NTNU – Trondheim
Norwegian University of
Science and Technology

Tissue

Tore Egil Halse
Thomas Tøkje

Master of Science in Engineering Cybernetics
Submission date: June 2012
Supervisor: Amund Skavhaug, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Tissue

Thomas Tøkje
Tore Egil Halse

Report
Submission date: June 2012
Supervisor: Amund Skavhaug

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Summary

In this thesis, the development of a web application for designing electronic circuits has been initiated and documented. The application will feature some unique features regarding the design process of electronic circuits. Among them are interface based routing, a plugin-friendly environment and a collaborative resource database.

At the start of working on this thesis, there were no known web-based EDA software available. This provided an unique opportunity to fill this gap.

The application has been implemented using HTML5 and JavaScript for the interactive front-end (The web browser), and Google Go and MongoDB for the backend (The Server).

The basic building blocks of this application has been implemented, and together serves as an tech demo, available under a GPL licence.

Sammendrag

I denne hovedoppgaven har har uviklingen av en nett-applikasjon for design av elektronikk blitt startet og dokumentert. Programvaren vil inneha flere unike egenskaper rundt design av elektroniske kretser. Blandt de er interface basert routing, et plugin-vennlig miljø og samarbeidsdrevet ressursdatabase.

Ved påbegynnelse av oppgaven, var det ingen kjente web-basert EDA-programvare tilgjengelig. Dette åpnet en unik mulighet for å fylle dette gapet.

Programvaren har blitt implementert ved å benytte HTML5 og JavaScript for den interaktive frontapplikasjonen (Som kjører i nettleseren), og med Google Go og MongoDB i bakapplikasjonen (Som kjører på serveren).

De grunnleggende byggeklossene for programvaren har blitt implementert, og fungerer sammen som en tech-demo. Programvaren er GPL lisensert.

Contents

List of Figures

1 Introduction

1.1 Problem description

The objective of this thesis is to develop a GPL-licensed[2] web application for rapid prototyping of electronic circuits. Important qualities for the application includes ease of use, reusability and schematics readability.

Some of the tasks to be performed are listed below:

- Survey existing electronic design automation software (EDA).
- Describe essential and nice features for an EDA application.
- Study necessary background information to implement such an application
- Decide which tools and languages the software will be built upon, and learn how to best utilize these within the project.
- Implement the essentioal features of the application, and a plugin-platform which other features can built on top of.
- Evaluate the implementation.

1.2 Existing EDA solutions

Electronic Design Automation (EDA) software has been around since the early days of computers. Printed circuit boards had, until that point, mostly been drawn by hand, which imposes great difficulty when the complexity of the board increases. By the mid-80's, many companies had been born which sole focus was EDA software.

Today, many software solutions are available for design of printed circuit boards, and they vary as much in quality as they do in pricing and functionality. The cost for one licence vary from completely free (RS Components Design Spark, gEDA, CadSoft EAGLE with limitations to board dimensions) to \$9,995.00 (OrCAD PCB Designer Professional with PSpice). There is undoubtedly some correleation between cost and quality, however, the cost of the high-end tools render them unaffordable to the normal hobbyist.

The author has mainly used CadSoft EAGLE and Altium, and so, the observations below will mainly be based upon these editors. However, upon watching other people use other editors (e.g. Proteus), it is easily observable that their workflow is pretty similar.

1.2.1 Observations from using Altium and CadSoft EAGLE

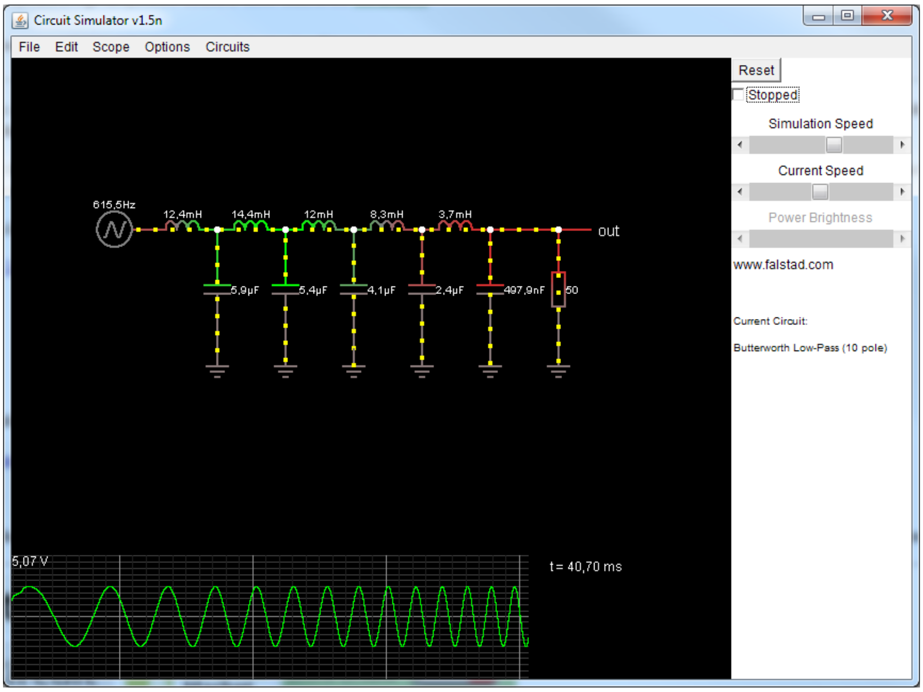
The workflow in EDA tools today can be argued to be quite awkward. First, an overview of what the overlaying system should look like, needs to be specified. This often includes wiring together different abstract, functional modules by means of communication interfaces. These interfaces may be implemented as e.g. *SPI*, I^2C or an analog signal wire.

After having specified the module topology of the project, the user will have to create implementations of all the modules in the design. Circuit implementation is usually done on component level, and so, the components to be used in the design must be decided upon now. Usually, even resistors and capacitors have to be retrieved from a huge component library and chosen based on their component value and package. Said in another way: The specific components (complete with it's physical package) to be used has to be decided from the start, regardless of how close they are to ideal components. Concequently, the user can't decide to use a NPN-type BJT transistor, he will have to choose between, among others, *BC550C*, *2N5551* and *BC337*. Although these transistors have different characteristics, their schematic representation should still be the same, and the user shouldn't have to worry about choosing which one of these to use until he knows what the requirements for his application are.

Next, all integrated circuits needs to be wired up to match the specifications in the datasheet. This usually implies, among other things, that between each V_{CC} and GND pair, a decoupling capacitor of specified size needs to be connected. These decoupling capacitors shouldn't have to be represented graphically, they are required by the IC.

1.2.2 Falstad Circuit Simulator

Although not strictly an EDA application, we find Falstad Circuit Simulator[3] to be an inspiring piece of software. It is able to simulate circuits with a wide range of components including: Resistors, capacitors, inductors, operational amplifiers, a variety of transistors, diodes, and much more. Circuits are colored based on voltage, and dots along wires and components illustrating charge move in proportion to the currents. Figure 1.1 shows a picture of Falstad Circuit Simulator in action.



Current Circuit:

Butterworth Low-Pass (10 pole)

Figure 1.1: Screenshot of falstad

1.3 A new approach to circuit design

1.3.1 Project goal

As shown in section 1.2, there are many circuit layout tools available today, and although some new EDA tools has emerged and gained momentum in recent years (i. e. RS Components' Design Spark), the possibility of success by means of a “me too”-application seems rather unlikely, as it would not be able to stand out among the other editors. However, if the main focus of the project is instead on creating a “different” EDA application, the program would hopefully gain much more acknowledgement, and also a larger user base. In this section, the main task will be to:

- Explain who the primary targeted user base for this application is
- Explain some of the core ideas for making the application stand out among other EDA tools
- Explain the strategy for turning the application into a healthy open-source project

The item list below is yet to be placed somewhere in this chapter.

- Top-down design
- Reduced schematic verbosity
- User-generated editors to create modules (filter design, etc)
- Collaborative component library
- Interface-level routing
- Built-in circuit simulator

In section 1.2, it was argued that the way in which circuit schematics are created seems to have been established since the 80's. Although all tools have their variations and supplements to this method, the basics are still the same: The schematics software offers little more than a graphical way of generating the net- and component lists for the circuit layout. In other words, everything on the printed circuit board is in one or another way created by the user in the schematic editor, and although the principles behind many of the modules used in the design are well known (e.g. low-order RLC filters), there is no automated way to ease the process of applying and modifying the module. This is also true for routing of interfaces. Indeed, most applications allow bundling together wires into buses, but buses contain no extra information about how the interconnection between the wires should be. Thus, components such as termination resistors, pull-up resistors, etc. has to be appended to the schematic editor by the user himself.

Scripted schematic modules

The example in figure 1.2 shows a band-pass filter created for use as a guitar effect. The schematics in the figure are created in CadSoft Eagle. Although the mathematics behind this circuit is fairly obvious for an electronics engineer, it would still take some time to calculate the needed component values for the desired input impedance and center frequency, choose which physical components to use, place the components in the schematic editor and connect the components together with wires. Some tools may let the user create third-party scripts to allow for extra functionality, e.g. filter design scripts, however, these scripts usually limit themselves to auto-placement and auto-wiring of filter components. There are no EDA tools (to the author's knowledge) that allow the user to place modules in the schematic editor where the internals of the module are defined, not necessarily by schematics, but rather by user-created dialog windows, scripts, or whatever else the user decides. By having a band-pass filter

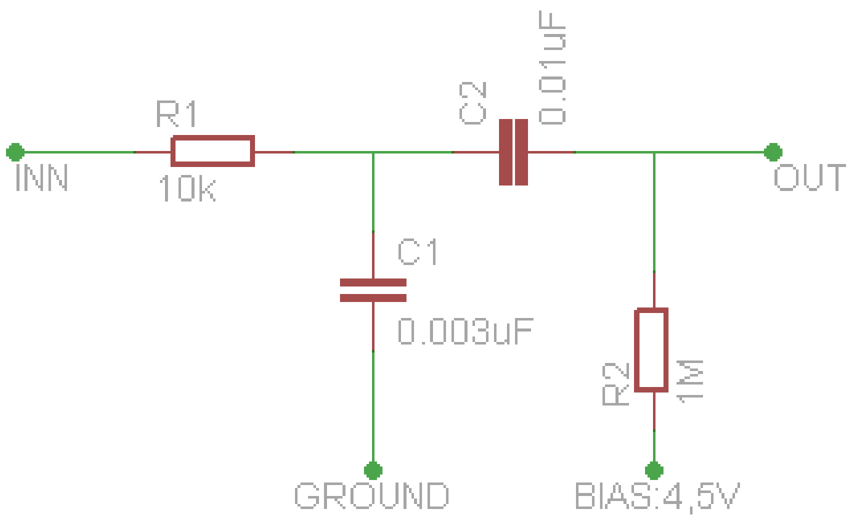


Figure 1.2: Band-pass filter created in CadSoft Eagle.

defined as an abstract module, the user would then be able to write his own filter implementation, wherein the net- and component lists of the module would be changed according to configurable parameters. The user could then share his script with others through a object repository. The different signal processing modules in section 1.3.2 explains scripted schematic modules rather well.

Less verbose circuit design

In subsection 1.2.1, it was argued that existing EDA software tend to be unncessarily verbose, meaning that every net in the design needs to have graphical components connected to it in order to work. Although some EDA tools (e.g. *Altium*) support hierarchical schematics, which does indeed clean up the schematic, the editor still has to explicitly place and connect together all components in the design on some level in the hierarchy.

The application to be developed should hide away unnecessary schematic information. As an example, decoupling capacitors between V_{CC} and GND will be omitted from the schematic, as their values are defined in the component's datasheet. Additionaly, when connecting an interface to the component, the interface should connect to the correct pins, although only the interface name should be visible.

The user should also be able to select which component will realize his component based on which interfaces are used.

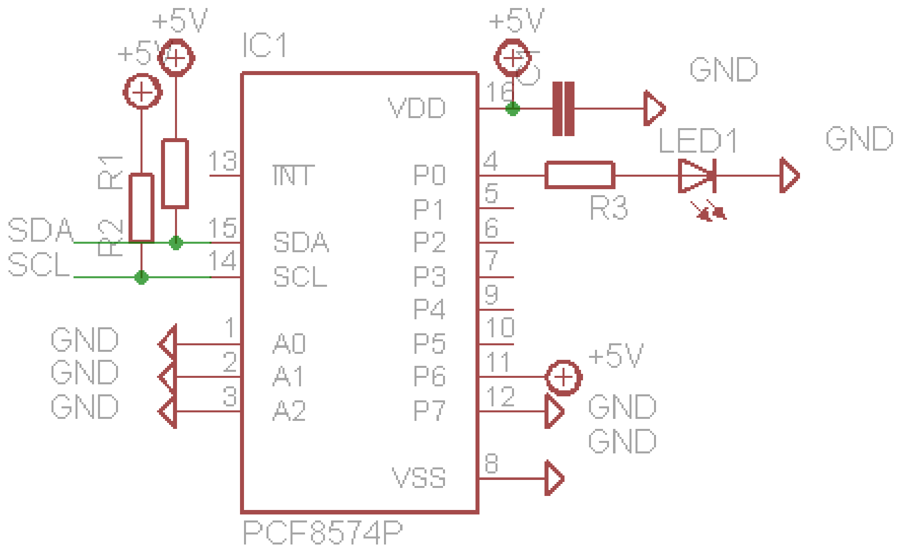


Figure 1.3: IO expander created in CadSoft Eagle, showing manual interface routing [1]

Interface-based routing

The example in figure 1.3 shows an IO expander accessed with an I^2C-interface. The schematics are created in CadSoft Eagle, but the way in which the component is connected to the I^2C-bus is the typical way to do it in most EDA software. Although fairly readable for this application due to the low complexity of the circuit, it is easy to see that schematics with higher complexity may suffer from bad schematic readability, due to only parts of the interface displayed at once. A better solution would be to show *only one* interface wire, called I^2C, as an input to the component. By knowing that this is an I^2C-interface, it would implicitly follow that there are pull-up resistors on both *SCK* and *SDA*, and the component would know where to connect both *SCK* and *SDA*. There should also be a way to show all endpoints connected to the interface, along with other components necessary to implement the interface. The user should also be able to define his own interfaces and share these through the object repository.

Plugin-based architecture

Many EDA tools have the ability for the user to write scripts or plugins to extend the functionality of the software. However, these scripts are only granted a subset of the software's API; their typical use is for auto-generation of different components, e.g. PCB footprints.

The architecture of the application to be developed in this thesis should be one where most of the user interactivity should be done through plugins. The software platform itself should contain only graphical building blocks and application- and project control functions, and all editors should be able to utilize and extend upon these as seen fit.

Built-in circuit simulator

Electronic circuit simulation is a very useful tool for both design- and debugging purposes. Many EDA tools have circuit simulation either built into the application or available as an (often expensive) extension to the software.

The application to be developed throughout this thesis should be able to perform circuit simulations in both time and frequency domains. The simulator should also be able to mix analog and digital signals in a seamless manner.

Hardware and software modeling

Describe both layout and functionality of modules.

Centralized object database

For components, modules (schematic and layout), plugins, etc.

When using EDA software, it is very useful to have a way of sharing component libraries with other users. Many companies do this by keeping a private revision controlled folder with all the libraries the company uses in its designs.

A centralized database should be created for loading and storing component libraries, plugins, schematic and circuit layout data, etc. This will greatly reduce the workload for the user, such that he will be able to collaborate with others on creating components.

1.3.2 Example: Analog audio processing

In this example, the goal is to create an analog audio processor to be used in the effect chain of an electric guitar, etc. This example was chosen due to that many musicians want to create their own guitar effects, but may be demotivated by the amount of research needed to be done before starting out creating their own effects. First, they will need to learn enough about electronic circuits to know how to create the necessary filters and signal gain for their effect. Secondly, they will need to learn how to use some EDA tool for creating the circuit board for their guitar effect. As mentioned in subsection 1.2.1, using an EDA for the first time can be quite overwhelming.

Starting out with a module chain for the effect processor, none of which has been chosen implementations for yet. Double clicking on the DC bias module should bring up a configuration dialog for choosing a way to implement the DC offset, set the input cutoff frequency, and choosing bias voltage.

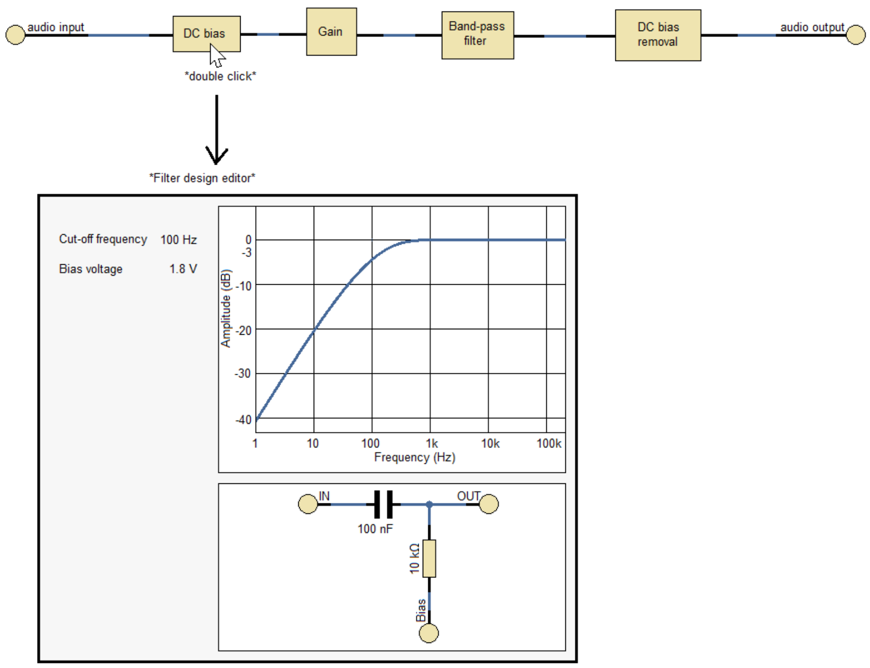
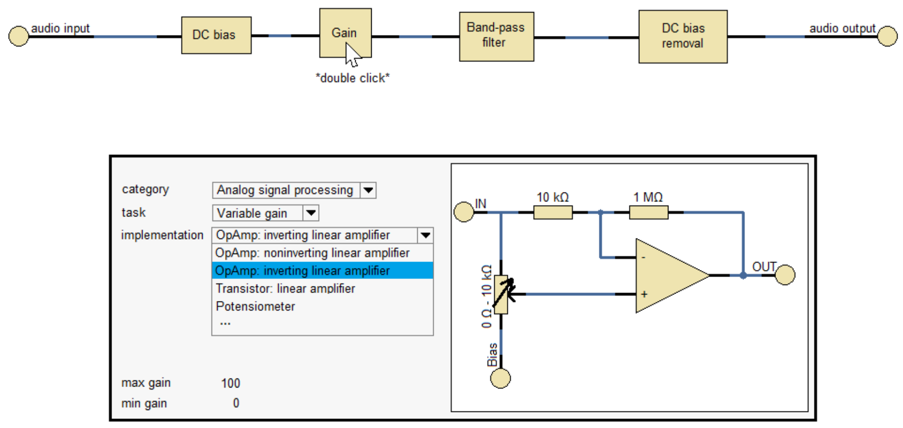


Figure 1.4: Starting out with an unimplemented module chain.



category Analog signal processing
task Variable gain
implementation OpAmp: inverting linear amplifier
OpAmp: noninverting linear amplifier
OpAmp: inverting linear amplifier
Transistor: linear amplifier
Potensiometer
...
max gain 100
min gain 0

Figure 1.5: Double click to bring up config dialog for gain stage.

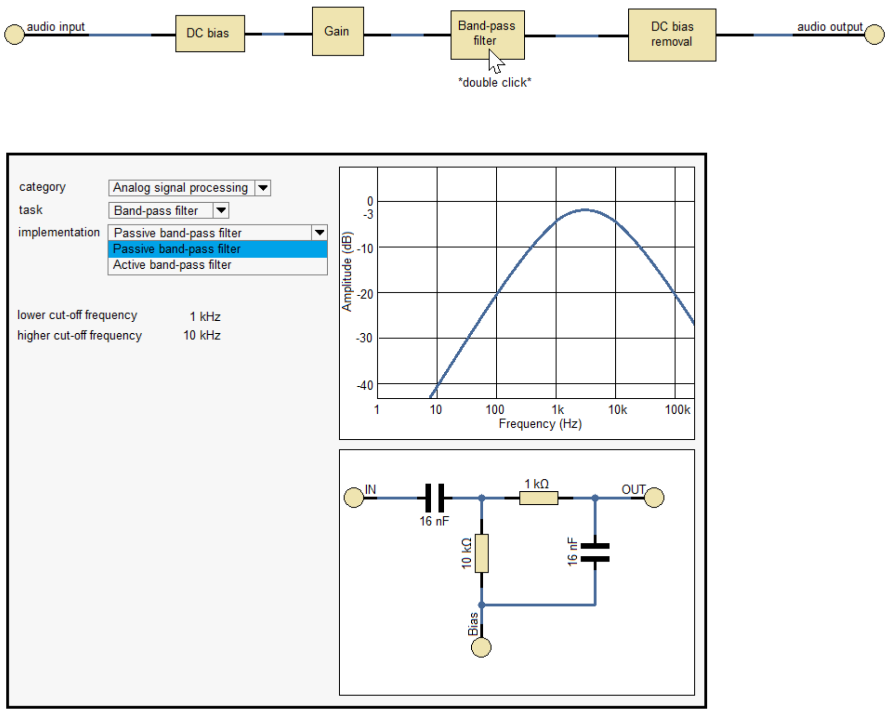
Double clicking on the gain module should bring up a configuration dialog for choosing a way to implement the gain stage, with parameters for maximum and minimum gain.

Double clicking on the band pass filter module should bring up a band-pass filter configuration dialog, with parameters for lower and upper cutoff frequencies.

Double clicking on the DC offset removal module should bring up a DC offset removal configuration dialog, with a parameter control for setting the cutoff frequency.

1.3.3 Example: Module interconnection with CAN

In this example, the goal is to connect separate modules together by a *CAN* interface. *CAN* is a serial interface, consisting of two wires, *CANH* and *CANL*, and terminating resistors of 120 Ω between these on both ends of the interface.



category	Analog signal processing
task	Band-pass filter
implementation	Passive band-pass filter
	Passive band-pass filter
	Active band-pass filter
lower cut-off frequency	1 kHz
higher cut-off frequency	10 kHz

Figure 1.6: Double click to bring up config dialog for the band-pass filter.

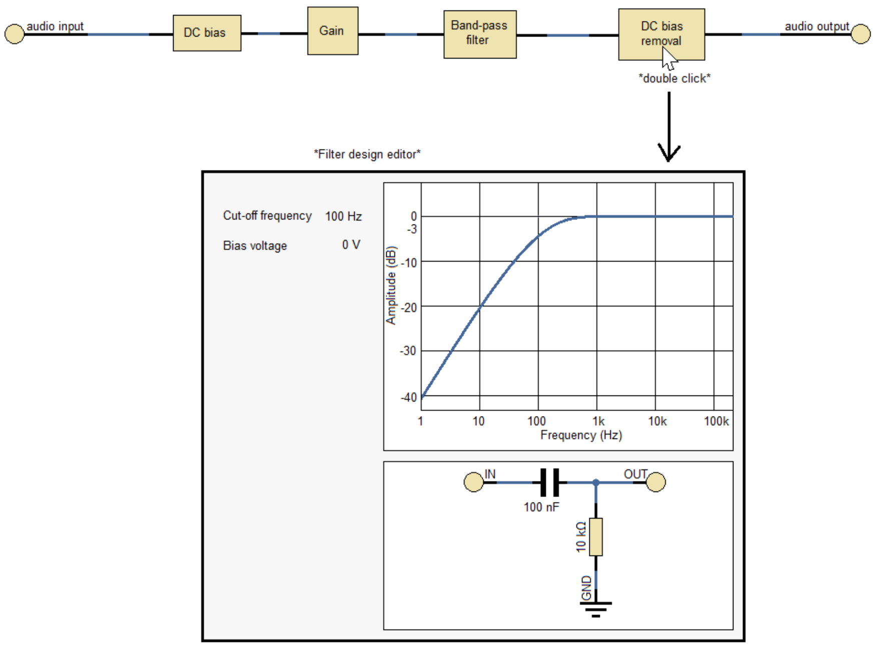


Figure 1.7: Double click to bring up config dialog for DC offset removal.

Usually when routing *CAN* interfaces, the user has to pay attention to placing and routing the termination resistors as well, although they are part of the *CAN* specification and the protocol simply won't work without them. This example demonstrates one of the main features of the application to be developed in this thesis, in which the netlist and component list for the interface is part of the interface itself and does not need to be manually routed in the schematic editor.

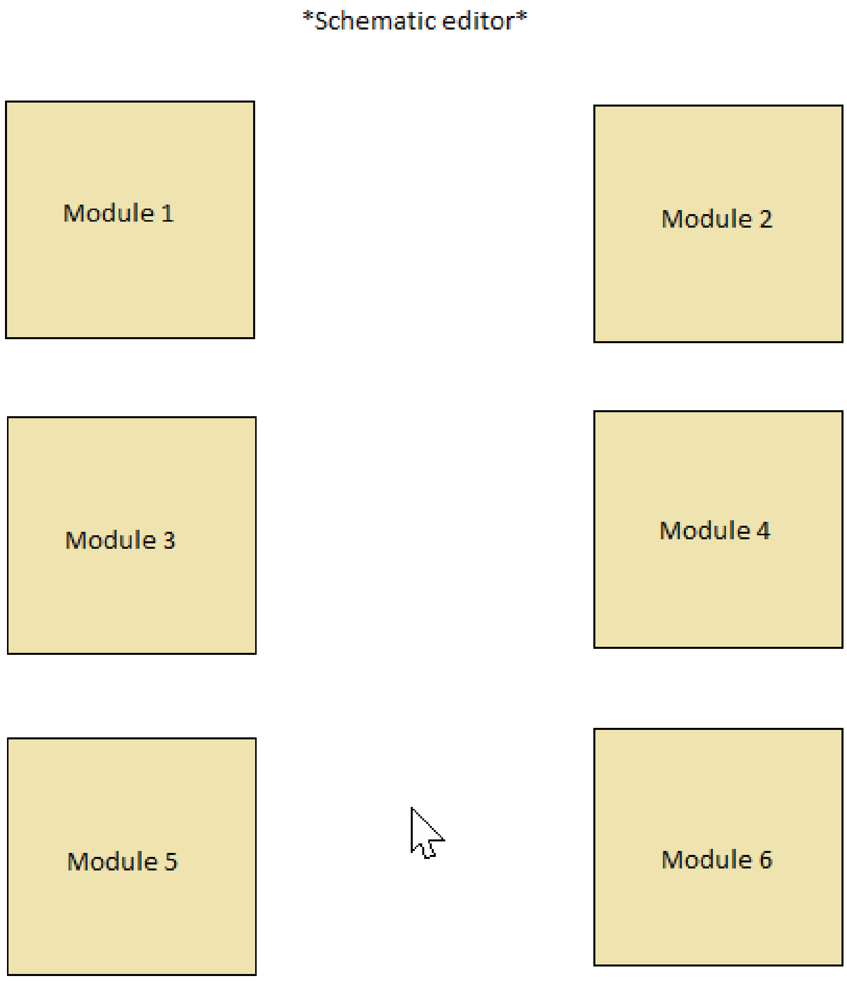


Figure 1.8: Starting out with six unconnected modules.

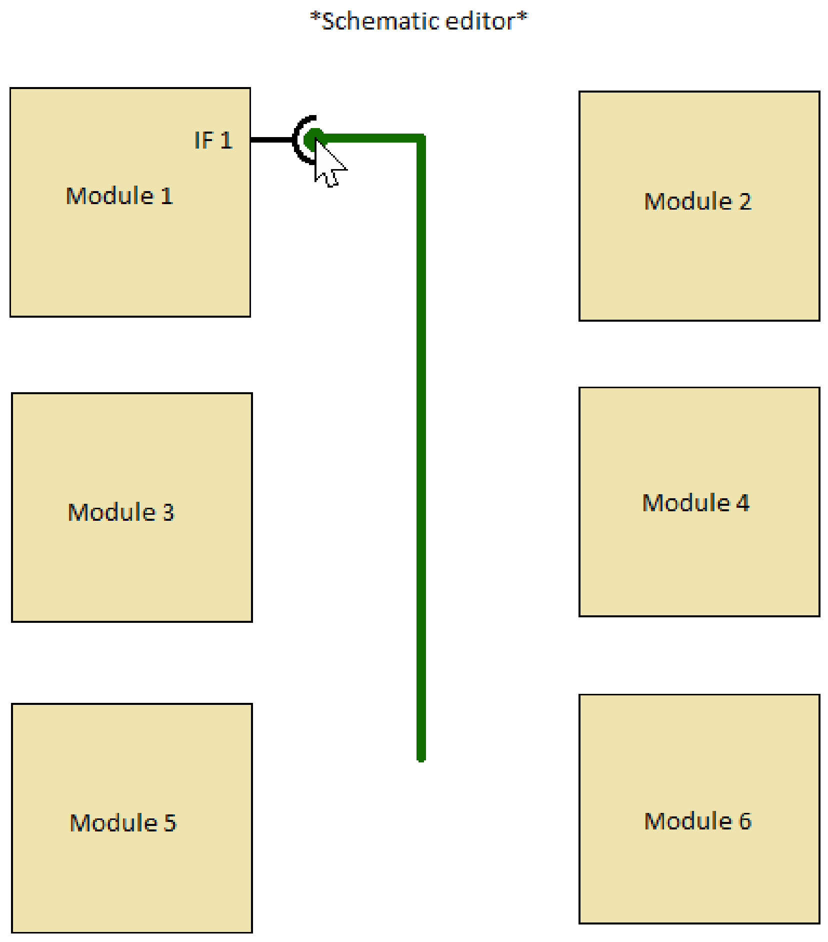


Figure 1.9: Creating an interface wire and connecting it to the first module.

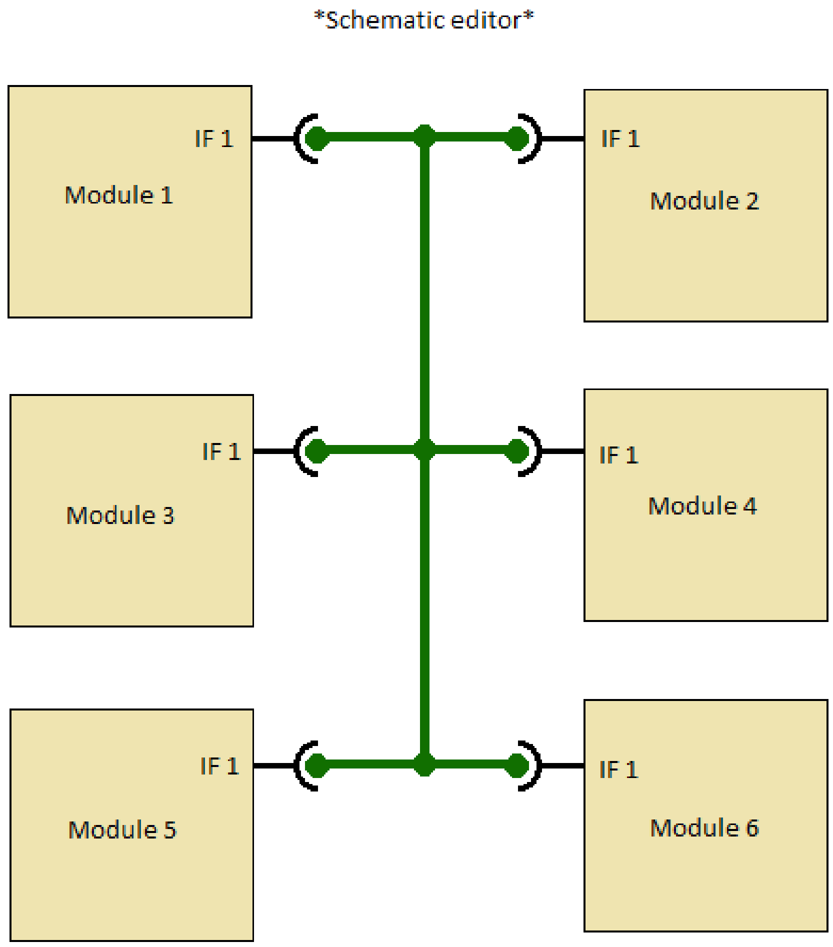


Figure 1.10: Connecting the interface wire to the rest of the modules.

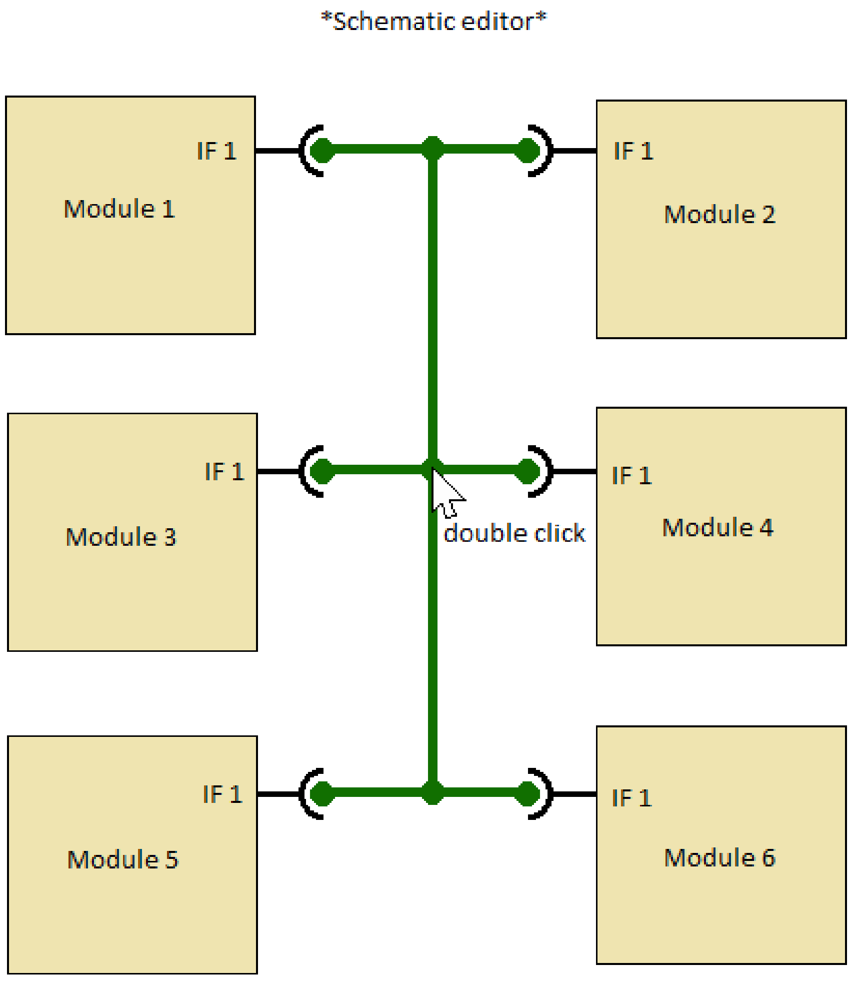


Figure 1.11: Double clicking on the interface will bring up either the interface config dialog or interface selection dialog.

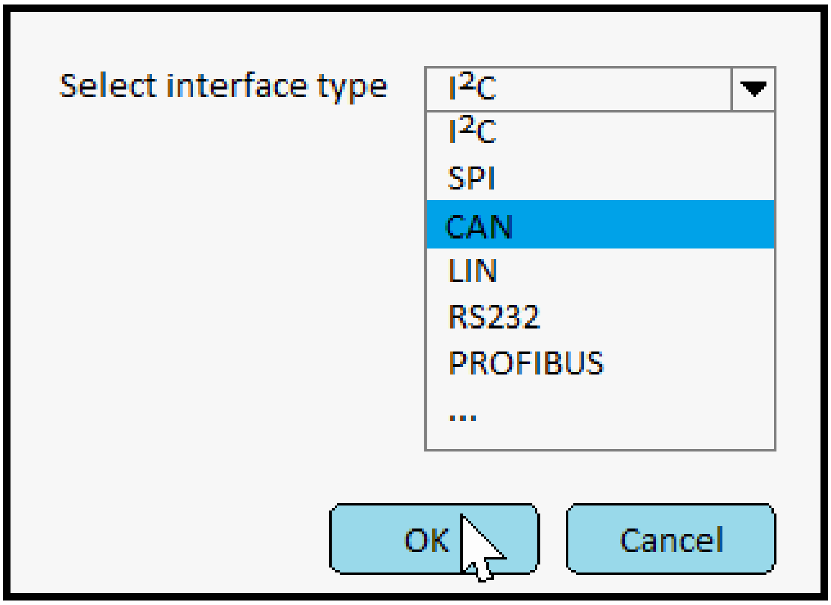


Figure 1.12: Choosing interface type.

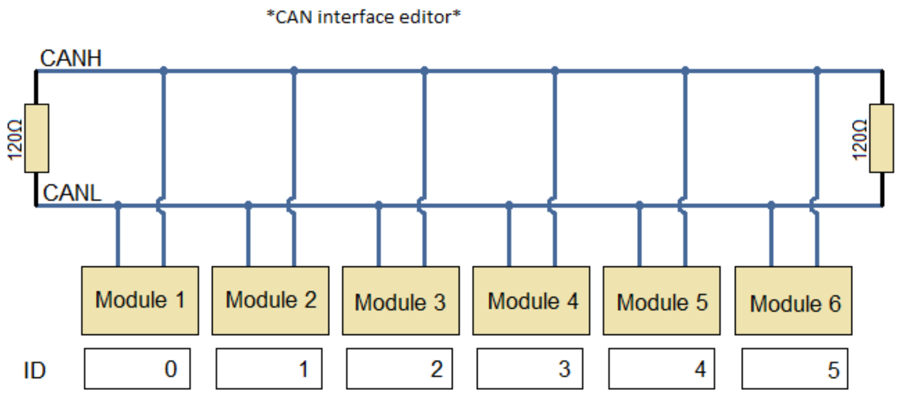


Figure 1.13: Interface configuration dialog for *CAN* will pop up.

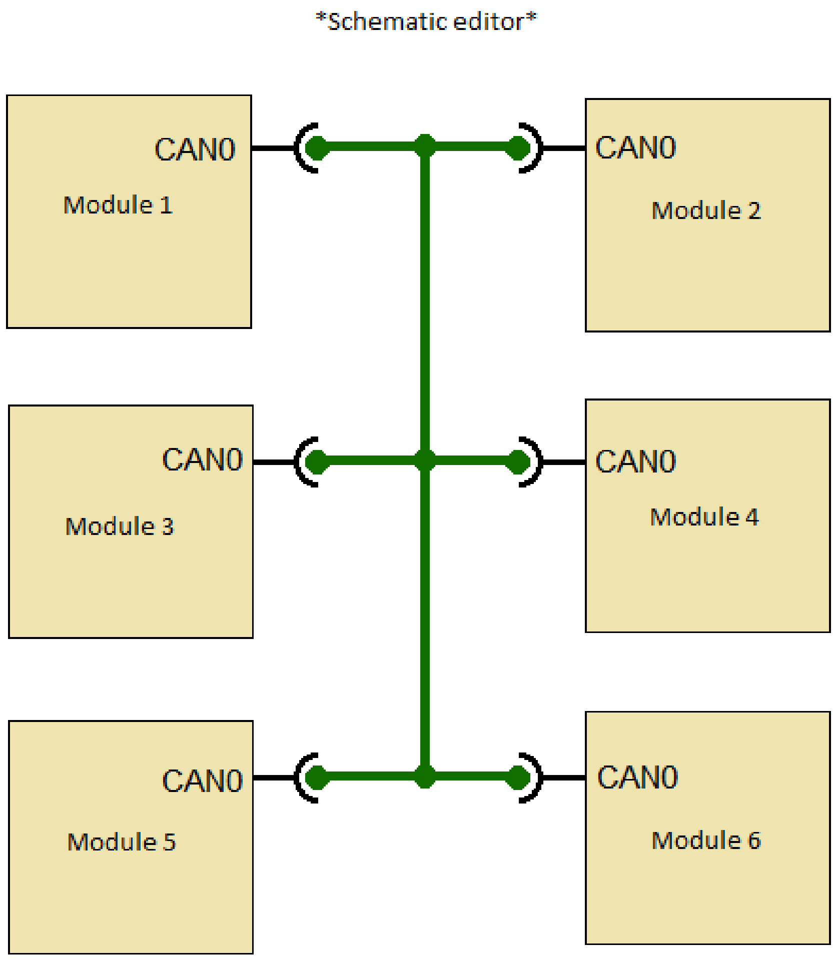


Figure 1.14: Upon saving the interface configuration dialog, the name of the interface will change in all modules.

1.3.4 Example: Module interconnection with SPI

In this example, the goal is to connect two modules and a memory card together on an *SPI* bus. The *SPI* bus consists of at least four lines called `MISO`, `MOSI`, SCK and one or more $\overline{\texttt{CS}}$. The bus master controls the data transfers by pulling low the $\overline{\texttt{CS}}$ line connected to the module it wants to exchange data with. The bus master usually controls the $\overline{\texttt{CS}}$ lines through `GPIO` pins controlled in software. Although the connection of *SPI* modules doesn't require any other external components, it will still appear pretty messy in the schematic editor.

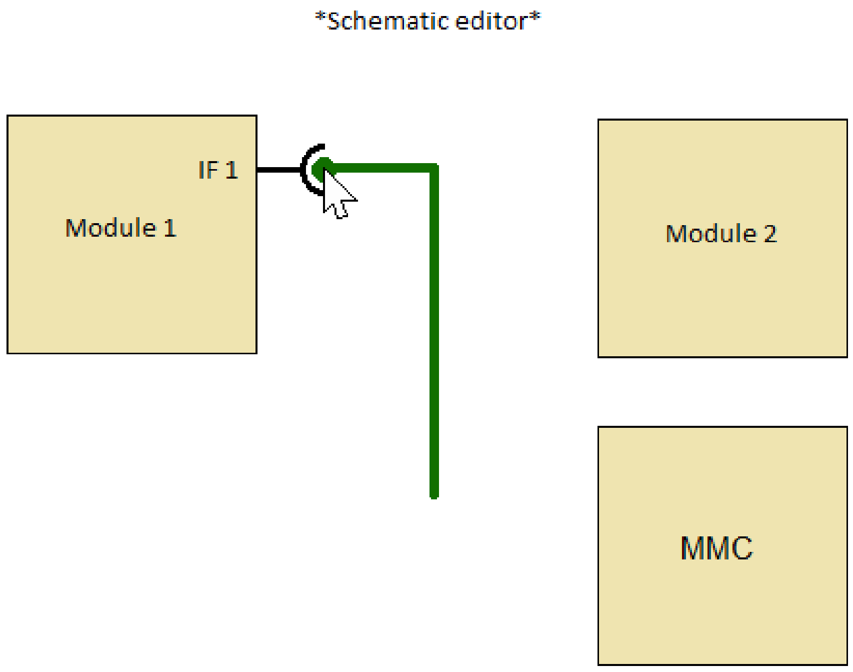


Figure 1.15: Starting out with two modules and a memory card, creating an interface wire and connecting it to the first module.

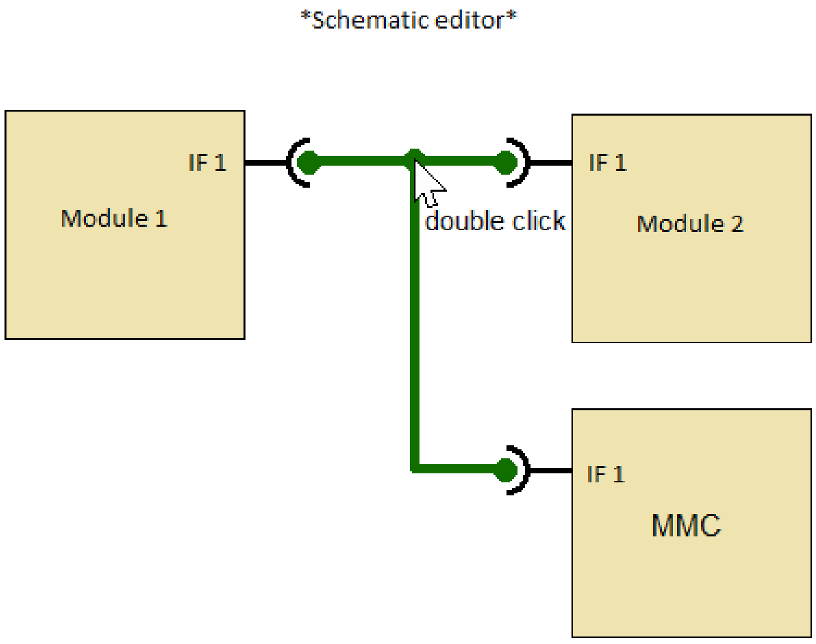


Figure 1.16: Connecting the interface wire to the other module and the memory card, and double clicking on the interface to bring up the interface config dialog or the interface selection dialog.

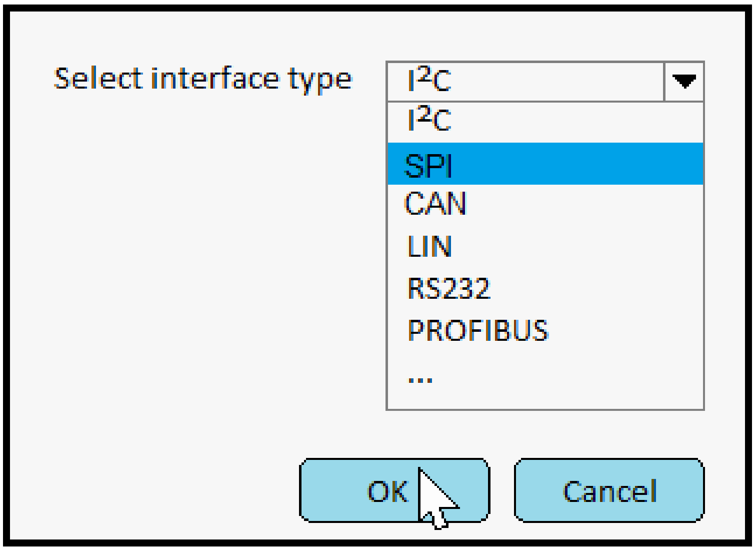


Figure 1.17: Choosing *SPI* as interface type. Memory cards typically only support *SPI* as secondary connection interface, so all the other interface types shown here should be greyed out or omitted.

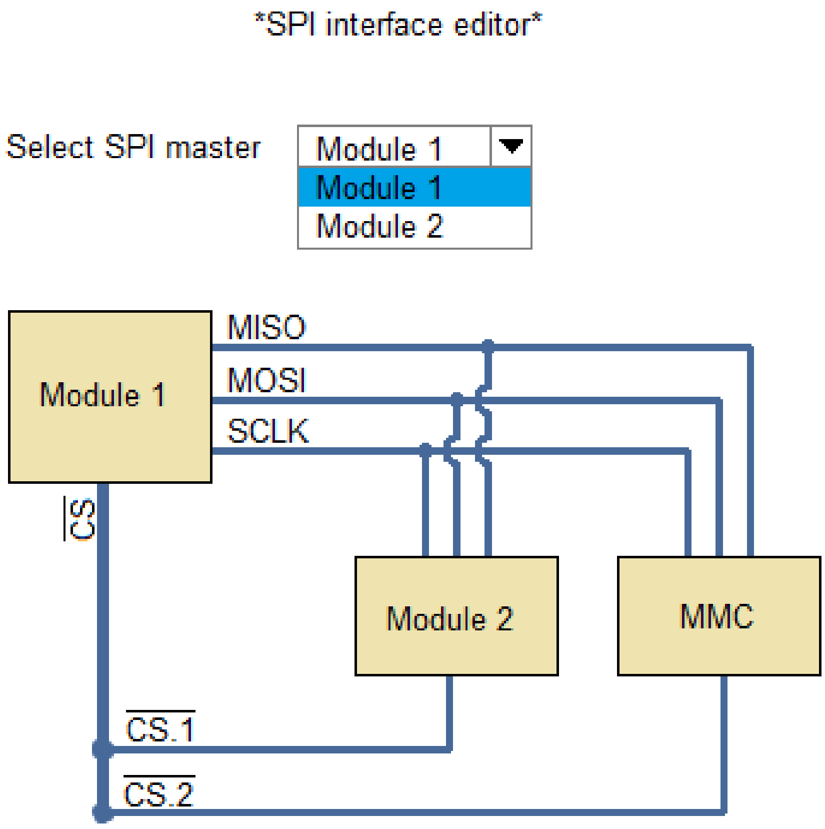


Figure 1.18: Upon selecting *SPI* as interface type, the interface configuration dialog for *SPI* will appear. The user will be able to configure, among other things, which module to operate as *SPI* master. It should also be possible to route other additional interfaces or wires in the interface, e.g. interrupt lines from the slaves to *SPI* master.

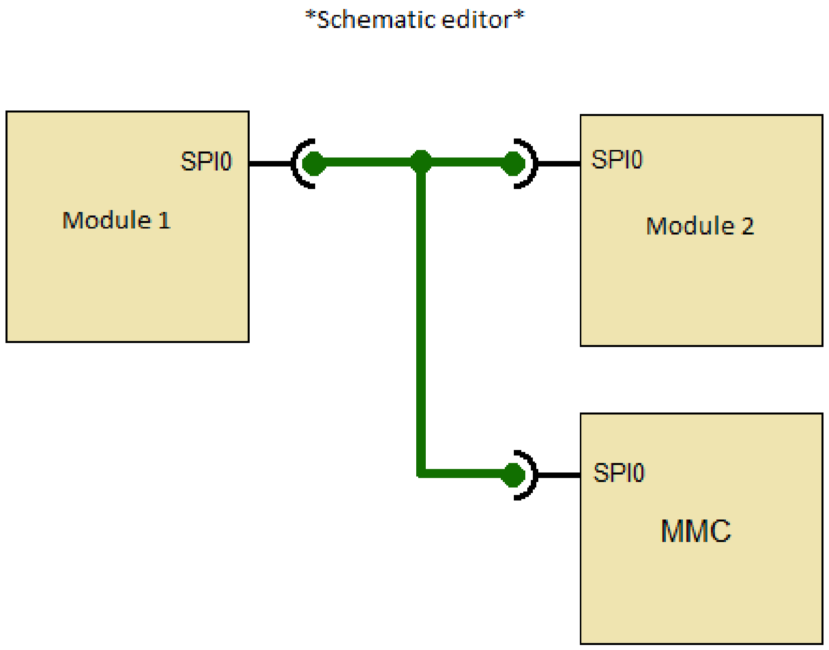


Figure 1.19: Upon saving the interface configuration dialog, the name of the interface will change in all modules.

1.3.5 Example: Module interconnection with I^{2}C

In this example, the goal is to connect six modules together on an I^2C bus. The I^2C bus consists of two lines, SDA and SCK, both of which will have a pull-up of 10 $k\Omega$ to V_{CC}. Usually when connecting modules together with I^2C they have to make sure not to place several pull-up resistors on each line. This example will show how to connect the modules together and how to choose how the slave modules will be able to communicate back to their master modules.

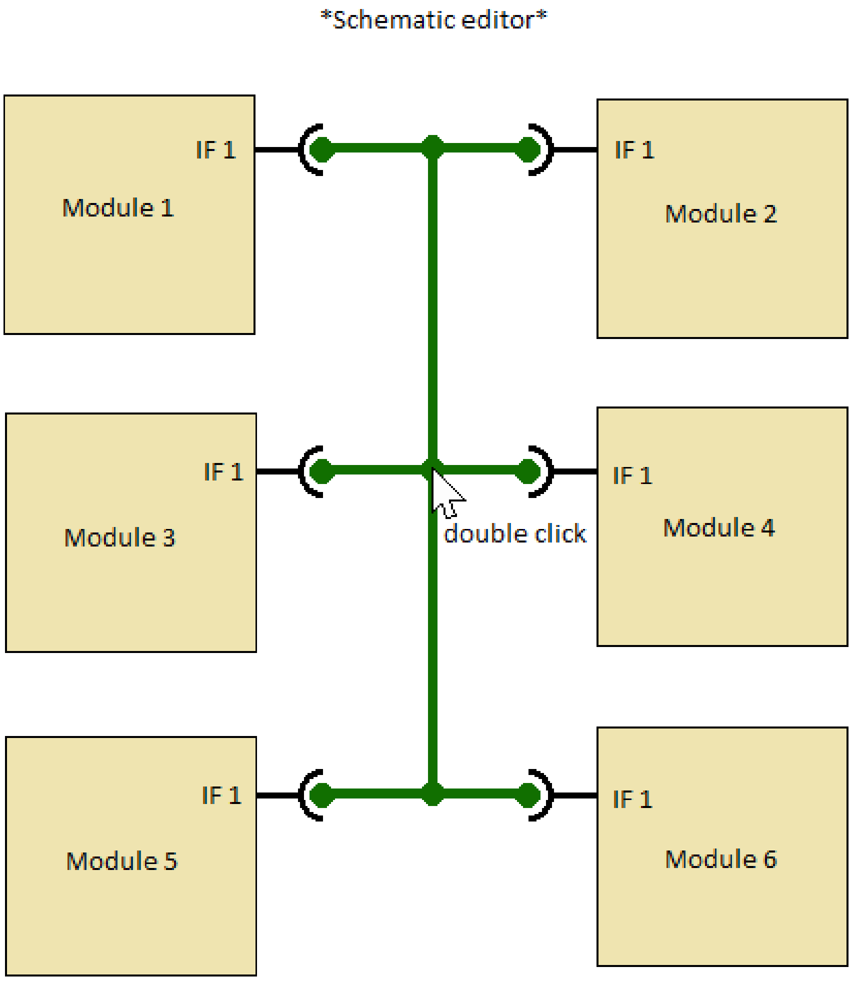


Figure 1.20: Starting out with six modules connected through a yet to be defined interface type. Double clicking on the interface wire should bring up the interface selection dialog.

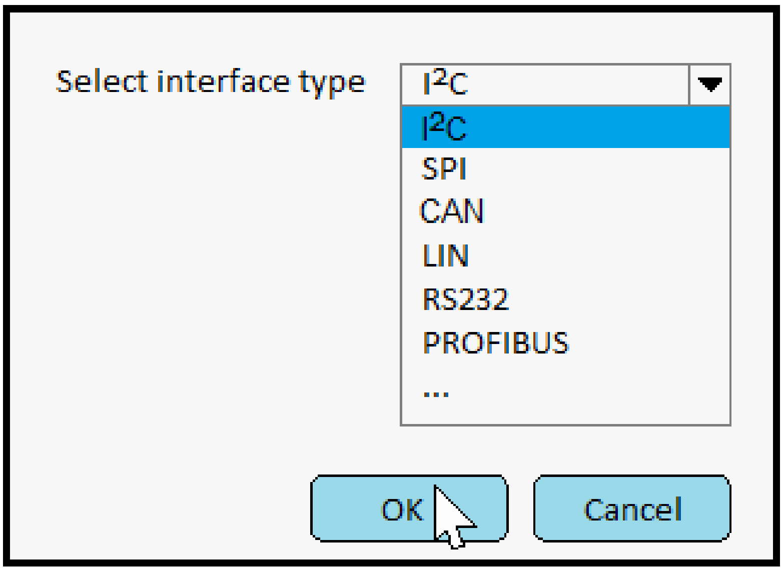


Figure 1.21: Choosing I^2C as interface type.

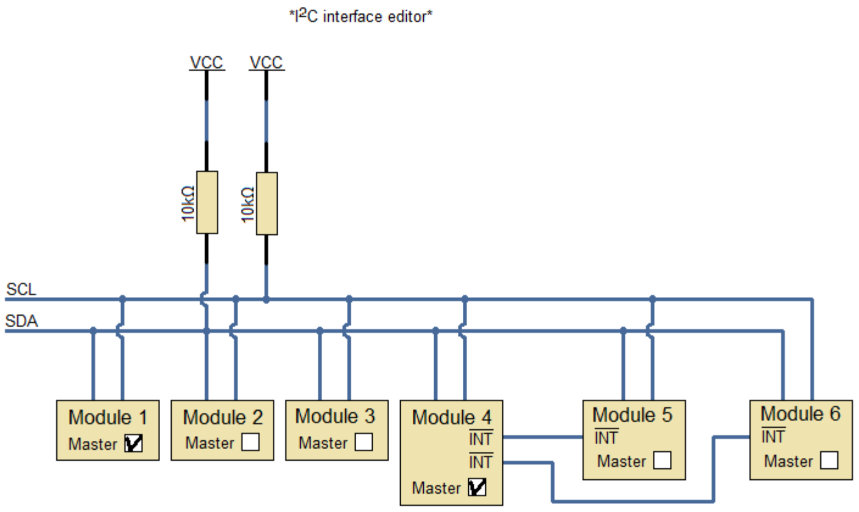


Figure 1.22: Upon selecting I^2C as interface type, the interface configuration dialog for I^2C should appear. Additional settings for the interface should be configurable here, for example choosing which modules to appear as I^2C masters and which to appear as slaves, as well as routing additional interfaces or wires to be used in the interface, e.g. interrupt lines from the slaves to the masters.

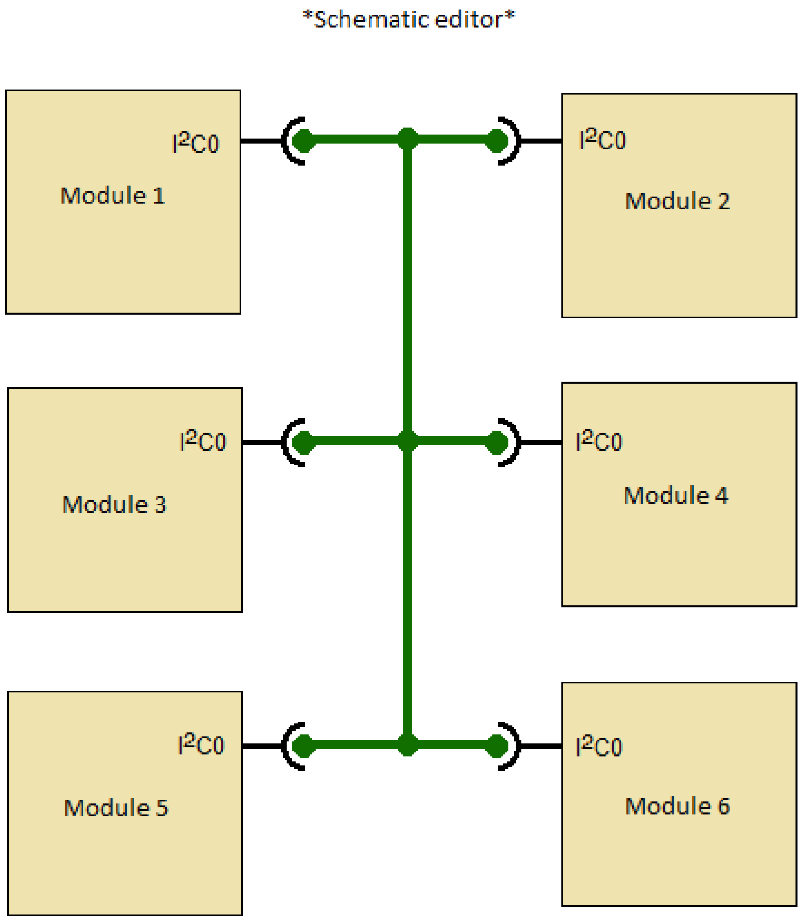


Figure 1.23: Upon saving the interface configuration dialog, the name of the interface will change in all modules.

2 Theory

2.1 Simulation

We consider to be a useful design tool, as it can give the designer immediate feedback on how a system will perform. Falstad Circuit Simulator, as mentioned in subsection 1.2.2 is a good example on how this can be done.

The state of an network of resistive components, voltage and current sources can be solved using Kirchhoff's current law: The sum of all currents emitting from a node is 0.

Kirchoff's current law can be applied to resistors, described by Ohm's law:

$$V = R \cdot I$$

When introducing reactive components, the equations become slightly more complex, but are still valid.

Capacitor:

$$C\frac{d}{dt}V = I$$

Inductor:

$$L\frac{d}{dt}I = V$$

Modified Nodal Analysis is formalization of Kirchoff's current law.

2.1.1 Modified Nodal Analysis

Modified nodal analysis is a method for determine the mathematical relations between different nodes and branches in a circuit. Resistors, capacitors, inductors, voltage sources and current sources are considered. This section (subsection 2.1.1) is a summary of the method described by [4].

Each component is considered a branch, and the connections between them nodes. Each component is given an orientation and for each component type a *incidence matrix* describing which branches and nodes are connected. The rows in the matrices represents nodes, and the columns branches. A value of *1* represents the branch starting in the specified node, and *-1* ending in the specified node. No connection is represented by a 0, and connections to ground are omitted. The incidence matrix is split by component type into the fellowing matrices:

- $\mathbf{A_R}$, representing resistors;
- $\mathbf{A_L}$, representing inductors;
- $\mathbf{A_C}$, representing capacitors;
- $\mathbf{A_V}$, representing voltage sources and
- $\mathbf{A_I}$, representing current sources.

In addition, the component values are represented by the fellowing diagonal matrices.

- $\mathbf{C}$, all capacitances.
- $\mathbf{G}$, all conductances.
- $\mathbf{L}$, all inductances.

The columns in the component value matrices corresponds to the columns in the incidence matrices. The current sources and voltage sources are represented by the vectors I and E, respectively. The vectors e, i_V i_L represent node voltages, current through voltage sources and current through inductors, respectively.

The fellowing matrix equation then describes the circuit.

$$\begin{bmatrix} A_C C A_C' & 0 & 0 \\ 0 & L & 0 \\ 0 & 0 & 0 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} e \\ i_L \\ i_V \end{bmatrix} + \begin{bmatrix} A_R G A_R' & A_L & A_V \\ -A_L' & 0 & 0 \\ A_V' & 0 & 0 \end{bmatrix} \begin{bmatrix} e \\ i_L \\ i_V \end{bmatrix} = \begin{bmatrix} -A_I I \\ 0 \\ E \end{bmatrix} \tag{2.1}$$

The article further does describe how to perform DC analysis to these equations, and how to calculate impedances.

2.1.2 Numerical Simulation

The goal of simulation is to provide the user with some feedback on how a circuit will perform. Providing a symbolic solution, and plotting this solution neither feasible for all possible circuits, nor is it flexible when introducing new non-linear components. Instead numerical simulation can provide a prediction of the circuit's behaviour.

Modified Nodal Analysis results in a system of differential algorithmic equations on the form:

$$F_1\dot{\mathbf{x}} + F_2\mathbf{x} + \mathbf{k} = \mathbf{0} \tag{2.2}$$

The matrix F_1 usually contain some empty(zero) columns, and the system can be transformed into the following form:

$$\begin{aligned} \dot{\mathbf{y}} &= F_y\mathbf{y} + F_z\mathbf{z} + \mathbf{f_c}, \\ \mathbf{0} &= G_z\mathbf{z} + G_y\mathbf{y} + \mathbf{g_c} \end{aligned} \tag{2.3}$$

Which is a semi-explicit differential algorithmic system. To simulate this system, an implicit Runge-Kutta method can be used(see Discusion).

$$\begin{aligned}\mathbf{x_{n+1}} &= \mathbf{x_n} + h[b_1\mathbf{k_1} + b_2\mathbf{k_2}] \\ \mathbf{k_1} &= f(\mathbf{x_n} + h[a_{11}\mathbf{k_1} + a_{12}\mathbf{k_2}], t_n + c_1 h) \\ \mathbf{k_2} &= f(\mathbf{x_n} + h[a_{21}\mathbf{k_1} + a_{22}\mathbf{k_2}], t_n + c_2 h)\end{aligned} \tag{2.4}$$

where

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

are the parameters from the methods' butcher tableau.

First we put the (continuous) system (Equation 2.3) on implicit form:

$$\dot{\mathbf{y}} = f(\mathbf{y}, t) = A\mathbf{y} + \mathbf{b} \tag{2.5}$$

where:

$$\begin{aligned} A &= F_y - F_z G_z^{-1} G_y \\ \mathbf{b} &= \mathbf{f_c} - F_z G_z^{-1} \mathbf{g_c} \end{aligned} \tag{2.6}$$

Combining the system (Equation 2.5) with an implicit Runge-Kutta method (Equation 2.4) yields:

$$\begin{aligned}\mathbf{y_{n+1}} &= \mathbf{y_n} + h(b_1\mathbf{k_1} + b_2\mathbf{k_2}) \\ \mathbf{k_1} &= A(\mathbf{y_n} + h[a_{11}\mathbf{k_1} + a_{12}\mathbf{k_2}]) + \mathbf{b} \\ \mathbf{k_2} &= A(\mathbf{y_n} + h[a_{21}\mathbf{k_1} + a_{22}\mathbf{k_2}]) + \mathbf{b}\end{aligned} \tag{2.7}$$

Solving $y_{n+1}$for y_n (See Equation A.4 - Equation A.12) yields:

$$\begin{aligned}\mathbf{y_{n+1}} &= \mathbf{y_n} + h(b_1\mathbf{k_1} + b_2\mathbf{k_2}) \\ \mathbf{k_1} &= M_1\mathbf{y_n} + N_1\mathbf{b} \\ \mathbf{k_2} &= M_2\mathbf{y_n} + N_2\mathbf{b}\end{aligned} \tag{2.8}$$

where:

$$
\begin{aligned}
M_1 &= (I - h^2 a_{12} a_{21} E_1 A E_2 A)^{-1} E_1 A (I + h a_{12} E_2) \\
M_2 &= (I - h^2 a_{21} a_{12} E_2 A E_1 A)^{-1} E_2 A (I + h a_{21} E_1) \\
N_1 &= (I - h^2 a_{12} a_{21} E_1 A E_2 A)^{-1} E_1 (h a_{12} A E_2 + I) \\
N_2 &= (I - h^2 a_{21} a_{12} E_2 A E_1 A)^{-1} E_2 (h a_{21} A E_1 + I) \\
E_1 &= (I - h a_{11} A)^{-1} \\
E_2 &= (I - h a_{22} A)^{-1}
\end{aligned}
\tag{2.9}
$$

2.1.3 Example System

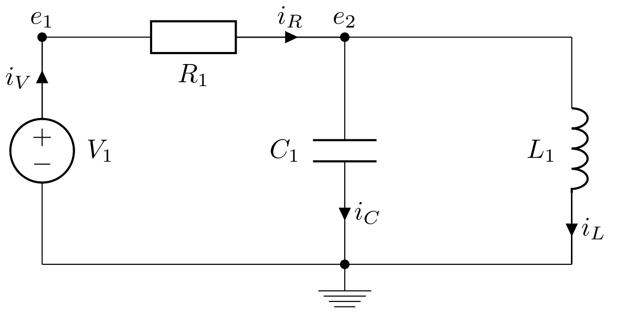


Figure 2.1: A simple RLC circuit

To compare some different numerical methods, the system in Figure 2.1 is considered.

Using MNA, we get:

$$
\begin{gathered}
\mathbf{A_R} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{A_C} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{A_L} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{A_V} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \\
\mathbf{G} = \begin{bmatrix} \frac{1}{R_1} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} C_1 \end{bmatrix}, \mathbf{L} = \begin{bmatrix} L_1 \end{bmatrix}, \mathbf{E} = \begin{bmatrix} V_1 \end{bmatrix}
\end{gathered}
\tag{2.10}
$$

Applying these values to the Equation 2.1, yield:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & C_1 & 0 & 0 \\ 0 & 0 & L_1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} e_1 \\ e_2 \\ i_L \\ i_V \end{bmatrix} + \begin{bmatrix} \frac{1}{R_1} & \frac{-1}{R_1} & 0 & -1 \\ \frac{-1}{R_1} & \frac{1}{R_1} & 1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ i_L \\ i_V \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -V_1 \end{bmatrix} \tag{2.11}$$

This can be transformed into:

$$\begin{aligned} \frac{d}{dt} \begin{bmatrix} e_2 \\ i_L \end{bmatrix} &= \begin{bmatrix} \frac{-1}{C_1 R_1} & \frac{-1}{C_1} \\ \frac{1}{L_1} & 0 \end{bmatrix} \begin{bmatrix} e_2 \\ i_L \end{bmatrix} + \begin{bmatrix} \frac{V_1}{C_1 R_1} \\ 0 \end{bmatrix} \\ \begin{bmatrix} e_1 \\ i_V \end{bmatrix} &= \begin{bmatrix} 0 & 0 \\ \frac{-1}{R_1} & 0 \end{bmatrix} \begin{bmatrix} e_2 \\ i_L \end{bmatrix} + \begin{bmatrix} V_1 \\ \frac{V_1}{R_1} \end{bmatrix} \end{aligned} \tag{2.12}$$

This system's exact solution has been solved in subsection A.3.1. The system has been simulated using a modified version of the application's numerical solver. Instructions on how to do this, and generate plots are in Chapter B.1: The fellowing numerical methods [5, from pp. 528 & 539] has been implemented and tested.

Explicit euler, Figure 2.2

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

Implicit euler, Figure 2.3

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

LobattoIIIC, order 2 Figure 2.5

$$\begin{array}{c|cc} 0 & \frac{1}{2} & -\frac{1}{2} \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Explicit Runge-Kutta. Figure 2.4

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array}$$

Figure 2.2: Simulation of RLC circuit using the Euler's method, with step size = 0.001s

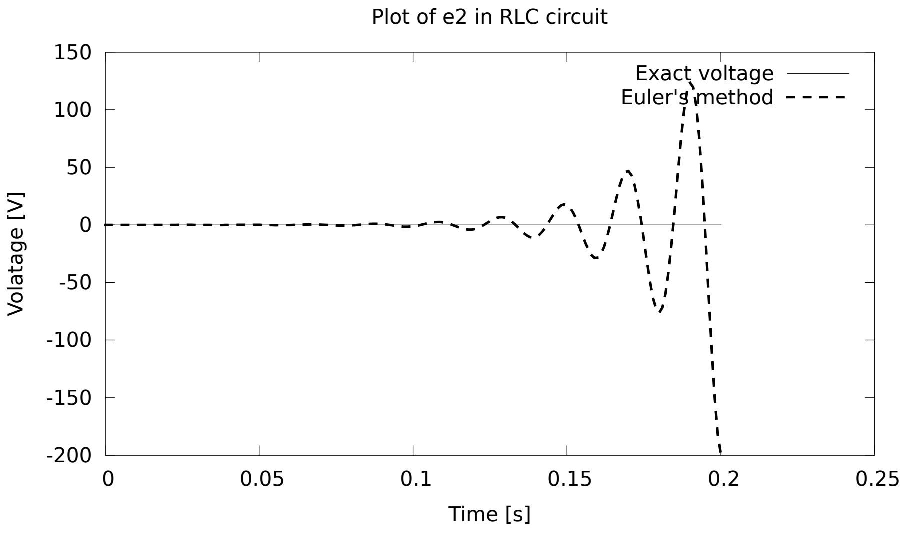


Figure 2.3: Simulation of RLC circuit using the implicit Euler method, with step size = 0.001s

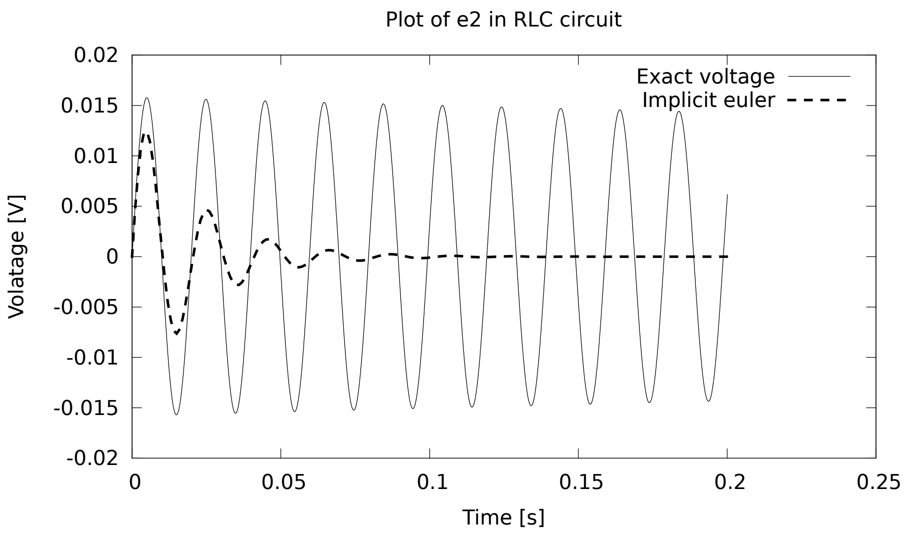


Figure 2.4: Simulation of RLC circuit using the Runge-Kutta method, with step size = 0.001s

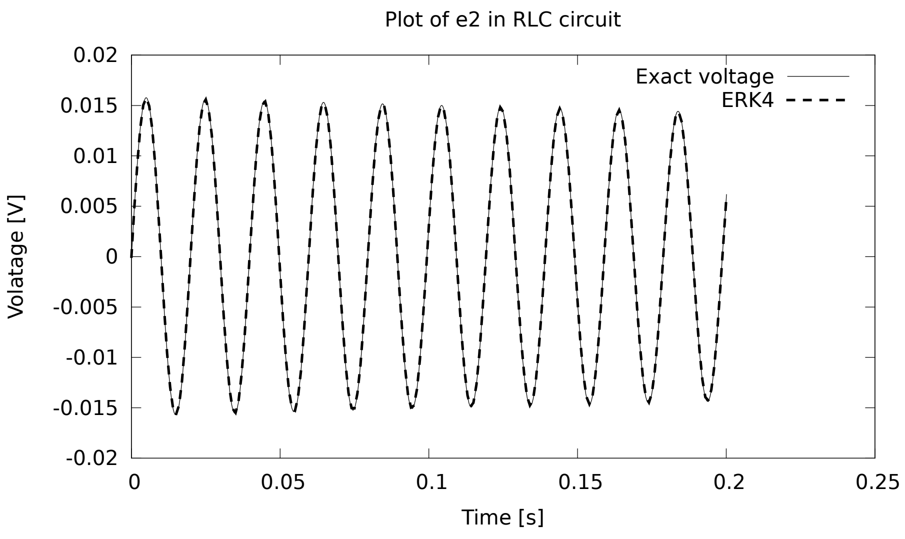
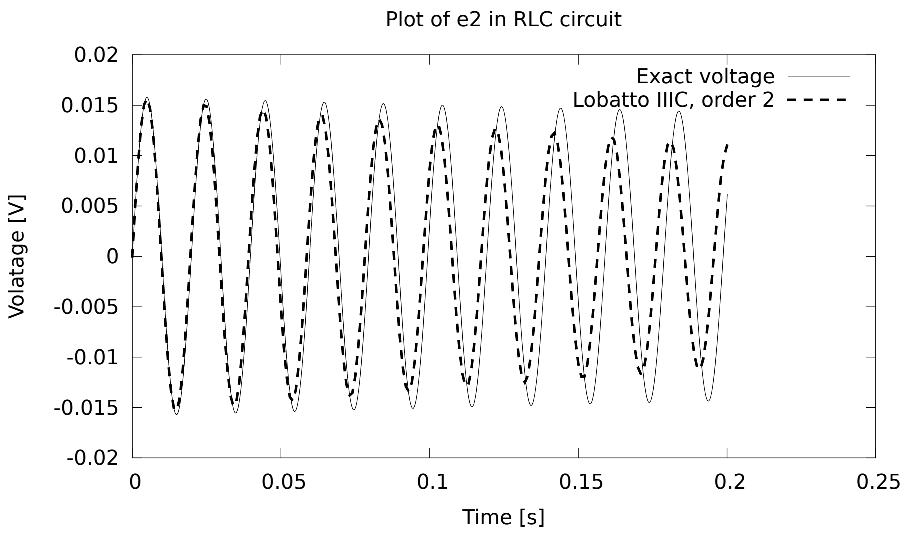


Figure 2.5: Simulation of RLC circuit using Lobatto IIIC, with step size = 0.001s



2.2 Web Applications

2.2.1 Interactivity

Since we have chosen to develop the EDA as a web-application, we will be implementing parts of the application in JavaScript. JavaScript run in the user's web-browser, and allows use of mouse position and movement and key-presses for interaction.

It is possible to build an web based application without using JavaScript, this is however highly impractical. Web-pages contain links, that allows you to open new pages, and the address of these pages can be interpreted by the server to generate new pages. Such pages are hereby referred to as dynamic web-pages. This technique is often used to build blogs and online newspapers, where the server mixes some data from a database with html and returns this to the user. Putting this to the extreme, one could build a web-application, where each web-page is unique, and the interactive elements on the page are unique links. Each these links are recognisable by the server, which has some action associated with each of these links. When an URL is requested, the action is performed, and a new web-page is generated for the user.

This approach is highly impractical for the following reasons: The server need to keep track of all the possible actions of all the possible users, including simple visual changes to the user interface. This requires a lot of resources. Each action must be sent from the user's web-browser to the server, where some appropriate action is taken, a new web-page is generated and sent to the user web-browser, where it must be rendered. This causes significant latency for all user actions, and simply isn't tolerable.

Instead of using JavaScript, one could choose to build the application using some browser plug-in, such as Flash or Java.

2.2.2 Rendering

The EDA will have a graphical interface to display what is currently being worked on. This graphical interface must be able to display complex graphical shapes. It is possible to draw the graphics on the server-side (in the backend), but this would cause intolerable strain on the server and its network connection. This would also result in significant latencies between a user action, and a response on the user's screen.

Our approach is to let the client's web-browser draw the graphics. There are several methods available:

SVG

Short for Scalable vector graphics. SVG is a XML based format for describing vector graphics, and can be embedded using the `<svg>` tag since HTML5. The shapes inside the SVG namespace are represented by XML tags, which can be made interactive by associating javascript functions to their mouse event properties. When shapes are clicked, the browser automatically triggers the shape's associated event function. The shapes can be modified, and are redrawn automatically. This redrawing does however tend to be very resource heavy when the shape complexity increases.

The most important advantages of SVG is:

- Build-in event detection.
- Only changes need to be updated.
- Recognised by other software.

Canvas 2D

HTML 5 introduces the `<canvas>` tag, and a API for drawing inside this element. Canvas appears to be better suited for dynamic content than SVG; it is quicker at drawing shapes. The drawback is a lower level API. The canvas only contains the rendered image, and figuring out what must be redrawn must be done by the application. The application must also generate events using the mouse position. Fortunately the canvas provides the option to draw an invisible path, and test whether a coordinate is within this path.

2.2.3 Persistent Storage

A user must be able to recover his work each time he uses an application. In traditional programs, running in the user's operating system, this is usually done by storing the user's work in the file system. Web-applications do not have direct access to the user's file system, and must store the data in some other way.

Copy-Paste

This is perhaps the simplest solution to implement. When the user is done working on some data, the web-application dumps the data describing his work in text form to the web page. This data can later be pasted into a textfield, and allow the user to continue working. The user is responsible for extracting, storing and retrieving the data. This method is clearly unacceptable from a usability perspective.

URL encoded data

Instead of the user copying and pasting the data, the web-application can generate a URL with the data structures describing the user's work embedded in the URL. This allows the user to download his work as if it was stored in a file. Later this file can be uploaded in order to continue working on the data described by the file. This method is similar to some applications running in the operating system, but is also infeasible when operating of several interconnected documents.

Web storage

The HTML5 specification adds *Web Storage*. Web Storage defines an API for persistent data storage of key-value pair data in the user's web browser. An important advantage of using Web-storage, is its ease of use. Data is stored with `localStorage.setItem("name of item","data");`, and retrieved with `localStorage.getItem("name of item");`. There is an upper limit of 10 megabytes that can be stored for each web-site in this storage. Data stored in the local storage is local to the user's webbrowser, and is lost if the user changes computers or deletes his browser history. Browsers are also free to expire old data. These factors makes Web Storage a non-useful location for persistent storage. It is however a very interesting caching location.

Server side storage

By storing the users data in the server, the user can access his data from any internet-connected computer with a web-browser. Some user id can be associated to the data each user generates. This is userful to determine who has access to what. Important drawbacks of server side storage is the need for user accounts, the resposibility of the user's data, and the increased network activity between server and client.

Security

When storing user information, protecting the user's data becomes an issue. If some third-party with bad intentions get a hold of the users passwords and usernames, he could potentially use this to compromise some of the users' other accounts, such as email. It is thus important to reduce the potential for a third party to gain access to the user's passwords.

An attacker on the same network as the client or server, or anywhere between, can see the data transferred between the client or server. This is solved by using HTTPS instead of HTTP. HTTPS encrypts the transmission between the client and server using a technique known as *public key cryptography.* When used correctly, this ensures that only the intended recipient is able to read the encrypted message. It is possible to break such an encryption, but the computational effort required to do so is very high. When an HTTPS connection is established, communication between server and client is secure; An attacker can pose as neither the server nor the client. However, when establishing the connection an attacker can pose as the server to the client, and as the client to the server; implementing what is called an *man in the middle* attack. A man in the middle attacks are avoided by using signed certificates. The certificate contains the public key of the server, and a signature from a certificate authority. This signature can be verified against the certificate authority's public key, which is usually signed by some other certificate authority. Although it is hypothetically possible to be able to imitate several certificate authorities, it is highly impractical even when controlling all data between the client and the internet.

An attacker able to gain control of the server could download information of all the users. Sometimes this results in peoples credit card information becoming public, as well as passwords and email accounts. To reduce the harm of a third-party acquiring the users' information, passwords should not be stored. Instead the passwords should be obfuscated using a cryptographic hashing algorithm and a salt, and this hash and salt is stored instead. A cryptographic hashing algorithm takes an arbitrary input and generates a fixed-size output. The input of a hashing function should not be computable from the hash, and small changes should cause large change in the output. By trying a lot of different inputs, a *brute-force attack*, a input generating hash matching the original hash will eventually be found. If the original input is short compared to the hash, the found solution is very likely to be equal to the original input.

The salt is a sequence of random characters used to add entropy to the hash, by e.g. prepending it to the password before generating the hash. A unique salt should be generated for each user. If no salt is used a brute-force attack becomes a lot more practical, as each generated hash can be tested against all users. List of precomputed hashes for common passwords can also be used, given the same hashing algorithm is used. When a salt is used, an attacker having gained posession of both the users hashes and salts must attack each user's password independently For passwords with sufficient entropy, this becomes infeasible.

2.3 JavaScript

2.3.1 Object Model

JavaScript has quite a few features that can be considered odd from the perspective of other languages. Especially the objects can be seem a bit odd when coming from C++ or Java; it does not have classes. It does however have constructors and inheritance (prototype inheritance).

Objects in JavaScript are a collections of named values, or key value pairs. These named values can be strings, numbers, arrays, functions or objects. Strings, numbers, arrays and functions are also objects, and can be assigned key-value pairs. This is where the "Everything is an object" phrase stems from. Apart from these special case objects, objects are created either using curly brackets or using the `new` keyword.

Curly brackets notation

As demonstrated in Listing 2.1, objects can be created with or without initial values. The names of the values and the values are separated with colons, and the values separated with commas. It is possible create empty objects using an empty pair of brackets. After an object is created it is still possible to add and change it's values.

```javascript
var foo = {};
var bar = {
  strVal: "Lorem Ipsum",
  numVal: 42,
  arrVal: [2,7,1,8,2,8,1,8,2,8],
  objVal: { yo: "dawg" },
  fncVal: function(arg1) {
    return arg1 * numVal;
  }
};
bar.postVal = 5;
```

Listing 2.1: Object creation with curly brackets

The curly bracket notation is very similar to JSON (JavaScript Object Notation). Listing 2.2 is a JSON document with the same values as in the object bar in Listing 2.1. The value `objVal` has not been included since JSON does not support function. Also note that the name of the values, the keys, are enclosed in quotation marks.

```
{
  "strVal": "Lorem Ipsum",
  "numVal": 42,
  "arrVal": [2,7,1,8,2,1,8,2,8],
  "objVal": {"yo": "dawg"}
}
```

Listing 2.2: Example of JSON

The `new` keyword

The `new` keyword creates a new object and sets the object's *prototype* (`__proto__`) to a reference of the constructors function's `prototype` property. `new` then calls the specified constructor on the newly created object. In the constructor `this` refers to the newly created object.

```
var foo = new Object();
var Bar = function() {
  this.baz = 42;
};
var qux = new Bar();

console.log(foo.bar); //prints undefined
console.log(qux.bar); //prints 42
```

Listing 2.3: Object creation with `new`.

The special constructor function `Object` constructs an empty object, which is equivalent to assigning a pair of empty curly brackets.

2.3.2 Prototype inheritance

JavaScript, as mentioned, has no classes, but it does have inheritance. What this means is an object can inherit some other object's properties. This is done by using the `new` keyword. After `new` has created a new object, it assigns it an prototype (i.e. its (__proto__) property). The prototype is set to the object pointed to by the constructor's `prototype` property. By modifying an constructor's `prototype` property, all objects created by the constructor will inherit it's values. Note that the prototype is copied by reference, not value, so a change to an object's prototype will affect all other objects created by the same constructor.

```
var Foo = function(){};
Foo.prototype.bar = 42;
var baz = new Foo();
var qux = new Foo();
console.log(baz.bar);       //prints 42
console.log(qux.bar);       //prints 42
Foo.prototype.bar = 2.718281828904590;
console.log(baz.bar);       //prints 2.71828182890459
console.log(qux.bar);       //prints 2.71828182890459
```

Listing 2.4: Example of prototype inheritance

In Listing 2.4 `baz` and `qux` has the same prototype, i.e. their (`__proto__`) refer to the same object. This causes any change to this object to affect both `baz` and `qux`. However, changing the Foo.prototype to a new object will affect neither `baz` nor `qux`, since they will be referencing the old one. The relations of most of the objects involved in Listing 2.4 is described in Figure 2.6.

What actually happens is: When some variable is referenced, the interpreter first looks for the variable in the specified object. If the variable is not found, the interpreter then looks for the variable in the object's prototype. This is done recursively until the *default object*, `Object.prototype`, has been searched. The search will stop since the *default object* (`Object.prototype`) has no prototype (`Object.prototype.__proto__` is `null`). If the requested variable has not been found, a reference error is thrown.

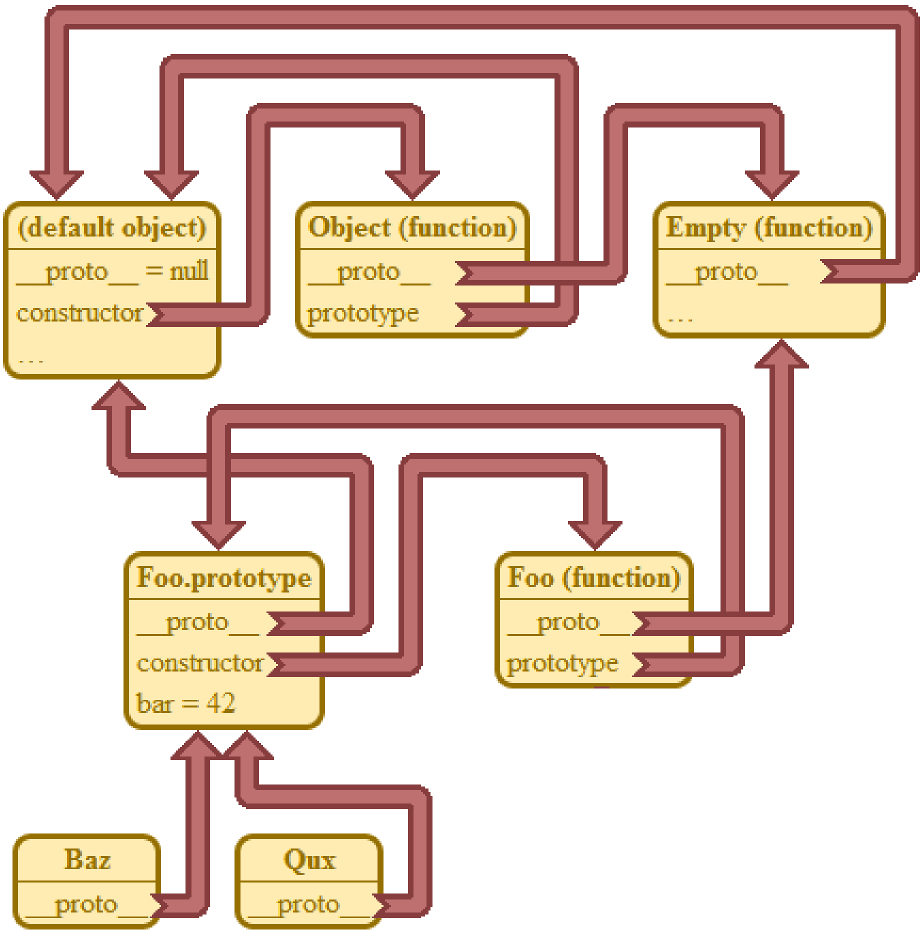


Figure 2.6: The red arrows are references. Note: The value 42 of Foo.prototype is actually an object too; it has the prototype `Number.prototype`.

By adding values to the default object, all objects will inherit these values. This can lead to slightly bizarre results, as demonstrated in Listing 2.5 Although any function can be used as a constructor, and all functions have the `prototype` property; if a function returns an object, the newly created object with an assigned prototype will be discarded in favour of the returned object. Thus, if a function returns an object, using the `new` keyword on this function only affects the `this` keyword inside the function used as an constructor.

```
Object.prototype.foo = "bar";
var baz = 42;
//The fellowing lines all evaluates as true:
baz.foo == "bar"
Object.foo == "bar"
{}.foo == "bar"
false.foo.foo.foo.foo.foo == "bar"
```

Listing 2.5: Effects of manipulating the default object.

2.3.3 Scope

When a variable is defined, it is available by its defined name in some limited part of the program. Where this variable is available is referred to as its scope. In JavaScript the scope is defined by functions. A variable defined inside a function, is only available inside the function. New function declarations inside the function where the variable is defined are also part of the variable's scope.

```
(function(){                      //outer function
  var foo = "bar";
  (function(){              //inner function
    var baz = "qux";
    console.log(foo); //outputs "bar"
    console.log(baz); //outputs "qux"
  })();

  console.log(foo);         //outputs "bar"
  console.log(baz);          //throws a ReferenceError.
})();
```

Listing 2.6: Demonstration of scope

In case Listing 2.6 is confusing: functions can be defined anonymously (with no associated name). Functions are just objects just as all other values, and can be passed as arguments and assigned to variables. It is also possible to call functions in place as they are defined, which is exactly what happens in Listing 2.6.

Note that it is possible to define functions referencing variables not yet defined, as long as the function is executed after the variable has been defined. As demonstrated in Listing 2.7, although `foo` is defined before `baz`, which it references, since `baz` is defined before `foo` is called, no `RefrenceError` is thrown.

```
(function(){                           //outer function
  var foo = function(){                   //inner function
    var bar = 1;
    console.log(bar); //outputs 1
    console.log(baz); //outputs 2
  };
  var baz = 2;
  foo();

  console.log(baz);          //outputs 2
  console.log(bar);          //throws a ReferenceError.
})();
```

Listing 2.7: Function definitions can preceed the variables it refrences

Unlike some languages such as *C*, JavaScript does not have block scope. Thus control structures, i.e. conditional structures and loops, does affect the scope of variables. Variables defined inside control structures does not go out of scope when exiting the control structure.

```
for(var i = 0; i < 5; i++) {
  var foo = i;
}
console.log(foo); //outputs 4
```

2.3.4 Closures

A function's *Closure* refer to the variables whose scope includes the aforementioned function. When function's are passed around, their closures are not affected.

```
var setbar;
var getbar;
(function(){
  var foo;
  setfoo = function(val) {
    foo = val;
  };
  getfoo = function() {
    return foo;
  };
})();
setfoo(25);
console.log(getfoo());//Outputs 25;
console.log(foo)      //Throws an ReferenceError
```

Listing 2.8: Demonstration of closures

In Listing 2.8, `foo` is part of `setfoo`'s and `getfoo`'s closure. When the anonymous function in which `setfoo` and `getfoo` are defined returns, `foo` goes out of scope, and is no longer defined anywhere but in `setfoo`'s and `getfoo`'s closures. This provides a form of encapsulation, a hiding of inner variables from the outer world. This is similar to the effect of the `private` keyword in languages such as *Java* and *C++*.

Variables that are not referenced anywhere when they go out of scope are garbage collected. Using closures is thus a good way of keeping memory use down when using a lot of temporary variables. Exploiting closures is thus a useful design pattern, not only for the encapsulation it provides.

```
var fibclosure = function() {
  var someValue = 1;
  var someList = [0, 0];
  var iterate = function() {
    someValue += someList.shift();
    someList.push(someValue);
    return someValue;
  };
  var lastval = function() {
    return someValue;
  };
  return {
    iterate: iterate,
    lastval: lastval
  };
}();
```

Listing 2.9: Fibonacci numbergenerator with hidden variables.

Listing 2.9 is an example of an design pattern exploring closures. It is a rather stupid implementation of a Fibonacci number generator, but it demonstrates the creation of an object with hidden variables. Public methods must be explicitly added to the returned object, while private and temporary variables are kept in generating function.

Example output:

```
console.log(fibclosure.iterate());  //Returns 1
console.log(fibclosure.iterate());  //Returns 1
console.log(fibclosure.iterate());  //Returns 2
console.log(fibclosure.iterate());  //Returns 3
console.log(fibclosure.iterate());  //Returns 5
console.log(fibclosure.lastval());  //Returns 5
console.log(fibclosure.someValue);  //Returns undefined
```

2.3.5 The `this` keyword

In *C++* and *Java*, when `this` is encountered, it refers to the object which contain the function. In JavaScript, this is not always the case, as `this` depend on how a function is called. Any function call will change what `this` refers to.

When `this` is used inside a constructor, `this` refer to the object being initialized.

```
var Foo = function() {
  this.bar = 1;
};
var foo = new Foo();
console.log(foo.bar);//outputs 1
```

When used as a member function of an object, `this` will refer to the object. var foo = bar: function() console.log(this) ; foo.bar(); //outputs the object foo;

When function is called directly, `this` refer to the global object. In a web-browser the global object is `Window`.

```
var foo = function() {
  console.log(this);
};
foo(); //outputs the object Window;
```

Note that `this` in an outer function may not be the same as `this` in an inner function; How the function is called, not how it is defined determines what `this` is.

```
var Foo = function() {
  var bar = function(){
    console.log(this);
  };
  bar(); //Will output the object Window
  this.bar = bar;
};
var foo = new Foo();
var baz = foo.bar;
foo.bar();          //will output the object foo.
baz()               //will output the object Window.
```

If the `this` value is needed inside an inner function, `this` can be assigned to a variable. This is particularly useful when using callbacks, where `this` will refer to the object calling the callback. In event-handlers for HTML, `this` will refer to the element causing the event.

```
function foo(callback) {
  callback("bar");
};

function Baz() {
  var obj = this;
  foo(function(res){
    obj.text = res;
  });
};
var baz = new Baz();
console.log(baz.text) //outputs "bar"
```

2.3.6 Third-party libraries for JavaScript

In web development, it is fairly common to use third-party libraries (e.g. Dojo, JQuery or MooTools) to add functionality to the web page. There are several reasons for this. First of all, many web developers come from a designer background and has a limited understanding of programming. These frameworks allow the developer to add common functionality to the web site with as little coding effort as possible. Examples of this is image galleries, etc. Secondly, these frameworks have (usually) put an effort into reducing (where possible) the difference in functionality between platforms (i.e. web browsers) where they deviate from the W3 standard. Thus, using a web framework will in general reduce the development and debugging effort, due to only having to write one base of code for all platforms. Thirdly, they tend to extend the JavaScript language with additional functionality, both in the objects already in the JavaScript standard (e.g. JQuery's inArray(object) method addition to the array object) as well as new objects and methods (e.g. MooTools' Class() constructor).

Typically, the third-party libraries also have subframeworks for typical GUI widgets for web applications. These include tree views, movable and resizable in-browser windows, drop-down menus, tabs, sliders, buttons, etc. The features of these frameworks vary, as does their intention and documentation.

2.3.7 Design Patterns: Asynchronous Module Definition (AMD)

File loading in JavaScript is not as easy as writing "include", as you do in many other languages. This is due to the fact that your web browser has to load every file from a web server somewhere in the world, and so every include statement has to prepare a http request packet, send it, and wait for a response. This is a slow process, and halting the program flow to wait for a file to load may cause your program to suffer from slow loading time. The solution for this is that every http request is provided a callback function to be executed when the file has finished loading, like below.

```
var client = new XMLHttpRequest();
client.open('GET', '/foo.txt');
client.onreadystatechange = function() {
  alert(client.responseText);
}
```

```
client.send();
```

There are several issues with this way of loading external files. First of all, there is no way of knowing if there is already a pending request for the file you want to load, or if it is loaded already. Thus, you might end up slowing down your program due to unnecessary file requests. Secondly, a program module is often dependent on several files, and so all file request callbacks will have to check if all other files are loaded, and if they are, execute the initialization code for the module.

```
var scr1, src2;
var init = function() {
  // module initialization code
}
var req1 = new XMLHttpRequest();
var req2 = new XMLHttpRequest();
req1.open('GET', '/foo.js');
req1.onreadystatechange = function() {
  scr1 = eval(req1.responseText);
  if(scr1 && scr2) {
    init();
  }
}
req1.send();

req2.open('GET', '/bar.js');
req2.onreadystatechange = function() {
  scr2 = eval(req2.responseText);
  if(scr1 && scr2) {
    init();
  }
}
req2.send();
```

In Listing 2.3.7, the ”init” function will not be called before both scr1 and scr2 are loaded.

The task described above is called Asynchronous Module Definition, or AMD for short. The idea is to allow the user to load parts of the code during runtime, allowing the developer to easily structure his application into modules, and load these when needed. The AMD library also knows which modules are already loaded, and if two modules requires the same dependency module, it is loaded only the first time one of them requests it.

Considering the same application as in Listing 2.3.7, loading the same files through the Dojo framework is done like this:

```
define("foo", [], function() {
  return "Foo!";
});
define("bar", [], function() {
  var printBar = function() {
    console.log("Bar!");
  }
  return printBar;
});
define("myModule", ["foo", "bar"], function(Foo, Bar) {
  // module initialization code;
  // called when both foo.js and bar.js are loaded

  console.log(Foo); // Will output "Foo!";
  Bar();     // Calls the function printBar,
        // thus outputing "Bar!"
});
```

Dojo will then store a new key-value pair with ”myModule” as the key and the return value from the callback function as the value. This return value will then be easily accessable from another module by listing it in its dependency array. Thus, one module can be a string of text; another, a JavaScript function; a third, an block of application initiation code.

2.3.8 Design Patterns: Mixins

As explained in 2.3.1, JavaScript's object model is quite different from those in traditional object-oriented programming languages (e.g. Java), and thus allows for some unusual design patterns. One of these is the mixin pattern. The concept is somewhat similar to multiple inheritance in C++, although the mixin pattern is arguably more flexible in nature.

The concept of the mixin pattern is to mix in (hence the name) functionality, properties, etc. into an already initiated object. This is different from inheritance in the sense that it works on object level (and after initialization), rather than on class level. This allows for an infinite number of different ”classes”, in that two objects created with the same constructor can later be assigned with different additional properties, such as interaction events, color, etc.

There are two ways to implement the mixin pattern, although they are just different ways of doing the same thing, that is, assigning new properties to an object during runtime. One way is to iterate through all the properties of the parent and assigning these to the child as well. In JavaScript, all objects are implemented as hash tables, and hence, writing ”object.property” is the same as writing ”object[”property”]”, so assigning new properties to an object is easily done like in 2.10.

```
function mixin(giver, taker) {
  for(par in giver) {
    taker[par] = giver[par];
  }
}
function Car() {
  this.drive = function() {
    console.log("driving");
  }
}
function Red() {
  this.color = "red";
}
function Blue() {
  this.color = "blue";
}

var myRedCar = new Car();
mixin(Red, myRedCar);

var myBlueCar = new Car();
mixin(Blue, myBlueCar);

console.log(myRedCar.color);  // Will output "red"
console.log("myBlueCar.color);  // Will output "blue"
```

Listing 2.10: Mixin using a giver-taker function

Another way is to call a mixin function from inside of an object's scope. This is done by invoking the function's ”call” function (JavaScript functions are also objects), which takes the scope object as first parameter, followed by the function's own parameters. By assigning a value to a variable preceded with the ”this” keyword within the function will then assign the variable to the object in scope. Likewise, accessing a variable preceded with the ”this” keyword will access the object's member variable with that keyword.

```
function Car() {
  this.drive = function() {
    console.log("driving");
  }
}
function Red() {
  this.color = "red";
}
function Blue() {
  this.color = "blue";
}

var myRedCar = new Car();
Red.call(myRedCar);

var myBlueCar = new Car();
Blue.call(myBlueCar);

console.log(myRedCar.color);  // Will output "red"
console.log(myBlueCar.color); // Will output "blue"
```

Listing 2.11: Mixin using the `this` keyword

2.3.9 Passing references of references

A particular problem we encountered during development was the need to have two objects sharing access to a string or number. The problem is: When either objects change the string or value, the change only occur on one of the objects, while what was needed was the change to occur in both.

```
var foo = { string: "foo's string" };
var bar = { string: foo.value };
bar.string = "bar's string";
console.log(foo.string); //outputs "foo's string"
```

Strings and numbers are objects; they are also immutable objects, i.e they cannot change. So *foo.value* and bar.value are references, both initially to the object (”foo's string”). When we try to change the value in bar, a new object(”bar's string”) is created, and `bar.value`'s is changed to a reference to the newly created object.

The solution used in the application is to wrap the shared values in an object.

```
var foo = { string: { value: "foo's string" } };
var bar = { string: foo.string };
```

```
bar.string.value = "bar's string";
console.log(foo.string); //outputs "bar's string"
```

3 Implementation

3.1 Platform

In order to create an easily extendable software application, a software platform needs to be established, and the application will then be built on top of this. One of the main features of this application will be giving the user the ability to extend the code base with new plugins and sharing these with others. The platform will provide a set of tools, widgets, objects and functions with which the application itself will be created. In fact, all editors provided in the application (schematic editor, component editor, layout editor, etc.) will be implemented as plugins, and such, the user will be able to develop his own set of tools for creating his application.

3.2 Frontend

The frontend is the part of the application that runs in the user's web browser, and consists HTML files for graphical layout, and JavaScript files for adding functionality to the application.

In this project, which framework to use, if any, had to be decided upon. Among the frameworks considered were

- JQuery
- MooTools
- Dojo

- Require.js
- Common.js

The choice eventually fell upon Dojo, due to two factors. First of all, Dojo comes with a complete GUI package called Dijit, which would ease the development of the graphical interface for the project to a great extent. Drop down menus, tab areas and dialog windows were all needed for the project, and these are all part of the Dijit package. Secondly, the need for a module management library was stated early on during development. Several AMD-specific libraries, like require.js and common.js, were considered, but Dojo was chosen due to having many features valuable for this project bundled in one package.

Please note that this application is still early in development, and some of the modules described below is planned for major refactoring.

3.2.1 Frontend building blocks

As explained in the beginning of this chapter, all the different editors will be built as application plugins, and so, the application needs some highly customizable building blocks in order to ease the coding effort as much as possible. In this section, the building blocks which the frontend consists of, will be explained.

WidgetBase

This mixin provides the widget with basic GUI functionality. It is intended as a middle-layer between the layout provider (i.e. Dojo) and the application. By doing this, the application will not be heavily dependent on the chosen layout provider to work properly, thus allowing for an easy switch to another layout provider or writing one from scratch.

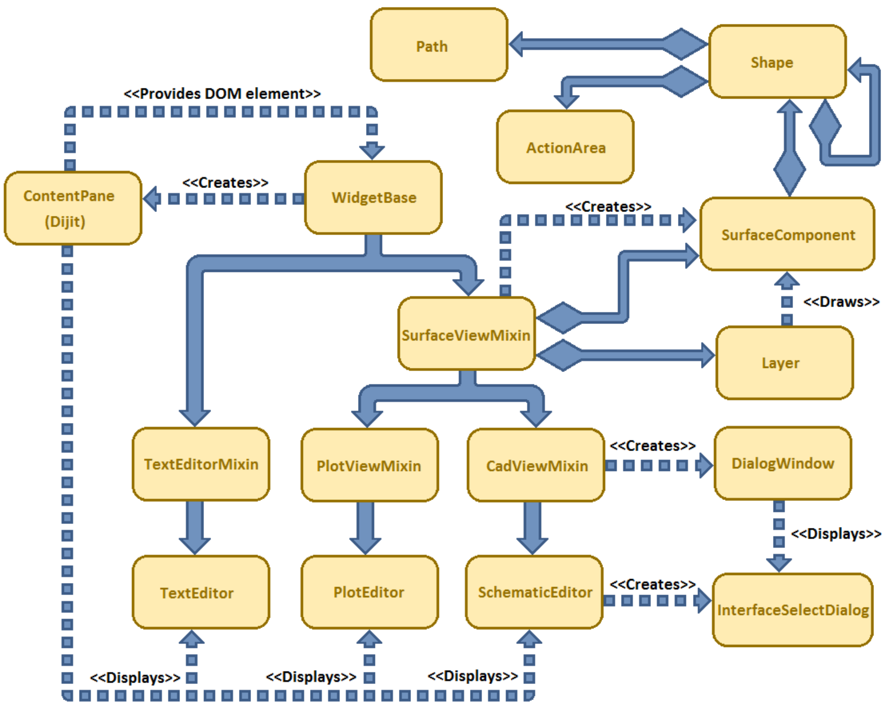


Figure 3.1: Class diagram for the current implementation of the frontend.

The `WidgetBase` mixin will provide the caller object with the functions `addEventListener` and `removeEventListener` to add event listeners, as well as providing the widget with event triggers for `close`, `show` and `resize` events, and `triggerEvent` for triggering these events. If the event for the event listener is not defined, it will be created while registering the event listener. For consistency reasons, the `addEventListener` and `removeEventListener` functions are identical to the JavaScript DOM Elements' functions in usage. Listing 3.1 shows how to add a resize event listener to the widget, such that it will be called every time the widget resizes.

```
// New resize event listener to be added to the widget
var newResizeEventListener = function(event) {
  // Outputs new size of the widget to the console
  console.log("new size: " + event.w + ", " + event.h);
}

// Attach newResizeEventListener to myWidget's "resize" event
myWidget.addEventListener("resize", newResizeEventListener, false);
```

Listing 3.1: Attaching a resize event listener to the widget

The `WidgetBase` mixin will also create and append to the widget the `content` object, which is the HTML DOM element which operates as the base element for the widget.

Additionaly, `WidgetBase` provides the widget with keypress event handlers, which will only fire if the widget is currently focused and uses editor modes.

In the current state of the application, this module is implemented as a mixin. In retrospect, this has been considered to be an unfortunate implementation; it would make more sense to let a widget creation function create an instance of this object, and then appending the desired additional functionality to this object through mixins.

It would also make sense to let all the event-related content be its own mixin, and then mix in this functionality into the `WidgetBase` upon creation. By doing this, other objects may mix in event handling functionality as well.

The keypress handler should get its own mixin, since not all widgets necessarily needs keyboard input.

SurfaceView

This mixin provides the widget with functionality for creating and displaying surface layers. It requires the widget to have a member DOM element called `content` for it to work, as all new surface layers will be appended to this element as HTML `canvas` elements. Listing 3.2.1 shows how to create a widget and mix in `SurfaceView` functionality.

```

define("myWidget", ["widget/surfaceviewmixin"],
function(SurfaceView) {
  // returns the constructor to the new widget
  return function(parentNode) {
    // parentNode needs to be set to a DOM element in order
    // to work
    this.parentNode = parentNode;

    // Mix in SurfaceView functionality
    SurfaceView.call(this);
  }
});
```

The `SurfaceView` mixin provides the widget with a `createLayer` function, which takes as parameters layer name and layer z-order, then creates and returns the new surface layer, i.e. a `canvas` element. It also adds a resize event to the widget so that the surface will be automatically resized too upon widget resize.

The widget is also provided with functionality for drawing graphical objects. To draw shapes to the screen, a low level drawing API, Canvas 2D, described in Chapter 2.2.2 is used. Describing shapes with code is rather inconvenient, so SurfaceView provides an internal data format and an interpreter of this format. The interpreter is called “Mustache”, and objects used to describe shapes are called “Mustache objects”.

New shapes can be added in one of two ways: Firstly, an object can be drawn directly onto a layer by invoking the layer’s `draw` method. This method takes the graphical object as a parameter. Secondly, one can store the object in the layer’s `objects` array, which will then be drawn each time the canvas is cleared. By invoking the layer’s `redrawLayer` method, the layer will first be cleared, and then all objects in the array will be drawn upon the canvas sequencially. Listing 3.2 shows how to create a graphical layer in the widget, and create a graphical object and draw it on the layer.

```

var myLayer = this.createLayer("myLayer", 1);

var Square = function(width) {
  this.fillStyle = "red";
  this.strokeStyle = "black";
  this.path = [
    {
      type: "line",
      points: [
        {x: -width/2, y: -width/2},
        {x: -width/2, y: width/2},
        {x: width/2, y: width/2},
        {x: width/2, y: -width/2}
      ]
    }
  ];
}

```

```
var mySquare = new Square(10);
myLayer.draw(mySquare);
```

Listing 3.2: Drawing a graphical object on a surface layer

CadView

This mixin provides the widget with functionality common for *CAD* editors, such as multi-layer surfaces, mouse interactivity and editor modes. It is intended as a base widget for the schematic and layout editors, among others. Listing 3.3 shows how to mix in CadView functionality to a widget.

```

define("myWidget", ["widget/cadviewmixin"],
function(CadView) {
  // returns the constructor to the new widget
  return function(parentNode) {
    // parentNode needs to be set to a DOM element in order
    // to work
    this.parentNode = parentNode;

    // Mix in SurfaceView functionality
    CadView.call(this);
  }
});
```

Listing 3.3: Creating a widget with *CadView* functionality

The CadView mixin creates a surface for mouse interactivity which has a high z-index to ensure that it is always above all other surfaces, and thus is the surface to catch the mouse events. It will capture both mouse movement and click, and fire events for these. For mouse movement, `CadView` will translate the new mouse coordinate on the screen to the current editor coordinate for the cursor. If the member variable `snapping` is defined, the widget will fire the `cursorMove` event; else, the event will fire every time the mouse moves.

The editor is also keep control over all the graphical components, and will update their positions every time they are moved, and it will also test if there are objects under the cursor every time it moves. It will also trigger relevant mouse events on objects, such as `onmouseover`, `onmouseout`, `onmousedown`, etc.

The `CadView` mixin also introduces the concept of *action areas*. These are areas on the surface with which the user can interact in an editor-specified manner. They can be attached to graphical components, to make these components interactive. This is useful in e.g. the schematic editor, where the user can connect two component pins together by clicking in the action areas of each pin while in the *place wire* mode. Listing 3.4 shows how to create an action area.

```
// A basic component without any graphics, only
// a position and an angle, as well as an attachment
// point for the action area
var myComponent = {
  pos: {x: 100, y: 100},
  angle: 0,
  somePoint = {
    pos: {x: 0, y: 100}
  }
};
myWidget.objects.push(myComponent);

// Updates the global position and angle of all component
// objects in the editor.
myWidget.updateGlobalPositions();

// Creates a new action area and attach it to the
// global position of myComponent.somePoint,
// as well as assigning some events to the action area.
var newActionArea = myWidget.createActionArea({
  type: "componentActionPoint",
  shape: Circle,
  pos: myComponent.somePoint.globalPos,
  objects: [],
  onMouseOver: function(event) {
    console.log("over area!");
  },
  onMouseOut: function(event) {
    console.log("out from area!");
  },
  onMouseClick: function(event) {
    console.log("clicked on object!");
  }
});
```

Listing 3.4: Creating an action area in a *CadView* widget

The `CadView` mixin introduces the concept of *editor modes*, which is a modifiable state machine, in which all the states have their own input handlers, entry and exit events, etc. The state editor mode model is fully customizable, and a mode can be entered by calling `setEditorMode` with the new mode as parameter. This will invoke the current state's exit event, and then, the new mode's enter event. An object for storing all editor states, called `editorStates`, is also provided by the mixin, although using it is not strictly necessary in order to use custom modes. Listing 3.5 shows how to create and use a custom editor mode.

```

var customEditorMode = function() {
  // Event to be triggered when the mode is entered
  this.onEnterMode = function() {
    console.log("entering custom editor mode.");
  }
  // Event to be triggered when mouse is clicked while the editor
  // is in this mode.
  this.onMouseDown = function(event) {
    console.log("mouse clicked while in custom editor mode.");
  }
  // Event to be triggered when leaving the mode
  this.onLeaveMode = function() {
    console.log("leaving custom editor mode.");
  }
  // Event to be triggered when 'a' is pressed on the keyboard
  // while the editor is in this mode.
  this.keyBindings["A".charCodeAt(0)] = function() {
    console.log("a pressed in custom editor mode.");
  }

  // Store the mode in the editorMode object to make it reusable
  myWidget.editorModes.customEditorMode = customEditorMode;

  // Set the editor mode to customEditorMode.
  myWidget.setEditorMode(customEditorMode);
}
```

Listing 3.5: Creating a custom editor mode and entering it

PlotView

This mixin provides the widget with multi-layered plotting functionality. It provides the widget with the `createPlot` function, which takes a parameter object as its only parameter. This function creates a new plot layer with `content` as parent DOM element, and returns this new layer. The `parameters.name` string decides the name of the plot.

The plot layer defines the `addPoint` function, which takes as parameter the new point to be added to the plot, as an object consisting of one `x` and one `y` parameter.

It should be noted that this widget is under construction and thus lacks most of its intended functionality.

TextWidget

This mixin turns the widget `content` element's surface area into a text editor, complete with syntax highlighting, line numbering, etc. The current implementation of this uses the MIT-like-licensed *CodeMirror* text editor.

3.3 Backend

Any web-application must be hosted by a web server. The server returns data requested by the client (the user's web browser). This server can also perform additional task, such as providing persistent storage (see section 2.2.3). Any web server can host files, but providing persistent storage and user authentication requires some additional programming. During development lighttpd was used to host files.

The *backend* refer to the parts of the application implemented on the server side, as opposed to the *frontend* which refer to the code and content running in the user's web browser.

The programming language *Go* was chosen to implement the backend. *Node.js* was also considered, but dismissed for lacking multi-threading and some difficulties compiling it. Go has a few similarities to javascript, such as first class functions and closures. This allows using some of the same design patterns in both the backend and the frontend. Go has some very useful packages: The `net/http` package allows for easy creation of web servers. It also contain a file hosting function. The `encoding/json` is used to generate JSON containers. JSON is easily interpreted in javascript, thus useful for backend-frontend data exchange.

The backend relies on *MongoDB*, a document database for storing user data. The package *mgo* provides an interface to the database.

Exchange of information between the frontend and backend was an important consideration when choosing the language. JSON is used to wrap data. Data from the backend to the client is sent as HTTP response messages with *Content-*

Type set to *application/json*. The JSON encoded data is put into the message body. Data from the frontend are sent to the backend as HTTP request messages, with the data encoded in the request body. The path field in the HTTP header is used by the backend to determine what to do with the provided data. The backend provides the options: login, logout and register for user accounts; store, list and get for documents. In order to user the document function store, the user must first have successfully performed the login function.

storage.js is used to communicate with the backend. In addition to the arguments needed for the functions in the backend, it takes a callback function. It encodes the arguments passed to its functions, and encode them as a field values in a HTTP request form. When it receives a response, it converts the returned JSON into a JavaScript object, and calls the callback function with the decoded object.

3.3.1 Login

The function `login` in *storage.js* creates an HTTP request with username and password encoded as form data. In the backend, the user with a matching username is fetched from the database. The sha256 sum of the concatenation of the salt from the database and the password from the HTTP request is calculated. If no user is found, or the calculated sum does not match the one in the user database, the fellowing JSON object is returned:

```
{ "error": "Wrong username/password" }
```

If the sums match, the login is successful, and an unique session ID (USID) is generated. The USID is stored in a session database along with the username. To the user the fellowing JSON object is returned:

```
{ "usid": <base64 encoded USID> }
```

3.3.2 Register

Parameters passed as form data in the http request are: Username, email and password. If the username provided is already taken, the fellowing JSON object is returned:

```
{ "error": "Username taken" }
```

And if no username was given, the following:

```
{ "error": "No username given" }
```

If the username is not in use, a salt is generated. Then the sha256 hashing algorithm is applied to the concatenation of the salt and password. This hash is stored together with the username, email and salt of the user in the database. A login operation is then performed.

3.3.3 Logout

A USID and username is provided from the client. If a record with both matching username and USID is found, this record is removed. No value is returned.

3.3.4 Store

The fellowing arguments are stored when the operation is successful:

Name Used to refer to the item.

Implements Some other item that can be replaced by this item.

Published Whether the item is availibe to use by all users or only the owner and the maintainers.

Owner Person with privileges to update the item and its metadata.

Maintainers Users listed as maintainers are allowed to update the item's data.

Readers A list of users that is allowed to use the item if it is unpublished.

Data The item's data.

In addition the USID is used to obtain the username and verify if the user har the required privilegies. If not an error is returned:

```
{ "error": "Not authorized" }
```

3.3.5 Load

Arguments: Name, USID. This function searches the item database for the provided name. If no objects are found, an error is returned:

```
{ "error": "Not found" }
```

An item is marked as published will be returned along with its metadata. If the item is not marked as published, the username is located from the provided USID in the session database, and checked against the item's owner, maintainer list and reader list. Provided the user is listed in one of these groups, the item is returned along with its metadata. If not, an error is returned:

```
{ "error": "Not authorized" }
```

3.3.6 List

This function lists all objects the user can use. If no USID is provided, only items with the published flag enabled are listed. The result is returned as a JSON object of the fellowing form:

```
{ "names": ["item1","item2","item3",...] }
```

If no items are found, an empty array is retured.

3.4 Provided plugins

This section explains some of the plugins that has been developed or are intened to be implemented in the near future.

3.4.1 Schematic editor

This plugin is a graphical widget that will manipulate the document's net- and component lists based on interaction from the user. It derives its functionality from the `CadView` mixin, providing it with useful functionality, e.g. surface layers and the editor modes state machine.

The plugin adds editor states for placement of components, wires and interfaces. All component placement modes (e.g. `placeResistorMode`) mix in the `placeComponentMode` object, which provides it with an entry event, exit event, component placement upon left click, component rotation upon right click, and a component config dialog which pops up when the *tab* key is pressed. The `placeComponentMode` function takes a component as an argument, which will be the component to be placed by the mode.

The schematic editor plugin defines the `createComponent` function, which creates a surface component (see section 3.2.1), mixes in *CAD component* functionality (selection, movability, etc), and appends the new component to the `objects` array, which is inherited from `SurfaceView`. Listing 3.6 shows how to create a basic schematic component.

```

define("myComponent", [], function() {
  return function(parameters) {

    // a pointer to the instance of the object currently called
    var myComponent = this;

    if(parameters) {
      parameters.pos = {x: 0, y: 0};
      parameters.angle = 0;
      if(parameters.pos) myComponent.pos = parameters.pos;
      if(parameters.angle) myComponent.angle = parameters.angle;
      if(parameters.view) myComponent.view = parameters.pos;
    }
    this.type = "myComponent";
    this.boundingPath = [
      {
        type: "line",
        points: [
          {x: -50, y: -50},
          {x: -50, y: 50},
          {x: 50, y: 50},
          {x: 50, y: -50}
        ]
      }
    ];
    var view = this.view;
    this.pins = [
      view.createPin({angle: Math.PI, pos: {x:0, y:-50}, length:
          25, parent: myComponent});
      view.createPin({angle: 0, pos: {x:0, y:50}, length: 25,
          parent: myComponent});
    ];
    this.shape = {
      this.pins,
      new function() {
        this.fillStype = "red";
        this.strokeStyle = "black";
        this.closed = true;
        this.path = [
          {
            type: "line",
            points: [
              {x: -50, y: -50},
              {x: -50, y: 50},
              {x: 50, y: 50},
              {x: 50, y: -50}
            ]
          }
        ]
      }
    }
  }
});
```

Listing 3.6: Creating a new circuit component.

The plugin define modes for placing wires and interface wires, called `placeWireMode` and `placeInterfaceMode`. These modes call `createComponent` with `Wire` and `Interface`, respectively, as parameter. Upon creation, the `Wire` component will append itself to the schematic editor's `wireList` array (this has not yet been implemented for `Interface`). The `wireList` array is currently the only representation of the net list, although in the future, a net list controller will be implemented, and so this part of the plugin will undergo some changes.

These modes have a lot in common, and for reusability reasons, the functionality of these will be moved into a mixin from which both modes will inherit functionality. This mixin will also be used in the layout editor wire routing mode.

The schematic editor also assigns `idleMode` key bindings for entering component placement modes, e.g. entering `placeResistorMode` by pressing 'r'. In the future, the user will be able to configure key bindings and store these in his user profile.

Figure 3.2 shows how the class diagram for and around the schematic editor will look like after some code cleanup.

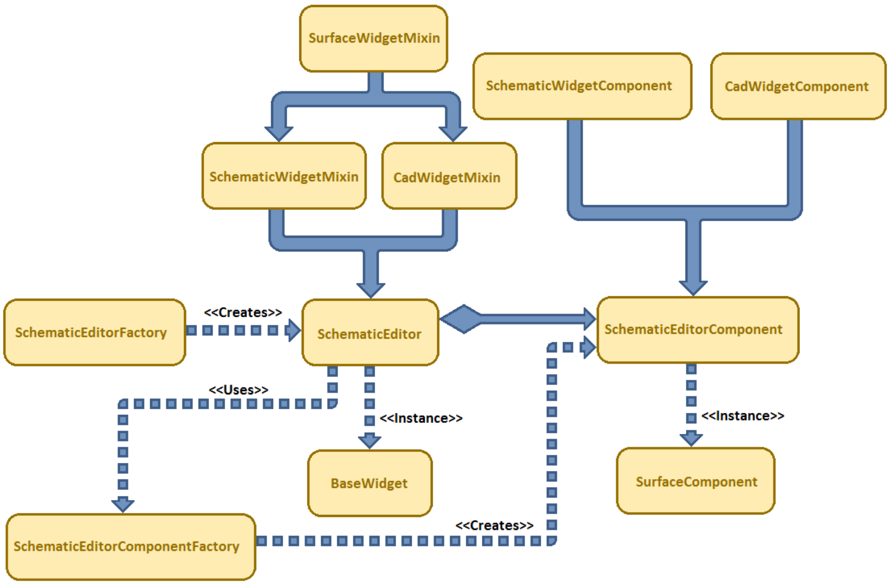


Figure 3.2: Future class diagram of the schematic editor, with factory methods for creating the schematic editor and the schematic editor components

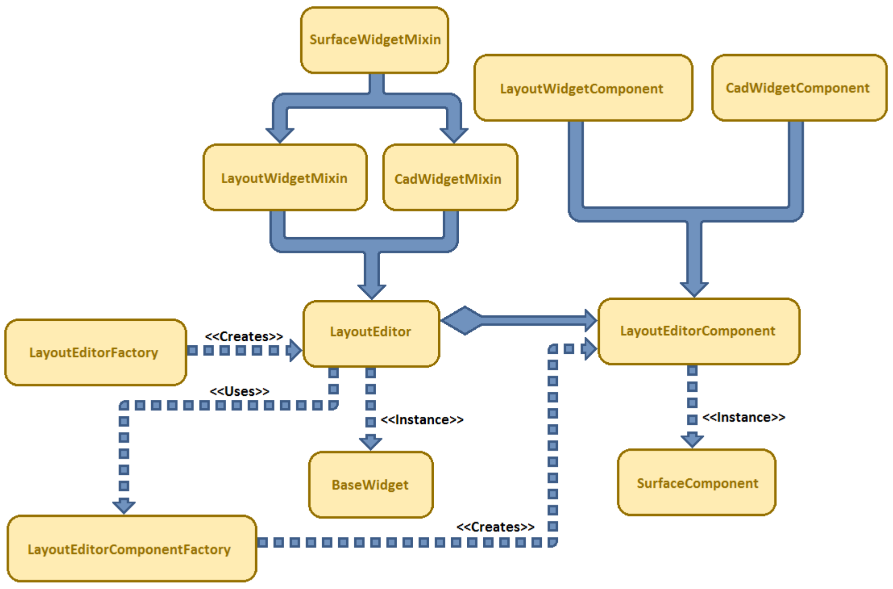


Figure 3.3: Future class diagram of the layout editor, with factory methods for creating the layout editor and the layout editor components

3.4.2 Layout editor

This plugin is used for generating the net- and component lists into printed circuit board layout. Due to time limitations, this plugin has not been written yet, because the schematic editor needs to be able to manipulate the net list before this editor will actually serve a purpose.

The planned class diagram for and around the layout editor is shown in Figure 3.3.

3.4.3 Simulator

The simulator implementation is incomplete, and the simulator plugin currently only contains the numerical simulator, and a menu to run a hard-coded example system.

Generation of the incidence matrix has been implemented, but is currently part of the schematic editor.

Numerical Solver

The numerical solver is defined by the module `simulator/solver`, and solves systems on the fellowing form:

$$\dot{\mathbf{y}} = f(\mathbf{y}, t) = A\mathbf{y} + \mathbf{b} \tag{3.1}$$

The module defines a function: `function(h, A, b, y0)`, which returns a new function for stepping forward in time. The arguments are: `h`, the step size for the simulation; `A`, same as $\mathbf{A}$ from the system in Equation 3.1; `b`, same as $\mathbf{b}$ from the system in Equation 3.1; and `y0`, ($\mathbf{y_0} = \mathbf{y}(t_0)$), the initial state of the system.

The function calculates the parameters given in Equation 2.9, which become part of the returned function's closure (discussed in 2.3.4). The returned function calculates and returns $\mathbf{y_{n+1}}$ as defined in Equation 2.8.

3.4.4 Plotting tool

This plugin is for viewing simulator scope signals, and it mixes in the `PlotView` mixin. Due to this tool currently being in development, it is currently only drawing a red blob of noise.

3.4.5 Text editor

This plugin is for creating and displaying text, and can be used for writing plugins, creating component descriptions, and much more. It mixes in `TextWidget`, and appends additional functionality for saving and loading text, change language for syntax highlighting, etc. Due to time limitations, this extra functionality is not implemented yet.

3.4.6 Planned plugins

Not all the plugins needed for the application could be developed during the timeframe given, and so some plugins

3.4.7 Interface editor

This plugin is for configuring connection interfaces, as described in section FIXME. Due to time limitations, this plugin has not been written yet, although it plays a vital role for getting the schematic editor to work correctly.

3.4.8 ComponentEditor

This plugin is for creating schematic and layout components. Due to time limitations, it has not been implemented yet. Interface-pins

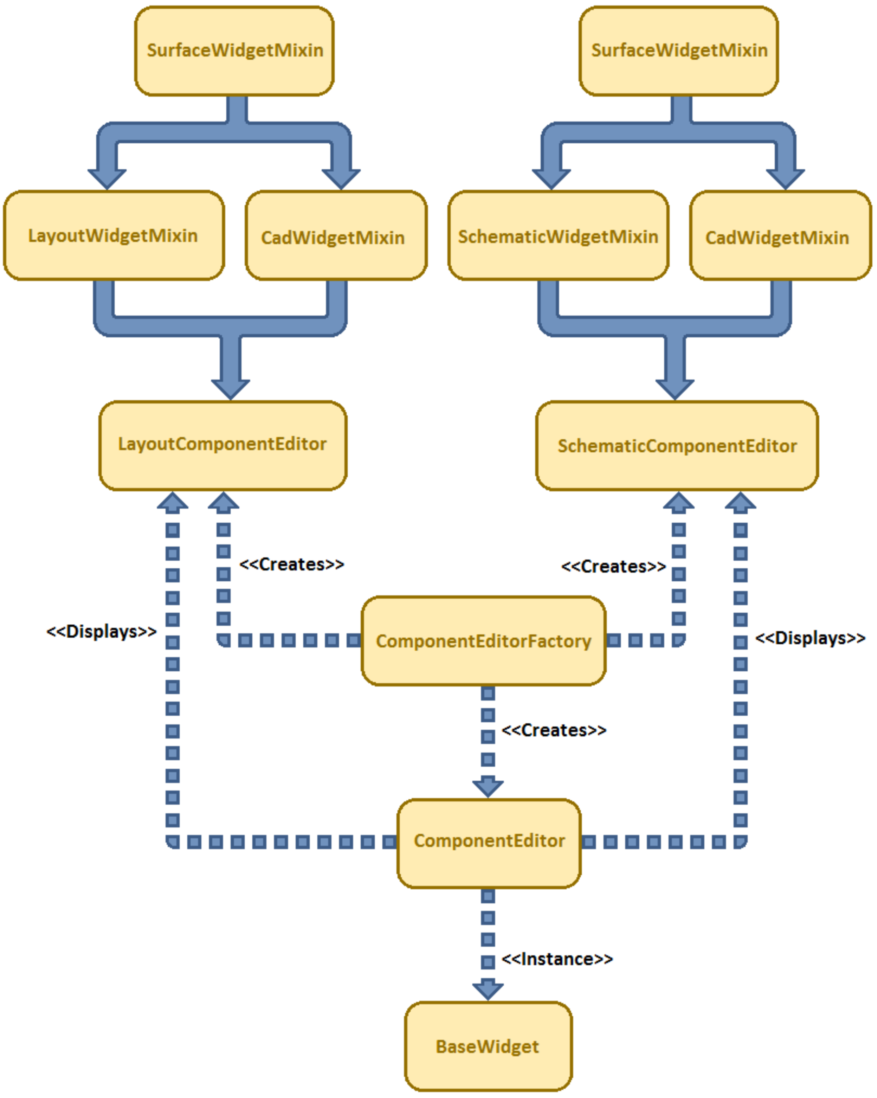


Figure 3.4: Future class diagram of the component editor, with factory method for creating the component editor, the schematic component editor and layout editor

4 Discussion, Conclusion and Further Work

4.1 Discussion

4.1.1 The Simulator

One of the first problems attacked during our work, was to find a way to simulate circuits. The first attempt used the explicit Euler method, and a plotting tool was implemented to debug the implementation. The explicit Euler method proved to be very inaccurate, and the Runge-Kutta (explicit Runge-Kutta, order 4) method was implemented.

The Runge-Kutta method seemed to work well, until we tried simulation more complex circuits. Actually, introducing a single inductor to a RC circuit caused the simulation to fail. Significant time was spent trying to understand why this did not work, and after a discussion with FIXME, an attempt was made using the implicit Euler method.

While implicit euler worked, it was very inaccurate, and required very small time steps. This in turn made simulation unacceptably slow, and an higher order method was implemented. Lobatto IIIC provided a much better simulation as demonstrated in Figure 2.5 compared to Figure 2.3.

When reimplementing explicit Runge-Kutta 4 to provide plots for the report, explicit Runge-Kutta appeared to be very accurate for the RLC circuit, contrary to earlier results. Since the simulator has many features yet to be implemented, having a implicit Runge-Kutta method available may prove useful because of it's stability characteristics.

The attached files contains code examples allowing the reader to test and plot himself the differences between the different numerical methods. A description on how to do this is provided in Appendix B.1.

The Simulator has one significant problem, which is that it relies on *Sylvester.js*. Sylvester.js was chosen because it appeared to do what we needed, and it has an efficient matrix inversion algorithm. Unfortunately Sylvester is unable to handle 1-by-1 matrices properly. Specifically, it is unable to invert a 1-by-1 matrix, which becomes rather inconvenient when extra code is needed for testing the size of the matrix, and handling the special cases as scalar values. It is also unable to us 0-by-anything matrices, which would have been neat. Using Sylvester.js may not have been the wisest choice, as it must be replaced or improved, but implementing a matrix library would have cost us significant time.

In the current implementation analog simulation was prioritized, and we did not have enough time to implement digital simulation. The digital simulation could be performed with discrete event simulation. Behaviour could be described e.g. using function blocks. A change in the input of a block could schedule an output event, as described by the block. To interface analog and digital components analog to digital converters could be used. These could be provided with a sampling interval, and generate output events each sample, while Digital to analog converters would modify the analog process' matrices. We haven't had time to explore these ideas in more detail.

The choice to focus on analog simulation was influenced by the Falstad Circuit Simulator, and the authors' interest in audio and radio electronics. Choosing to focus on analog simulation may have provided earlier results than focusing on discrete event simulation would have. An discrete event simulator requires an event scheduler, a time advancement algorithm, and some callback system. A very simple analog simulator could use Euler's method, $y_{n+1} = y_n + h(Ay_n + b)$, with hard coded values for A and b, running in a loop printing y_n. One of the disadvantages of focusing on analog simulation, is that it is not well suited for simulating more abstract behaviour, such as software.

4.1.2 JavaScript

As reflected by section 2.3, a significant part of our time was used to familiarize our self with JavaScript. Neither of the authors had any previous experience in programming with JavaScript, and as a consequence, much of the code written in the early stages of development have been refactored or rewritten. By choosing to build a web-application, we had already accepted to code in JavaScript, and we consider the time spent on “getting on terms” with JavaScript, to have been necessary and useful.

4.1.3 Swapping of tasks

Early in development, Thomas was working on the simulator, and Tore Egil on the GUI. In order to provide better feedback from the simulator, the first component of the GUI to be implemented was the plotting tool. *Mustache* was implemented as part of the plotting tool. With a simulator able to plot the voltage curve of a RC-circuit in a web-browser, the speed of development unfortunately came to a halt: Tore Egil found himself spending a lot of time attempting to make event handling and *Mustache* work together. Thomas was stuck in development of the simulator, having trouble finding a consistent way of including inductors. Having spent too much time in this barely productive phase, the tasks were switched: Tore Egil took over development of the simulator, while Thomas took over the GUI development. This switch proved effective: Thomas scratched Tore Egil’s low level approach, and used *Dojo* to provide a framework for the GUI. *Mustache* was however kept, and is still used to draw shapes to the canvas element. Tore Egil found a paper describing MNA[4] more properly than the sources Thomas had used earlier, which had been aimed at resistive network analysis.

4.1.4 The Backend

The backend, as described in section 3.3 is implemented in *go* and provides hosting of static files, user control, and persistent storage for users. It is closely matched with the module *storage.js*, which communicate with the backend, and presents an clean api for the rest of the application to use. *MongoDB* was chosen rather arbitrarily, and ease of use was the main concern. The early idea was to just push everything into a hashmap, and store and retrieve values from this. While the idea was fairly easy to implement, providing functionality for searching and filtering posts became too cumbersome. Thus the first solution was ditched in favor of a simpler one using MongoDB. The implementation of the backend is incomplete, as with many other parts of the application. User registration, login and logout works, and users' sessions can be remembered. The document storage is however incomplete, mainly because we have not had time to implement the testing facilities. The testing GUI is able to perform user registration, login and logout, and the connection is made over HTTPS. Had we chosen to use *MongoDB* at an earlier stage, we might have had a functional version of the backend.

4.2 Conclusion

The objective of this thesis was to develop a GPL-licensed web application for rapid prototyping of electronic circuits. Listed as important qualities for the application was ease of use, reusability and schematics readability.

4.2.1 The tasks performed

Survey existing electronic design automation software (EDA).

Some existing EDA programs are discussed in section 1.2. Inspiration has been drawn for some of them, in particular Falstad Circuit Simulator and CadSoft EAGLE. Lessons taken are the poor simulator usability, and schematic verbosity.

Describe essential and nice features for an EDA application

Essential and nice features of an EDA are described in chapter 1.3. One of the most significant features for this application it is implemented as a web-application. Other significant features are to delay the choice of components, and interface based routing.

Study necessary background information to implement such an application

Significant effort has been put to understanding JavaScript, and learning to utilize it. This effort has mostly been practical, experimenting with the language. Modified Nodal Analysis has been important in order to simulate analog circuits.

Decide which tools and languages the software will be built upon, and learn how to best utilize these within the project.

We have chosen JavaScript to implement the frontend. *Dojo* provides module loading, and a graphical user interface framework. *Sylvester.js* provides addition, subtraction, multiplication and inversion of matrices. The backend is implemented in *Google Go*, with the package *mgo* providing an interface to *MongoDB*.

Implement the essential features of the application, and a plugin-platform which other features can built on top of.

Several parts of the application has been implemented as plug-in modules. The schematic editor, the numerical solver, the plotting tool and the text editor.

Evaluate the implementation.

Several of the application's modules are only partially implemented; most of them do however provide some functionality. The application can therefore be considered a tech demo.

4.3 Further Work

4.3.1 Refactoration of code

Parts of the application has not been fully integrated, while much of the required implementation is in place.

4.3.2 Integrating persistant storage

While *storage.js* is able to perform user registrations, logins and logouts, as well as storing and retrieving user data; it has not been integrated with other parts of the frontend. The primary reason is that the rest of the application, has not yet reached a maturity level where this functionality is needed.

4.3.3 Improving the simulator

Implement a new matrix library

Sylvester.js which is used by the Simulator is a third-party library. It lacks proper support for 1-by-1 matrices, and has no support for smaller matrices, which is unacceptable. Implementing a new matrix library, or patching Sylvester.js should therefore be one of the first things to look at.

Integrating the simulator with the schematic editor

A working numerical solver has been implemented, and *Schematic* is able to generate the incidence matrix. What needs to be done after a better matrix library has been implemented, is to implement an algorithm to transform these values into a usable form for the numerical solver, and display the results in an informative way.

4.3.4 Implementing discrete event simulation

Having the ability to simulate digital components is a rather important feature.

and using components using interfaces was one of the of the features that would make this feasible.

The current parts of the simulator implementation does not include any code to perform discrete event simulation. Discrete event simulation could provide simulation capabilities for the functional design of electronics. Analog and discrete event simulation could be combined using adc and dac blocks.

A Analog Simulation

A.1 Implicit Methods

Implicit Runge-Kutta methods can be used to estimate the solutions of systems on the form:

$$\dot{\mathbf{y}} = f(\mathbf{y}, t) = A\mathbf{y} + \mathbf{b} \tag{A.1}$$

A general 2nd order Implicit Runge-Kutta method has the form:

$$\begin{aligned}
\mathbf{x_{n+1}} &= \mathbf{x_n} + h[b_1\mathbf{k_1} + b_2\mathbf{k_2}] \\
\mathbf{k_1} &= f(\mathbf{x_n} + h[a_{11}\mathbf{k_1} + a_{12}\mathbf{k_2}], t_n + c_1 h) \\
\mathbf{k_2} &= f(\mathbf{x_n} + h[a_{21}\mathbf{k_1} + a_{22}\mathbf{k_2}], t_n + c_2 h)
\end{aligned} \tag{A.2}$$

Applying (A.2) to (A.1) yields:

$$\begin{aligned}
\mathbf{y_{n+1}} &= \mathbf{y_n} + h(b_1\mathbf{k_1} + b_2\mathbf{k_2}) \\
\mathbf{k_1} &= A(\mathbf{y_n} + h[a_{11}\mathbf{k_1} + a_{12}\mathbf{k_2}]) + \mathbf{b} \\
\mathbf{k_2} &= A(\mathbf{y_n} + h[a_{21}\mathbf{k_1} + a_{22}\mathbf{k_2}]) + \mathbf{b}
\end{aligned} \tag{A.3}$$

Solving (A.3) for y_{n+1}:

$$\begin{aligned}
\mathbf{k_1} &= A\mathbf{y_n} + ha_{11}A\mathbf{k_1} + ha_{12}A\mathbf{k_2} + \mathbf{b} \\
\mathbf{k_2} &= A\mathbf{y_n} + ha_{21}A\mathbf{k_1} + ha_{22}A\mathbf{k_2} + \mathbf{b}
\end{aligned} \tag{A.4}$$

$$\begin{aligned}
\mathbf{k_1} - ha_{11}A\mathbf{k_1} &= A\mathbf{y_n} + ha_{12}A\mathbf{k_2} + \mathbf{b} \\
\mathbf{k_2} - ha_{22}A\mathbf{k_2} &= A\mathbf{y_n} + ha_{21}A\mathbf{k_1} + \mathbf{b}
\end{aligned} \tag{A.5}$$

$$\begin{aligned}(I - ha_{11}A)\mathbf{k_1} &= A\mathbf{y_n} + ha_{12}A\mathbf{k_2} + \mathbf{b}\\(I - ha_{22}A)\mathbf{k_2} &= A\mathbf{y_n} + ha_{21}A\mathbf{k_1} + \mathbf{b}\end{aligned} \tag{A.6}$$

$$\begin{aligned}\mathbf{k_1} &= E_1A\mathbf{y_n} + E_1ha_{12}A\mathbf{k_2} + E_1\mathbf{b}\\\mathbf{k_2} &= E_2A\mathbf{y_n} + E_2ha_{21}A\mathbf{k_1} + E_2\mathbf{b}\end{aligned} \tag{A.7}$$

where:

$$\begin{aligned}E_1 &= (I - ha_{11}A)^{-1}\\E_2 &= (I - ha_{22}A)^{-1}\end{aligned} \tag{A.8}$$

$$\begin{aligned}\mathbf{k_1} &= E_1A\mathbf{y_n} + ha_{12}E_1A(E_2\mathbf{y_n} + E_2ha_{21}A\mathbf{k_1} + E_2\mathbf{b}) + E_1\mathbf{b}\\\mathbf{k_2} &= E_2A\mathbf{y_n} + ha_{21}E_2A(E_1\mathbf{y_n} + E_1ha_{12}A\mathbf{k_2} + E_1\mathbf{b}) + E_2\mathbf{b}\end{aligned} \tag{A.9}$$

$$\begin{aligned}(I - h^2a_{12}a_{21}E_1AE_2A)\mathbf{k_1} &= E_1A(I + ha_{12}E_2)\mathbf{y_n} + E_1(ha_{12}AE_2 + I)\mathbf{b}\\(I - h^2a_{21}a_{12}E_2AE_1A)\mathbf{k_2} &= E_2A(I + ha_{21}E_1)\mathbf{y_n} + E_2(ha_{21}AE_1 + I)\mathbf{b}\end{aligned} \tag{A.10}$$

$$\begin{aligned}\mathbf{y_{n+1}} &= \mathbf{y_n} + h(b_1\mathbf{k_1} + b_2\mathbf{k_2})\\\mathbf{k_1} &= M_1\mathbf{y_n} + N_1\mathbf{b}\\\mathbf{k_2} &= M_2\mathbf{y_n} + N_2\mathbf{b}\end{aligned} \tag{A.11}$$

where:

$$\begin{aligned}M_1 &= (I - h^2a_{12}a_{21}E_1AE_2A)^{-1}E_1A(I + ha_{12}E_2)\\M_2 &= (I - h^2a_{21}a_{12}E_2AE_1A)^{-1}E_2A(I + ha_{21}E_1)\\N_1 &= (I - h^2a_{12}a_{21}E_1AE_2A)^{-1}E_1(ha_{12}AE_2 + I)\\N_2 &= (I - h^2a_{21}a_{12}E_2AE_1A)^{-1}E_2(ha_{21}AE_1 + I)\\E_1 &= (I - ha_{11}A)^{-1}\\E_2 &= (I - ha_{22}A)^{-1}\end{aligned} \tag{A.12}$$

A.2 Example: RC circuit

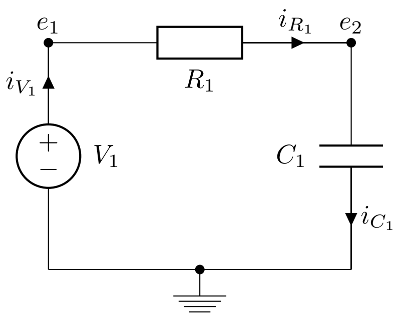


Figure A.1: A simple RC circuit

Using MNA, we get:

$$\begin{gathered}\mathbf{A_R} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{A_C} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{A_V} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ \mathbf{G} = \begin{bmatrix} \frac{1}{R_1} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} C_1 \end{bmatrix}, \mathbf{E} = \begin{bmatrix} V_1 \end{bmatrix}\end{gathered} \tag{A.13}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & C_1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} e_1 \\ e_2 \\ i_{V_1} \end{bmatrix} + \begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} & 1 \\ -\frac{1}{R_1} & \frac{1}{R_1} & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ i_{V_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ V_1 \end{bmatrix} \tag{A.14}$$

Splitting Equation A.14 into implicit and explicit parts yields:

$$\begin{bmatrix} C_1 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{R_1} \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} -\frac{1}{R_1} & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ i_{V_1} \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix} \tag{A.15}$$

$$\begin{bmatrix} \frac{1}{R_1} & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ i_{V_1} \end{bmatrix} + \begin{bmatrix} -\frac{1}{R_1} \\ 0 \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} = \begin{bmatrix} 0 \\ V_1 \end{bmatrix} \tag{A.16}$$

Solving the algebraic part for $\left[\begin{smallmatrix} e_1 \\ i_{V_1} \end{smallmatrix}\right[$:

$$\begin{bmatrix} \frac{1}{R_1} & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ i_{V_1} \end{bmatrix} = \begin{bmatrix} \frac{1}{R_1} \\ 0 \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} 0 \\ V_1 \end{bmatrix} \tag{A.17}$$

$$\begin{bmatrix} e_1 \\ i_{V_1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -\frac{1}{R_1} \end{bmatrix} \begin{bmatrix} \frac{1}{R_1} \\ 0 \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1 & -\frac{1}{R_1} \end{bmatrix} \begin{bmatrix} 0 \\ V_1 \end{bmatrix} \tag{A.18}$$

$$\begin{bmatrix} e_1 \\ i_{V_1} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{R_1} \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} V_1 \\ -\frac{V_1}{R_1} \end{bmatrix} \tag{A.19}$$

Inserting Equation A.19 into Equation A.15

$$\begin{bmatrix} C_1 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{R_1} \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} -\frac{1}{R_1} & 0 \end{bmatrix} \left(\begin{bmatrix} 0 \\ \frac{1}{R_1} \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} V_1 \\ -\frac{V_1}{R_1} \end{bmatrix} \right) = \begin{bmatrix} 0 \end{bmatrix} \tag{A.20}$$

$$\begin{bmatrix} C_1 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{R_1} \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} -\frac{1}{R_1} V_1 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix} \tag{A.21}$$

$$\frac{d}{dt} \begin{bmatrix} e_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{R_1 C_1} \end{bmatrix} \begin{bmatrix} e_2 \end{bmatrix} + \begin{bmatrix} \frac{V_1}{R_1 C_1} \end{bmatrix} \tag{A.22}$$

A.2.1 Exact Solution

Stripping the matrix notation as it is unneeded:

$$\frac{d}{dt} e_2 = -\frac{1}{R_1 C_1} e_2 + \frac{V_1}{R_1 C_1} \tag{A.23}$$

$$\frac{d}{dt} e_2 = (e_2 - V_1) \frac{-1}{R_1 C_1} \tag{A.24}$$

$$\frac{de_2}{e_2 - V_1} = \frac{-1}{R_1 C_1} dt \tag{A.25}$$

$$\int \frac{de_2}{e_2 - V_1} = \int \frac{-1}{R_1 C_1} dt \tag{A.26}$$

Using integration rule from [6, 4, p. 133]:

$$\int \frac{dx}{ax+b} = \frac{1}{a} \ln C(ax+b)$$

$$\ln K_1(e_2 - V_1) = \frac{-t}{R_1C_1} + K_2 \tag{A.27}$$

$$K_1(e_2 - V_1) = K_2 e^{\frac{-t}{R_1C_1}} \tag{A.28}$$

$$e_2 = V_1 + K e^{\frac{-t}{R_1C_1}} \tag{A.29}$$

Using the initial condition: $e_2(t=0) = 0$

$$e_2(t=0) = V_1 + K e^{\frac{-0}{R_1C_1}} \tag{A.30}$$

$$0 = V_1 + K \tag{A.31}$$

$$K = -V_1 \tag{A.32}$$

thus:

$$e_2(t) = V_1(1 - e^{\frac{-t}{R_1C_1}})$$

A.2.2 Plots of numerical solutions

Figure A.2: Simulation of RLC circuit using Euler's method, with step size = 0.001s

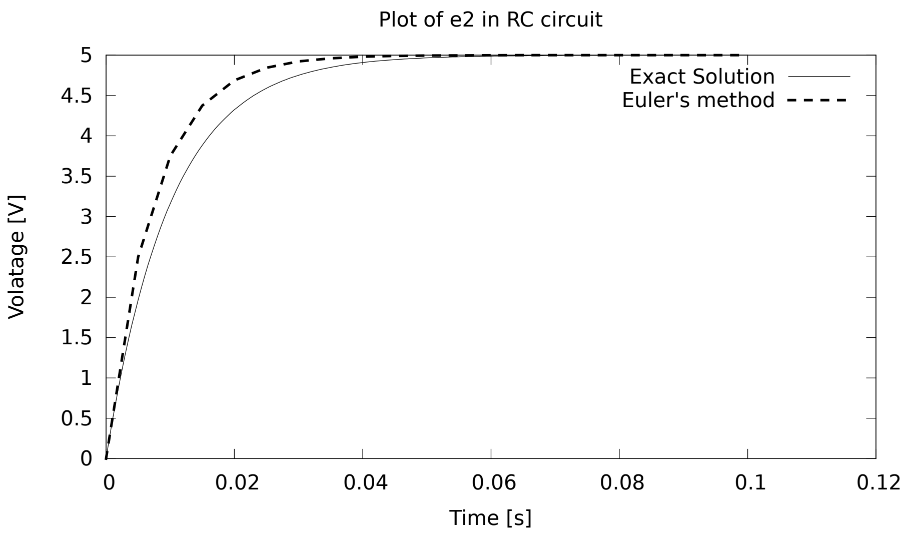


Figure A.3: Simulation of RLC circuit using the implicit Euler method, with step size = 0.001s

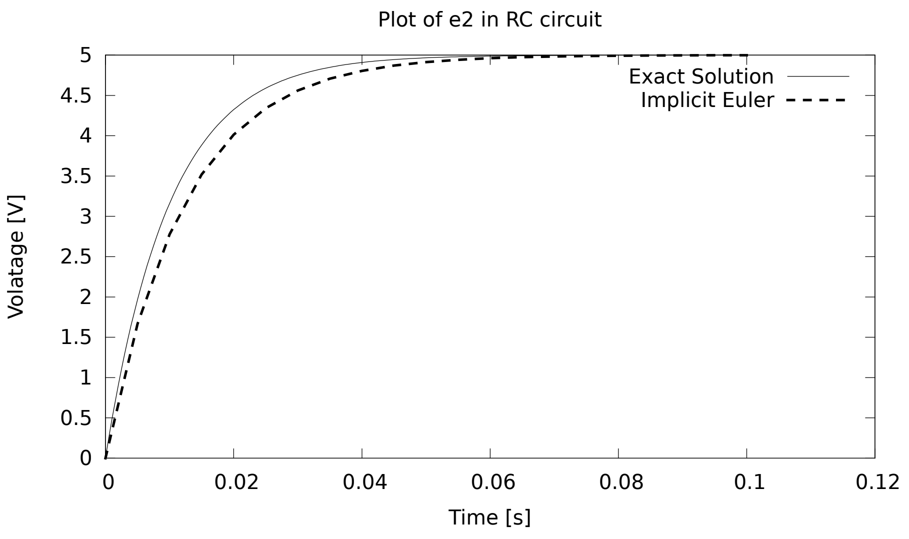


Figure A.4: Simulation of RLC circuit using the Runge-Kutta method, with step size = 0.001s

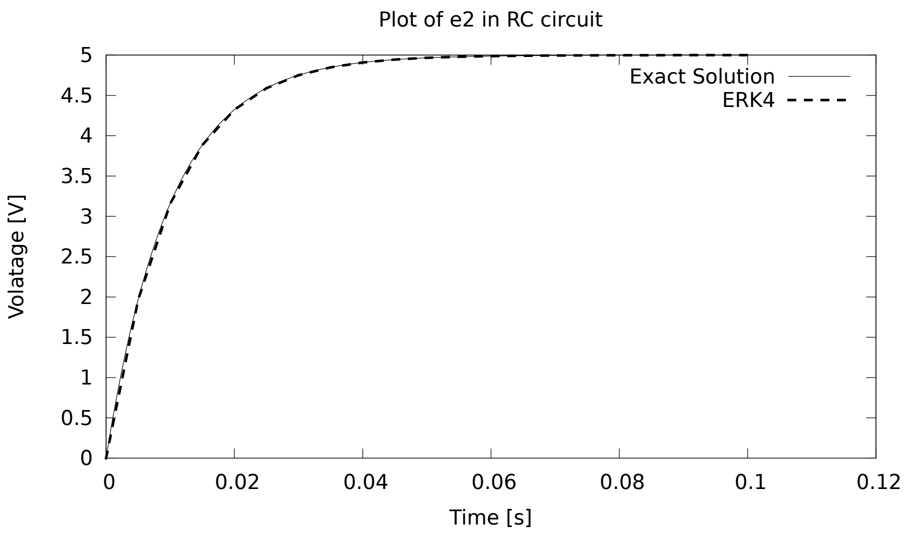
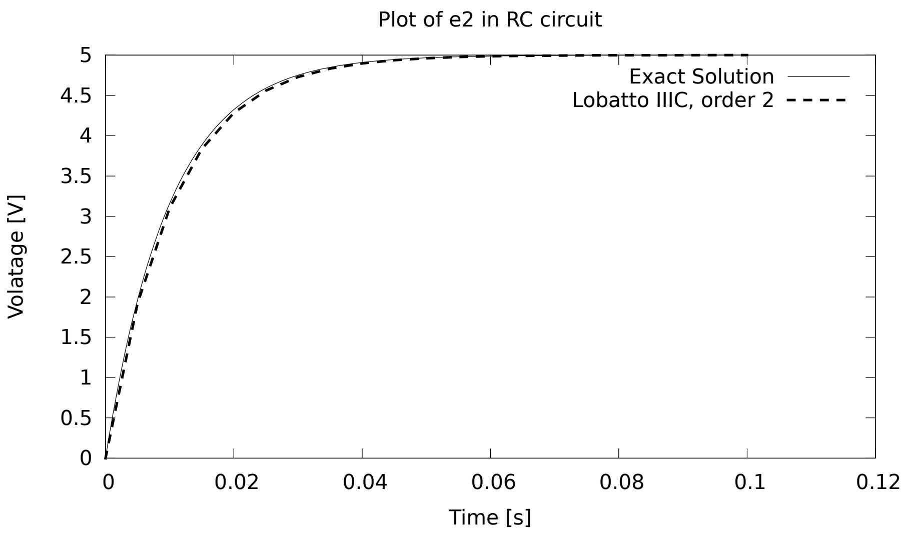


Figure A.5: Simulation of RLC circuit using Lobatto IIIC, with step size = 0.001s



A.3 Example: RLC circuit

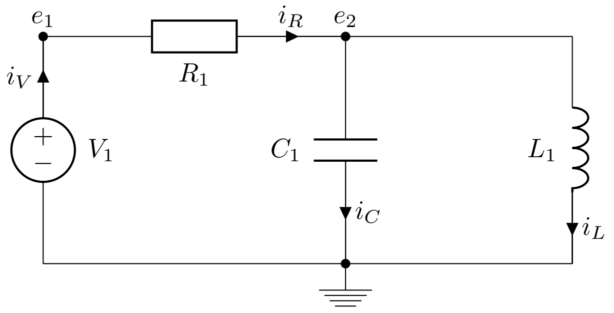


Figure A.6: Example circuit: $R_1 = 1000\Omega$,$L_1 = 10\text{mH}$,$C_1 = 1\text{mF}$ and $V_1 = 5\text{V}$

.

Using MNA, we get:

$$\mathbf{A_R} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{A_C} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{A_L} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{A_V} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$
$$\mathbf{G} = \begin{bmatrix} \frac{1}{R_1} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} C_1 \end{bmatrix}, \mathbf{L} = \begin{bmatrix} L_1 \end{bmatrix}, \mathbf{E} = \begin{bmatrix} V_1 \end{bmatrix} \tag{A.33}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & C_1 & 0 & 0 \\ 0 & 0 & L_1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} e_1 \\ e_2 \\ i_L \\ i_V \end{bmatrix} + \begin{bmatrix} \frac{1}{R_1} & \frac{-1}{R_1} & 0 & -1 \\ \frac{-1}{R_1} & \frac{1}{R_1} & 1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ i_L \\ i_V \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -V_1 \end{bmatrix} \tag{A.34}$$

$$\begin{bmatrix} C_1 & 0 \\ 0 & L_1 \end{bmatrix} \frac{d}{dt} \begin{bmatrix} e_2 \\ i_L \end{bmatrix} + \begin{bmatrix} \frac{-1}{R_1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ i_V \end{bmatrix} + \begin{bmatrix} \frac{1}{R_1} & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} e_2 \\ i_L \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} \frac{1}{R_1} & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ i_V \end{bmatrix} + \begin{bmatrix} \frac{-1}{R_1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e_2 \\ i_L \end{bmatrix} = \begin{bmatrix} 0 \\ V_1 \end{bmatrix} \tag{A.35}$$

$$\begin{aligned}\frac{d}{dt}\begin{bmatrix}e_2\\i_L\end{bmatrix} &= -\begin{bmatrix}\frac{1}{C_1} & 0\\0 & \frac{1}{L_1}\end{bmatrix}\begin{bmatrix}\frac{-1}{R_1} & 0\\0 & 0\end{bmatrix}\begin{bmatrix}e_1\\i_V\end{bmatrix} - \begin{bmatrix}\frac{1}{C_1} & 0\\0 & \frac{1}{L_1}\end{bmatrix}\begin{bmatrix}\frac{1}{R_1} & 1\\-1 & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix}\\ \mathbf{0} &= \begin{bmatrix}\frac{1}{R_1} & -1\\-1 & 0\end{bmatrix}\begin{bmatrix}e_1\\i_V\end{bmatrix} + \begin{bmatrix}\frac{-1}{R_1} & 0\\0 & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix} - \begin{bmatrix}0\\-V_1\end{bmatrix}\end{aligned} \tag{A.36}$$

$$\begin{aligned}\frac{d}{dt}\begin{bmatrix}e_2\\i_L\end{bmatrix} &= \begin{bmatrix}\frac{1}{C_1R_1} & 0\\0 & 0\end{bmatrix}\begin{bmatrix}e_1\\i_V\end{bmatrix} + \begin{bmatrix}\frac{-1}{C_1R_1} & \frac{-1}{C_1}\\\frac{1}{L_1} & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix}\\ \mathbf{0} &= \begin{bmatrix}\frac{1}{R_1} & -1\\-1 & 0\end{bmatrix}\begin{bmatrix}e_1\\i_V\end{bmatrix} + \begin{bmatrix}\frac{-1}{R_1} & 0\\0 & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix} - \begin{bmatrix}0\\-V_1\end{bmatrix}\end{aligned} \tag{A.37}$$

$$\begin{bmatrix}\frac{1}{R_1} & -1\\-1 & 0\end{bmatrix}\begin{bmatrix}e_1\\i_V\end{bmatrix} = -\begin{bmatrix}\frac{1}{R_1} & 0\\0 & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix} + \begin{bmatrix}0\\-V_1\end{bmatrix} \tag{A.38}$$

$$\begin{bmatrix}e_1\\i_V\end{bmatrix} = -\begin{bmatrix}0 & -1\\-1 & \frac{-1}{R_1}\end{bmatrix}\begin{bmatrix}\frac{-1}{R_1} & 0\\0 & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix} + \begin{bmatrix}0 & -1\\-1 & \frac{-1}{R_1}\end{bmatrix}\begin{bmatrix}0\\-V_1\end{bmatrix} \tag{A.39}$$

$$\begin{bmatrix}e_1\\i_V\end{bmatrix} = \begin{bmatrix}0 & 0\\\frac{-1}{R_1} & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix} + \begin{bmatrix}V_1\\\frac{V_1}{R_1}\end{bmatrix} \tag{A.40}$$

$$\frac{d}{dt}\begin{bmatrix}e_2\\i_L\end{bmatrix} = \begin{bmatrix}\frac{1}{C_1R_1} & 0\\0 & 0\end{bmatrix}\left(\begin{bmatrix}0 & 0\\\frac{-1}{R_1} & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix} + \begin{bmatrix}V_1\\\frac{V_1}{R_1}\end{bmatrix}\right) + \begin{bmatrix}\frac{-1}{C_1R_1} & \frac{-1}{C_1}\\\frac{1}{L_1} & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix} \tag{A.41}$$

$$\frac{d}{dt}\begin{bmatrix}e_2\\i_L\end{bmatrix} = \begin{bmatrix}\frac{-1}{C_1R_1} & \frac{-1}{C_1}\\\frac{1}{L_1} & 0\end{bmatrix}\begin{bmatrix}e_2\\i_L\end{bmatrix} + \begin{bmatrix}\frac{V_1}{C_1R_1}\\0\end{bmatrix} \tag{A.42}$$

A.3.1 Exact Solution

$$\dot{e_2} = -\frac{1}{C_1 R_1} e_2 - \frac{1}{C_1} i_L + \frac{V_1}{C_1 R_1} \tag{A.43}$$

$$\ddot{e_2} = \frac{d}{dt}\left(-\frac{1}{C_1 R_1} e_2 - \frac{1}{C_1} i_L + \frac{V_1}{C_1 R_1}\right) \tag{A.44}$$

$$\ddot{e_2} = -\frac{1}{C_1 R_1}\dot{e_2} - \frac{1}{C_1}\dot{i_L} \tag{A.45}$$

$$\ddot{e_2} = -\frac{1}{C_1 R_1}\dot{e_2} - \frac{1}{C_1 L_1} e_2 \tag{A.46}$$

$$\ddot{e_2} + \frac{1}{C_1 R_1}\dot{e_2} + \frac{1}{C_1 L_1} e_2 = 0 \tag{A.47}$$

From [7, p. 56] we have for a system $y'' + ay' + by = 0$, where $a^2 - 4b < 0$, the solution is $y = e^{ax/2}(A\cos\omega x + B\sin\omega x)$, where A and B are arbitrary.

We have:

$$\left(\frac{1}{C_1 R_1}\right)^2 < 4\cdot\left(\frac{1}{C_1 L_1}\right) \tag{A.48}$$

$$1 < 40 \tag{A.49}$$

Thus, the solution is:

$$e_2(t) = e^{-\frac{1}{C_1 R_1}\frac{t}{2}}\left[P\sin\left(\sqrt{\left|\frac{1}{C_1 R_1} - 4\frac{1}{C_1 L_1}\right|}\frac{t}{2}\right) + Q\cos\left(\sqrt{\left|\frac{1}{C_1 R_1} - 4\frac{1}{C_1 L_1}\right|}\frac{t}{2}\right)\right] \tag{A.50}$$

Using the initial conditions:

$$\begin{bmatrix} e_2(0) \\ i_L(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$e_2(0) = P\sin(0) + Q\cos(0) = 0 \tag{A.51}$$

$$\Downarrow \tag{A.52}$$

$$Q = 0 \tag{A.53}$$

$$\dot{e_2}(0) = -\frac{1}{C_1R_1}e_2(0) - \frac{1}{C_1}i_L(0) + \frac{V_1}{C_1R_1} = \frac{V_1}{C_1R_1} \tag{A.54}$$

$$\frac{d}{dt}\left(e^{-\frac{1}{C_1R_1}\frac{t}{2}}P\sin\left(\sqrt{\left|\frac{1}{C_1R_1} - 4\frac{1}{C_1L_1}\right|}\frac{t}{2}\right)\right)|_{t=0} = \frac{V_1}{C_1R_1} \tag{A.55}$$

$$\frac{d}{dt}\left(e^{-\frac{1}{2C_1R_1}t}P\sin\left(\frac{\sqrt{\frac{4}{C_1L_1} - \frac{1}{R_1C_1}}}{2}t\right)\right)|_{t=0} = \frac{V_1}{C_1R_1} \tag{A.56}$$

$$= \left((\dots)\sin((\dots)t) + e^{(\dots)t}P\left(\frac{\sqrt{\frac{4}{C_1L_1} - \frac{1}{R_1C_1}}}{2}\right)\cos((\dots)t)\right)|_{t=0} \tag{A.57}$$

$$P\left(\frac{\sqrt{\frac{4}{C_1L_1} - \frac{1}{R_1C_1}}}{2}\right) = \frac{V_1}{C_1R_1} \tag{A.58}$$

$$P = \frac{2}{\sqrt{\frac{4}{C_1L_1} - \frac{1}{R_1C_1}}}\frac{V_1}{C_1R_1} \tag{A.59}$$

$$P = \frac{2V_1}{R_1\sqrt{\frac{4C_1}{L_1} - \frac{C_1}{R_1}}} \tag{A.60}$$

$$e_2 = \frac{2V_1}{R_1\sqrt{\frac{4C_1}{L_1} - \frac{C_1}{R_1}}}e^{-\frac{1}{2C_1R_1}t}\sin\left(\frac{\sqrt{\frac{4}{C_1L_1} - \frac{1}{R_1C_1}}}{2}t\right) \tag{A.61}$$

$$e_2 = \frac{2V_1}{R_1\sqrt{\frac{4C_1}{L_1} - \frac{C_1}{R_1}}}e^{-\frac{1}{2C_1R_1}t}\sin\left(\frac{\sqrt{\frac{4}{C_1L_1} - \frac{1}{R_1C_1}}}{2}t\right) \tag{A.62}$$

$$\frac{d}{dt}i_L = \frac{1}{L_1}e_2 \tag{A.63}$$

$$\frac{d}{dt}i_L = \frac{1}{L_1}\frac{2V_1}{R_1\sqrt{\frac{4C_1}{L_1}-\frac{C_1}{R_1}}}e^{-\frac{1}{2C_1R_1}t}\sin\left(\frac{\sqrt{\frac{4}{C_1L_1}-\frac{1}{R_1C_1}}}{2}t\right) \tag{A.64}$$

$$\frac{d}{dt}i_L = \frac{1}{L_1}\int\left(\frac{2V_1}{R_1\sqrt{\frac{4C_1}{L_1}-\frac{C_1}{R_1}}}e^{-\frac{1}{2C_1R_1}t}\sin\left(\frac{\sqrt{\frac{4}{C_1L_1}-\frac{1}{R_1C_1}}}{2}t\right)\right)dt \tag{A.65}$$

$$\frac{d}{dt}i_L = \frac{1}{L_1}\frac{2V_1}{R_1\sqrt{\frac{4C_1}{L_1}-\frac{C_1}{R_1}}}\int\left(e^{-\frac{1}{2C_1R_1}t}\sin\left(\frac{\sqrt{\frac{4}{C_1L_1}-\frac{1}{R_1C_1}}}{2}t\right)\right)dt \tag{A.66}$$

Using integration rule from [6, 132, p. 144]:

$$\int e^{ax}\sin bx dx = \frac{e^{ax}}{a^2+b^2}(a\sin bx - b\cos bx) + C \tag{A.67}$$

With the initial condition $i_L(t=0)=0$ this becomes:

$$0 = k_4e^{k_2t}(k_2\sin k_3t - k_3\cos k_3t) + C\Big|_{t=0} \tag{A.68}$$

$$0 = k_4e^0(k_2\sin 0 - k_3\cos 0) + C \tag{A.69}$$

$$0 = -k_4k_3 + C \tag{A.70}$$

$$C = k_3k_4 \tag{A.71}$$

$$\begin{bmatrix} e_1 \\ e_2 \\ i_L \\ i_V \end{bmatrix} = \begin{bmatrix} 5 \\ k_1 e^{k_2 t}\sin(k_3 t) \\ k_4 e^{k_2 t}(k_2 \sin k_3 t - k_3 \cos k_3 t) + k_3 k_4 \\ \frac{-k_1}{R_1} e^{k_2 t} sin(k_3 t) \end{bmatrix}, \text{where:} \tag{A.72}$$

$$k_1 = \frac{2V_1}{R_1\sqrt{\frac{4C_1}{L_1} - \frac{C_1}{R_1}}} \tag{A.73}$$

$$k_2 = -\frac{1}{2C_1 R_1} \tag{A.74}$$

$$k_3 = \frac{\sqrt{\frac{4}{C_1 L_1} - \frac{1}{R_1 C_1}}}{2} \tag{A.75}$$

$$k_4 = \frac{1}{L_1}\frac{k_1}{k_2^2 + k_3^2} \tag{A.76}$$

When inserting values for R_1,L_1, C_1 and V_1, (A.72) becomes:

$$\begin{bmatrix} e_1 \\ e_2 \\ i_L \\ i_V \end{bmatrix} = \begin{bmatrix} 5 \\ 1.58 \cdot 10^{-2} e^{-0.5t}\sin(316t) \\ 1.58 \cdot 10^{-5} e^{-0.5t}(-0.5\sin(316t) - 316\cos(316t)) + 0.005 \\ -1.58 \cdot 10^{-5} e^{-0.5t}\sin(316t) \end{bmatrix} \tag{A.77}$$

B Attached files

The attached archive file contains the source code for this report, some plot generating functions and the latest revision of tissue.

B.1 Plot generation

Copy-pasting plot-data from the web-browser was too tiresome and error prone, so the process of generating plots has been automated. By using a javascript engine running on a GNU operating system, some slightly modified code from the web-application can be run with out an web-browser. Tools used (tested with): js (JavaScript-C 1.8.5), make (GNU Make 3.82) and gnuplot (gnuplot 4.6).

B.1.1 RC circuit

Files in location: plots/rc:

method Directory containing the numerical methods used; all share the same interface.

- **bweuler.js** Implicit Runge-Kutta, order 1, specifically The implicit/backwards Euler method.
- **erk4.js** Explicit Runge-Kutta, order 4, specifically The Runge-Kutta method.
- **euler.js** Explicit Runge-Kutta, order 1, specifically The Euler method.

  - **lobatto.js** Implicit Runge-Kutta, order 2, specifically Lobatto IIIC, order 2.

- **exact.js** Code generating the exact solution of the system in rlc. Note: it only supports an underdamped system.
- **numerical.js** Code generating an numerical solution of the same system, using one of the numerical methods above.
- **plot** Directory containing the plots after using the Makefile.
- **rc.js** Defines the resitance, capacitance and source voltage for an RC system. It also specifies the step-length for the numerical methods, sampling rate for the exact solution, and length of simulation.
- **Makefile** File for automatically generating plots for each of the numerical methods, plotted along with the exact solution.

B.1.2 RLC circuit

Files in location: plots/rlc:

- **method** Directory containing the numerical methods used; all share the same interface.
  - **bweuler.js** Implicit Runge-Kutta, order 1, specifically The implicit/backwards Euler method.
  - **erk4.js** Explicit Runge-Kutta, order 4, specifically The Runge-Kutta method.
  - **euler.js** Explicit Runge-Kutta, order 1, specifically The Euler method.
  - **lobatto.js** Implicit Runge-Kutta, order 2, specifically Lobatto IIIC, order 2.
- **data** Directory containing the results from the simulation and sampling.

exact.js Code generating the exact solution of the system in rlc. Note: it only supports an under damped system.

numerical.js Code generating an numerical solution of the same system, using one of the numerical methods above.

plot Directory containing the plots after using the Makefile.

rlc.js Defines the resitance, inductance, capacitance and source voltage for an RLC system. It also specifies the step-length for the numerical methods, sampling rate for the exact solution, and length of simulation.

sylvester.js Library for matrix calculations.

Makefile File for automatically generating plots for each of the numerical methods, plotted along with the exact solution.

B.1.3 Usage

Just type make in either of the aforementioned locations.

```
$ cd plot/rlc
$ make
```

This will generate plots in both pdf and png format in the plot subdirectory. It is of course possible to generate the simulated values manually:

```
$ cd plot/rlc
$ cat sylvester.js method/erk4.js rlc.js numerical.js | js
$ cat rlc.js exact.js | js
```

The first line outputs the numerical solution, with a time and e_2 voltage separated by a comma on each line. The second line outputs the exact solution in the same format.

B.2 Tissue

This folder contains the latest revision of our application.

cert.pem Sample certificate for the application.

key.pem Sample encryption key for the application.

main.go The backend.

static Folder containing hosted files.

> **tissue** The frontend.
>
> **storage** Some test facilities for *storage.js*.

To run tissue, you'll need MongoDB (2.0.5 or newer) and Google Go (1.0 or newer) To try out the application, first start MongoDB, and run the backend with:

```
$ cd tissue
$ go run main.go
```

This creates a https server running on localhost. By accessing `https://localhost:4443/static/` you can either try out the schematic editor in `tissue`, or the user control system in `storage`.

Bibliography

[1] [Online]. Available: http://hackaday.com/2008/12/27/parts-8bit-io-expander-pcf8574/

[2] I. Free Software Foundation, “Gnu general public license, version 3,” 2007. [Online]. Available: http://www.gnu.org/licenses/gpl-3.0.html

[3] P. Falstad, “Falstad circuit simulator.” [Online]. Available: http://www.falstad.com/circuit/

[4] M. Hanke, “An introduction to the mofied nodal analysis,” 2006. [Online]. Available: http://www.nada.kth.se/kurser/kth/2D1266/MNA.pdf

[5] O. Egeland and J. T. Gravdal, *Modeling and Simulation for Automatic Control*, 2nd ed. Marine Cybernetics AS, 2002.

[6] K. Rottmann, *Matematisk Formelsamling*, 12th ed. Spektrum forlag, 2010.

[7] E. Kreyszig, *Advanced Engineering Mathematics.* John Wiley & Sons, inc., 2006.

[8] I. 10gen, “mongodb.” [Online]. Available: www.mongodb.org

[9] J. Coglan, “Sylvester.” [Online]. Available: sylvester.jcoglan.com