Automated Circuit Simulation Control Toolchain

Mitja Stachowiak

Masterthesis – avril 2023

Tutor: Andrea Zingariello, M.Sc.

TECHNISCHE UNIVERSITÄT DARMSTADT

LEA

Mitja Stachowiak	
Matrikelnummer:	2010632
Studiengang:	Elektrotechnik und Informationstechnik
Studienrichtung:	Datentechnik
Masterthesis:	Automated Circuit Simulation Control Toolchain
MA 1512-2022	
Eingereicht:	10.04.2023
Betreuer:	Andrea Zingariello, M.Sc.
	Prof. Dr.-Ing. Gerd Griepentrog

Institut für Stromrichtertechnik und Antriebsregelung
Fachbereich Elektrotechnik und Informationstechnik
Technische Universität Darmstadt
Fraunhoferstraße 4
64283 Darmstadt

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Reinheim, den 10.04.2023 Mitja Stachowiak

Mitja Stachowiak

Abstract

The automation of measurement and simulation for creating detailed transfer characteristics and other circuit analysis becomes more and more important. Currently, scripts to do such automation are usually written in Matlab. This is because Matlab has a huge library to interact with common circuit simulation software.

The drawback is, that most of this scripts are weakly documented and designed for one special purpose only. The Matlab scripting language has no advanced code features and maintaining of software, written in Matlab, is very difficult.

This is the motivation of creating an automation toolchain written in a true programming language, that abstracts from proprietary interfaces but delivers a general interface to modify parameters of arbitrary simulations and returns the result data.

Based on this, it should become simple to write custom simulation drivers, that can control common simulation software and/or process signals from real measurement devices.

This thesis presents such a toolchain with two specialized drivers that can map the operating regions of power electronic circuits or synchronize two simulations or one simulation and a set of measurements by comparing their resulting signals.

The software can be found on https://mitjastachowiak.de/projects/simulationtoolchain/

Keywords: simulation automation, semiconductor regions, map conduction modes, compare simulation and measurement, find parasitic elements

Table of Contents

List of Figures

List of Tables

Glossar

Abbreviations

C2L	Coupled inductor boost converter. The circuit regarded in this thesis is shown in Appendix A2.
CSV	Comma/Character Separated Values: File format to store table-like data
GUI	Graphical User Interface
HCL	Hue-Chroma-Luminance: A color description in three components
PC	Personal Computer
PCB	Printed Circuit Board
RPC	Remote Procedure Call: Protocol to transfer commands and data between applications. The format of the transferred data can be XML, bus also JSON is used.
SRC	Series Resonant Converter: Halve bridge with capacitor in current path, that gives a resonant tank together with the parasitic inductance of the transformer. The circuit regarded in this thesis is shown in Appendix A1.
UML	Unified Modeling Language: Graphical description of software modules
XML	Extensible Makeup Language: Document format with nestable elements.
ZCS	Zero Current Switching: There is a resonant tank in the alternating current path, so that the current performs a sinusoidal half wave after one transistor was turned on. The transistor gets turned off, after this wave becomes zero, so there is no turn-off current through the transistor.
ZVS	Zero Voltage Switching: The turn-off of one transistor causes a tail current to commute to the other transistor and (dis)charges its parasitic capacity so that the voltage on this transistor is zero, when it turns on.

Physical symbols

F	Farad
H	Henry
Hz	Hertz
I	electric current
t	time
V	voltage

Circuit symbols and indices

C	capacitor
D	diode
L	inductance
Q, S	transistor
R	resistor
Tr	transformer

1 Software Design

There already is a basic software, created during an industrial practical in 2018. It is written in Freepascal/Lazarus and can drive simulations in Plecs and xFEMM. PSpice is supported as well, but due to some re-engineering and the lack of a PSpice license not ready-to-use yet.

The initial project was developed for Windows, but now runs on Linux. It is possible and planned to deploy builds for both operating systems later.

In this context, simulation software is described with *tools*.

1.1 General design

There are simulation drivers, that do the simulation run control and tool support units, that implement several abstract classes and interfaces to allow the software to control the simulation tools. Both types of classes can be written in one unit each, that just need to be included in one uses-list to show up in the menu. And both inherit from the so called TFrame, which is a blank GUI form to allow easy development of graphical interfaces.

The software is designed to run several simulations in parallel. To archive this, there is a task-queue concept and the so called netlists are clonable. This are the lists of available elements and parameters in the different simulation tools, that can be modified by the software. For parallelization, each thread can work on a copy of the initial netlist.

The simplified UML-diagram looks like this:

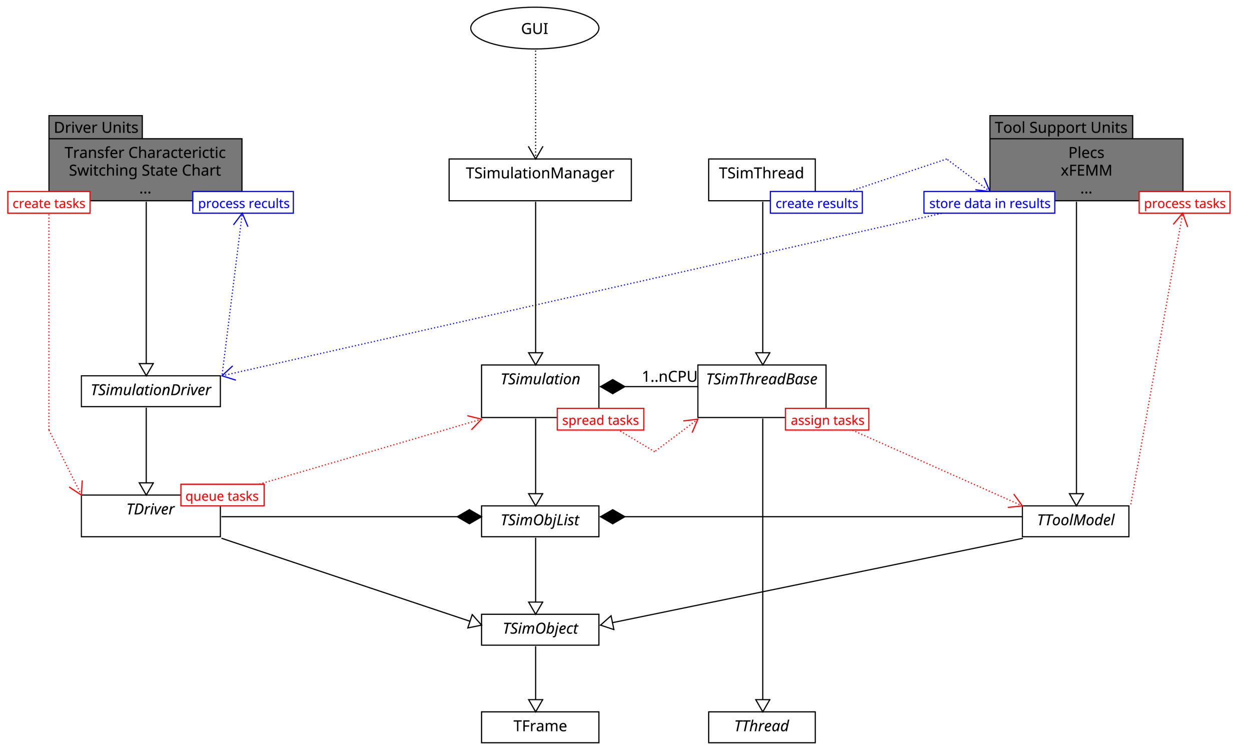


Figure 1.1: UML diagram of important classes

1.2 Simulation tool support and abstraction

The abstraction of tool support orients on classic circuit simulations, but can also address arbitrary simulations. When automation is used, the algorithms usually apply value changes to several parameters or elements, but don't process topological modifications, like reassigning of wire connections, to the simulation.

Each tool support is represented by a successor of TToolModel, which instances are called models. This class contains several sub-classes, that interfere with the tool chain.

The basic structure to hold information on elements and parameters to be modified, is the TNetlist-class. Each TTask, created by the drivers, holds information, which parameters of the netlist should be modified. The tool support implements the abstract class TTaskOperator, that knows the current task, netlist and thread. When a task is ready to be processed, the TTaskOperator's prepareProcess-function is called. The TTaskOperator's basic implementation of this function hands over the task and netlist to the driver, which can then apply its modifications specified in the task.

The TTaskOperator's abstract execProcess-function must be implemented to execute the external simulation tool.

After this, the TTaskOperator's evalProcess-function hands over the previously parsed result data to the driver and destroys the current task. This is the functionality of TTaskOperator.

In addition, each tool support needs to implement a custom successor of TNetlist. Each netlist has one main circuit, that can hold several elements or sub-circuits. Each element has multiple values. This yields a tree-structure of parameters, that can be modified by the drivers. The classes may be named netlist, circuit or element, but can also apply i. e. for field simulations, where an element could be a point in space.

Finally there is the TResultData class, that can be inherited to apply a custom type of data storage, that best fits the requirements for a certain tool. There are sub classes like TProbe1D, TProbeScalar, etc., that can store typical data and be used in the drivers.

1.2.1 Tool Support for Plecs

The Plexim spice simulation software offers a XML-RPC interface to enable control from external tools. Plecs can run simulations in parallel, but this is currently not possible or at least not documented using the XML-RPC interface.[1] So the tasks for Plecs are limited to just one thread.

The XML-RPC result is very weird: Each float value of a simulated signal is packed in a tree of XML-Elements. Interpreting this structure with a common XML-parser would cost too much computation time – even more, than the simulation itself. So a rapid finite-state-machine was written, that goes through the XML without creating any heap object. The common stringToFloat-function of freepascal is also very slow and was replaced by a readFloatFromStream inline function, that stops at the first character, that doesn't belong to the float.

It turned out, that string to float conversions in general can be inexact[2]: The problem is, that IEEE 754 floating point numbers work with an exponent base of two, while the human readable format uses base 10. There is a table to convert between both numbers. But doing this with double precision[1] can cause inaccuracy. A sorted record of double values converted to string and back to double can cause flips in sequence if the exponent conversion table is just double precision. Since this table was stored in extended precision[2], this problem did not occur anymore – which does not mean, that the conversion is accurate in all cases. On platforms, that don't support extended precision, the problem is potentially not solved.

1.2.2 Tool Support for xFEMM

The very often used field simulation software FEMM was improved by a fast, simplified non-GUI version called xFEMM [3]. This can load a LUA-script, that can control the post processor. The raw field simulation result is unhandy, as it describes potentials at the tiling points. So there is a post processor in xFEMM, that offers functions for interpolating the field strength at a certain point in space.

The xFEMM tool support has a text memo, where short LUA snippets can be written, that print such values to the std-out. This is then converted into a result data probe.

For modification, a parser for .fem-files was written. This can completely store a fem-simulation in the TNetlist and print it to a new file. This way, a copy of the fem-file is created for each thread and the simulations run in parallel.

1.2.3 Tool Support for oscilloscopes

To get use of true measures in the tool chain, there is a support class for CSV-based oscilloscope exports. Actually the Tektronix TDS2024B is supported, which creates a folder with several CSV-files for each export. In addition, a file named scalar.csv can be places in the folder, where for example values of multimeters, that belong to the oscilloscope signals, can be stored.

The tool then contains no elements in its netlist but holds only one index parameter in its netlist's simulation parameters record. Using this index, the desired measurement export can be chosen.

With this oscilloscopes support, measurements can be used like the signals of simulations in the tool.

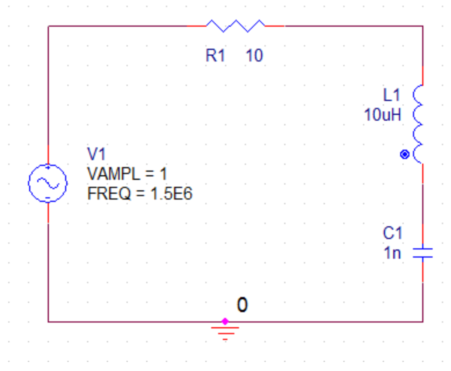
1.3 Simulation drivers

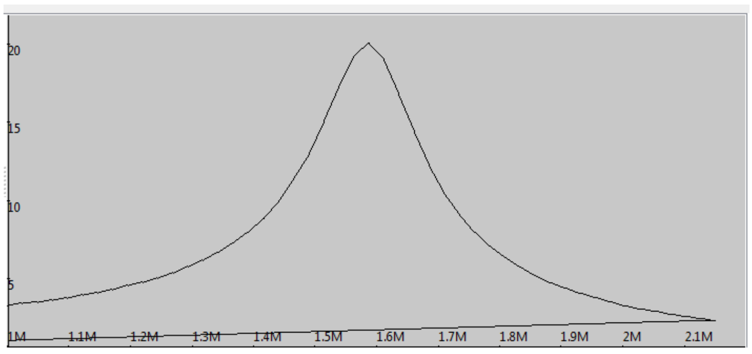
Simulation drivers create tasks and/or manage modifications to the netlists. Each driver can implement a successor of TTask. Each task is assigned to one model, that should perform a simulation run.

1 double precision has 53 bits for the mantissa
2 extended precision is a weakly standardized format that, for x86 processors, holds a 64-bit mantissa

1.3.1 Transfer Characteristic

A very basic driver is the transfer characteristic. It can for example calculate a transfer characteristic of a RLC-Circuit for a frequency sweep:





The result is multidimensional: An arbitrary number of parameters, that should be modified, can be selected. The first one is used for the chart's x-axis. If this should contain 20 steps at different frequencies and in addition, the value or R should be modified in 5 steps, the driver iterates through 100 simulations.

For each simulation, the characteristic can store multiple values, computed from the result data. There are helpers, that can find the maximum, minimum, average or similar values from the 1D time transient signals.

1.3.2 Patch

A patch is a driver that doesn't create tasks but hooks into the model modification of other drivers and modifies parameters with custom functions of other parameters. For example a half bridge has two transistors, that switch with a half period delayed gate signal. For the simulation, this can be generated using two rectangular voltage sources. If i. e. a transfer characteristic wants to iterate through several switching frequencies, it only applies the values for one signal source and a patch can generate the second source's parameters from this.

2 Switching State Chart

The idea behind the switching state chart is, to map the *order* of switching states. Each semiconductor has different states of operation: cutoff -, ohmic -, saturation region and so on. By specifying the voltages and currents at the semiconductor device pins, the driver can detect, in which region the device operates.

During one period, the device can go through different states. For the complete circuit, this could look like [t1: Q1 goes from saturation to ohmic region | t2: D1 goes from reverse to "conductive" region |...]

Each of this transition sequences during one period can cause a different behavior of the circuit and probably needs extra care, regarding the boundary conditions of safe operating state.

The idea of the switching state chart is to automatically detect this states in arbitrary circuits and over wide multi-dimensional parameter sweeps.

2.1 Motivation

One typical topology, where this is of high interest, is the so-called coupled inductor boost converter (C2L). Under load, the current through a common boost converter's coil consists of a large direct part and a smaller ripple. The coil should have a high inductance to keep the ripple small. But a coil with high inductance, that can also handle a large (direct) current, is also large in space.

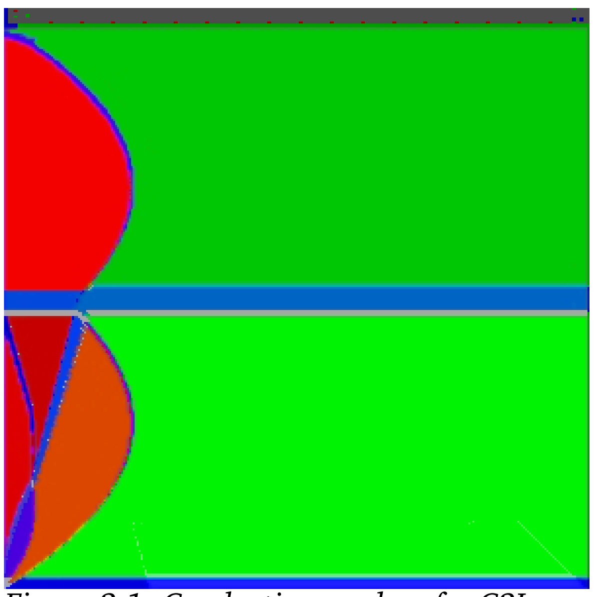


Figure 2.1: Conduction modes of a C2L converter, detected using Matlab

In the coupled inductor boost converter, there are two current paths with two transistors. The two coils share a larger part of their magnetic flux and are reversely wound, so that the direct current's magnetic fields nullify each other. This way, a lot of ferromagnetic material can be saved.

The drawback is, that the circuit is hard to control, as there is not just one continuous and one discontinuous conduction mode, but a variety of modes, where currents can become zero after each other or at the same time. In related work, a Matlab script was written, to map this modes over different output currents (x-axis) and duty cycles (y-axis) – see figure 2.1.

But this script can only map modes of exactly this topology. It is not that easy to rewrite this algorithm for other models, as there is a lot of optimization required.

2.2 Workflow

First of all, the driver needs a set of Variables, that should be iterated. A variable usually is a parameter of a circuit element. This parameter can be selected in an element dialog. For each variable, a start and a stop value can be given as well as the number of steps in between.

A set of three variables like

I_1: [1A .. 10A] in 10 steps

R_1: [1kΩ .. 2kΩ] in 50 steps

V_1: [100V .. 300V] in 30 steps

would create an array of $10 \cdot 50 \cdot 30 = 15000$ items. It can be seen, that many variables quickly result in an exponentially hard problem. To reduce the computational effort, there is a special interpolation: There are raw steps, where the state is always simulated. But the number of raw steps can be set to a low value and there is an additional interleave step count (See section 2.3).

Next task is to specify the semiconductors (transistors, diodes) in the circuit on the Switches-tab. This are the elements, that can operate in different regions. To determine the operating region in each time step, several voltages or currents on the device need to be given. By selecting a supported semiconductor in the circuit, a list of probes appears under the element, to specify the one-dimensional time signals of this voltages or currents in the result data.

The given signals are analyzed for one period. It is not guaranteed, that this time period is always correctly defined by capture start time and stop time. It is very common, to modify for example the switching frequency as one variable. To offer better opportunities for defining the period, the triggers are used. Models for circuit simulations, which mainly return time signals, have an interface, that can set up triggers. There is always an *analyze begin* and an *analyze end* trigger. This can be capture start - and stop time, but also a rising edge of a gate signal.

When variables, semiconductors and the period are specified, the simulation may run. Ont the States-tab, the switching state sequences are then detected automatically and get a color assigned in HCL color space:

- Hue/Color component is chosen by the index of the sequence – this appears as if it was randomized.
- Chroma component represents the frequentness of the sequence: States that occur over a large set of variable combinations get a strong color, states that occur seldom get gray-scaled.
- Luminance component represents the number of states of the sequence. When there is for example high-resonant ringing in the circuit, a high number of semiconductor

state transitions can occur. This is indicated with a bright color, while the simple sequences get a dark color.

It turned out, that there are often nearly similar switching states frequently alternating in some areas, which discrimination is of no interest. To avoid the computation of many interleave points in this areas, the related states can be defined as equal. If a state A is defined equal to state B, state B will be stored for all points, where state A is detected. This can be done using the = -Button at the detected states or by right-clicking in the chart.

Once detected states and their changeable colors are re-used for newer simulations. If a semiconductor is modified, added or removed, all stored states from earlier simulations have no meaning anymore and get cleared.

Finally it is not only interesting to map the switching states, but also different extra probe values for each result array item. They can be selected on the Capture -tab. Because there can be set an arbitrary number of such probe values, the memory size of the result data items is not fixed but variable as well. So the access to the result data array is done with some pointer casting.

When simulations have run, the captured probes can be painted by selecting the related probe item in the Capture -tab. Going to the States -tab will show the switching states again. The exact number values of the probes can be displayed by moving the cursor over the chart.

Which variable is x- and which is the y-axis can be selected on the Variables -tab, by using the X or Y -button at the bottom after selecting a variable. All other variables will get a track bar, where their value can be set. The chart shows a two-dimensional cut through the result data hyper cube.

A mouse down click on the chart will show the exact x and y-position of the cursor in the Variables -tab. Releasing the mouse will then trigger an extra simulation run at the selected point, so the probe signals can be calmly analyzed in the signal monitor(s).

2.3 Interpolation algorithm

The number of steps in which each variable is iterated is separated in two types: Raw steps and interleave steps. The number of raw steps is the second edit field. At this points, the simulation is definitely run in a first pass. The first edit field defines, how many interleave steps lay between two raw steps.

Setting for example 5 interleave steps between 10 raw steps would result in $10+(10-1)\cdot 5=55$ final steps. When the first pass for computing the raw steps is finished, the interleave steps in the middle of two raw steps are computed in the second pass. If the number of interleave steps is even, the lower step next to the middle is taken. This halving of interval is repeated till all steps are known.

For interleave steps, a simulation is only triggered, if the surrounding points have different state sequences. If they are equal, it is assumed, that the same sequence also occurs at the interleave step's point. The additional probe values are interpolated linearly.

For one dimension with 6 interleave steps, the order of interleave iteration could look like

1	3	4	2	4	3	4	1

with the blue raw-steps in first pass.

For multiple dimensions, the problem is more complex: There is not just one center interleave point, but with regard in which dimension(s) the interval is halved, many center points exist in each pass. Generally spoken, 2^n raw points span a n-dimensional hyper cube between adjacent raw grid steps. In each hypercube, there is always one "volume" center point, where the interval in all dimensions is halved. If all edge points are of the same sequence, this center point can be assumed to be of the same sequence as well and can be interpolated. Otherwise a simulation has to be done for this point.

In next pass, the interval is halved in n-1 dimensions, which gives 2n interleave points to regard. In the non-halved dimension, the previous volume center points become neighbors where same state sequence is required for interpolation.

In the next pass, the interval is halved in n-2 dimensions, which gives $4 \cdot \binom{n}{n-2}$ interleave points to regard, and so on...

Figure 2.2 shows, how this works in 3 dimensions: The raw step points (1) are the corners of the cube. In first pass, the volume centers (a) are regarded, in second pass the surface centers (b), and in third pass the edge centers (c). The red highlighted b illustrates, which neighbors need to be of the same sequence to allow interpolation.

All points of (a), (b), (c) belong to the first halving of the interleaves. For next halving, the cube is divided into 8 smaller cubes, whose corners are the marked points (1), (a), (b), (c).

With this algorithm, smoothly pointed convexities in the regions of equal sequence can be retraced.

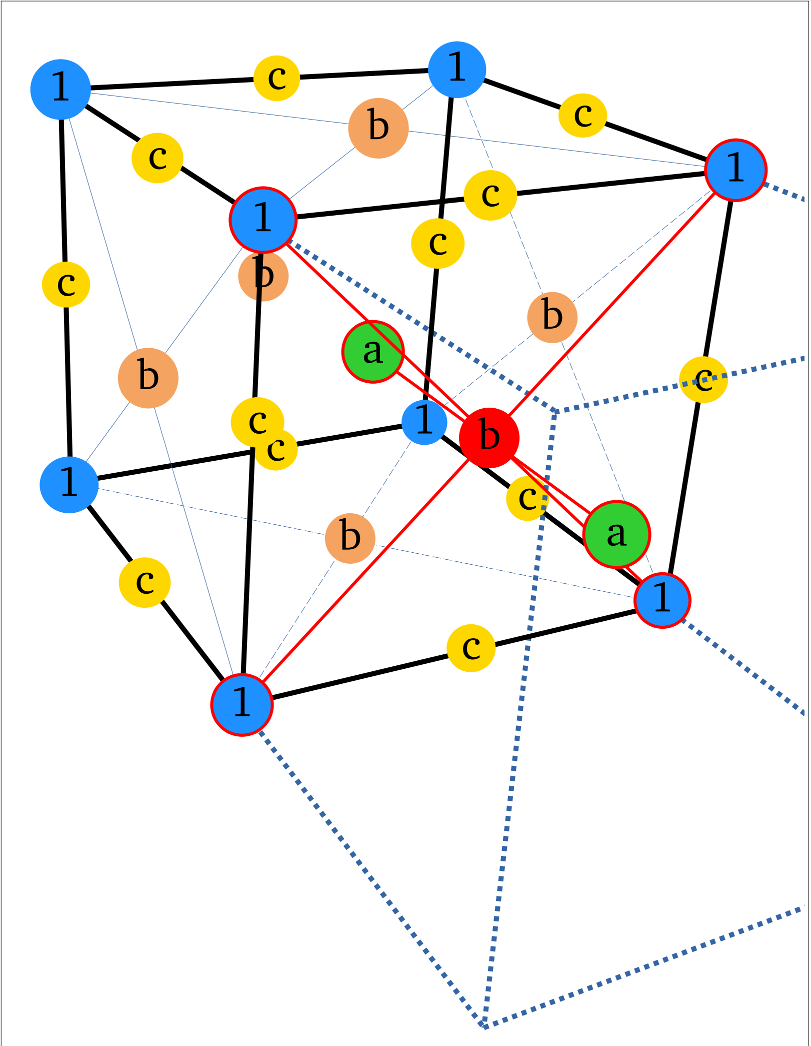


Figure 2.2: 3D-example of interleave halving

During simulations, the interpolation looks like Figure 2.3: The gray pixels are the step points with queued tasks to be simulated. The image shows the last two halvings of interleave step interval. In the second last iteration, there are black pixels, which will remain empty in this iteration and are going to be interpolated or simulated in the final iteration.

It can be seen, that the queued tasks only occur near the borders of equal sequence regions.

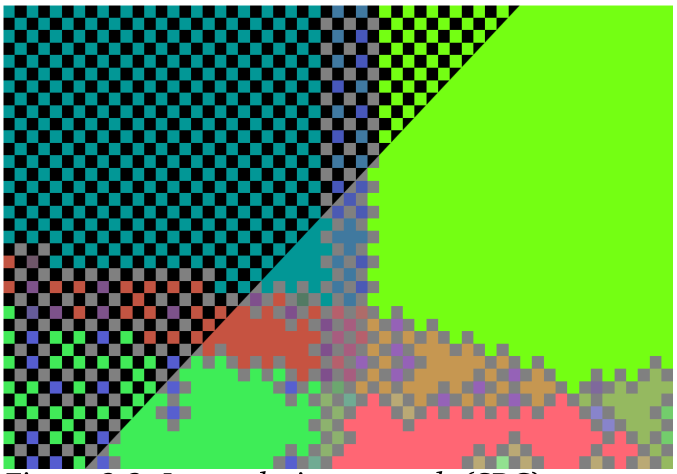


Figure 2.3: Interpolation at work (SRC)

For the tested SRC-chart here, only the parameters f_s, duty cycle and I_{dc} were changed in a certain range that gives 21.090 points.
In the beginning, two negligible sequences in the large areas were synchronized. For this test, the number of simulated and interpolated points was counted as shown in Figure 2.4. The interpolation algorithm could save more than $\frac{2}{3}$ of the simulations.

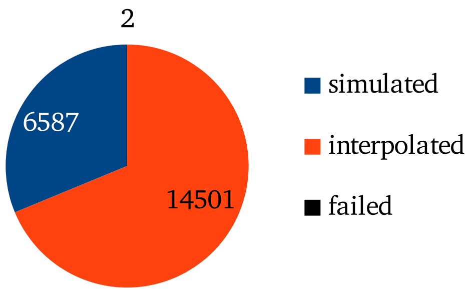


Figure 2.4: Reduction by interpolation

2.4 State chart results

First, the results of the Matlab script, mentioned in section 2.1, should be reconstructed. The same model (A2) was generated in Plecs. The following 3D parameter sweep was done:

Table 2.1: Range of parameter sweep for C2L-circuit

Circuit Element	Description	Start	Stop	Raw Steps	Interleave Steps
Iref	Desired output current	1 A	50 A	10	5
Pulse Generator	Duty cycle of low transistor's gate signal; copied to the high pulse generator using a patch driver (see 1.3.2)	0.05	0.95	10	5
Vin	Input voltage	300 V	500 V	3	1

The driver then generated a chart with the expected look (Figure 2.9):

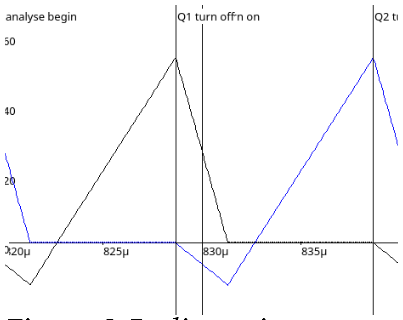


Figure 2.5: discontinuous conduction mode currents with negative current

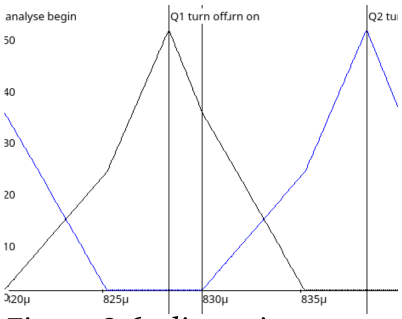


Figure 2.6: discontinuous conduction mode currents

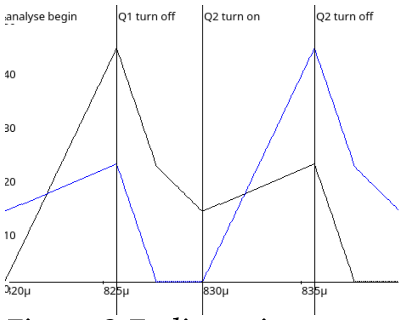


Figure 2.7: discontinuous conduction mode currents

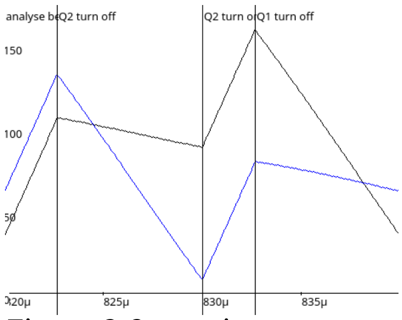


Figure 2.8: continuous conduction mode currents

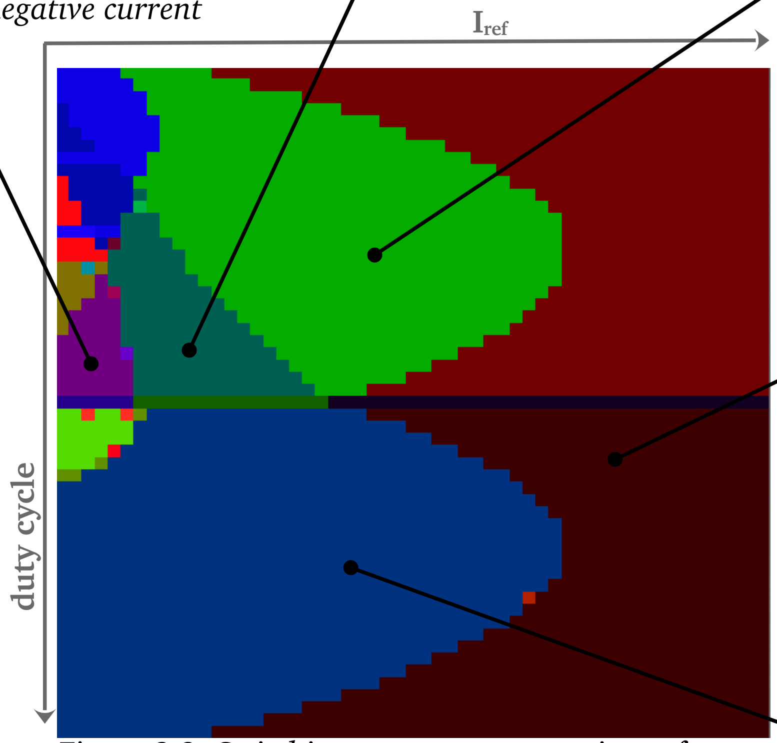


Figure 2.9: Switching state sequence regions of coupled inductance boost converter at 400 V

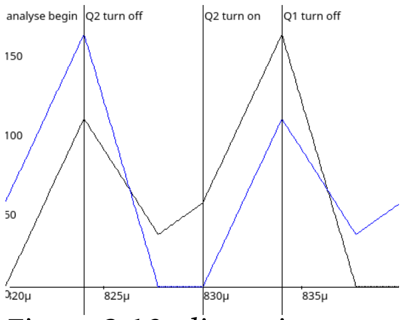


Figure 2.10: discontinuous conduction mode currents

An other interesting use case of the switching state driver was the analysis of a series resonant converter (A1). For this model, 4 parameters were changed over a wider range to map the regions, where ZVS and ZCS happen.

Table 2.2: Range of parameter sweep for SRC-circuit

Parameter	**Description**	**Start**	**Stop**	**Raw Steps**	**Interleave Steps**
f_s	Frequency of high- and low gate pulse generator	200 kHz	280 kHz	12	5
duty cycle	Duty cycle of high- and low gate pulse generator	0.495	0.47	10	5
I_{dc}	Output current sink I_dc	6	8	4	2
L1	Main inductance	1.1 mH	50 μH	6	4

After days of simulation, the state sequences were mapped:

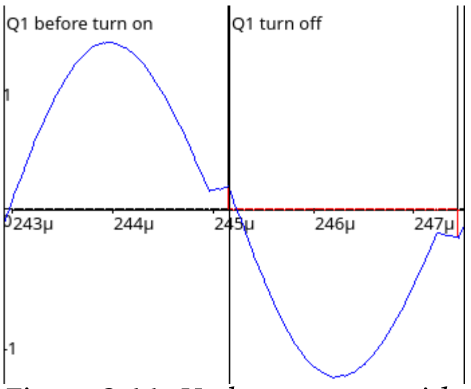


Figure 2.11: Under resonant with ZVS and nearly ZCS

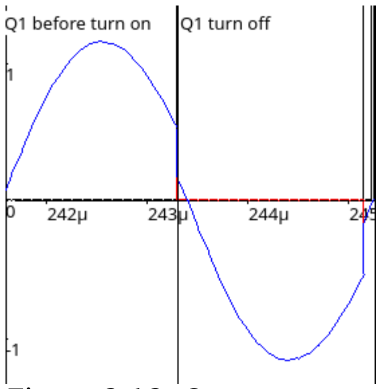


Figure 2.12: Over resonant with loss of ZVS

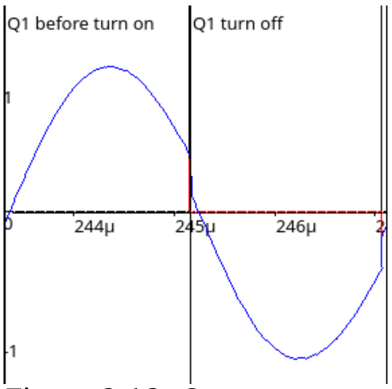


Figure 2.13: Over resonant with ZVS

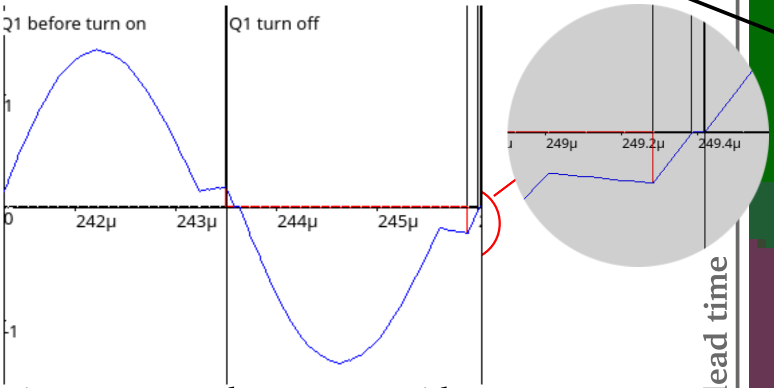


Figure 2.15: Under resonant with loss of ZVS due to long dead time

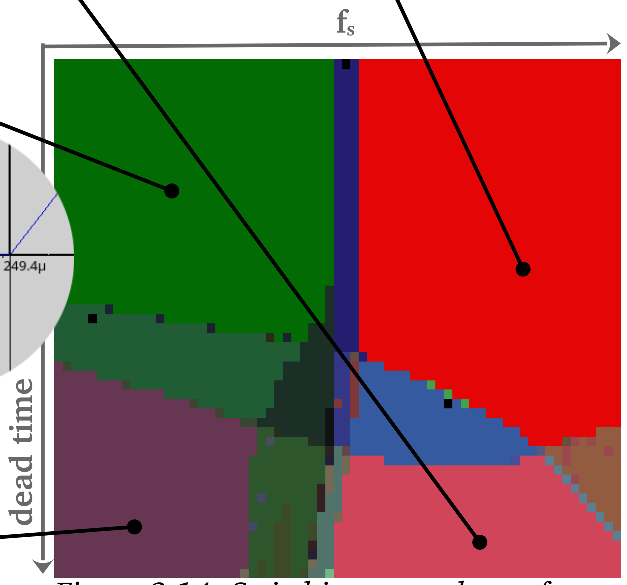


Figure 2.14: Switching state chart of SRC, cut through 4D-result at I_{dc}=8 A, L_1=1.1 mH

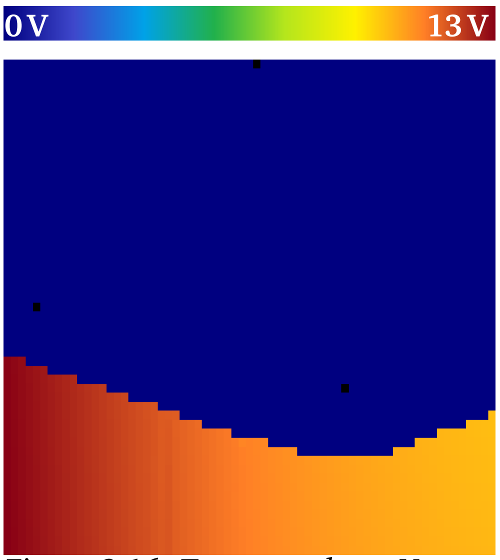


Figure 2.16: Turn on voltage $V_{ds,S+}$ at I_{dc} = 8 A, L_1 = 1.1 mH

By plotting the voltage on i. e. high transistor at the time of this transistor's turn on trigger (Figure 2.16), a raw estimate, where high switching losses can occur, can be made – as well by plotting the current, when the transistor turns off (Figure 2.17).

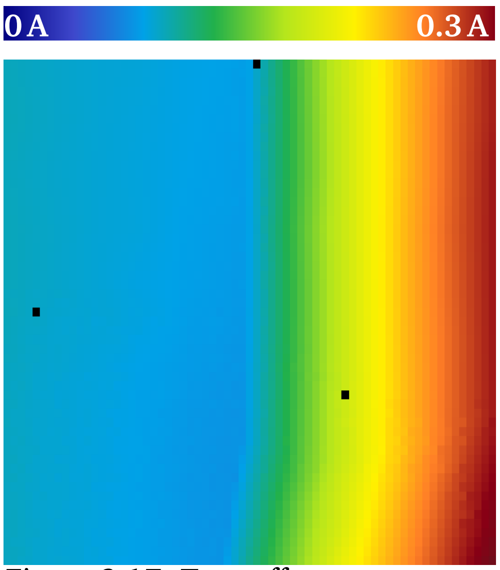


Figure 2.17: Turn off current at I_{dc} = 8 A, L_1 = 1.1 mH

3 Simulation Synchronizer

The simulation synchronizer driver is meant to minimize differences between signals of two different models. This can be useful for many applications – one common case could be to search for fitting values of parasitic elements in a circuit. But also to adjust a circuit to get a desired signal.

3.1 Motivation

By developing resonant switching power converters, parasitic elements can have significant influence. The stray inductance of the transformer is not a parasitic but a scheduled part. But it is difficult, usually impossible, to build the transformer with exactly the planned stray inductance. Even after the transformer is built, it is difficult to measure the inductance, as it is non-linear.

A common task for engineers is, to watch the oscilloscope signals when launching the PCB and to readjust frequency, air gap or resonant capacity. For further development, it is then often necessary, to tune the model til it represents the real device.

This can be an annoying and time-consuming job. It can happen, that measurement and simulation look almost similar in one comparison, but at an other frequency, it appears, that they aren't. Maybe because one parameter is too large and an other to small and only for the regarded conditions, the mistakes nullify.

The idea of the synchronizer driver is, to make comparisons of many signals – for example of different frequencies – easy. Finally it could be very nice to have an algorithm, that automatically reduces the differences between simulations and measures over night.

The job before in the laboratory then is to generate oscilloscope exports of many different operating conditions. Especially to test extremely long dead times, where ringing over a long time can happen, and other weird configurations can help to allow adjusting different parameters against each other.

3.2 Workflow

3.2.1 Specify models and parameters

First step is to specify the models on the Models and Parameters -tab. There is one column for the so called variable model and one for the destination model. The variable model is the one with unknown parameters, that should be found so that the output signals best fit the ones from the destination.

Below the model selector, there is a field to enter the number of simulations, that should be made for each comparison of both models. Usually this number is the same for variable and destination model. But it can for example happen, that multiple CSV-exports were made under same conditions, like when stepping through different trigger events using the oscilloscope's run/stop-key.

The track bar is meant to select the index of simulation within each comparison. At the bottom, variables can be added to the models using the + -buttons. On the variable model's column, there is the additional + unknown -button, where those parameters can be specified, that should be modified in the variable model for each comparison. Variables specify the parameters, that should be modified for each simulation within the comparisons to generate the different conditions.

For an oscilloscope measure, you just want to add one variable and select the index parameter of the netlist's simulation parameters record. Then enter an "i" in the expression edit. This identifier represents the index of simulation within the related comparison, as selectable by the track bar. This way, all data exports of the oscilloscope were iterated in each comparison.

Any other identifier in the edit field will cause an extra entry below to appear, where, like in other drivers, a circuit element's parameter can be linked. Especially for this synchronizer driver's page, identifiers can also be a table-like function of the index. Use the ↻ -button of an identifier's entry to convert it into a table. Using the track bar, a different value can be entered for this identifier at each index.

3.2.2 Link probes

Next step is to link the equivalent signals of both models on the Probes -tab. Currently supported are only 1D-signals with real values. But scalar values are intended to be used as well. Each probe can have a *weight*, used for total result. If left empty, the weight is 1.

3.2.3 Find signal matches

On the Matches and Constraints -tab, there can then be defined sections of the previously selected signals, that should be taken into account for comparison. Each section can occur (match) once for each relation of destination and variable model result and has a *weight*, like the probes. So if there are 5 destination and 3 variable simulations in each comparison and a match section without any constraint is added, it will mach $5 \cdot 3 = 15$ times.

To get a meaningful result, the sections need constraints. There can be exactly one duration constraint in each section, that specifies the duration in time for which the section should hold. In addition, there can be triggers. The first trigger of the destination or variable signals is used to align the signals in time. All other triggers then are simply constraints and must occur within the defined section window to get a match.

There can be added multiple expression constraints. One often used, simple approach would be to use $0 = \mathrm{iVar} - \mathrm{iDst}$. For all other identifiers an entry will be added, which can be a parameter of a circuit element, an evaluation of a result probe or the time, when a trigger occurs (change using the ↻ -button).

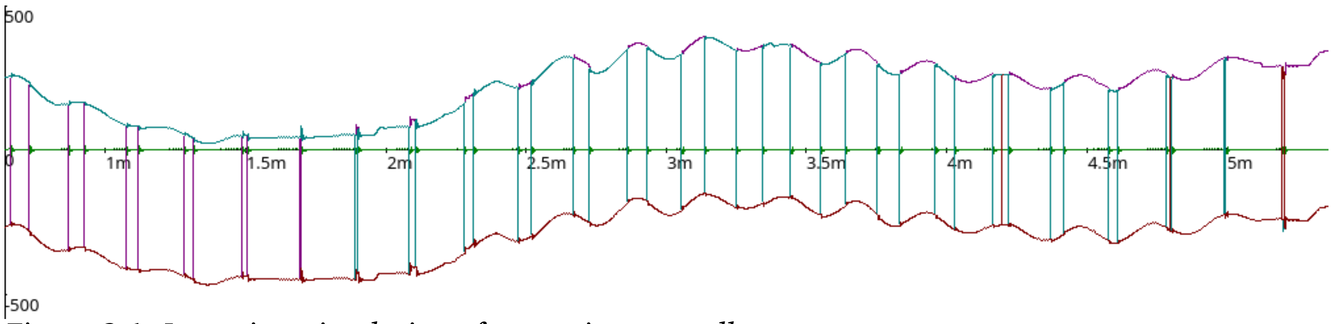


Figure 3.1: Long time simulation of an engine controller

To match a signal part in a complex scenario, where multiple transistors can switch (like in Figure 3.1), several constraints are required. In this scenario, the ringing after different transistors switching was measured with totally 32 oscilloscope exports for 3 different motor frequencies to find plausible values for the cable capacitance and inductance. While the simulation only needs 3 runs for the different frequencies, there can match multiple oscilloscope measures for each simulation.

Let's create a window of 18μs, beginning 4μs before channel 3 (one phase's voltage) rises:

- duration = 18E-6
- Variable Trigger | ch3 rise | Shift: `-4E-6`
- Destination Trigger | ch3 rise | Shift: `-4E-6`

This causes 27 matches – most of them are wrong, because for example a situation, where in the simulation only channel 3 rises, while channel 1 and 2 remain low, can match an oscilloscope measure, where channel 1 is already high and channels 2 and 3 rise at the same time. It occurs seldom, that multiple channels switch at the same time, but this causes very strong ringing, so this events can easily be triggered by the oscilloscope.

To exclude the wrong matches, several constraints can be added. The compare operator can be changed by clicking on it.

- 0 < minimum(dst.ch1)
- 0 < minimum(dst.ch2)
- 0 < minimum(var.ch1)
- 0 < minimum(var.ch2)

The minimum values of destination and variable result's channels 1 and 2 within the defined window must be larger than 100.

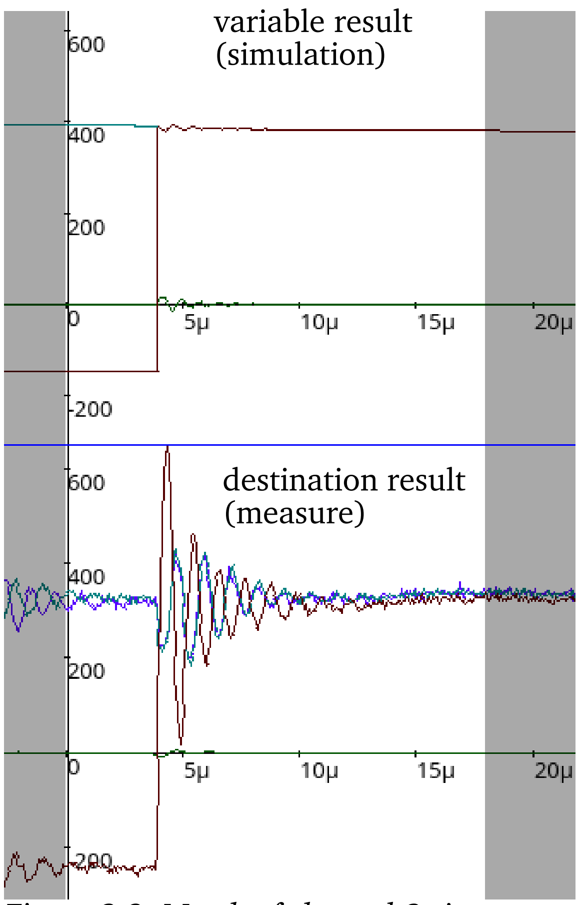


Figure 3.2: Match of channel 3 rise

To find the correct match as shown in Figure 3.2, the algorithm tests all combinations of variable and destination results. For each pair, the first variable trigger and the first destination trigger are taken. If one of the signals causes no trigger, this means no match. If the signals trigger, the other constraints are checked. If one fails, the next occurrence of variable trigger is tested, until a match is found or no further trigger in the variable result is possible. In this case the next occurrence of destination trigger is chosen and the variable trigger search starts again from beginning.

There are several optimizations to speed this up for not testing further triggers, if, according to the constraints, a match can never happen in the regarded pair.

3.2.4 Regard differences

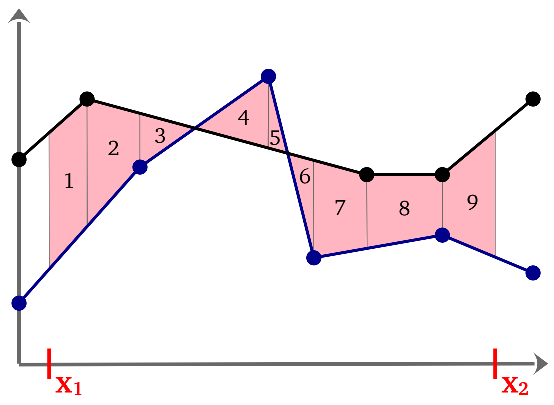


Figure 3.3: Summation of trapezes to integrate difference between two signals from x_1 til x_2

In the Difference and History -tab the previously selected matches can be compared. There is one supported method for difference measure: The *integrated difference*. This means, that the area between two signals gets summed. For signals with scattered x-axis steps, the algorithm needs to branch from point to point, to sum the area of all the trapezes, as shown in Figure 3.3.

There are not many test cases to verify the correctness of this summation. One thing, that was tested, is the result when both signals are the same. By default, the sum isn't zero then, as there can be rounding mistakes. With some extra if-cases, the function now returns zero for identical signals.

For each linked signal from the Probes -tab and each match from the Matches and Constraints -tab, one such difference can be computed. This results are displayed in the table at the bottom of the Difference and History -tab. When starting a simulation from this or the Matches and Constraints -tab, a tuple of simulations for all indexes of variable and destination model is done to allow full comparison.

The difference table and the current patch for the unknown parameters (which can also be edited on the Difference and History -tab) is then stored in the so-called history. The cell background of the difference table indicates, weather this value has increased (red) or decreased (green) since the previous history entry. Using the track bar, the history entry to show up, can be chosen.

There is a weighted result of each difference table, which is displayed in the chart. The idea is to compute one real average number of each table. Because the signals can have different physical units or simply be of totally different range, there are weighting factors for probes and matches, as described before. The sum is then computed as

$$\sum_{s=1}^{n_{\text{sections}}} \left(\sum_{m=1}^{n_s} \sum_{p=1}^{n_{\text{probes}}} \text{diff}_{p;s,m} \cdot \text{probeWeight}_p \cdot \text{sectionWeight}_s - \text{MCF} \cdot n_s \right)$$

n_s is the number of matches of the current section s. *MCF* is the match count factor, which can be set on the Difference and History -tab as well. This could be necessary if an extremely wrong patch can prevent some constraints from allowing matches. If less matches are found, there are less signal differences to integrate. This could mean, that completely wrong patches get a low (zero, if no match is found) total difference. By setting a large value for *MCF*, comparisons with less matches get downrated.

The weighting factors can be changed later and the chart gets updated with the new weighted total results.

3.2.5 Start auto tuning

Finally on the Auto Optimize -tab, a simulated annealing can be started to auto tune the unknown patch parameters. There is no range to be specified for the parameters, but they get changed in each step by a multiplicative factor:

$$f = a + b \cdot \frac{T_{current}}{T_{start}} \quad x_i' \in \left[\frac{1}{f} \cdot x_i \quad \dots \quad f \cdot x_i \right] \text{ while } p(x_i' < x_i) = p(x_i' > x_i) = 50\,\%$$

T is the virtual temperature of the algorithm, which should initially be roughly in the range of the total difference.

f is meant to be slightly greater than 1. For example 1.3 means, that the unknown values x_i get changed by 30 % for each comparison. The idea behind the part *b* of the factor *f* is, that for low temperature, we assume to be near the global minimum and do only small steps.

The parameter for cooling speed describes, how many temperature steps were taken for the annealing. The temperature is then linearly decreased from T_{start} to 0.

The simulated annealing can be stopped and continued in an other launch of the program or after changing some parameters manually. This makes it very usable to supplement human actions of tuning the parameters. To reset the algorithm to T_{start}, use the Reset Current Temperature -button.

3.3 Synchronizer results

The engine controller scenario is not suitable for a simulated annealing, as one simulation needs up to halve a minute to finish. As the true parasitics are even unknown, we cannot verify the results. For a proof of concept, the SRC-circuit was chosen and instead of oscilloscope measures, the same simulation model was set as destination. Then there was a patch-driver added to modify three parameters:

Table 3.1: SRC synchronizer destination parameters

Parameter	Destination value	Description
L_1	1.5 mH	Transformer’s main inductance, projected to primary side
L_R	5 μH	Transformer’s stray inductance, projected to primary side
C_R	85 nF	Resonant tank capacitor

Several runs of the simulated annealing were done with this destination parameters but different start parameters, temperatures and step change factors *f*. With good parameters, the global minimum can be found. Two interesting results are the following:

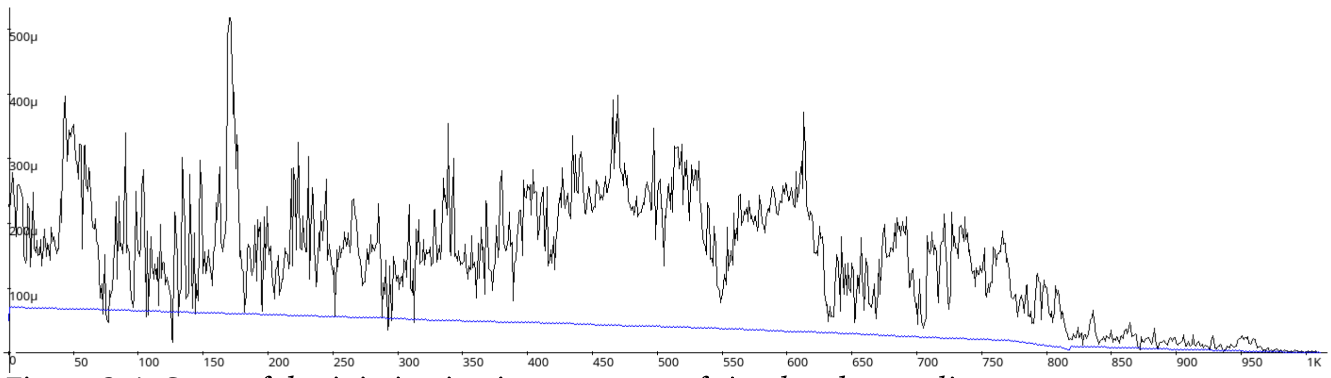


Figure 3.4: Successful minimization in two passes of simulated annealing

A run with $T_{start} = 70\mu$, $a = 1.3$, $b = 0.1$ was done over 3000 temperature steps and finally looks like having found the correct value combination.

A refinement minimization with $T_{start} = 10\mu$, $a = 1$, $b = 0.2$ was then started to further explore the minimum.

Parameter	start value	value after first minimization	value after refinement
L_1	0.8 mH	1.4799 mH	1.5047 mH
L_R	1 μH	4.0857 μH	5.0042 μH
C_R	200 nF	84.104 nF	84.72 nF
weighted result	228.72 μ	20.862 μ	0.5005 μ

Table 3.2: Successful minimization in numbers

It seems, that only for the lower temperatures in the right hand side of Figure 3.4 a true minimization happens. But prior to minimization it is not easy to decide, which start parameters are “good”. There is a short extra log on the Auto Optimize -tab, that prints for each comparison, weather a result was accepted or rejected and with which probability.

Starting with low temperatures will not work, as there *are* local minima. Figure 3.5 demonstrates, that a run, starting from the same parameters (Table 3.3) with $T_{start} = 1\mu$, $a = 1$, $b = 0.2$ stucks in a local minimum. Only 44 of 1000 steps were accepted. For steps 0 to 6, it seems, the minimization asymptotically converged to an even worse minimum and just jumped to the final minimum by a good random step.

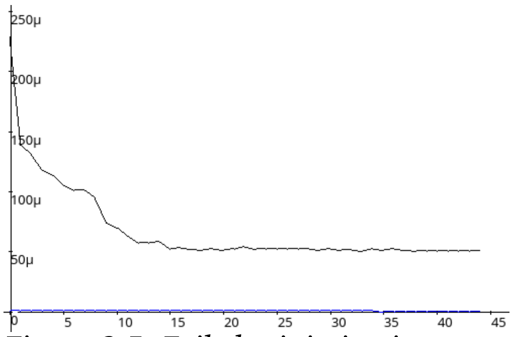


Figure 3.5: Failed minimization

Parameter	start value	final value
L_1	0.8 mH	0.9818 mH
L_R	1 μH	3.1929 μH
C_R	200 nF	133.84 nF

Table 3.3: Failed minimization in numbers

We can say, this minimization failed as we know the true result and that a much better approximation of it can be found.

But by just comparing the signals (Figure 3.6), many engineers might say, the simulation was close to measurement and that the chosen parameters are not that bad. We can ask the question, in how many cases, the model accuracy is not limited by topology but simply by inaccurate parameters...

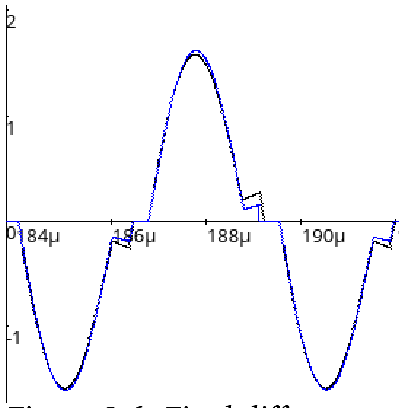


Figure 3.6: Final difference

4 Conclusion

The software toolchain could demonstrate, that it can solve several helpful analysis jobs with a comparatively low effort of configuration. The simulations may run for hours or days, but this happens without further assistance.

None of the results is spectacular or necessarily better than a Matlab script's result. But it can be created with lower effort. An other important benefit is, that this toolchain can run simulations in parallel. Matlab supports a lot of ways for parallelization, but it still costs extra effort to take use of it in each new script. There is a helpful GUI to explore results and interactively start further simulations at certain points. This brings human experience together with the systematic analysis of computer algorithms.

The software can be extended by extra drivers for special purposes while not being bound to a special simulation software, as the tool support interface is very generalized and interaction with other simulation software (than Plecs or xFemm) can be implemented.

4.1 Problems to be solved and outlook

Regarding the Plecs tool support, the opportunity of parallelization and a faster, string-conversion free data exchange should be investigated, as both together could allow 12 times more simulations within the same time even on a common PC.

It turned out, that when doing long time simulations (more than a few hours), the program gets very slow. The cause of this slowdown was not investigated. Possibly a loop to clear past log entries or queued tasks is not working as expected... For common use cases, the software has no unfreed memory blocks.

The interpolation of the switching state chart can cause artifacts of the interleave halving in multiple dimensions, if a signal changes rapidly in one dimension but not in the others. Generally causes adding of more dimensions a higher amount of points that need to be simulated in the existing dimensions, because the algorithm then knows, if a sequence change in the previously unknown dimension is near. One could think about concepts to make the decision, weather a simulation is required, smarter.

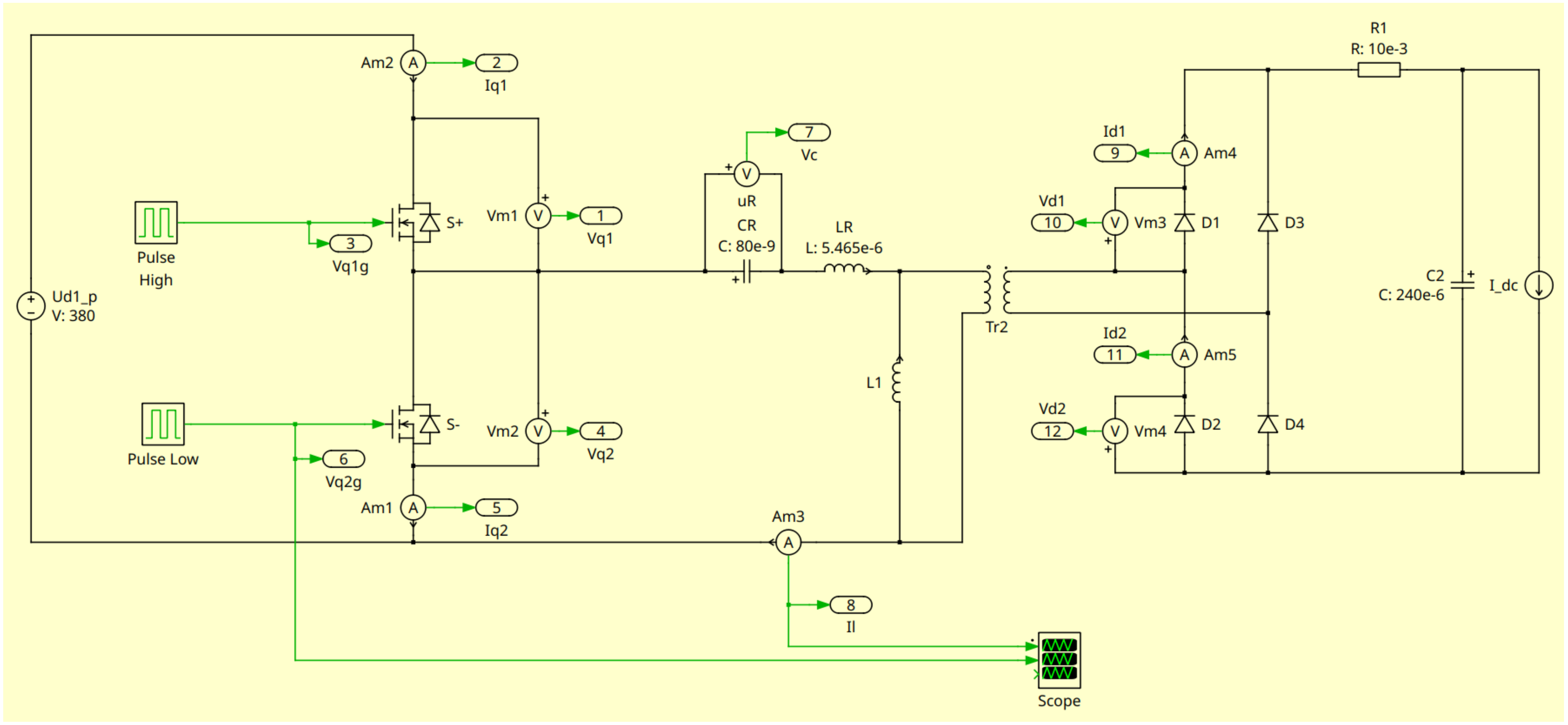
The simulation synchronizer offers many opportunities to become faster. The tested simulated annealing algorithm is easy and applicable to arbitrary problems, but also very slow and not suitable for simulations with a longer computation time. The chosen difference measure is not very smart. By applying Fourier analysis or other orthogonal transformations, the evaluation of peaks in frequency space would be possible. A minimization algorithm can take advance of which matches or which characteristics of the signals are affected by which parameters. By using the concept of modifying known simulations for destination model, it could be possible to generate training data for some machine learning approaches.

With the current work, a basis for such approaches is generated. But much more and different models are required to continue algorithmic research.

Appendix

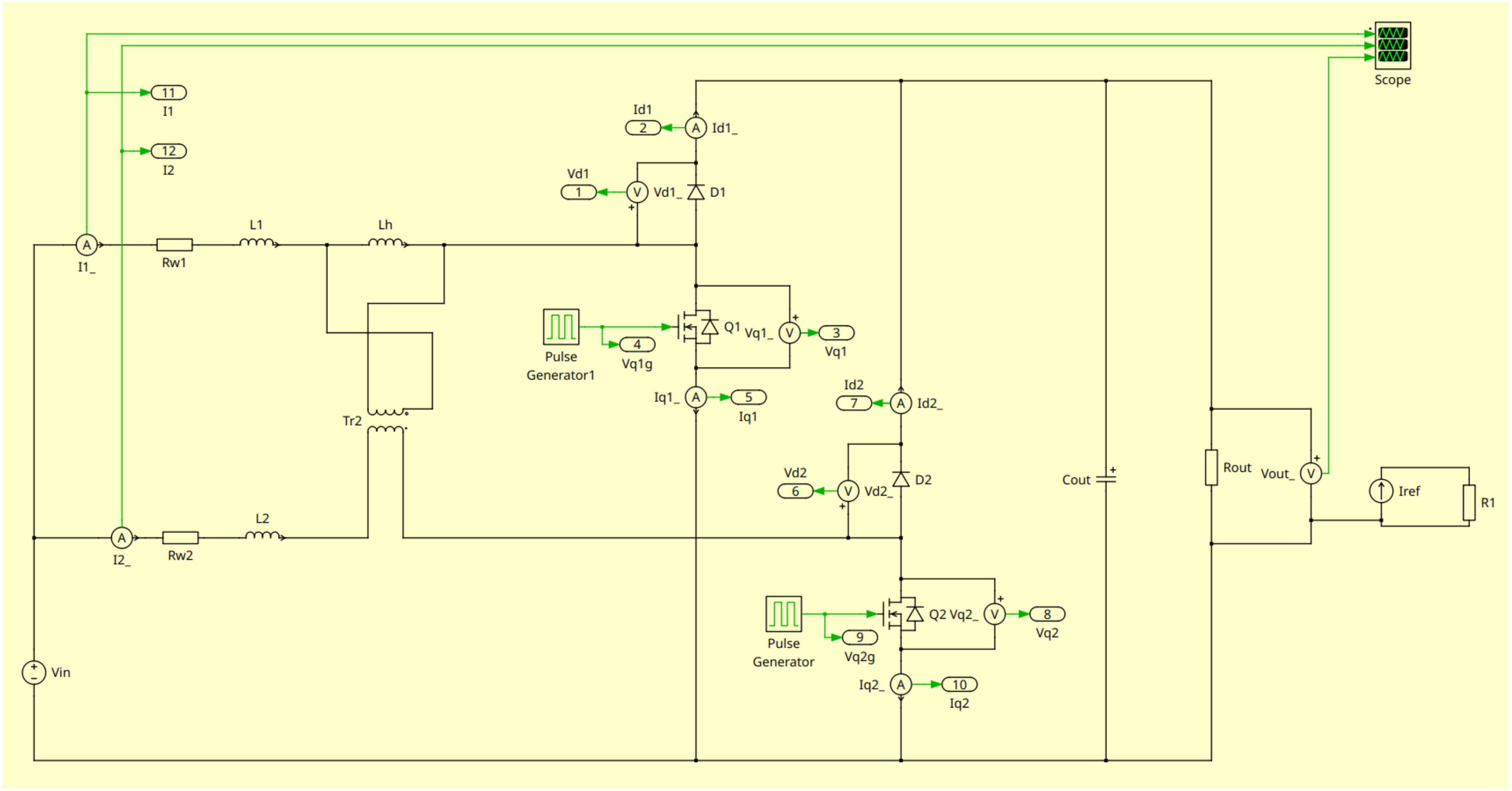
Models used in simulation

A1 Series resonant converter



This is mainly the circuit of my bachelor thesis[4], reconstructed in Plecs with some simplifications – there is no parasitic capacitance at the transistors, as this is not provided in Plecs without some tricks.

A2 Coupled inductor boost converter



Rw1, L1, Rw2, L2, Lh and Tr2 represent the inductance with partly coupled magnetic flux and winding resistance. For the transistors and diodes, the related currents and voltages are given to output. The “useless” Iref and R1-block is used to receive a targeted output current. The resistor Rout is then chosen by a patch to best fit this current. A model with current sink load would have a long oscillation phase from startup. Note that the x-axis for output current of this model uses the desired current Iref, while the true average current through Rout can be slightly different.

Bibliography

[1] Plecs Forum: Run multiple simulations in parallel using XML-RPC: https://forum.plexim.com/874/run-multiple-simulations-in-parallel-using-xml-rpc (called in avril 2019)
[2] Lazarusforum: Rechenfehler mit Double-Gleitkommewerten: https://www.lazarusforum.de/viewtopic.php?p=134826#p134826 (called in avril 2023)
[3] Source Forge page of xFEMM-project: https://sourceforge.net/p/xfemm/wiki/Home/ (called in 2023)
[4] Stachowiak, M.: Design of a compact 200 WDC/DC converter, 2017 (https://mitjastachowiak.de/projects/netzteil/)