int n = 0 ; n < wire_connections.length() ; n++)
    {
        // For each wire attached to the current node:
        for(int i = 0 ; i < wire_connections[n].length() ; i++)
        {
            // If the current node is connected, [and the current wire has not actually
            // been created yet?? Necessary check?]
            if(node_connections[n] && (global_wire_connections.length() !=
wire_connections[n][i]))
            {
                // Remove the wires from the canvas
                // Originally, a single wire was used to connect components, but for
visual
                // enhancement, a 3-point wire connection was used (2 wires). These two
wires
                // are purely cosmetic (visual), all of the connection analysis and
circuit
                // topology information is still taken from the global_wire_connections
array
                // which is now invisible to the user. Using the single wire makes
connection
                // logic much simpler.
                //canvas->removeItem(global_wire_connections[wire_connections[n][i]]);
                canvas->
>removeItem(global_wire_connection_lines[wire_connections[n][i]][0]);
                canvas->
>removeItem(global_wire_connection_lines[wire_connections[n][i]][1]);
                // If the current node was the first point in the wire connection, adjust
                // P1 of the wire to be at the "moved" node location
                if(p1[n][i])
                {
                    //QDebug() << "Adjusting P1";
                    // Move the logical single wire connection (hidden to the user)
                    global_wire_connections[wire_connections[n][i]]-
>setLine(QPointF(node_locations[n],global_wire_connections[wire_connections[n][i]]-
>line().p2()));
                    // Move the cosmetic 2-wire connection (visible to the user)
                    temp_line_mid =
                    QPointF(node_locations[n].x(),global_wire_connections[wire_connections[n][i]]-
>line().p2().y());
                    global_wire_connection_lines[wire_connections[n][i]][0]-
>setLine(QPointF(node_locations[n], temp_line_mid));
                    global_wire_connection_lines[wire_connections[n][i]][1]-
>setLine(QPointF(temp_line_mid, global_wire_connection_lines[wire_connections[n][i]][1]-
>line().p2()));
                    //QDebug() << global_wire_connections[wire_connections[i]]-
>line().p1();
                }
                // If the current node was the end point in the wire connection, adjust
                // P2 of the wire to be at the "moved" node location
                else

```

```

        {
            //QDebug() << "Adjusting P2";
            // Move the logical single wire connection (hidden to the user)
            global_wire_connections[wire_connections[n][i]]-
>setLine(QLineF(global_wire_connections[wire_connections[n][i]]-
>line().p1(),node_locations[n]));
            // Move the cosmetic 2-wire connection (visible to the user)
            temp_line_mid =
QPointF(global_wire_connections[wire_connections[n][i]]-
>line().p1().x(),node_locations[n].y());
            global_wire_connection_lines[wire_connections[n][i]][0]-
>setLine(QLineF(global_wire_connection_lines[wire_connections[n][i]][0]->line().p1(),
temp_line_mid));
            global_wire_connection_lines[wire_connections[n][i]][1]-
>setLine(QLineF(temp_line_mid, node_locations[n]));
            //QDebug() << global_wire_connections[wire_connections[i]]-
>line().p2();
        }
        // Add the new lines/wires back to the canvas
        //global_wire_connections[wire_connections[n][i]] = canvas-
>addLine(QLineF(global_wire_connections[wire_connections[n][i]]-
>line().p1(),global_wire_connections[wire_connections[n][i]]->line().p2()));
        global_wire_connection_lines[wire_connections[n][i]][0] = canvas-
>addLine(QLineF(global_wire_connection_lines[wire_connections[n][i]][0]-
>line().p1(),global_wire_connection_lines[wire_connections[n][i]][0]->line().p2()));
        global_wire_connection_lines[wire_connections[n][i]][1] = canvas-
>addLine(QLineF(global_wire_connection_lines[wire_connections[n][i]][1]-
>line().p1(),global_wire_connection_lines[wire_connections[n][i]][1]->line().p2()));
        // Send the wires to the back so they are not placed in front of the node
rectangles
        global_wire_connection_lines[wire_connections[n][i]][0]->setZValue(-1);
        global_wire_connection_lines[wire_connections[n][i]][1]->setZValue(-1);
    }
}
}
// Rotate the specified component by specified number of degrees
// (whatever the current rotation of the component is +90 degrees)
void Main::rotateComponent(int component_index, int degrees)
{
    // Save all component center node scene locations to a temporary QPointF array
    QList<QPointF> temp;
    for(int nl = 0 ; nl < comp[component_index]->node_locations.length() ; nl++)
    {
        temp.append(comp[component_index]->node[nl]->sceneBoundingRect().center());
    }
    // Rotate the symbol/component
    // qFloor is needed to only rotate components an integer amount of pixels, not float
    // This is to keep the node rect hit boxes hoverable...not sure why...
    QTransform transform = QTransform();
    transform.translate(qFloor(comp[component_index]->boundingRect().width()/2),
qFloor(comp[component_index]->boundingRect().height()/2));
    transform.rotate(degrees);
    transform.translate(qFloor(-comp[component_index]->boundingRect().width()/2),
qFloor(-comp[component_index]->boundingRect().height()/2));
    comp[component_index]->setTransform(transform);
    // Transform the identifier with the component (same technique, except negative
degrees are used)
    transform = QTransform();
    transform.translate(qFloor(comp[component_index]->identifier-
>boundingRect().width()/2), qFloor(comp[component_index]->identifier-
>boundingRect().height()/2));
    transform.rotate(-degrees);
    transform.translate(qFloor(-comp[component_index]->identifier-
>boundingRect().width()/2), qFloor(-comp[component_index]->identifier-
>boundingRect().height()/2));
    comp[component_index]->identifier->setTransform(transform);
    // If the value slider was vertical, make it horizontal and adjust its position
    if(comp[component_index]->value_slider->orientation() == Qt::Vertical)
    {

```

```

        comp[component_index]->value_slider_offset.setY(comp[component_index]-
>value_slider_offset.y()-20);
        comp[component_index]->value_slider->setOrientation(Qt::Horizontal);
        comp[component_index]->value_slider->move(comp[component_index]->pos().x()-
comp[component_index]->value_slider_offset.x(), comp[component_index]->pos().y());
    }
    // If the value slider was horizontal, make it vertical and adjust its position
    else
    {
        comp[component_index]->value_slider_offset.setY(comp[component_index]-
>value_slider_offset.y()+20);
        comp[component_index]->value_slider->setOrientation(Qt::Vertical);
        comp[component_index]->value_slider->move(comp[component_index]-
>pos().x()+comp[component_index]->value_slider_offset.x(), comp[component_index]-
>pos().y());
    }
    // Swap the width and height of the value slider to conform to the new orientation
    int tw = comp[component_index]->value_slider->width();
    comp[component_index]->value_slider->setFixedWidth(comp[component_index]-
>value_slider->height());
    comp[component_index]->value_slider->setFixedHeight(tw);
    // Set the components current rotation after the 90 degree rotation
    // Mod it with 360 to avoid getting too large and to keep the numbers
    // known (as there are only 4 possible values: 0, 90, 180 and 270)
    comp[component_index]->current_rotation = (comp[component_index]->current_rotation +=
90)%360;
    //qDebug() << comp[component_index]->current_rotation;
    // For each of the nodes
    for(int nl = 0 ; nl < comp[component_index]->node_locations.length() ; nl++)
    {
        // If the rotated_delta variable hasn't been set (no prior rotations),
        // or the rotated_delta for the current node location hasn't been appended
        // to the rotated_delta list yet, append the rotated_delta for the node
        if(comp[component_index]->rotated_delta.isEmpty() || comp[component_index]-
>rotated_delta.length() == nl)
        {
            comp[component_index]->rotated_delta.append(temp[nl]-comp[component_index]-
>node[nl]->sceneBoundingRect().center());
        }
        // If the rotated_delta variable has already been created for the current
        // node location, then just modify it (no need to append a new value)
        else
        {
            comp[component_index]->rotated_delta[nl] += temp[nl]-comp[component_index]-
>node[nl]->sceneBoundingRect().center();
        }
        //qDebug() << "rotated delta" << comp[component_index]->rotated_delta[nl];
    }
    // Update node/wire locations and redraw the screen (update()) if necessary
    comp[component_index]->updateNodeLocations(false, QPointF(0,0));
    update();
}
// Remove the wire at the specified global wire index
void Main::removeWire(int index)
{
    //qDebug() << "Removing wire at index:" << index;
    // Go through all of the componenets
    for(int i = 0 ; i < comp.length() ; i++)
    {
        // Go through all of the nodes on each component
        for(int n = 0 ; n < comp[i]->wire_connections.length() ; n++)
        {
            //qDebug() << "INITIAL wire_connections: " << comp[i]->wire_connections[n];
            // If the current component contains the index of the connection to be
deleted
            if(comp[i]->wire_connections[n].contains(index))
            {
                // Remove the connection
                comp[i]->removeConnection(n, comp[i]->wire_connections[n].indexOf(index));
            }
        }
    }
}

```

```

        // Shift down all of the indices after the specified global index
        for(int s = 0 ; s < comp[i]->wire_connections[n].length() ; s++)
        {
            if(comp[i]->wire_connections[n][s] > index)
            {
                comp[i]->wire_connections[n][s]--;
            }
        }
        // If the node is no longer connected to anything, change the
        // color back to white and set the node_connection flag false
        if(comp[i]->wire_connections[n].isEmpty())
        {
            comp[i]->setNodeColor(n,255,255,255);
            comp[i]->node_connections[n] = false;
        }
        //qDebug() << "wire_connections after: " << comp[i]->wire_connections[n];
    }
}
// Remove both the cosmetic lines from the canvas
//canvas.removeItem(global_wire_connections[index]);
canvas.removeItem(global_wire_connection_lines[index][0]);
canvas.removeItem(global_wire_connection_lines[index][1]);
// Remove the cosmetic lines at the specified index
global_wire_connection_lines.removeAt(index);
// Remove the logical line at the specified index
global_wire_connections.removeAt(index);
}
// Set the specified node_index of a component as connected (append the most
// recently added logical wire connection and then set the node_connection
// flag true)
void Component::setNodeConnected(int node_index)
{
    wire_connections[node_index].append(global_wire_connections.length()-1);
    node_connections[node_index] = true;
}
// Note: index parameter has already been translated to the actual list index of
wire_connections
void Component::removeConnection(int node_index, int index)
{
    // Remove the index where the value is stored
    wire_connections[node_index].removeAt(index);
}
// Remove component at the specified index in the comp[] QList.
void Main::removeComponent(int index)
{
    // Create temporary buffer to store the identifier
    char buffer [5] = {0};
    // For each node on component
    for(int n = 0 ; n < comp[index]->wire_connections.length() ; n++)
    {
        // Remove each wire attached to the current node
        int initial_connections = comp[index]->wire_connections[n].length();
        for(int w = 0 ; w < initial_connections ; w++)
        {
            removeWire(comp[index]->wire_connections[n][0]);
        }
    }
    // Shift the identifier (reference designator #) down for all following components
    for(int i = index + 1 ; i < comp.length() ; i++)
    {
        memset(buffer,0,5);
        // Store the existing identifier
        const char* temp = comp[i]->identifier->toPlainText().toLocal8Bit().data();
        // If the second character is a D, then ref des was VDC and is more than
        // just a one char change. Else, keep the char at index 0 and shift down the #
        if(temp[1] == 'D')
        {
            sprintf(buffer,"%cD%d",temp[0],i-1);
        }
        else

```



```

        {
            sprintf(buffer, "%c%d", temp[0], i-1);
        }
        // Update the identifier text string
        comp[i]->identifier->setPlainText(QString(buffer));
    }
    // Hide the controls (otherwise the canvas will keep them
    // and once the component is removed, the main function
    // will no longer have access to these proxy widgets
    // and will not be able to remove them.)
    comp[index]->value_slider->hide();
    comp[index]->spin_box->hide();
    comp[index]->scaler_button->hide();
    // Remove the component from the canvas
    canvas.removeItem(comp[index]);
    // Remove the component object from the comp[] list
    comp.removeAt(index);
}
// Set the RGB color of the inside of the node bounding box (rectangle) for the specified
node index
void Component::setNodeColor(int node_index, int r, int g, int b)
{
    // If the node_index specified is -1, that is a flag
    // to set all nodes of a component to a specified color
    if(node_index == -1)
    {
        for(int i = 0 ; i < node.length() ; i++)
        {
            node[i]->setBrush(QBrush(QColor(r,g,b)));
        }
    }
    // Set the node at node_index to the specified RGB color
    else
    {
        node[node_index]->setBrush(QBrush(QColor(r,g,b)));
    }
}
// Create a boundingRect for the node starting at an origin
// of poffset (top-left corner) and the size (9x79)
QRectF Component::boundingRect() const
{
    // Return defined 'size' and located rect
    return QRectF(poffset, size);
}
// Component paint function, the way Qt actually draws the component to the screen
// The pixmap stays in the image boundingRect and is drawn to the screen
void Component::paint(QPainter *painter, const QStyleOptionGraphicsItem*, QWidget*)
{
    // Scale QGraphicsPixmapItem to wanted 'size' and keep the aspect ratio using
    boundingRect()
    painter->drawPixmap(boundingRect().toRect(), image);
}
// Function to detect a mouse hover enter event: Performs an
// action when the mouse enters a node rectangle
void Component::hoverEnterEvent(QGraphicsSceneHoverEvent * event)
{
    // If we are not in delete, rotate or place mode
    if(!delete_mode && !rotate_mode && !place_mode)
    {
        // The current component being hovered over is this one
        line_end_comp = comp.indexOf(this);
        // For each node
        for(int i = 0 ; i < node_connections.length() ; i++)
        {
            // If the mouse is in your bounding rectangle
            if(node[i]->rect().contains(event->pos()))
            {
                // And you you are not already connected
                if(!node_connections[i])
                {

```

```

        // Set node color to blue
        setNodeColor(i,0,200,200);
    }
    // If in voltage probe mode and the iprobe_nodes
    // array does not contain the current node in it
    else if(vprobe_mode && !iprobe_nodes.contains(i))
    {
        // Set node color to red
        setNodeColor(i,255,0,0);
    }
    // If in current probe mode and the vprobe_nodes
    // array does not contain the current node in it
    else if(iprobe_mode && !vprobe_nodes.contains(i))
    {
        // Set node color to yellow
        setNodeColor(i,255,255,0);
    }
    // Set the hover variable for your node true
    hover[i] = true;
}
}
}
// Function to detect a mouse hover leave event: Performs an
// action when the mouse leaves a node rectangle
void Component::hoverLeaveEvent(QGraphicsSceneHoverEvent * event)
{
    // The current component is no longer being hovered over
    line_end_comp = -1;
    // For each node
    for(int i = 0 ; i < node_connections.length() ; i++)
    {
        // If you are not connected
        if(!node_connections[i])
        {
            // Set node color back to white
            setNodeColor(i,255,255,255);
        }
        // If you are in either probe mode and the node wasn't
        // added to either probe list, set back to default color
        else if((vprobe_mode || iprobe_mode) && !vprobe_nodes.contains(i) &&
!iprobe_nodes.contains(i))
        {
            // Set node color back to black
            setNodeColor(i,0,0,0);
        }
        // Set the hover variable for your node false
        hover[i] = false;
    }
}
// Function to detect a mouse press event within a component:
// Main line/wire creation/connection logic
void Component::mousePressEvent (QGraphicsSceneMouseEvent * event)
{
    // If the temporary line starting point has not been created
    if(temp_line_start.isNull())
    {
        // You can move the component
        setFlag(QGraphicsItem::ItemIsMovable);
    }
    // If the left button is pressed within the component area
    if (event->button() == Qt::LeftButton) {
        // For each node
        for(int n = 0 ; n < node.length() ; n++)
        {
            // If the current node is being hovered over
            // and you are not in delete or rotate mode
            if(hover[n] && !delete_mode && !rotate_mode)
            {

```

```

        // If in either probe mode and the node is connected and no lines have
        been started
        if((iprobe_mode || vprobe_mode) && node_connections[n] &&
temp_line_start.isNull())
        {
            // If in voltage probe mode and the iprobe_nodes
            // array does not contain the current node in it
            if(vprobe_mode && !iprobe_nodes.contains(n))
            {
                // If the node is set to already probe voltage
                // remove the probe and reset the node color
                if(vprobe_nodes.contains(n))
                {
                    // Remove the probe
                    vprobe_nodes.removeAt(vprobe_nodes.indexOf(n));
                    updateDetected = true;
                    // Set the node color back to black
                    setNodeColor(n,0,0,0);
                }
                // The probe does not exist at the node yet
                else
                {
                    // Add the probe
                    vprobe_nodes.append(n);
                    // Flag a simulation update
                    updateDetected = true;
                    // Set the node color to red
                    setNodeColor(n,255,0,0);
                }
            }
            // If in current probe mode and the vprobe_nodes
            // array does not contain the current node in it
            else if(iprobe_mode && !vprobe_nodes.contains(n))
            {
                // If the node is set to already probe current
                // remove the probe and reset the node color
                if(iprobe_nodes.contains(n))
                {
                    // Remove the probe
                    iprobe_nodes.removeAt(iprobe_nodes.indexOf(n));
                    updateDetected = true;
                    // Set the node color back to black
                    setNodeColor(n,0,0,0);
                }
                // The probe does not exist at the node yet
                else
                {
                    // Add the probe
                    iprobe_nodes.append(n);
                    // Flag a simulation update
                    updateDetected = true;
                    // Set the node color to yellow
                    setNodeColor(n,255,255,0);
                }
            }
        }
    }
    else
    {
        // If the temporary line starting point has not been created
        if(temp_line_start.isNull())
        {
            // Set pl variable of the node at a new wire index to true
            pl[n][wire_connections[n].length()] = true;
            // Set the component the line is starting at to "this" one
            line_start_comp = comp.indexOf(this);
            //qDebug() << "Line start at node " << n << "on" <<
component_name;

            // Set the node connection variable to true
            node_connections[n] = true;

```

```

// Set the starting point of the wire connection to this node
location
temp_line_start = node_locations[n];
// Disable component movement
setFlag(QGraphicsItem::ItemIsMovable, false);
// If the node is not being probed, set it to black (connected)
if(!iprobe_nodes.contains(n) && !vprobe_nodes.contains(n))
{
    // Set the node color to black
    setNodeColor(n, 0, 0, 0);
}
}
// Else, if the temporary line ending point has not been created
else if(temp_line_end.isNull())
{
    // Set the found and same_component variables to false
    // for duplicate and invalid wire connection detections
    bool found = false;
    bool same_component = false;
    // Set the ending point of the wire connection to this node
location
temp_line_end = node_locations[n];
// For each node
for(int nc = 0 ; nc < node.length() ; nc++)
{
    // If the line started at any other node
    // of the same component, flag it
    if(temp_line_start == node_locations[nc])
    {
        same_component = true;
        // Delete the endpoint of the temporary line
        temp_line_end = QPointF();
    }
}
// If the start and end points of the line are not the same
// and the line is not being drawn to the same component
if((temp_line_end != temp_line_start) && !same_component)
{
    // If there are other wire connections
    if(!global_wire_connections.isEmpty())
    {
        //qDebug() << "Checking, wire_connections.length() = " <<
wire_connections[n].length();
        //qDebug() << "wire_connections: " <<
wire_connections[n];
        //qDebug() << "global wire_connections length: " <<
global_wire_connections.length();
        // For each wire connection on the current node
        for(int j=0 ; j < wire_connections[n].length() ; j++)
        {
            //qDebug() << "global_wire_connections: " <<
global_wire_connections[n][j];
            // If a line is already drawn between the same nodes
            // of components, don't create another one...
            if((temp_line_end ==
global_wire_connections[wire_connections[n][j]]->line().p1()
&& temp_line_start ==
global_wire_connections[wire_connections[n][j]]->line().p2())
|| (temp_line_end ==
global_wire_connections[wire_connections[n][j]]->line().p2()
&& temp_line_start ==
global_wire_connections[wire_connections[n][j]]->line().p1()))
            {
                qDebug("Wire already exists!");
                // Delete the endpoint of the temporary line
                temp_line_end = QPointF();
                // Set the line found variable to true
                found = true;
            }
        }
    }
}
}

```

```

    }
    // If a duplicate wire was not found, create a wire
connection
    if(!found)
    {
        // Since you are the component the wire ended on, you are
P2, not P1
        p1[n][wire_connections[n].length()] = false;
        // Remove the temporary cosmetic lines
        canvas->removeItem(temp_line);
        canvas->removeItem(temp_line1);
        // Add the line/wire to the global wire connections array
and to the canvas
        //global_wire_connections.append(canvas-
>addLine(QLineF(temp_line_start,temp_line_end)));
        // Create a logical line
        global_wire_connections.append(new
QGraphicsLineItem(QLineF(temp_line_start,temp_line_end)));
        // Draw the actual lines (cosmetic) to the canvas
        QList<QGraphicsLineItem*> t_graphics_item;
        // Determine the mid-point of the cosmetic line
intersection
        if(temp_line_mid.x() <= temp_line_start.x() + 9 &&
temp_line_mid.x() >= temp_line_start.x() - 9)
        {
            temp_line_mid.setX(temp_line_start.x());
        }
        else
        {
            temp_line_mid.setX(temp_line_end.x());
        }
        if(temp_line_mid.y() <= temp_line_start.y() + 9 &&
temp_line_mid.y() >= temp_line_start.y() - 9)
        {
            temp_line_mid.setY(temp_line_start.y());
        }
        else
        {
            temp_line_mid.setY(temp_line_end.y());
        }
        // Add the two cosmetic lines to the canvas
        t_graphics_item.append(canvas-
>addLine(QLineF(temp_line_start,temp_line_mid)));
        t_graphics_item.append(canvas-
>addLine(QLineF(temp_line_mid,temp_line_end)));
        // Send the wires to the back so they are not placed in
front of the node rectangles
        t_graphics_item[0]->setZValue(-1);
        t_graphics_item[1]->setZValue(-1);
        global_wire_connection_lines.append(t_graphics_item);
        // For each component
        for(int j = 0 ; j < comp.length() ; j++)
        {
            // For each node
            for(int l = 0 ; l < comp[j]->node_locations.length()
; l++)
            {
                // If the node location equals the starting point
of the line
                if(comp[j]->node_locations[l] == temp_line_start)
                {
                    // Tell the other component's node they are
connected
                    comp[j]->setNodeConnected(l);
                    //qDebug() << comp[j]->component_name <<
"connected on node " << l;
                }
            }
        }
    }
}

```



```

        // Return the newPos to the parent method
        return newPos;
    }
    // Return to the parent method if not a position change
    return QGraphicsItem::itemChange(change, value);
}
// Component slot to set the value of the spin box from the slider
void Component::setValue(int value)
{
    spin_box->setValue(value);
    updateDetected = true;
}
// Component slot to set the fvalue of the fspin box from the fslider
void Component::fsetValue(int value)
{
    fspin_box->setValue(value);
    updateDetected = true;
}
// Component slot to set the value of the slider from the spin box
void Component::setSpinValue(double value)
{
    value_slider->setValue((int)value);
}
// Component slot to set the fvalue of the fslider from the fspin box
void Component::fsetSpinValue(double value)
{
    fvalue_slider->setValue((int)value);
}
// Figure Editor constructor, acts as a parent to the canvas
// and is the central widget of the main window. (Main class)
FigureEditor::FigureEditor(
    QGraphicsScene& c, QWidget* parent,
    const char* name, Qt::WindowFlags f) :
    QGraphicsView(&c, parent)
{
    setObjectName(name);
    setWindowFlags(f);
    setRenderHints(QPainter::Antialiasing | QPainter::SmoothPixmapTransform);
}
// EventFilter for the class Main
bool Main::eventFilter(QObject *obj, QEvent *event)
{
    // If it is a keypress event
    if (event->type() == QEvent::KeyPress)
    {
        // Cast the event as a key event
        QKeyEvent *keyEvent = static_cast<QKeyEvent *>(event);
        // If Escape button pressed
        if (keyEvent->key() == Qt::Key_Escape)
        {
            // Stop the simulation and reset the mode
            stopSimulating();
            message = "None";
            delete_mode = false;
            rotate_mode = false;
            // Set the cursor back to the default
            // Although the cursors stopped showing
            // up differently after a while...??
            setCursor(Qt::ArrowCursor);
            // If a line/wire has been started
            if(!temp_line_start.isNull())
            {
                // For each component
                for(int j = 0 ; j < comp.length() ; j++)
                {
                    // For each node
                    for(int n = 0 ; n < comp[j]->node_locations.length() ; n++)
                    {
                        // If the line started on the current node
                        if(comp[j]->node_locations[n] == temp_line_start)

```

```

        {
            // If there are no other wire connections on
            // the current node, set the node color back to
            // black and the node_connection flag to false
            if(comp[j]->wire_connections[n].length() == 0)
            {
                comp[j]->setNodeColor(n,255,255,255);
                comp[j]->node_connections[n] = false;
            }
            break;
        }
    }
    // Remove the temp line start/end points
    temp_line_start = QPointF();
    temp_line_end = QPointF();
    // Remove the temp cosmetic lines from the canvas
    canvas.removeItem(temp_line);
    canvas.removeItem(temp_line1);
}
// If a component is being placed and Escape is pressed
else if(place_mode)
{
    // Remove the temp comp from the screen and
    // stop the place mode
    canvas.removeItem(comp[comp.length()-1]);
    comp.removeAt(comp.length()-1);
    place_mode = false;
    message = "None";
}
}
// If Delete key pressed
else if(keyEvent->key() == Qt::Key_Delete)
{
    // Enable delete mode
    delete_mode = true;
    message = "Remove Wire/Component";
    setCursor(Qt::ForbiddenCursor);
}
// If 'R/r' key pressed
else if(keyEvent->key() == Qt::Key_R)
{
    // Enable rotate mode
    rotate_mode = true;
    message = "Rotate Component";
    setCursor(Qt::SizeHorCursor);
}
}
// If the event is a left double click and we are not rotating a component or making
wire connections
else if ((event->type() == QEvent::MouseButtonDbClick) && !rotate_mode &&
temp_line_start.isNull())
{
    // Static cast as a mouse move event
    QMouseEvent *mouseEvent = static_cast<QMouseEvent*>(event);
    // For each component
    for(int c = 0 ; c < comp.length() ; c++)
    {
        // If it has controls
        if(comp[c]->controls)
        {
            // If the mouse double clicked on the current component
            if(QRectF(comp[c]->pos(),QSizeF(comp[c]->boundingRect().width(),comp[c]-
>boundingRect().height())).contains(mouseEvent->posF()))
            {
                // If the controls are visible, make them invisible
                if(comp[c]->value_slider->isVisible())
                {
                    comp[c]->value_slider->setVisible(false);
                    comp[c]->spin_box->setVisible(false);
                }
            }
        }
    }
}

```



```

        comp[c]->scaler_button->setVisible(false);
        if(comp[c]->component_name == QString("vacsource"))
        {
            comp[c]->fvalue_slider->setVisible(false);
            comp[c]->fspin_box->setVisible(false);
            comp[c]->fscaler_button->setVisible(false);
        }
    }
    // Else the controls are invisible, make them visible
    else
    {
        comp[c]->value_slider->setVisible(true);
        comp[c]->spin_box->setVisible(true);
        comp[c]->scaler_button->setVisible(true);
        if(comp[c]->component_name == QString("vacsource"))
        {
            comp[c]->fvalue_slider->setVisible(true);
            comp[c]->fspin_box->setVisible(true);
            comp[c]->fscaler_button->setVisible(true);
        }
    }
}

// If the event is a mouse button press event
else if (event->type() == QEvent::MouseButtonPress)
{
    // If in place mode
    if(place_mode)
    {
        // Disable place mode
        place_mode = false;
        message = "None";
    }
    // If in delete mode
    else if(delete_mode)
    {
        //qDebug("Tried to delete line!");
        // Static cast the event as a mouse event
        QMouseEvent *mouseEvent = static_cast<QMouseEvent*>(event);
        // If it was a left button press
        if(mouseEvent->button() == Qt::LeftButton)
        {
            // For each component
            for(int c = 0 ; c < comp.length() ; c++)
            {
                // If the mouse left clicked on the current component
                if(QRectF(comp[c]->pos(), QSizeF(comp[c]-
>boundingRect().width(), comp[c]->boundingRect().height())).contains((mouseEvent-
>posF())))
                {
                    // Remove the clicked component and disable delete mode
                    removeComponent(c);
                    delete_mode = false;
                    message = "None";
                    break;
                }
            }
            // For each global wire connection (logical)
            for(int j = 0 ; j < global_wire_connections.length() ; j++)
            {
                // Adjust the size of the temp wire/line bounding rects to make line
                // Makes it so you can be within ballpark (+/-5 pixels on all sides
                // of wire)
                QRectF b_rect0 = QRectF(global_wire_connection_lines[j][0]-
>boundingRect());

```

```

        b_rect0.setSize(QSizeF(global_wire_connection_lines[j][0]-
>boundingRect().width()+5,global_wire_connection_lines[j][0]-
>boundingRect().height()+5));
        QRectF b_rect1 = QRectF(global_wire_connection_lines[j][1]-
>boundingRect());
        b_rect1.setSize(QSizeF(global_wire_connection_lines[j][1]-
>boundingRect().width()+5,global_wire_connection_lines[j][1]-
>boundingRect().height()+5));
        // If the mouse cursor is within the new temp wire/line bounding
rects
        if((b_rect0.contains(mouseEvent->posF())) ||
(b_rect1.contains(mouseEvent->posF())))
        {
            // Remove the logical wire at the current index and disable
delete mode
            // The removeWire() function will also remove the 2 cosmetic
wires who
            // are associated with the single logical wire specified
            removeWire(j);
            delete_mode = false;
            message = "None";
            setCursor(Qt::ArrowCursor);
            break;
        }
    }
}
// If in rotate mode
else if(rotate_mode)
{
    // Static cast the event as a mouse event
    QMouseEvent *mouseEvent = static_cast<QMouseEvent*>(event);
    // If it was a left button press
    if(mouseEvent->button() == Qt::LeftButton)
    {
        // For each component
        for(int j = 0 ; j < comp.length() ; j++)
        {
            // If the current component contains the mouse cursor
            if((comp[j]->sceneBoundingRect().contains(mouseEvent->posF())))
            {
                // Rotate the component 90 degrees CCW
                rotateComponent(j,comp[j]->current_rotation+90);
                break;
            }
        }
    }
}
}
// If the event is a mouse move event
else if (event->type() == QEvent::MouseMove)
{
    // Static cast the event as a mouse event
    QMouseEvent *mouseEvent = static_cast<QMouseEvent*>(event);
    // If a line/wire has been started
    if (!temp_line_start.isNull()) {
        // Remove the existing temp lines if there are any
        // Comment these 2 lines out to see a pretty cool
        // glitch when drawing wires! :)
        canvas.removeItem(temp_line);
        canvas.removeItem(temp_line1);
        // If the line is inside a valid area (node rect) on a component
        if(line_end_comp != -1)
        {
            // If the hover value is true on the ending component node
            if(comp[line_end_comp]->hover.indexOf(true) != -1)
            {
                // If the y component of the ending component node is less than the y
component

```

```

// of the starting component node or if the y component of the start
component node
// is less than the y component of the scene position of the ending
component node
// This will give the effect of a constant X/Y locked effect when
drawing wires
// Definitely needs some work, sometimes the lines do not orient the
way they
// are naturally supposed to. This part is the "auto" snap feature
if(comp[line_end_comp]->node_locations[comp[line_end_comp]-
>hover.indexOf(true)].y() < temp_line_start.y() || temp_line_start.y() <
comp[line_end_comp]->scenePos().y())
{
    // Constant X
    //qDebug("Constant X");
    // Calculate the temp midpoint of the 2 temp lines
    temp_line_mid = QPointF(temp_line_start.x(),mouseEvent-
>pos().y());
    // Add the new cosmetic temp lines
    temp_line = canvas.addLine(QPointF(temp_line_start,
temp_line_mid));
    temp_line1 = canvas.addLine(QPointF(temp_line_mid, mouseEvent-
>pos()));
}
else
{
    // Constant Y
    //qDebug("Constant Y");
    // Calculate the temp midpoint of the 2 temp lines
    temp_line_mid = QPointF(mouseEvent-
>pos().x(),temp_line_start.y());
    // Add the new cosmetic temp lines
    temp_line = canvas.addLine(QPointF(temp_line_start,
temp_line_mid));
    temp_line1 = canvas.addLine(QPointF(temp_line_mid, mouseEvent-
>pos()));
}
}
// Else the line is not ending on a valid node location yet
// So instead of following the wire/line autosnap functionality,
// the temp wires are just drawn with constant x or constant y
// values depending on the location of the mouse position relative
// to the line/wire starting position
else
{
    // If the y component of the current mouse position is less than the
    // y component of the temp line starting point
    if(mouseEvent->pos().y() < temp_line_start.y())
    {
        // Constant X
        //qDebug("Constant X");
        // Calculate the temp midpoint of the 2 temp lines
        temp_line_mid = QPointF(temp_line_start.x(),mouseEvent->pos().y());
        // Add the new cosmetic temp lines
        temp_line = canvas.addLine(QPointF(temp_line_start, temp_line_mid));
        temp_line1 = canvas.addLine(QPointF(temp_line_mid, mouseEvent-
>pos()));
    }
    // Else the y component of the current mouse position must be greater
    // y component of the temp line starting point
    else
    {
        // Constant Y
        //qDebug("Constant Y");
        // Calculate the temp midpoint of the 2 temp lines
        temp_line_mid = QPointF(mouseEvent->pos().x(),temp_line_start.y());
        // Add the new cosmetic temp lines
        temp_line = canvas.addLine(QPointF(temp_line_start, temp_line_mid));

```

```

        temp_line1 = canvas.addLine(QLineF(temp_line_mid, mouseEvent-
>pos()));
    }
    }
    // If in place mode
    else if(place_mode)
    {
        // Move the component being placed to the current mouse location
        comp[comp.length()-1]->setPos(mouseEvent->pos().x(),mouseEvent->pos().y());
    }
    // Update the status bar with messages (shows mouse cursor location as well)
    statusBar()->showMessage(QString("Grid Position (%1,%2): Current Mode:
%3").arg(mouseEvent->pos().x()).arg(mouseEvent->pos().y()).arg(message));
    }
    return false;
}
// Main constructor (parent of FigureEditor and canvas)
Main::Main(QGraphicsScene& c, QWidget* parent, Qt::WindowFlags f) :
    QMainWindow(parent, f),
    canvas(c)
{
    // Create the editor and pass the canvas object to it
    editor = new FigureEditor(canvas,this);
    // Install the event filter to capture mouse events from the
    // Main class (qApp)
    qApp->installEventFilter(this);
    // Create a menu bar, which is semi-functional and still needs major work
    QMenuBar* menu = menuBar();
    QMenu* file = new QMenu("&File", menu );
    file->addAction("&New schematic", this, SLOT(clear()), Qt::CTRL+Qt::Key_N);
    file->addAction("E&xit", qApp, SLOT(quit()), Qt::CTRL+Qt::Key_Q);
    menu->addMenu(file);
    QMenu* edit = new QMenu("&Edit", menu );
    edit->addAction("&Rotate Component", this, SLOT(rotateComponent()),
Qt::ALT+Qt::Key_R);
    edit->addAction("&Delete Component", this, SLOT(deleteComponent()),
Qt::ALT+Qt::Key_D);
    menu->addMenu(edit);
    QMenu* view = new QMenu("&View", menu );
    view->addAction("&Zoom in", this, SLOT(zoomIn()), Qt::CTRL+Qt::Key_Plus);
    view->addAction("Zoom &out", this, SLOT(zoomOut()), Qt::CTRL+Qt::Key_Minus);
    menu->addMenu(view);
    menu->addSeparator();
    QMenu* help = new QMenu("&Help", menu );
    help->addAction("&About", this, SLOT(help()), Qt::Key_F1);
    menu->addMenu(help);
    // Create the status bar
    statusBar();
    // Set the central widget as the editor object
    // and set the min/max height of the FigureEditor
    // window. Those parameters determine the screensize
    // or boundaries of the canvas. Very finnick...
    setCentralWidget(editor);
    editor->setMinimumHeight(905);
    editor->setMaximumHeight(905);
    // Call the initialization function
    init();
}
// Rotate component method
void Main::rotateComponent(void)
{
    rotate_mode = true;
    message = "Rotate Component";
}
// Delete component method
void Main::deleteComponent(void)
{
    delete_mode = true;
    message = "Remove Wire/Component";
}

```

```

}
// Create component method
void Main::createComponent(char type)
{
    qDebug("Creating component...");
    QVector<QPointF> node_coords;
    message = "None";
    delete_mode = false;
    switch(type)
    {
        // node_coords STRUCTURE:
        // node_coords.append(node1)
        // node_coords.append(node2)
        // node_coords.append(node3)
        // ...
        // node_coords.append(nodeX)
        // node_coords.append(Scaler Button Offset)
        // node_coords.append(Slider Bar Offset)
        // node_coords.append(Spin Box Offset)
        // node_coords.append(Identifier Offset)
        // ComponentType object: ComponentType(identifier/name,
        //                                     component symbol pixmap (in resources),
        //                                     size/scaling of the component symbol,
        //                                     the node_coords vector,
        //                                     the pixmap offset (for moving around the
        //                                     image to keep
        //                                     floating point locations.)
        //                                     the node coordinates away from
        // Ground
        case 'g':
        {
            node_coords.append(QPointF(18,0));
            node_coords.append(QPointF(150,0));
            node_coords.append(QPointF(30,0));
            node_coords.append(QPointF(50,0));
            node_coords.append(QPointF(-25,30));
            ComponentType ground = ComponentType(QString("ground"),
            QPixmap(":/images/ground.png"), .6,node_coords,QPointF(-.5,0));
            comp.append(new
            Component(&canvas,ground.pixmap,ground.poffset,ground.size,ground.name,ground.node_coords
            ,0));
            break;
        }
        // Resistor
        case 'r':
        {
            node_coords.append(QPointF(10,0));
            node_coords.append(QPointF(10,88));
            node_coords.append(QPointF(140,0));
            node_coords.append(QPointF(30,0));
            node_coords.append(QPointF(50,0));
            node_coords.append(QPointF(-25,30));
            ComponentType resistor = ComponentType(QString("resistor"),
            QPixmap(":/images/resistor.png"), .5,node_coords,QPointF(1,0));
            comp.append(new
            Component(&canvas,resistor.pixmap,resistor.poffset,resistor.size,resistor.name,resistor.n
            ode_coords,0));
            break;
        }
        // Capacitor
        case 'c':
        {
            node_coords.append(QPointF(16,0));
            node_coords.append(QPointF(16,88));
            node_coords.append(QPointF(150,0));
            node_coords.append(QPointF(40,0));
            node_coords.append(QPointF(60,0));
            node_coords.append(QPointF(-30,35));
            ComponentType capacitor = ComponentType(QString("capacitor"),
            QPixmap(":/images/capacitor.png"), .5,node_coords,QPointF(1.25,0));

```

```

        comp.append(new
Component(&canvas, capacitor.pixmap, capacitor.poffset, capacitor.size, capacitor.name, capaci
tor.node_coords, 0));
        break;
    }
    // Inductor
    case 'i':
    {
        node_coords.append(QPointF(3, -1));
        node_coords.append(QPointF(3, 88));
        node_coords.append(QPointF(140, 0));
        node_coords.append(QPointF(30, 0));
        node_coords.append(QPointF(50, 0));
        node_coords.append(QPointF(-25, 30));
        ComponentType inductor = ComponentType(QString("inductor"),
QPixmap(":/images/inductor.png"), .5, node_coords, QPointF(-.75, 0));
        comp.append(new
Component(&canvas, inductor.pixmap, inductor.poffset, inductor.size, inductor.name, inductor.n
ode_coords, 0));
        break;
    }
    // Diode
    case 'd':
    {
        node_coords.append(QPointF(9, 0));
        node_coords.append(QPointF(9, 88));
        node_coords.append(QPointF(150, 0));
        node_coords.append(QPointF(30, 0));
        node_coords.append(QPointF(50, 0));
        node_coords.append(QPointF(-30, 35));
        ComponentType diode = ComponentType(QString("diode"),
QPixmap(":/images/diode.png"), .5, node_coords, QPointF(-.5, 0));
        comp.append(new
Component(&canvas, diode.pixmap, diode.poffset, diode.size, diode.name, diode.node_coords, 0));
        break;
    }
    // Opamp
    case 'o':
    {
        node_coords.append(QPointF(0, 32));
        node_coords.append(QPointF(0, 65));
        node_coords.append(QPointF(145, 48));
        node_coords.append(QPointF(150, 0));
        node_coords.append(QPointF(30, 0));
        node_coords.append(QPointF(50, 0));
        node_coords.append(QPointF(75, 5));
        ComponentType opamp = ComponentType(QString("opamp"),
QPixmap(":/images/opamp.png"), .5, node_coords, QPointF(0, 0));
        comp.append(new
Component(&canvas, opamp.pixmap, opamp.poffset, opamp.size, opamp.name, opamp.node_coords, 0));
        break;
    }
    // NMOS
    case 'n':
    {
        node_coords.append(QPointF(44, 0));
        node_coords.append(QPointF(0, 43));
        node_coords.append(QPointF(44, 88));
        node_coords.append(QPointF(150, 0));
        node_coords.append(QPointF(30, 0));
        node_coords.append(QPointF(50, 0));
        node_coords.append(QPointF(5, 0));
        ComponentType nmos = ComponentType(QString("nmos"),
QPixmap(":/images/nmos.png"), .5, node_coords, QPointF(0, 0));
        comp.append(new
Component(&canvas, nmos.pixmap, nmos.poffset, nmos.size, nmos.name, nmos.node_coords, 0));
        break;
    }
    // BJT
    case 'b':

```

```

        {
            node_coords.append(QPointF(46,0));
            node_coords.append(QPointF(0,43));
            node_coords.append(QPointF(46,88));
            node_coords.append(QPointF(150,0));
            node_coords.append(QPointF(30,0));
            node_coords.append(QPointF(50,0));
            node_coords.append(QPointF(5,0));
            ComponentType bjt = ComponentType(QString("bjt"),
QPixmap(":/images/bjt.png"), .5,node_coords,QPointF(-.5,0));
            comp.append(new
Component(&canvas,bjt.pixmap,bjt.poffset,bjt.size,bjt.name,bjt.node_coords,0));
            break;
        }
        // Current source (isource)
        case 's':
        {
            node_coords.append(QPointF(18,0));
            node_coords.append(QPointF(18,88));
            node_coords.append(QPointF(150,0));
            node_coords.append(QPointF(40,0));
            node_coords.append(QPointF(60,0));
            node_coords.append(QPointF(-25,35));
            ComponentType isource = ComponentType(QString("isource"),
QPixmap(":/images/isource.png"), .5,node_coords,QPointF(-.5,0));
            comp.append(new
Component(&canvas,isource.pixmap,isource.poffset,isource.size,isource.name,isource.node_c
oords,0));
            break;
        }
        // Voltage source (DC)
        case 'v':
        {
            node_coords.append(QPointF(19,0));
            node_coords.append(QPointF(19,88));
            node_coords.append(QPointF(150,0));
            node_coords.append(QPointF(40,0));
            node_coords.append(QPointF(60,0));
            node_coords.append(QPointF(-45,35));
            ComponentType vsource = ComponentType(QString("vsource"),
QPixmap(":/images/dc.png"), .5,node_coords,QPointF(-.5,0));
            comp.append(new
Component(&canvas,vsource.pixmap,vsource.poffset,vsource.size,vsource.name,vsource.node_c
oords,0));
            break;
        }
        // Voltage source (AC)
        case 'a':
        {
            node_coords.append(QPointF(19,0));
            node_coords.append(QPointF(19,88));
            node_coords.append(QPointF(180,0));
            node_coords.append(QPointF(40,0));
            node_coords.append(QPointF(90,0));
            node_coords.append(QPointF(-25,35));
            ComponentType vacsource = ComponentType(QString("vacsource"),
QPixmap(":/images/ac.png"), .5,node_coords,QPointF(-.5,0));
            comp.append(new
Component(&canvas,vacsource.pixmap,vacsource.poffset,vacsource.size,vacsource.name,vacsou
rce.node_coords,0));
            break;
        }
        default:
        {
            qDebug("Oops! That's not a component!");
            return;
        }
    }
    // Move the component to a random position on the canvas...not sure why I put this
    here...

```

```

    comp[comp.length()-1]-
>setPos(qrand()%int(canvas.width()),qrand()%int(canvas.height()));
// Set node highlight color to white (-1 means all node indices)
comp[comp.length()-1]->setNodeColor(-1,255,255,255);
// Signal/Slot Connections for component class
connect(comp[comp.length()-1]-
>value_slider,SIGNAL(valueChanged(int)),comp[comp.length()-1],SLOT(setValue(int)));
connect(comp[comp.length()-1]-
>spin_box,SIGNAL(valueChanged(double)),comp[comp.length()-1],SLOT(setSpinValue(double)));
connect(comp[comp.length()-1]->scaler_button,SIGNAL(clicked()),comp[comp.length()-
1],SLOT(on_scale_button_clicked()));
if(comp[comp.length()-1]->component_name == QString("vacsource"))
{
    connect(comp[comp.length()-1]-
>fvalue_slider,SIGNAL(valueChanged(int)),comp[comp.length()-1],SLOT(fsetValue(int)));
connect(comp[comp.length()-1]-
>fspin_box,SIGNAL(valueChanged(double)),comp[comp.length()-
1],SLOT(fsetSpinValue(double)));
connect(comp[comp.length()-1]-
>fscaler_button,SIGNAL(clicked()),comp[comp.length()-
1],SLOT(on_fscale_button_clicked()));
}
// Add component to canvas
canvas.addItem(comp[comp.length()-1]);
// Enable place mode which allows the movement and placement of the newly created
component
place_mode = true;
message = "Place Component";
}
// Initialization Slot
void Main::init()
{
    // Create button header
    QPushButton* ground_button = new QPushButton(QString("Ground"));
    QPushButton* resistor_button = new QPushButton(QString("Resistor"));
    QPushButton* capacitor_button = new QPushButton(QString("Capacitor"));
    QPushButton* inductor_button = new QPushButton(QString("Inductor"));
    QPushButton* diode_button = new QPushButton(QString("Diode"));
    QPushButton* opamp_button = new QPushButton(QString("Opamp"));
    QPushButton* nmos_button = new QPushButton(QString("NMOS"));
    QPushButton* bjt_button = new QPushButton(QString("BJT"));
    QPushButton* isource_button = new QPushButton(QString("DC Current Source"));
    QPushButton* vsource_button = new QPushButton(QString("DC Voltage Source"));
    QPushButton* vsourceac_button = new QPushButton(QString("AC Voltage Source"));
    QGridLayout* grid_layout = new QGridLayout();
    grid_layout->addWidget(vsource_button,0,0);
    grid_layout->addWidget(vsourceac_button,0,1);
    grid_layout->addWidget(isource_button,0,2);
    grid_layout->addWidget(ground_button,0,3);
    grid_layout->addWidget(resistor_button,1,0);
    grid_layout->addWidget(capacitor_button,1,1);
    grid_layout->addWidget(inductor_button,1,2);
    grid_layout->addWidget(diode_button,1,3);
    grid_layout->addWidget(opamp_button,1,4);
    grid_layout->addWidget(nmos_button,1,5);
    grid_layout->addWidget(bjt_button,1,6);
    QWidget* widget = new QWidget();
    widget->setAttribute(Qt::WA_NoBackground);
    widget->setLayout(grid_layout);
    widget->move(QPoint(0,0));
    QGraphicsProxyWidget *button_proxy = canvas.addWidget(widget);
    // Create simulate, vprobe, iprobe buttons and line divisor
    simulate_button = new QPushButton(QString("Start Simulation"));
    vprobe_button = new QPushButton(QString("Voltage Probe"));
    iprobe_button = new QPushButton(QString("Current Probe"));
    // Add the buttons to the canvas
    canvas.addWidget(simulate_button);
    canvas.addWidget(vprobe_button);
    canvas.addWidget(iprobe_button);
    // Move the buttons to the specified location

```



```

    simulate_button->move(QPoint(5,470));
    vprobe_button->move(QPoint(150,470));
    iprobe_button->move(QPoint(280,470));
    // Add the horizontal line divider
    QGraphicsLineItem* line = new
QGraphicsLineItem(QPointF(5,460),QPointF(1015,460));
    canvas.addItem(line);
    // Connect buttons with their respective slots
    connect(ground_button,SIGNAL(clicked()),this,SLOT(groundButtonClicked()));
    connect(resistor_button,SIGNAL(clicked()),this,SLOT(resistorButtonClicked()));
    connect(capacitor_button,SIGNAL(clicked()),this,SLOT(capacitorButtonClicked()));
    connect(inductor_button,SIGNAL(clicked()),this,SLOT(inductorButtonClicked()));
    connect(diode_button,SIGNAL(clicked()),this,SLOT(diodeButtonClicked()));
    connect(opamp_button,SIGNAL(clicked()),this,SLOT(opampButtonClicked()));
    connect(nmos_button,SIGNAL(clicked()),this,SLOT(nmosButtonClicked()));
    connect(bjt_button,SIGNAL(clicked()),this,SLOT(bjtButtonClicked()));
    connect(isource_button,SIGNAL(clicked()),this,SLOT(isourceButtonClicked()));
    connect(vsource_button,SIGNAL(clicked()),this,SLOT(vsourceButtonClicked()));
    connect(vsourceac_button,SIGNAL(clicked()),this,SLOT(vsourceACButtonClicked()));
    connect(simulate_button,SIGNAL(clicked()),this,SLOT(simulateButtonClicked()));
    connect(vprobe_button,SIGNAL(clicked()),this,SLOT(vprobeButtonClicked()));
    connect(iprobe_button,SIGNAL(clicked()),this,SLOT(iprobeButtonClicked()));
}
// Header button slots
void Main::groundButtonClicked()
{
    createComponent('g');
}
void Main::resistorButtonClicked()
{
    createComponent('r');
}
void Main::capacitorButtonClicked()
{
    createComponent('c');
}
void Main::inductorButtonClicked()
{
    createComponent('i');
}
void Main::diodeButtonClicked()
{
    createComponent('d');
}
void Main::opampButtonClicked()
{
    createComponent('o');
}
void Main::nmosButtonClicked()
{
    createComponent('n');
}
void Main::bjtButtonClicked()
{
    createComponent('b');
}
void Main::isourceButtonClicked()
{
    createComponent('s');
}
void Main::vsourceButtonClicked()
{
    createComponent('v');
}
void Main::vsourceACButtonClicked()
{
    createComponent('a');
}
// Simulate button slot
void Main::simulateButtonClicked()

```

```

{
    // If current probe mode
    if(iprobe_mode)
    {
        // Disable iprobe_mode and
        // return the button color
        // to the default color
        iprobe_mode = false;
        iprobe_button->setStyleSheet("");
    }
    // If voltage probe mode
    if(vprobe_mode)
    {
        // Disable vprobe_mode and
        // return the button color
        // to the default color
        vprobe_mode = false;
        vprobe_button->setStyleSheet("");
    }
    // If simulate button text is set to Start
    if(simulate_button->text() == QString("Start Simulation"))
    {
        // Start the simulation
        message = "Simulating";
        // Change the simulate_button text
        simulate_button->setText("Stop Simulation");
        // Create the nets and make sure there are no errors
        qDebug("Creating Nets");
        if(!createNets())
        {
            // Error occurred and was sent to the status
            // bar messages. Most likely a ground symbol
            // wasn't included or there is an unconnected
            // node. Stop the simulation.
            simulate_button->setText("Start Simulation");
            return;
        }
        // Initially set updateDetected and simulating flags
        // to true
        updateDetected = true;
        simulating = true;
        // While the simulating flag is true
        while(simulating)
        {
            // If either in delete or place mode
            if(delete_mode || place_mode)
            {
                // Stop the simulation
                stopSimulating();
            }
            // If an update was detected
            // (circuit measurement/value changed)
            if(updateDetected)
            {
                // Create the netlist and make sure there are no errors
                qDebug("Writing Netlist");
                if(!formatNetlist())
                {
                    // Error occurred and was sent to the status
                    // bar messages. Stop the simulation.
                    // Usually a circuit topology error or
                    // other netlist error.
                    stopSimulating();
                    return;
                }
            }
            qDebug("Run Spice");
            system("ngspice -b -o /root/results.txt /root/netlist.cir");
            qDebug("Format the output file");
            formatSpiceOutput();
            qDebug("GNUPlot the waveforms");
        }
    }
}

```

```

        gnuPlot();
        // Reset the update detected flag
        updateDetected = false;
    }
    // Process the events in the while loop so that
    // the GUI stays responsive and processes its events
    QApplication::processEvents();
    // Remove the plot if there is one
    canvas.removeItem(plot);
    // Add the updated plot to the canvas
    plot = canvas.addPixmap(QPixmap("/root/plot.jpg"));
    // Move the plot to specified position
    plot->setPos(50,500);
}
}
else
{
    // Stop the simulation if "Stop" clicked
    stopSimulating();
}
}
// Stop simulating slot
void Main::stopSimulating()
{
    simulating = false;
    simulate_button->setText("Start Simulation");
    message = "Simulation stopped";
}
// Voltage Probe button slot
void Main::vprobeButtonClicked()
{
    if(iprobe_mode)
    {
        iprobe_mode = false;
        iprobe_button->setStyleSheet("");
    }
    if(vprobe_mode)
    {
        vprobe_mode = !vprobe_mode;
        vprobe_button->setStyleSheet("");
        message = "None";
    }
    else
    {
        vprobe_mode = !vprobe_mode;
        vprobe_button->setStyleSheet("background-color: rgb(255, 0, 0); color: rgb(255, 255, 255)");
        message = "Voltage Probe";
    }
}
// Current Probe button slot
void Main::iprobeButtonClicked()
{
    if(vprobe_mode)
    {
        vprobe_mode = false;
        vprobe_button->setStyleSheet("");
    }
    if(iprobe_mode)
    {
        iprobe_mode = !iprobe_mode;
        iprobe_button->setStyleSheet("");
        message = "None";
    }
    else
    {
        iprobe_mode = !iprobe_mode;
        iprobe_button->setStyleSheet("background-color: rgb(255, 255, 0);");
        message = "Current Probe";
    }
}

```

```

}
// Destroy main function
Main::~Main()
{
    #if !defined(Q_OS_SYMBIAN)
        delete printer;
    #endif
}
// Clear all components from the screen
void Main::clear()
{
    for(int i = 0 ; i < comp.length() ; i++)
    {
        removeComponent(i);
    }
}
// Help slot
void Main::help()
{
    static QMessageBox* about = new QMessageBox( "ReActive-Spice v2.0 - Real-time
Interactive SPICE Simulator",
        "<h3>ReActive-Spice v2.0 - Real-time Interactive SPICE Simulator Help</h3>"
        "<ul>"
            "<li> Press r to rotate components."
            "<li> Press \"delete\" to remove components."
            "<li> Press \"Esc\" to stop rotating/deleting/placing components."
            "<li> Draw lines between the nodes to create a circuit."
            "<li> Drag the components around."
            "<li> Read the code!"
        "</ul>", QMessageBox::Information, 1, 0, 0, this, 0);
    about->setButtonText( 1, "Dismiss" );
    about->show();
}
// About slot
void Main::aboutQt()
{
    QMessageBox::aboutQt( this, "Qt Canvas Example" );
}
// Zoom in slot
void Main::zoomIn()
{
    editor->scale( 2.0, 2.0 );
}
// Zoom out slot
void Main::zoomOut()
{
    editor->scale( 0.5, 0.5 );
}
// Print slot
void Main::print()
{
    #if !defined(Q_OS_SYMBIAN)
        if ( !printer ) printer = new QPrinter;
        QPrintDialog dialog(printer, this);
        if(dialog.exec()) {
            QPainter pp(printer);
            canvas.render(&pp);
        }
    #endif
}
void Main::findAllLinks(QList<QPointF> points, int net_index)
{
    qDebug() << "Finding nodes for net " << net_index;
    for(int i = 0 ; i < points.length() ; i++)
    {
        for(int g = 0 ; g < global_wire_connections.length() ; g++)
        {
            if(global_wire_connections[g]->line().p1() == points[i] ||
global_wire_connections[g]->line().p2() == points[i])
            {

```

```

        // As long as it doesn't already contain the points
        if(!net[net_index].contains(global_wire_connections[g]->line().p1()))
        {
            net[net_index].append(global_wire_connections[g]->line().p1());
            qDebug() << "Net " << net.length()-1 << " now contains " <<
net[net.length()-1];
        }
        if(!net[net_index].contains(global_wire_connections[g]->line().p2()))
        {
            net[net_index].append(global_wire_connections[g]->line().p2());
            qDebug() << "Net " << net.length()-1 << " now contains " <<
net[net.length()-1];
        }
        if(!points.contains(global_wire_connections[g]->line().p1()))
        {
            points.append(global_wire_connections[g]->line().p1());
        }
        if(!points.contains(global_wire_connections[g]->line().p2()))
        {
            points.append(global_wire_connections[g]->line().p2());
        }
    }
}
return;
}
int Main::createNets()
{
    bool found_p;
    bool flag = false;
    net.clear();
    dummy_nets = 0;
    // Find ground and make that net 0, add all other global wires connected to ground to
net 0
    for(int i = 0 ; i < comp.length() ; i++)
    {
        if(comp[i]->identifier->toPlainText().toLocal8Bit().data()[0] == 'G')
        {
            for(int nc = 0 ; nc < comp[i]->node_connections.length() ; nc++)
            {
                if(comp[i]->node_connections[nc])
                {
                    QList<QPointF> temp_net;
                    temp_net.append(global_wire_connections[comp[i]-
>wire_connections[nc][0]]->line().p1());
                    temp_net.append(global_wire_connections[comp[i]-
>wire_connections[nc][0]]->line().p2());
                    net.append(temp_net);
                    //qDebug() << "Net " << net.length()-1 << " now contains " <<
net[net.length()-1];
                    findAllLinks(net[net.length()-1], net.length()-1);
                    flag = true;
                    break;
                }
            }
        }
        if(flag)
        {
            qDebug() << "Net " << net.length()-1 << " contains " << net[net.length()-1];
            break;
        }
    }
    if(!flag)
    {
        QString temp_string = QString("ERROR: Must have a ground connection!");
        qDebug() << temp_string;
        message = temp_string;
        return 0;
    }
    for(int g = 0 ; g < global_wire_connections.length() ; g++)

```

```

{
    flag = false;
    //qDebug() << "Wire " << g;
    for(int n = 0 ; n < net.length() ; n++)
    {
        //qDebug() << "net[" << n << "]" is " << net[n];
        //qDebug() << "global_wire_connections[g]->line().p1() = " <<
        global_wire_connections[g]->line().p1();
        //qDebug() << "global_wire_connections[g]->line().p2() = " <<
        global_wire_connections[g]->line().p2();
        if(net[n].contains(global_wire_connections[g]->line().p1()) ||
        net[n].contains(global_wire_connections[g]->line().p2()))
        {
            flag = true;
            break;
        }
    }
    if(!flag)
    {
        QList<QPointF> temp_net;
        temp_net.append(global_wire_connections[g]->line().p1());
        temp_net.append(global_wire_connections[g]->line().p2());
        net.append(temp_net);
        //qDebug() << "Net " << net.length()-1 << " now contains " <<
        net[net.length()-1];
        findAllLinks(net[net.length()-1], net.length()-1);
    }
}
/*
// Alternative method to create nets from wires
// PROBLEM: If this algorithm is used, the wire-creation
// order matters! The nets will need to be checked again
// after this algorithm and move the duplicates into the
// lowest net possible.
for(int k = 0 ; k < global_wire_connections.length() ; k++)
{
    QList<QPointF> temp_net;
    found_p = false;
    for(int j = 0 ; j < net.length() ; j++)
    {
        // For all wires after the current wire
        for(int ow = k + 1 ; ow < global_wire_connections.length() ; ow++)
        {
            // If any line after the current line contains a point that the current
            line contains, add that to the net first
            if((global_wire_connections[k]->line().p1() ==
            global_wire_connections[ow]->line().p1()) || (global_wire_connections[k]->line().p1() ==
            global_wire_connections[ow]->line().p2()) || (global_wire_connections[k]->line().p2() ==
            global_wire_connections[ow]->line().p1()) || (global_wire_connections[k]->line().p2() ==
            global_wire_connections[ow]->line().p2()))
            {
                if(net[j].contains(global_wire_connections[ow]->line().p1()) ||
                net[j].contains(global_wire_connections[ow]->line().p2()))
                {
                    if(net[j].contains(global_wire_connections[ow]->line().p1()))
                    {
                        net[j].append(global_wire_connections[ow]->line().p2());
                    }
                    else
                    {
                        net[j].append(global_wire_connections[ow]->line().p1());
                    }
                }
                //found_p = true;
                qDebug() << "Net " << j << " now contains " << net[j];
            }
        }
    }
    if(net[j].contains(global_wire_connections[k]->line().p1()) ||
    net[j].contains(global_wire_connections[k]->line().p2()))
    {

```

```

        if(net[j].contains(global_wire_connections[k]->line().p1()))
        {
            net[j].append(global_wire_connections[k]->line().p2());
        }
        else
        {
            net[j].append(global_wire_connections[k]->line().p1());
        }
        found_p = true;
        qDebug() << "Net " << j << " now contains " << net[j];
        break;
    }
}
if(!found_p)
{
    temp_net.append(global_wire_connections[k]->line().p1());
    temp_net.append(global_wire_connections[k]->line().p2());
    net.append(temp_net);
    qDebug() << "Net " << net.length()-1 << " now contains " << net[net.length()-
1];
}
}
*/
dummy_nets = net.length();
dummy_total = 0;
memset(dummy_sources,0,2000);
char buffer [200] = {0};
int dummy_voltage_net;
// Assign nodes to nets
for(int i = 0 ; i < comp.length() ; i++)
{
    comp[i]->assigned_net.clear();
    for(int n = 0 ; n < comp[i]->node_locations.length() ; n++)
    {
        if(comp[i]->node_connections[n] == false)
        {
            QString temp_string = QString("ERROR: Node %1 on component %2 is not
connected!").arg(n).arg(comp[i]->identifier->toPlainText().toLocal8Bit().data());
            qDebug() << temp_string;
            message = temp_string;
            return 0;
        }
        for(int r = 0 ; r < net.length() ; r++)
        {
            qDebug() << "net[" << r << "] is " << net[r] << "Does it contain " <<
comp[i]->node_locations[n] << "?";
            if(net[r].contains(comp[i]->node_locations[n]))
            {
                qDebug() << "Yes!";
                comp[i]->assigned_net.append(r);
                break;
            }
        }
    }
    // Take care of dummy voltage sources for current measurement
    if((!comp[i]->iprobe_nodes.isEmpty()) && (comp[i]->component_name !=
QString("vacsource")) && (comp[i]->component_name != QString("vsource")))
    {
        for(int n = 0 ; n < comp[i]->iprobe_nodes.length() ; n++)
        {
            dummy_voltage_net = comp[i]->assigned_net[comp[i]->iprobe_nodes[n]];
            comp[i]->assigned_net[comp[i]->iprobe_nodes[n]] = dummy_nets;
            memset(buffer,0,50);
            sprintf(buffer,"V%d %d %d DC
0\n",dummy_nets+50,dummy_voltage_net,dummy_nets);
            strcat(dummy_sources,buffer);
            qDebug() << dummy_sources;
            dummy_nets++;
            dummy_total++;
        }
    }
}

```

```

    }
    return 1;
}
int Main::formatNetlist()
{
    remove("/root/netlist.cir");
    std::ofstream spiceFile("/root/netlist.cir");
    char buffer [200] = {0};
    int opamp = 0;
    sprintf(buffer, "Generated Netlist\n\n");
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
    memset(buffer, 0, 200);
    sprintf(buffer, "option rshunt = 1.0e9\n");
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
    // Write dummy sources to file
    spiceFile.write(dummy_sources, strlen(dummy_sources));
    spiceFile.flush();
    for(int i = 0 ; i < comp.length() ; i++)
    {
        memset(buffer, 0, 50);
        if(comp[i]->identifier->toPlainText().toLocal8Bit().data()[0] == 'X')
        {
            sprintf(buffer, "%s %d %d %d opamp741\n", comp[i]->identifier-
            >toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
            comp[i]->assigned_net[2]);
            opamp = 1;
        }
        else if(comp[i]->identifier->toPlainText().toLocal8Bit().data()[0] == 'M')
        {
            sprintf(buffer, "%s %d %d %d nmos\n", comp[i]->identifier-
            >toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
            comp[i]->assigned_net[2]);
        }
        else if(comp[i]->identifier->toPlainText().toLocal8Bit().data()[0] == 'Q')
        {
            sprintf(buffer, "%s %d %d %d npn\n", comp[i]->identifier-
            >toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
            comp[i]->assigned_net[2]);
        }
        else if(comp[i]->identifier->toPlainText().toLocal8Bit().data()[0] == 'D')
        {
            sprintf(buffer, "%s %d %d\n", comp[i]->identifier-
            >toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1]);
        }
        else if(comp[i]->identifier->toPlainText().toLocal8Bit().data()[0] == 'I')
        {
            if(comp[i]->scaler_button->text().toLocal8Bit().data()[0] == 'M')
            {
                sprintf(buffer, "%s %d %d DC %d%s\n", comp[i]->identifier-
                >toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
                comp[i]->value_slider->value(), "meg");
            }
            else
            {
                sprintf(buffer, "%s %d %d DC %d%c\n", comp[i]->identifier-
                >toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
                comp[i]->value_slider->value(), comp[i]->scaler_button->text().toLocal8Bit().data()[0]);
            }
        }
        else if(comp[i]->identifier->toPlainText().toLocal8Bit().data()[0] == 'V' &&
        comp[i]->identifier->toPlainText().toLocal8Bit().data()[1] != 'D')
        {
            if(comp[i]->scaler_button->text().toLocal8Bit().data()[0] == 'M' && comp[i]-
            >fscaler_button->text().toLocal8Bit().data()[0] != 'M')
            {
                sprintf(buffer, "%s %d %d sin(0 %d%s %d%c 0 0 0)\n", comp[i]->identifier-
                >toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],

```



```

comp[i]->value_slider->value(), "meg", comp[i]->fvalue_slider->value(), comp[i]-
>fscaler_button->text().toLocal8Bit().data()[0]);
    }
    else if(comp[i]->scaler_button->text().toLocal8Bit().data()[0] == 'M' &&
comp[i]->fscaler_button->text().toLocal8Bit().data()[0] == 'M')
    {
        sprintf(buffer,"%s %d %d sin(0 %d%s %d%s 0 0 0)\n", comp[i]->identifier-
>toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
comp[i]->value_slider->value(), "meg", comp[i]->fvalue_slider->value(), "meg");
    }
    else if(comp[i]->scaler_button->text().toLocal8Bit().data()[0] != 'M' &&
comp[i]->fscaler_button->text().toLocal8Bit().data()[0] == 'M')
    {
        sprintf(buffer,"%s %d %d sin(0 %d%c %d%s 0 0 0)\n", comp[i]->identifier-
>toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
comp[i]->value_slider->value(), comp[i]->scaler_button->text().toLocal8Bit().data()[0],
comp[i]->fvalue_slider->value(), "meg");
    }
    else
    {
        sprintf(buffer,"%s %d %d sin(0 %d%c %d%c 0 0 0)\n", comp[i]->identifier-
>toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
comp[i]->value_slider->value(), comp[i]->scaler_button->text().toLocal8Bit().data()[0],
comp[i]->fvalue_slider->value(), comp[i]->fscaler_button-
>text().toLocal8Bit().data()[0]);
    }
    }
    else if(comp[i]->identifier->toPlainText().toLocal8Bit().data()[0] == 'G')
    {
    }
    else
    {
        if(comp[i]->scaler_button->text().toLocal8Bit().data()[0] == 'M')
        {
            sprintf(buffer,"%s %d %d %d%s\n", comp[i]->identifier-
>toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
comp[i]->value_slider->value(), "meg");
        }
        else
        {
            sprintf(buffer,"%s %d %d %d%c\n", comp[i]->identifier-
>toPlainText().toLocal8Bit().data(), comp[i]->assigned_net[0], comp[i]->assigned_net[1],
comp[i]->value_slider->value(), comp[i]->scaler_button->text().toLocal8Bit().data()[0]);
        }
    }
    qDebug() << buffer;
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
}
if(opamp)
{
    memset(buffer,0,50);
    sprintf(buffer, ".subckt opamp741 1 2 3\nrin 1 2 2meg\nrout 6 3 75\n");
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
    memset(buffer,0,50);
    sprintf(buffer, "e 4 0 1 2 100k\nrbw 4 5 0.5meg\nncbw 5 0 31.85nf\n");
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
    memset(buffer,0,50);
    sprintf(buffer, "eout 6 0 5 0 1\n.ends opamp741\n");
    spiceFile.write(buffer, strlen(buffer));
    spiceFile.flush();
}
memset(buffer,0,200);
sprintf(buffer, ".control\ntran 10n 100u\nprint ");
spiceFile.write(buffer, strlen(buffer));
spiceFile.flush();
iflag = false;
vflag = false;

```

```

dummy_nets = net.length();
// Find node voltages to be probed
for(int i = 0 ; i < comp.length() ; i++)
{
    if(!comp[i]->vprobe_nodes.isEmpty())
    {
        for(int n = 0 ; n < comp[i]->vprobe_nodes.length() ; n++)
        {
            qDebug() << getNet(comp[i]->node_locations[comp[i]->vprobe_nodes[n]]);
            int v_node = getNet(comp[i]->node_locations[comp[i]->vprobe_nodes[n]]);
            if(v_node >= 0)
            {
                vflag = true;
                memset(buffer,0,50);
                sprintf(buffer,"V(%d) ",v_node);
                qDebug() << buffer;
                spiceFile.write(buffer,strlen(buffer));
                spiceFile.flush();
            }
        }
    }
    if(!comp[i]->iprobe_nodes.isEmpty())
    {
        for(int n = 0 ; n < comp[i]->iprobe_nodes.length() ; n++)
        {
            qDebug() << getNet(comp[i]->node_locations[comp[i]->iprobe_nodes[n]]);
            int i_node = getNet(comp[i]->node_locations[comp[i]->iprobe_nodes[n]]);
            if(i_node >= 0)
            {
                iflag = true;
                memset(buffer,0,50);
                if((comp[i]->component_name == QString("vacsource")) || (comp[i]-
>component_name == QString("vsource")))
                {
                    sprintf(buffer,"I(%s) ",comp[i]->identifier-
>toPlainText().toLocal8Bit().data());
                    qDebug() << buffer;
                    spiceFile.write(buffer,strlen(buffer));
                    spiceFile.flush();
                }
                else
                {
                    sprintf(buffer,"I(V%d) ",dummy_nets+50);
                    qDebug() << buffer;
                    spiceFile.write(buffer,strlen(buffer));
                    spiceFile.flush();
                    dummy_nets++;
                }
            }
        }
    }
    v_and_c = vflag & iflag;
}
if(!vflag && !iflag)
{
    QString temp_string = QString("ERROR: You need to measure a current or a
voltage!");
    qDebug() << temp_string;
    message = temp_string;
    stopSimulating();
    return 0;
}
memset(buffer,0,50);
sprintf(buffer,"\n.endc\n.end");
spiceFile.write(buffer,strlen(buffer));
spiceFile.flush();
return 1;
}
int Main::getNet(QPointF node)
{

```

```

for(int j = 0 ; j < net.length() ; j++)
{
    if(net[j].contains(node))
    {
        return j;
    }
}
return -1;
}
void Main::formatSpiceOutput()
{
    remove("/root/formattedPart1.txt");
    remove("/root/formattedPart2.txt");
    remove("/root/formattedPart3.txt");
    std::ifstream inFile("/root/results.txt");
    std::ofstream outFile1("/root/formattedPart1.txt");
    std::ofstream outFile2("/root/formattedPart2.txt");
    std::ofstream outFile3("/root/formattedPart3.txt");
    std::string line;
    int indexFound = 0;
    int i = -2;
    int j = 1;
    int goodLine = 0;
    char indexCount[10];
    while(inFile.good())
    {
        getline(inFile,line);
        line.append("\n");
        if(!indexFound)
        {
            if(line.find("Index") == 0)
            {
                indexFound = 1;
            }
        }
        if(i >= 0)
        {
            sprintf(indexCount,"%d",i);
            if(line.find(indexCount) == 0)
            {
                goodLine = 1;
                i++;
            }
            else
            {
                if(line.find("0") == 0)
                {
                    i = 1;
                    goodLine = 1;
                    j++;
                }
            }
        }
        else
        {
            if(indexFound)
            {
                i++;
            }
        }
        if (i >= 0 && goodLine)
        {
            switch(j)
            {
                case 1:
                    outFile1.write(line.c_str(),line.length());
                    break;
                case 2:
                    outFile2.write(line.c_str(),line.length());
                    break;
            }
        }
    }
}

```

```

        case 3:
            outFile3.write(line.c_str(), line.length());
            break;
        }
        goodLine = 0;
    }
}
inFile.close();
outFile1.close();
outFile2.close();
outFile3.close();
return;
}
void Main::gnuPlot()
{
    remove("/root/gnuscrypt");
    std::ofstream script("/root/gnuscrypt");
    char buffer[500];
    char initiate[10] = "plot \\n";
    std::string fileName;
    int column = 3;
    int j, i, k = 0;
    bool flag = false;
    //int voltageCount = 0;
    //int currentCount = 0;
    //memset(nodeVoltageMeasured, 0, 10);
    //memset(nodeCurrentMeasured, 0, 10);
    memset(buffer, 0, 500);
    sprintf(buffer, "set term jpeg size 875,400\nset output \"/root/plot.jpg"\n");
    script.write(buffer, strlen(buffer));
    script.flush();
    memset(buffer, 0, 500);
    sprintf(buffer, "set size 1,1\nset origin 0,0\nset key outside\n");
    script.write(buffer, strlen(buffer));
    script.flush();
    // Setup the plot command
    memset(buffer, 0, 500);
    if(v_and_c) // IF voltage AND current!
    {
        sprintf(buffer, "set title 'Voltage/Current Plots'\nset xlabel 'Seconds'\nset
ylabel 'Voltage (V)'\nset y2label 'Current (A)'\nset ytics nomirror\nset y2tics
nomirror\nset xtics mirror\nset tics out\nset autoscale y\nset autoscale y2\n");
        script.write(buffer, strlen(buffer));
        script.flush();
    }
    else if(vflag)
    {
        sprintf(buffer, "set xlabel 'Seconds'\nset ylabel 'Volts'\nset title 'Voltage
Plots'\n");
        script.write(buffer, strlen(buffer));
        script.flush();
    }
    else if(iflag)
    {
        sprintf(buffer, "set xlabel 'Seconds'\nset ylabel 'Amps'\nset title 'Current
Plots'\n");
        script.write(buffer, strlen(buffer));
        script.flush();
    }
    script.write(initiate, strlen(initiate));
    script.flush();
    //qDebug("%d", numOfVMeasured + numOfIMeasured);
    // Add all traces to plot
    // Voltage Inputs
    int color_index = 0;
    for(int i = 0 ; i < comp.length() ; i++)
    {
        if(!comp[i]->vprobe_nodes.isEmpty())
        {
            for(int n = 0 ; n < comp[i]->vprobe_nodes.length() ; n++)

```

```

{
    int v_node = getNet(comp[i]->node_locations[comp[i]->vprobe_nodes[n]]);
    if(v_node >= 0)
    {
        if(flag)
        {
            sprintf(buffer, "\n\n");
            script.write(buffer, strlen(buffer));
        }
        flag = true;
        fileName = determineFile(k);
        sprintf(buffer, "%s\ using 2:%d with linespoints axis xlyl title
'V(%s) %d'", fileName.c_str(), column, comp[i]->identifier-
>toPlainText().toLocal8Bit().data(), comp[i]->vprobe_nodes[n]);
        script.write(buffer, strlen(buffer));
        column++;
        k++;
        // Set the node being measured the same color as the plotted line
        //gnuplot*line1Color: red
        //gnuplot*line2Color: green
        //gnuplot*line3Color: blue
        //gnuplot*line4Color: magenta
        //gnuplot*line5Color: cyan
        //gnuplot*line6Color: sienna
        //gnuplot*line7Color: orange
        //gnuplot*line8Color: coral
        int colors[9][3] =
        {
            {255,0 ,0 } , //red
            {0 ,195,0 } , //green
            {0 ,128,255} , //blue
            {190,0 ,255} , //magenta
            {0 ,240,235} , //cyan
            {190,65 ,0 } , //sienna
            {200,200,0 } , //orange
            {65 ,105,225} , //coral
            {100,100,100} //default grey
        };
        comp[i]->setNodeColor(comp[i]-
>vprobe_nodes[n], colors[color_index][0], colors[color_index][1], colors[color_index][2]);
        if(color_index < 8)
        {
            color_index++;
        }
    }
    if (column > 5)
    {
        column = 3;
    }
}

if(!comp[i]->iprobe_nodes.isEmpty())
{
    for(int n = 0 ; n < comp[i]->iprobe_nodes.length() ; n++)
    {
        int i_node = getNet(comp[i]->node_locations[comp[i]->iprobe_nodes[n]]);
        if(i_node >= 0)
        {
            if(flag)
            {
                sprintf(buffer, "\n\n");
                script.write(buffer, strlen(buffer));
            }
            flag = true;
            fileName = determineFile(k);
            sprintf(buffer, "%s\ using 2:%d with linespoints axis xlyl title
'I(%s) %d'", fileName.c_str(), column, comp[i]->identifier-
>toPlainText().toLocal8Bit().data(), comp[i]->iprobe_nodes[n]);
            script.write(buffer, strlen(buffer));
            column++;
        }
    }
}

```

```

        k++;
        // Set the node being measured the same color as the plotted line
        //gnuplot*line1Color: red
        //gnuplot*line2Color: green
        //gnuplot*line3Color: blue
        //gnuplot*line4Color: magenta
        //gnuplot*line5Color: cyan
        //gnuplot*line6Color: sienna
        //gnuplot*line7Color: orange
        //gnuplot*line8Color: coral
        int colors[9][3] =
        {
            {255,0,0}, //red
            {0,195,0}, //green
            {0,128,255}, //blue
            {190,0,255}, //magenta
            {0,240,235}, //cyan
            {190,65,0}, //sienna
            {200,200,0}, //orange
            {65,105,225}, //coral
            {100,100,100} //default grey
        };
        comp[i]->setNodeColor(comp[i]-
>iprobe_nodes[n],colors[color_index][0],colors[color_index][1],colors[color_index][2]);
        if(color_index < 8)
        {
            color_index++;
        }
    }
    if (column > 5)
    {
        column = 3;
    }
}

}
script.close();
system("gnuplot /root/gnuscript");
return;
}

std::string Main::determineFile(int i)
{
    std::string fileName = "";
    switch(i)
    {
        case 0:
        case 1:
        case 2:
            fileName = "/root/formattedPart1.txt";
            break;
        case 3:
        case 4:
        case 5:
            fileName = "/root/formattedPart2.txt";
            break;
        case 6:
        case 7:
        case 8:
            fileName = "/root/formattedPart3.txt";
            break;
    }
    return fileName;
}

void Component::on_scale_button_clicked()
{
    switch(scaler_button->text().toLocal8Bit().data()[0])
    {
        case '1':
            scaler_button->setText(scaler_button->text().replace(0,1,'k'));
            break;
    }
}

```

```

        case 'k':
            scaler_button->setText(scaler_button->text().replace(0,1,'M'));
            break;
        case 'M':
            scaler_button->setText(scaler_button->text().replace(0,1,'p'));
            break;
        case 'p':
            scaler_button->setText(scaler_button->text().replace(0,1,'n'));
            break;
        case 'n':
            scaler_button->setText(scaler_button->text().replace(0,1,'u'));
            break;
        case 'u':
            scaler_button->setText(scaler_button->text().replace(0,1,'m'));
            break;
        case 'm':
            scaler_button->setText(scaler_button->text().replace(0,1,' '));
            break;
    }
    updateDetected = true;
}
void Component::on_fscales_button_clicked()
{
    switch(fscaler_button->text().toLocal8Bit().data()[0])
    {
        case ' ':
            fscaler_button->setText(fscaler_button->text().replace(0,1,'k'));
            break;
        case 'k':
            fscaler_button->setText(fscaler_button->text().replace(0,1,'M'));
            break;
        case 'M':
            fscaler_button->setText(fscaler_button->text().replace(0,1,'p'));
            break;
        case 'p':
            fscaler_button->setText(fscaler_button->text().replace(0,1,'n'));
            break;
        case 'n':
            fscaler_button->setText(fscaler_button->text().replace(0,1,'u'));
            break;
        case 'u':
            fscaler_button->setText(fscaler_button->text().replace(0,1,'m'));
            break;
        case 'm':
            fscaler_button->setText(fscaler_button->text().replace(0,1,' '));
            break;
    }
    updateDetected = true;
}

```