Bibliographical report for the real-time digital simulation of power electronic systems

Pierre Haessig

February 2009

Contents

Abstract

This document describes the real-time digital simulation (RTDS) project at MIT that targets power electronic systems. After a short description of the project, we will give some notes on related issues, mainly about digital circuit simulation. Although a lot of work has been done on digital circuit simulation, there are no algorithms completely suitable for real-time simulated switching circuits. Section 4 gives more details on how we'll organize our tool to make real-time simulation possible.

1 Introduction

When designing power electronic systems, especially those dealing with high power, simulation is a helpful design, analysis and optimization tool.

However, the famous Spice and its modern successors are always in the *transient analysis* paradigm. The simulation is first run for some time on a computer and then results are displayed to the user. This prevents real-time interaction between the simulated circuit and some other real system (e.g. control hardware or software). This real-time interaction is the heart of Hardware In the Loop (HIL) paradigm.

Our aim is to achieve real-time simulation to enable HIL simulation for power electronic systems. We want to reach that goal by using both carefully selected algorithms and a dedicated hardware (FPGA or even ASIC) for doing the real-time computation.

While expensive real-time simulators already exist for big slow-dynamic power systems like power grids, our new tool will enable simulation of fast switching circuits involved in static power converters.

We believe our tool can be very useful to engineers developing today more and more complex power electronic systems.

2 Presentation of the project

2.1 The team

To work on our emerging project, we are, for the software part a team of three people. Ivan Celanovic, my research advisor, is a Research Engineer at the Institute for Soldier Nanotechnologies. Shireen Warnock, an undergraduate

student at MIT EECS department recently joined our group. I am a visiting student at the Laboratory for Electromagnetic and Electronic System for a six month research internship.

Another team in Serbia has already started preliminary work on the hardware part.

2.2 The simulator

We aim to build the fastest digital circuit simulator in the world. Real-time simulation is already achieved for instance by RTDS Technologies (`www.rtds.com`) from Manitoba, Canada. However they are targeting large power networks with slow time constants. An example use of RTDS simulator is the closed-loop testing of protective relays.

We want to simulate power electronic circuits, that means systems with smaller time constants. For example, wind-turbine inverters are internally switching currents at several kHz. For these circuits, we believe we need to update the calculation every *microsecond* to achieve near real-time.

We'll give more details on our implementation in section 4 because we want now to introduce the key expression of *Hardware in the Loop (HIL) simulation*. HIL helps getting a good understanding of our goal. [4] gives some more views on this special kind of simulation.

HIL is often used to test a controller of a big hardware like a car, airplane or a wind-turbine. Figure 1 gives an image of such a HIL application.

Not only is a wind-turbine expensive and big, but also it's impossible to put it in some *failure modes* to see how well the controller is reacting. And indeed these failure modes are of the most critical modes, where an improper behaviour of the controller can have catastrophic consequences.

In addition, more and more of the control software is being automatically generated, thus giving rise to necessary automatic testing procedure of the same. This is only achievable with HIL.

We believe that our simulator can help engineering companies developing faster some more reliable power electronic devices like wind turbines, hybrid electric vehicles or airplane power systems. This may help solving the current issue of sustainable development.

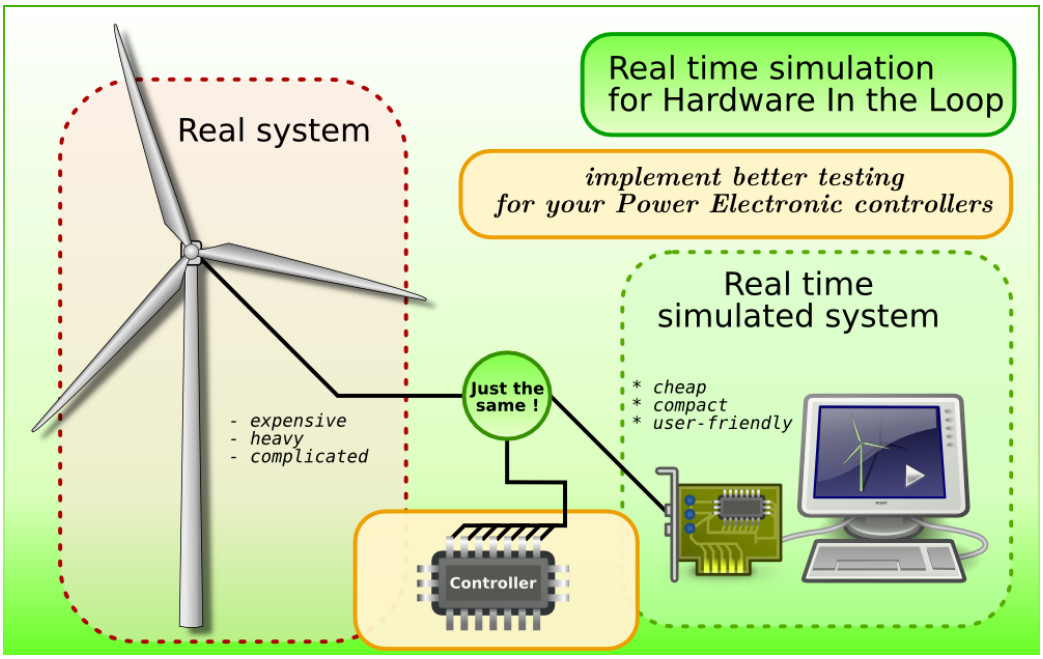

Figure 1: A targeted possible HIL simulation

2.3 A project extension

This is a special section dedicated to a possible further project that could completely change simulation software state-of-the-art.

We have started our project using Matlab for computation and Simulink as graphical interface. While these tools are relatively convenient, they suffer of some intrinsic problems

- Cost: a Matlab license is not affordable for everybody. While acceptable for big northern engineering companies, it can prevent the diffusion of our tool in other places like schools (indeed our simulator could be very useful for learning power electronic)

- Programming versatility: while Matlab is great for matrix computation, Matlab language is not adapted for real software programming For instance Matlab object implementation can be qualified at best as "simplistic". GUI programming and network communications are limited and/or unpractical.

- Moderate GUI quality: Simulink is not a great tool for drawing a circuit. We identified two main tracks for improvements:
  - *Better rendering engine (look):* we believe people deserve a more appealing interface not only because it can make electronic nicer to them, but also because a bad drawing is often ambiguous about what is connected and what is not.
  - *Better user experience (feel):* we believe a better ergonomy will protect people from wasting time on unnecessary repetitive tasks. For instance too much time is spent on clicking everywhere on the screen to reach common functionalities.

For these reasons we would like to create a new software, for systems and circuits (but not only electrical ones) modeling, somehow like Simulink. A modular architecture along with good design patterns would enable people adding their field-specific engines. On our side we would provide our engine for power electronic simulation.

We think it's possible to create a software using the highly flexible and powerful Python language. We can take advantage of some powerful libraries like

- Numpy/Scipy (`www.scipy.org`) for linear algebra computing,
- matplotlib (`matplotlib.sourceforge.net`) for data plotting
- QT widgets (from Trolltech, now Nokia, `www.qtsoftware.com`) to build a high quality GUI. [8] gives a comprehensive introduction on GUI programming using PyQT (`www.riverbankcomputing.co.uk/software/pyqt/`), the Python QT binding.

Our tool would not only be *open source and multi-platform* (Windows, Linux and Mac OS at least) but would also introduce new functionalities thanks to a good consistent core design. We especially have in mind introducing networking and versioning capabilities. These would be a huge help for *collaborative* work inside designing teams of scientists, students or engineers.

3 Notes on digital simulation

In this section we will give a brief overview of digital simulation of electrical circuits. By electrical circuit we mean a network of components governed

by Kirchhoff's circuit laws (KCL and KVL). This means we are interested in the same kind of simulation that for instance Spice has been doing since the 1970's ([3]). There are lot's of paths to simulate circuits and we want to choose the more appropriate ones to reach real-time simulation.

3.1 Nodal analysis

Speaking of Spice, let's introduce here nodal analysis, which can be seen as the most natural way of generating equations of a given circuit. Reference [5] gives a good introduction to the subject while more specific details on Spice's internals can be found in [3].

The basic idea is mixing the circuit equations (KCL and KVL) with the components equations (v-i relationships). For computing simplicity, everything is represented using matrix formalism.

3.1.1 simple DC analysis

An arbitrary network of resistors and constant current sources can be modeled as:

$$G\underline{v} = \underline{i}, \quad \underline{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} \underline{i} = \begin{pmatrix} i_1 \\ \vdots \\ i_n \end{pmatrix} G \in \mathcal{M}_n(\mathbb{R}), \tag{1}$$

where n is the number of nodes, without counting the ground. The unknowns are the nodes voltages $\underline{v}$ while $\underline{i}$ is the known source vector (full of zeros except except whenever there is a current source connected at the given node). G is the conductance (square) matrix which characterizes how resistors are connected between nodes. Briefly, the addition of a resistance R between nodes i an j of a network requires changing 4 coefficients of G:

$$G(i,i) \mathrel{+}= 1/R, \quad G(i,j) \mathrel{-}= 1/R, \quad G(j,i) \mathrel{-}= 1/R, \quad G(j,j) \mathrel{+}= 1/R \tag{2}$$

Then the solution is simply obtained by inverting the system, for instance using the Gauss-Jordan reduction. It's fairly straightforward. However a problem arises whenever G is *ill-conditioned* which can occur when resistors have really different orders of magnitude.

Also note that this method can be easily transposed to perform AC analysis in sinusoidal steady state by replacing the conductance matrix G by a complex admittance matrix Y where some $j\omega C$ and $1/j\omega L$ terms are handled.

3.1.2 Transient analysis

Most power electronic circuits contains capacitors and inductors whose v-i relationship is a differential equation. These circuits are studied during a certain period of time so that the simulation is run at different time steps, and in practice an iterative way: one time step after another.

Like in the previous section, there are two issues: how to get the equations modelling the circuit and then how to solve them. Numerical solving of these equations will be further covered in section 3.2.1.

Here we want to introduce quickly a method that enables a natural extension of the nodal analysis: *companion models*. The idea is to replace at each time step a capacitor (or similarly an inductor) by a current source and a resistor which will give a good approximation of the capacitor's operating point on the next time-step.

For instance at instant k, we know (v_k, i_k) from previous calculations. We also know an ideal capacitor obeys $i = C\frac{dv}{dt}$. Thus, calling h the time-step, and using Euler's backward approximation leads to:

$$i_{k+1} \approx C\frac{v_{k+1} - v_k}{h} = (\frac{h}{C})^{-1}v_{k+1} - \frac{C.v_k}{h} \tag{3}$$

We can here recognize the resistor $\frac{h}{C}$ and the current source $\frac{C.v_k}{h}$

This method, has the advantage of simplicity, and is quite protected against numerical instability thanks it implements a *backward* algorithm, rather than a *forward* one. However, good accuracy requires a time-step belows the smallest time constant.

3.1.3 Nonlinear networks

Whenever some nonlinear elements are involved, there is no more direct analytic solution, so that a search algorithm is needed. Finding the operating point for each nonlinear part can be expressed as finding the solution of an equation like $f(x) = 0,\ x \in \mathbb{R}$.

A popular search algorithm is Newton-Raphson's one, which tries to approach the solution iteratively using relation (4).

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \qquad k = 0, 1, \ldots \tag{4}$$

Of course this requires a starting guess x_0, which is a particularly crucial issue with Newton-Raphson, because the convergence is quadratic when close to

the solution, but can be extremely slow otherwise. [5] gives a self-speaking example for a diode

Regardless of this convergence problem, a key advantage of this algorithm is its ability to be easily plugged in a simple nodal analysis solver as presented in 3.1.1. Without getting into much details, a nonlinear element characterized by $i = g(v)$ can be replaced at $k+1$ step by a conductance $g'(v_k)$ in parallel with a current source $i = g'(v_k)v_k - i_k$. Please note that unlike the case of transient analysis, these iterations are made within the same time-step, and have to be *repeated* each new time-step.

This leads to a new approached bias point (v_{k+1}, i_{k+1}). Of course a control algorithm stops the iteration whenever a given criterion indicates that enough precision has been obtained.

A simple summary of this nonlinear analysis could be: nonlinear characteristics enable high precision analysis, but are very time-consuming. Moreover, like every iterative algorithms, the computation time is not predictable, which doesn't meet the requirements for real-time.

3.1.4 Piecerwise Linear networks

Some components have characteristics that are (or approached by) segments: Piecewise Linear (PWL) characteristics. The most famous are probably the idealized diode models. Circuit with such components don't need as a complex solver as describes in section 3.1.3. Indeed, while the circuit stays in the same "linear configuration", only a simple linear solver is required.

However PWL elements must now be monitored to check the bias point is still in the same linear subdomain (for instance, a conducting diode simply has to have $i_D \geq 0$). Whenever a boundary is crossed, a new task appears: determining the new linear state. A algorithm is needed to find it automatically.

[6] describes a method for representing PWL elements as a resistor in parallel with a current source. One specificity of their approach is their use of fixed resistance value regardless of the state. This way they end up with only one conductance matrix instead of 2^n if there are n switches.

Then their main contribution is to use a modified version of the Katzenelson algorithm to determine the new switch configuration whenever a boundary is crossed. However, even if efficient, this method is still iterative: changing one switch after another until a stable configuration is reached.

So the computation time is *not predictable*, which is not suitable for real-time simulation.

3.2 State space simulation

Now we'll move on to another way of simulating a circuit, by the use of state space equations.

State space equations are not restricted to circuit simulation. They can handle other kind of systems which can be modelized using *linear differential equations*. As a consequence, they can only deal with linear circuits (or PWL ones with some adaptations).

$$\underline{\dot{x}} = A\underline{x} + B\underline{u} \tag{5a}$$

$$\underline{y} = C\underline{x} + D\underline{u} \tag{5b}$$

For an electrical circuit, the state variables are related to the *energy storing* elements. So people usually choose voltage across capacitors and current through inductors because these variables are continuous.

Getting these state space equations automatically is trickier than nodal analysis ones. An algorithm is detailed in the reference book [2] but the same structure is described in several others like [7] and [5]. It uses some topological analysis and basically involves:

- Topological analysis to separate components in the *tree* from components in the *link*
- getting an incidence matrix with tree and links elements separated
- taking 16 slices of this incidence matrix
- introducing the v-i relationship of capacitors and inductors while eliminating undesirable variables using the slices

The words *tree* and *link* (or *cotree*) come from topological analysis of networks. They qualify the branches (ie the components). A tree is a subset of the branches connecting all nodes while containing no nodes. The link is composed of the remaining branches. [5] details the basic topological concepts needed.

Also topological analysis can reveal some properties of the circuit. We can point out that we should always be able to get a tree without current

sources and a link without voltage sources because otherwise there would be nodes of current sources or loops of voltage sources, thus violating KCL or KVL respectively.

The state space approach is used for instance by Plecs toolbox: `www.plexim.com`. They bring fast and accurate power electronic simulation to Simulink. They describe partly their method in [1], especially how they deal with switches. Their use of a detection of delta impulses to detect unstable switches configurations is an efficient solution. However it is iterative like Katzenelson algorithm in section 3.1.4. So, even if short, the computation time is not predictable, which is not suitable for real-time processing.

3.2.1 Solving state space equations

Here by solving we mean computing an acceptable numerical approximation as there cannot be an analytical solution when the input $\underline{U}$ is unknown.

The problem is part of the more general numerical integration issue. Again [5] and [7] give a good overview of the subject. We are mainly interested in fixed-step algorithms because our output is a fixed-frequency discretized signal. Of course some good algorithms are using "smart" variable steps but a real-time simulator cannot afford casting such time-slowing magic.

We want our state vector to be computed this way:

$$\underline{x}_{k+1} = \tilde{A}.\underline{x}_k + \tilde{B}.\underline{u}_k \tag{6}$$

with $\tilde{A}$ and $\tilde{B}$ depending on A, B and the discretizing algorithm.

Also we cannot afford implicit algorithms (that means ones which involve not already computed values) because they require a search algorithm.

This mainly let us with some of the simple integration algorithms, like Euler Forward Algorithm (EFA), or the higher order Runge-Kutta ones (RKA).

These algorithms are not robust when dealing with *stiff* systems: whenever there are time constants smaller than the time-step the risk of numerical instability is high. However [7] presents an interesting approach with an accurate approximation of matrix exponential that could be a great help to fight this instability. Moreover, Matlab provides an even better algorithm with its expm() function.

3.3 Power electronic circuits specificities

In this section we want to summarize the major issues related to our specific simulation problem. The two keywords are *switches* and *real-time*.

A power electronic circuit, for what we are interested in, is a network of linear elements plus switches (including internally controlled switches like diodes). So the system is piecewise linear and can be described as a *set of state space equations*. We have excluded nodal analysis because we want to add some interactions with a state-space-modeled mechanical system. So it's easier to be already in state space domain.

Then we had to choose how to deal with changing topology. Until recently community preferred "single matrix" approaches, that means a single equation is valid for all topologies, regardless of the switches configuration. This was mainly in order to save memory space.

We believe we now have largely enough memory. And indeed the most advanced Plecs toolbox computes and caches new matrices for each topology.

We want to go *further:* pre-compute in advance all the necessary matrices and then feed the memory of our hardware solver. This would largely increase our computational speed.

But there is still the issue of determining efficiently the new topology after a switching event (because for instance in a boost converter, opening the switch closes the diode).

3.3.1 Transitioning between configurations

We have described efficient approaches in sections 3.1.4 and 3.2 but we've seen that none of them is predictable in term of computation time.

We believe we can put more intelligence in our tool so that the transitions from one topology to another are summarized in an explicit *Finite State Machine algorithm*.

This means our simulator will have the knowledge to jump from one stable topology to the next stable one without wasting time on improper ones: the computation time will be completely deterministic.

The simulator knowledge will be gathered from a more extensive pre-analysis of the circuit where we will analyse what conditions on state vector makes a topology stable or not.

It is the important conclusion of this section to realize that the issue of switching between different topologies is the *key to reach real-time simulation*

and it explains why such simulation is not available as of today. Our FSM algorithm should make it possible.

4 Architecture of the simulator

In this section we describe our path toward real-time simulation of power electronics circuits.

4.1 System's structure

Most of the algorithms we'll be implemented in Matlab for the pre-computation, plus a hardware back-end stage actually running the real-time simulation.

This is because we have understood that our simulation will eventually run real-time only if the first stage of strong pre-computation and knowledge-gathering only let easy computation to the actual simulator device.

Figure 2 shows how we have structured our tool. From what we have just said, the real brain of the system is in the "Analysis" layer, while "Computation" stage is meant to be simple and fast.

Now we'll describe briefly these different stages.

4.2 Input GUI

In order to interact conveniently with the user, we want to provide a Graphical User Interface. Convenience also dictate us to use a tool which engineers are familiar with: that's why the whole program is Matlab-based and why the front end uses Simulink.

More precisely, we are using SimPowerSystems toolbox for drawing and connecting components (screen capture on figure 3). However, as we use SimPowerSytems blocks as empty boxes we'll provide our own toolbox to minimize the prerequirements for the user.

4.3 Netlist Extractor

To enable a good modelling algorithm, we need to access the circuit topological data a very simple way. But Simulink doesn't provide such "programming interface".

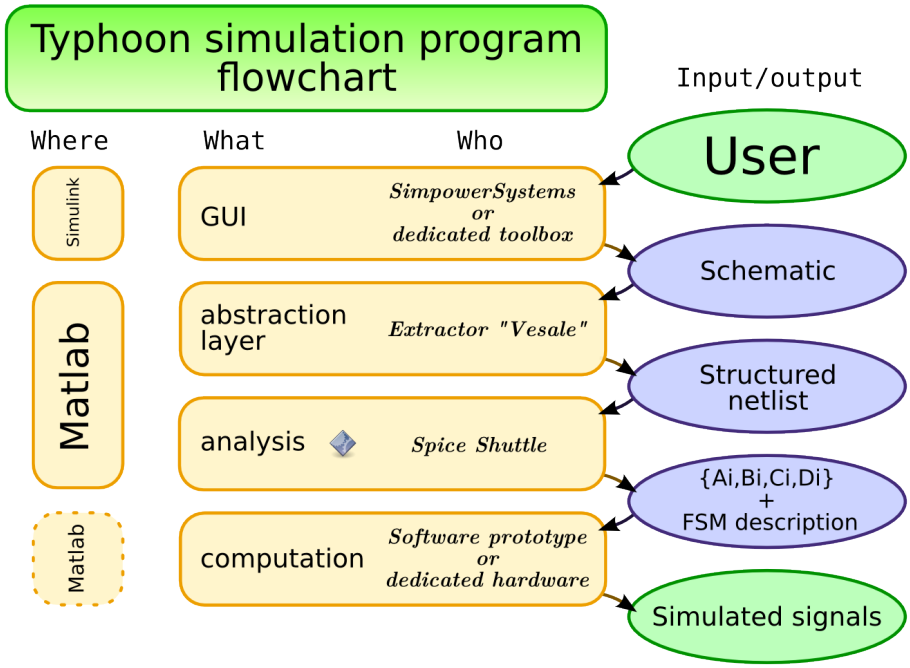

Figure 2: Simulation tool flowchart

That's why I've decided to create an abstraction layer which does the dirty job of *introspecting* Simulink and outputs a simple structure which summarizes all the needed informations the modeler need. This structure called "structured netlist" is implemented using a Matlab structure.

4.4 Modeler

This is the key layer that will enable real-time simulation. We have already discussed of what it should do and here is a summary:

This layer is doing a topological analysis of the circuit using the convenient "structured netlist". This analysis, repeated for each topology (that is each switch configuration), leads to a set of state space matrices. Figure 3 shows a graphical effect related to the tree-finding algorithm (note this is for debugging purpose only).

The second task of this layer is to anticipate the possible transitions between switches configurations. This analysis is then summarized in an

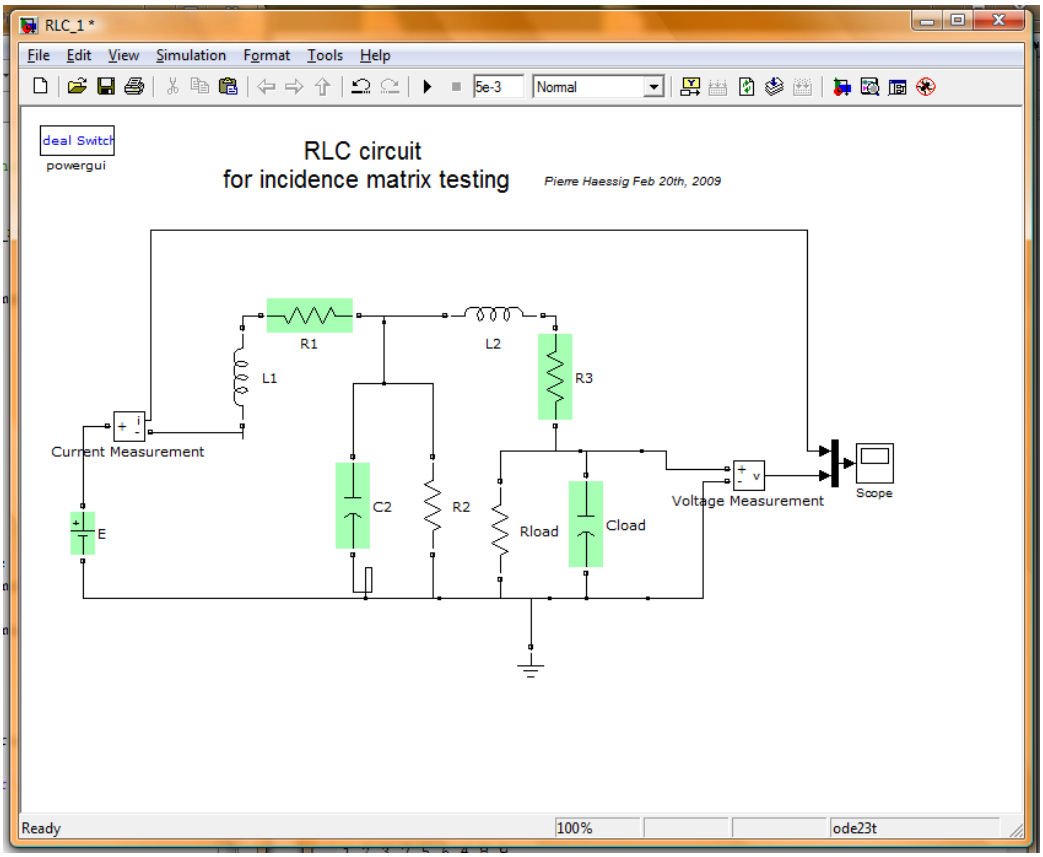

Figure 3: A circuit drown in Simulink get automated analysis from the modeling tool: components in the tree are green painted

FSM to be used by the solver.

4.5 Simulator

This layer runs the actual simulation.

Our project aims at running our simulation on a dedicated hardware. We are targeting FPGAs not because today computers lack computation power but because only FPGAs allow real high-speed low-latency communications with the real world.

Even people using computer-based computation HIL (like RTDS) are using FPGA dedicated cards for inputs and outputs. We want to go further

by porting also the computation on FPGA.

Manufacturers (Xilinx, Altera, ...) are providing some chips with around 50 integrated multipliers, running at several hundreds of MHz. This should be enough to run our simulation with reasonable circuit sizes. The natural increase in computation power over the years should overcome possible problems with bigger circuits.

Of course we'll first use software prototyping for both evaluate the simulation accuracy and the amount of calculation required. The latter will help choosing a suitable hardware, whose cost may vary from few hundreds to several thousands dollar.

5 Conclusion

After this introduction to the subject, I'll describe briefly my work.

A first global scope work is to help refining the architecture of our emerging tool as I'm getting more and more understanding of the subject. With Ivan we are defining how we want our layers to be organized to reach our goal: real-time simulation.

My first coding work was on the abstraction layer which looks into Simulink to get a structured netlist (cf section 4.3). This module is now working fine with SimPowerSystems toolbox. I'll now share the knowledge of low-level Simulink programming I gathered to help our new student, Shireen, to build our own toolbox.

Now I'm mainly focusing on the modeliser. I'm implementing the [2] algorithm to get the automated state space modeling. The purpose of doing it ourself is to get good understanding of the "inside" of these matrices. We believe they contain the information we need to build the switches configuration FSM. This FSM building is the second part of our modelling. I think it is the crucial issue because real-time simulation will only be possible if we are able to determine switches configuration using a time-deterministic non-iterative algorithm.

Whenever we have a good modeling algorithm which analyses *automatically* power electronic circuits, we'll be able to focus on studying, designing and scaling the hardware solver.

References

[1] J.H. Alimeling and W.P. Hammer. Plecs-piece-wise linear electrical circuit simulation for simulink. *Power Electronics and Drive Systems, 1999. PEDS '99. Proceedings of the IEEE 1999 International Conference on*, 1:355–360 vol.1, 1999.

[2] Leon O. Chua and Paul Y. Lin. *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*. Prentice Hall Professional Technical Reference, 1975.

[3] Ron M. Kielkowski. *Inside Spice: Overcoming the Obstacles of Circuit Simulation*. McGraw-Hill, Inc., New York, NY, USA, 1993.

[4] D. Maclay. Simulation gets into the loop. *IEE Review*, 43(3):109–112, May 1997.

[5] E. J. Mastacusa. *Computer-aided network analysis*. John Wiley & Sons, Inc., New York, NY, USA, 1988.

[6] P. Pejovic and D. Maksimovic. A new algorithm for simulation of power electronic systems using piecewise-linear device models. *Power Electronics, IEEE Transactions on*, 10(3):340–348, May 1995.

[7] V. Rajagopalan. *Computer-aided analysis of power electronics systems*. Marcel Dekker, New York, NY, USA, 1987.

[8] Mark Summerfield. *Rapid gui programming with python and qt: the definitive guide to pyqt programming*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.