ULB

ECOLE
POLYTECHNIQUE
DE BRUXELLES

beams

# Automated modeling and implementation of power converters on a real-time FPGA-based emulator

Ecole polytechnique de Bruxelles

2015

Promoteur : **FREDERIC ROBERT**
Co-Promoteur : **PIERRE MATHYS**

Thèse présentée
par **KEVIN DE CUYPER**
en vue de l'obtention du grade
de docteur en science de l'ingénieur
et technologie

## Abstract

Designing a new power electronic conversion system is a multi-step process that requires the R&D team(s) to go through an extended prototyping phase whose goal is to validate the design in its nominal state, as well as to test its behavior when it is subjected to abnormal conditions. To properly and safely validate all devices that are external to the power stage itself, such as the controllers and the protection systems, one of the best-suited device is a real-time emulator of the converter circuit, a platform that obeys the same mathematical laws and produces the same signals as the original device without actually realizing the power conversion.

Unfortunately, these models are often based on analog solvers which are difficult to build, must be redesigned for each modification and are subject to drift and aging. While multiple digital real-time emulators have appeared on the market in the last decades, they typically require powerful and expensive computing platforms to perform their calculations or are not generic enough to emulate the more complex power circuits.

In this work, we present a new framework that allows the rapid prototyping of a wide range of power converters by translating a power converter schematic drawn on a computer to a real-time equivalent set of equations which is processed by an FPGA with an emulation time-step of less than one microsecond. Contrary to the previously published works, our tools enable the use of entry-level FPGAs even for the emulation circuits composed of twenty switches or more.

This framework takes the form of a tool-chain that starts by extracting the necessary information and a standard description from the initial circuit. However, due to the intricate ways in which the switches and diodes can change their state, this raw information is too complex to be processed and emulated directly. Our first major contribution to the state of the art is a way to automatically analyze these changes in order to reduce the complexity of the problem as much as possible while keeping all the necessary information intact. In this thesis, we develop two tools that are able to find all possible changes in the state of the switches that may appear in the immediate future, thereby reducing the quantity of information required to emulate the circuit. Thanks to the global optimization provided by our tools, simulating a typical AC-to-DC converter composed of 12 switches could require 80% less resources when compared to existing emulators.

To enable the emulation or large power converters, we have created a partitioning method which divides the circuit in multiple sub-circuits which are analyzed and optimized separately. The performances of this partitioning are demonstrated by the emulation of a three-phase three-level converter with a relative error of a less that 5% on the signals.

To handle our new framework, a dedicated digital platform has been developed. In order to provide the best results even on small FPGAs, particular attention is given to the low resources usage and the low latency of our design.

Through multiple examples, we show that this inexpensive real-time emulation platform is able to accurately emulate many circuits in open- or closed-loop operation with a sampling rate higher than 1 MHz.

# Contents

# Chapter 1

# Introduction

## 1.1 Context

Imagine the following situation: as an engineer, you are in charge of the design of the power conversion system for a revolutionary electric car. After drawing your new circuit and performing the initial computations, you start making simulations on your computer to refine and optimize the structure and the components. After a few weeks, you have prototypes of the power converter and of all of its auxiliary modules, such as the controllers and protection subsystems, at your disposal. You are finally ready for the prototype run, everything runs smoothly . . . until the system fails. At this stage, it is not obvious where the fault comes from and the designers have to go back up to the simulation stage to understand what happened and wait for new prototypes to be built, costing you weeks in development. Because of the constant pressure on the development teams to deliver the final product as early as possible, this could mean missing you deadline.

This unwanted and expensive error could have been partially avoided by reducing the time to develop the prototypes, hence ensuring a much shorter development cycle. Before providing a more complete answer to this problem, let us first reintroduce the main steps required to develop a power converter.

The traditional design flow is presented in Figure 1.1. Based upon the specifications, the second step consists in designing and validating the converter and its auxiliaries (controller, protections. . . ) by paperwork and simulations, in normal and accidental conditions. The next step is the building of prototypes for the converter itself and for electronic controller. Most of the time those prototypes are built separately, by different teams, or even sub-contractors.

One of the most delicate parts is the integration of the two prototypes. Mistakes could have been made on both sides, and putting all things together, even carefully, can lead to bad surprises like in the previous example. The best response to this problem is to perform the most exhaustive tests possible on both prototypes before joining them. However, such tests require an accurate model of the devices connected to the unit under test (UUT). When this UUT is the circuit controller or the protection systems, we have to provide a real-time emulator of the converter circuit, i.e. a platform that obeys the same mathematical laws and produces the same signals as the original device without actually realizing the power conversion. By connecting the UUT to the emulator, we can easily and quickly tune the controller loop or assess the performance of the protections without

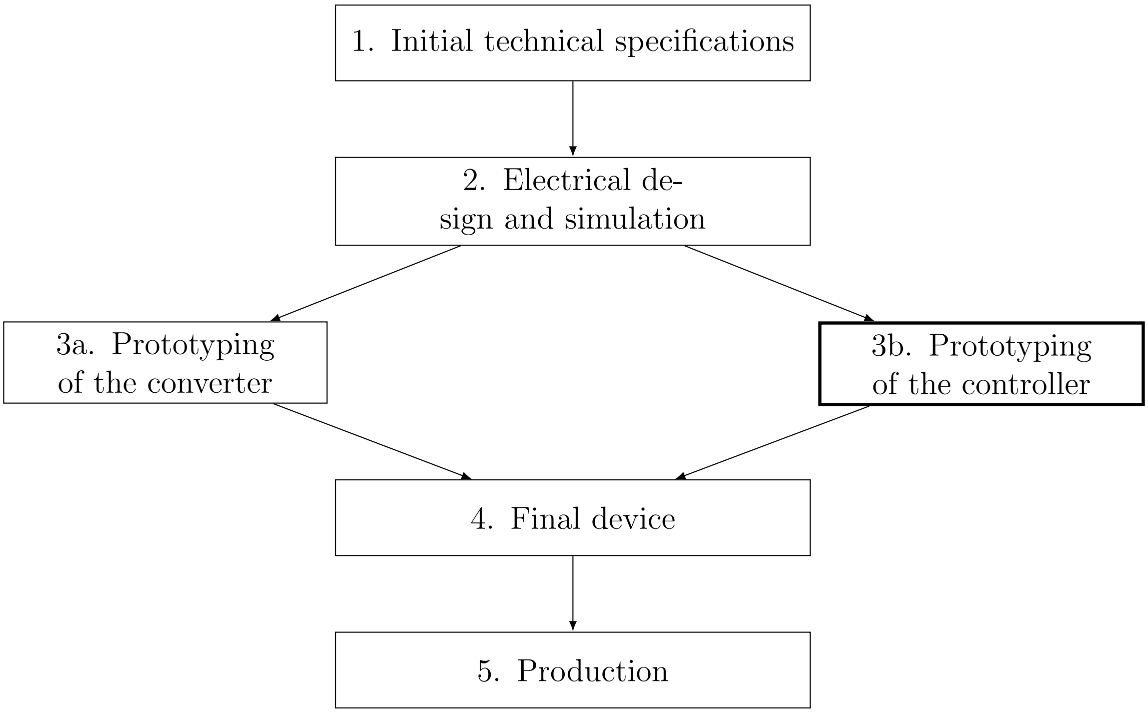

1. Initial technical specifications
2. Electrical design and simulation
3a. Prototyping of the converter
3b. Prototyping of the controller
4. Final device
5. Production

**Figure 1.1: The traditional steps required to design a power converter. The *prototyping of the controller* step is the main focus of this work**

facing any risks that could be engendered by an error not detected during the simulation stage or by a flaw in the UUT itself.

This thesis aims at helping the designer of the controller to produce a fully functional prototype without having the real power converter at his disposal. More precisely, our goal is to design a platform able to emulate in real-time a wide range of power converters and to provide the tools needed to quickly and conveniently implement any circuit on this platform, which significantly reduces the duration of the prototyping. With such a platform, we can indeed easily change some of the parameters of the system and view their impact on the emulated device without having to wait until all parts of the actual device are ready to be used.

Proposing an equivalent platform to a given power conversion circuit is not a novel idea. Indeed, analog circuit emulators based on operational amplifiers, multipliers and controlled switches have been around for decades and are still used today for the emulation of power electronics [1–3]. However, these circuits have the same downsides as the full-size converters: for each new converter, or even for each redesign and correction of errors, a new emulation platform must be rebuilt, sometimes from scratch. When the analog components of the platform get older, their electrical properties change, introducing an error between the simulator and the target power converter. These problems are avoided by the use of a *digital* real-time platform instead.

Using a computer to simulate electrical circuits is also a decades-long concept. Seminal works have been made in the sixties and seventies [4–7], introducing crucial methods for translating electrical circuits into algebraic or differential equation systems that could then be solved using any numerical method [8–10]. The application of digital simulation to power converters appeared in the eighties as a natural extension, just as the world was beginning the transition from linear converters to switched-mode power conversion systems [11–14].

Today, many commercial solutions are available for the offline computer simulation of power circuits such as the very popular *SPICE* [15] and the power-oriented *PLECS* [16] and *Saber*.

In the topic of this thesis, the last two decades have seen the rise of commercial *real-time* simulators dedicated to power electronics, with *RTDS* [17] and *OPAL-RT eMegaSim* [18] acting as the front runners. Thanks to their powerful hardware, these platforms are able to emulate a wide range of converters with time-steps as low as a few micro-seconds. Recent FPGA implementations have authorized sub-microsecond time-steps (see, for example, [19–22], other examples will be introduced further on).

Note: the works cited in this section are far from being a complete state of the art. A myriad of circuit simulators are available, commercially or not, and are characterized by their own forces and weaknesses. Instead of presenting all these works now, we have chosen to introduce them at various points in this thesis depending on where they provide a better illustration.

## 1.2 Why this work

Since digital prototyping solutions already exists, why have we chosen to explore this path any further? We shall answer this with the following two points:

- Most systems use traditional computers based upon microprocessor platforms and hence are limited to time-steps of a few microseconds or more. This restricts their use to the simulation of circuits characterized by switching frequencies of 10kHz-20kHz or lower

- They use the available hardware rather inefficiently. This is partially due to the use of the nodal analysis for solving the circuit, which is known for not providing equations close to the minimal representation circuit (this assertion will be studied with many more details in chapter 2). Combined with a lack of global study and optimization of the circuit, this leads to severe requirements for the processing power. The higher number of computations also typically leads to a higher latency, and hence forces us to use a larger time-step than needed, which in turn reduces the accuracy of the solution.

While FPGA implementations allow to overcome the first point, the second one typically remains a problem. Solutions have appeared very recently [23, 24] to get around this second limitation by splitting the circuit into smaller parts. While this idea is certainly interesting, it does not solve the core of the problem, which is that each part is suboptimal.

In this thesis, we provide a global solution to this problem by proposing a series of methods to improve the performance of the real-time platform. Thanks to this, we will show that we are able to emulate large power converters with sub-microsecond computing time-steps, even on low-cost entry-level hardware platforms.

The purpose of the proposed algorithms is to reduce the representation of the system thanks to a deep study of its dynamics *before* implementing it on the real-time platform. Hence, a global optimization is provided.

## 1.3 Structure of the work

This thesis is articulated around four main chapters. The first chapter introduces the problem of the simulation of power converters and provides the main requirements that will have to be met by our solution.

In Chapter 2, we introduce the different methods at our disposal to put electrical circuits into equations. The discussion will first be made for purely linear circuits before adventuring into the models for power-converters. This chapter provides the following points:

- introduction and comparison of the two main solving methods : the nodal analysis and the state-space analysis
- study of the models for the power switches
- introduction of the *hybrid automaton* paradigm, used to model the dynamics of the converter

Our first major contribution is presented in chapter 3. We introduce an automated offline method to reduce the complexity of the hybrid automaton. This study is based on the exploration of all possible transitions in the system, which can be due to the natural evolution of the signals or to a change in the command of the transistors. Thanks to our methods, we are able to only keep the transitions that are actually feasible by the circuits.
The provided solution also guarantees constant computation latency for each time-step, even when multiple diodes should be switched in sequence (which would typically lead to a recursive algorithm or to a transient extending over multiple time-steps), and avoids the use of compensation circuits in the model of the switches.
The performances, but also the limitations of the size of converters that can be handled by this tool are illustrated by a series of examples.

A solution to these limitations is provided in chapter 4, which introduces a novel way to partition larger converters using variable Thevenin/Norton sources. This partitioning further reduce the complexity of the system at the cost of some degradation of the accuracy of the simulation, and it will be shown to perform well in many different situations.

Finally, in chapter 5, the mathematical models provided by those methods are implemented on a custom-made real-time platform. This custom-made platform makes extensive use of the resources provided by the FPGA, since each part of the solver is studied and optimized. In this chapter, we demonstrate the performance of the simulator and of our extensive pre-processing by presenting the results of the real-time simulation of a medium size *AC to DC to three-phase AC* converter with a time step of 650ns. We show that, thanks to the implemented analysis methods, the emulation of this converter only requires a fraction of the resources of an entry-level FPGA while the relative error on the output signals, compared to a reference off-line simulator, is limited to about 1%.

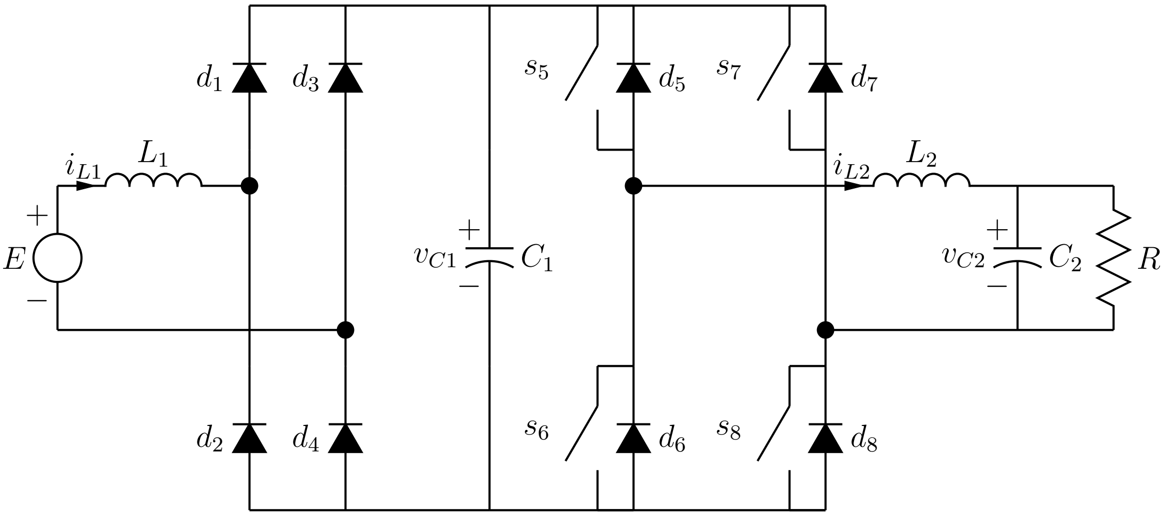


**Figure 1.2: A typical power converter composed of a rectifying stage followed by a full bridge inverter**

## 1.4 Problem analysis and Requirements

A typical power converter is represented in Figure 1.2. This circuit is composed of linear components (RLC passive components, sources), transistors and diodes. Depending on the application, these semiconductor devices can be controlled very quickly; in switched-mode power supplies, switching frequencies above 100kHz are quite common. If we want to accurately represent these signals, we must build a real-time emulator whose time-step is much lower than $10\mu s$. At the beginning of this thesis, the following objectives have been set for the real-time platform:

- a time-step of $1\mu s$ or less must be met. This value allows for a wide variety of power converters. This is also a requirement for the fastest control loops (such as sliding-mode control) which operate substantially faster than their controlled signals .

- the relative error, defined here as the ratio between the absolute error and the average value of the signal, must be kept under a few percents (typically 1%, maximum 5%). If we ever want to accurately represent the small variations in the signals and implement fast hysteresis current loops, such a requirement is unavoidable. The peak-to-peak variation of the signals must also present a similar error.

- there must be no limitation on the range of converters that can be represented by our solution. This implies that we must be able to emulate any existing topology by solving their equations in real-time. Of course larger converters might requires more powerful platforms.

- in line with the previous points, scalability and portability must be kept in mind. This means that the analysis and simulation portions of the work must be able to handle any circuit, and that the real-time solver must be usable with any future FPGAs without requiring major modifications.

# Chapter 2

# Circuit modeling

## 2.1 Introduction

Before developing a simulator of electronic power converters, we first have to answer the question of how to obtain a mathematical representation of these circuits. Obviously, the science of systematic circuit analysis is not a new subject, and our goal is not to develop a new representation but to study the existing methods and select the one that is most adapted to our usage.

Before studying power converters, we first take a look in section 2.2 at the two main representations of *linear* circuits: the nodal analysis and the state-space analysis. Since most power converters are, at their core, linear circuits used in combination with binary switches (diodes, transistors, ...), it makes sense to look first at the influence of the method on the complexity of the equations and the simplicity of the method itself. Since the equations will have to be ported on a digital computing platform, we will also take a look at how these equations behave when they are handled by a discrete-time solver.

The discussion is taken further into the subject of power converters in section 2.3. We compare the multiple ways of integrating the power switches into the equations, and also look at how to control these switches when the conditions inside the circuit change. To completely represent the dynamics resulting from mixing the linear components and the binary switches, we introduce the hybrid automaton paradigm in section 2.5, and a first simulation algorithm is presented in section 2.6.

## 2.2 Linear circuits

### 2.2.1 Introduction

Before simulating complete power converters, we first need to study how to simulate linear circuits (without any semiconductor device).

Therefore, we will first describe and compare two of the most used representations : the modified nodal analysis and the state-variable analysis. These two methods are used to obtain a set of equations describing the behavior of the circuit in response to its inputs, which is then simulated by a linear solver.

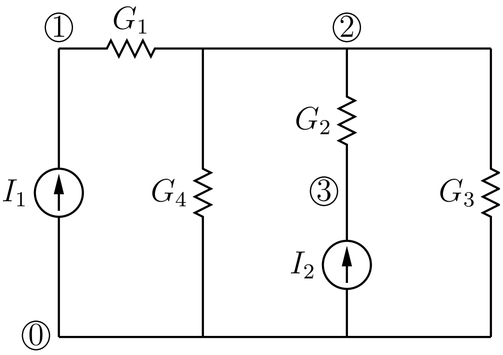


**Figure 2.1: A simple resistive circuit to illustrate the nodal analysis method. The encircled numbers represent the circuit nodes. The 0 node corresponds to the reference (i.e. to the ground node).**

### 2.2.2 The (Modified) Nodal Analysis

**The nodal analysis**

The original nodal analysis was one of the first systematic methods to obtain a mathematical model of an electric network [9]. The goal of this method is to write equations of the form

$$Y(s)V(s) = J(s) \tag{2.1}$$

where $V$ and $J$ are vectors containing respectively all node voltages and all current sources, $Y$ is the nodal admittance matrix of the circuit, and $s$ is the Laplace operator. If the circuit only contains resistors, the same equation can be used in the time domain. These equations are obtained through the use of graph analysis and of Kirchhoff's current law. To better illustrate this concept, we will put the circuit of figure 2.1 into equations of the form (2.1). First, we write the three nodal current equations :

$$\begin{aligned}
\text{Node 1 :}&(v_1 - v_2)G_1 &&= I_1\\
\text{Node 2 :}&(v_2 - v_1)G_1 + (v_2 - v_3)G_2 + v_2G_3 + v_2G_4 &&= 0\\
\text{Node 3 :}&(v_3 - v_2)G_2 &&= I_2
\end{aligned} \tag{2.2}$$

Or, in matrix form :

$$\begin{bmatrix} G_1 & -G_1 & 0\\ -G_1 & G_1 + G_2 + G_3 + G_4 & -G_3\\ 0 & -G_3 & G_3 \end{bmatrix} \begin{bmatrix} v_1\\ v_2\\ v_3 \end{bmatrix} = \begin{bmatrix} I_1\\ 0\\ I_2 \end{bmatrix} \tag{2.3}$$

Finally, the value of the node voltages is obtained by computing the inverse of the admittance matrix :

$$V(s) = Y^{-1}(s)J(s) \tag{2.4}$$

The nodal analysis is very efficient at modeling circuits with current sources. However, with the basic form given by (2.1), it does not allow the use of voltage sources or any other voltage controlled elements such as amplifiers, inductors or ideal transformers. A solution to these problems lies in the *modified* nodal analysis.

### The modified nodal analysis

The Modified Nodal Analysis (MNA) was first introduced during the seventies in [6] as a solution to the shortcomings of the classical nodal analysis, and has since been widely used by many offline simulators including the ever-popular SPICE [25]. It has also been ported on many off-the-shelf real-time simulators [17, 26, 27].
The increase in versatility comes from additional elements in the matrices defined in (2.1). The MNA matrices are defined as

$$\begin{bmatrix} Y & B \\ C & D \end{bmatrix} \begin{bmatrix} V \\ I \end{bmatrix} = H \begin{bmatrix} V \\ I \end{bmatrix} = \begin{bmatrix} J \\ F \end{bmatrix} \tag{2.5}$$

The additional $I$ in the unknown vector $\begin{bmatrix} V \\ I \end{bmatrix}$ represents branch currents, whose constitutive relations are defined by the $C$ and $D$ matrices. These branch currents are typically inductor currents or currents circulating through voltage sources. The $B$ matrix represents the effect of these added variables on the node voltages. The $F$ vector contains all additional excitations in the circuit. In the most basic form of the MNA, $F$ contains the voltage sources. The newly introduced branch equations may also be used to compute any current as part of the unknown vector, which is not possible with the nodal analysis paradigm [6].
The different matrices and vectors of (2.5) can be obtained by a powerful method called *stamping* : the effect of each circuit element is obtained separately and their contributions are summed together to obtain the final result. Each component (resistor, inductor, ...) has a different effect on the matrices. Some of them are presented in [6] and [28].

### Discrete-time MNA simulation

The MNA equations in (2.5) may not be directly used for discrete-time transient analysis, and we first need a discrete-time model of the circuit. To obtain this model, the circuit must be time-sampled before the equations are obtained, which requires the conversion of all reactive components to a discrete form. For a capacitor, the voltage after a sampling period $T$ may be obtained by:

$$i_c(t) = C\frac{\mathrm{d}v_c}{\mathrm{d}t} \Leftrightarrow v_c(kT) = v_c((k-1)T) + \frac{1}{C}\int_{(k-1)T}^{kT} i_c(t)dt \tag{2.6}$$

The discrete-time version is obtained by approximating the integral term (or the differential term), and is dependent on the discrete solver method method. For the backward Euler method:

$$\begin{aligned} i_c(t) &\approx i_c(kT) \\ i_c(kT) &= \frac{C}{T}[v_c(kT) - v_c((k-1)T)] = G_c v_c(kT) - J_c(kT) \end{aligned} \tag{2.7}$$

The term $J_C(kT)$ corresponds to the history of the signal and is introduced as a current source in parallel with a conductance $G_c$ of value $\frac{C}{T}$. The equivalent circuit is shown on

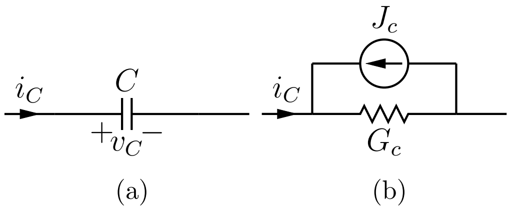


**Figure 2.2: A capacitor and its discrete time approximation. The value of the current source and of the conductance depend on the discretization method**

figure 2.2b. The approximation for the trapezoidal method yields

$$\begin{aligned} G_C &= \frac{2C}{T} \\ J_C(kT) &= \frac{2C}{T}v_C((k-1)T) + i_C((k-1)T) \end{aligned} \tag{2.8}$$

Inductors are discretized using a similar method :

$$\begin{aligned} G_L &= \frac{T}{L} \\ J_L(kT) &= i_L((k-1)T) \end{aligned} \tag{2.9}$$

Once the circuit is converted back to a purely resistive network, we apply the standard MNA procedure to write the equations in the form of (2.5). To simulate the circuit, we now have to repeatedly call the following computation for each new solving step (note that $T$ will not be written anymore) :

$$\begin{bmatrix} V(k) \\ I(k) \end{bmatrix} = H^{-1} \begin{bmatrix} J(k) \\ E(k) \end{bmatrix} \tag{2.10}$$

where $H^{-1}$ denotes the matrix inverse of $H$Before computing a new step, the controlled current sources must be updated according to the rules defined previously in (2.7) and (2.9).

**Example 1.** As an example, let us consider the circuit of figure 2.3a. Using the backward Euler method, we obtain the discretized circuit of figure 2.3b. The system matrix $H$ may be written using the stamping method, leading to

$$H \begin{bmatrix} v_1(k) \\ v_2(k) \\ v_3(k) \\ i_E(k) \\ i_L(k) \end{bmatrix} = \begin{bmatrix} G_1 & -G_1 & 0 & 1 & 0 \\ -G_1 & G_1+G_C & -G_C & 0 & 1 \\ 0 & -G_C & G_C+G_2 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & G_L & -G_L & 0 & -1 \end{bmatrix} \begin{bmatrix} v_1(k) \\ v_2(k) \\ v_3(k) \\ i_E(k) \\ i_L(k) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{C}{T}v_C(k-1) \\ -\frac{C}{T}v_C(k-1) \\ E \\ -i_L(k-1) \end{bmatrix} \tag{2.11}$$

Finally, the evolution law is obtained by inverting the relation:

$$\begin{bmatrix} v_1(k) \\ v_2(k) \\ v_3(k) \\ i_E(k) \\ i_L(k) \end{bmatrix} = H^{-1} \begin{bmatrix} 0 \\ \frac{C}{T}v_C(k-1) \\ -\frac{C}{T}v_C(k-1) \\ E \\ -i_L(k-1) \end{bmatrix} \tag{2.12}$$

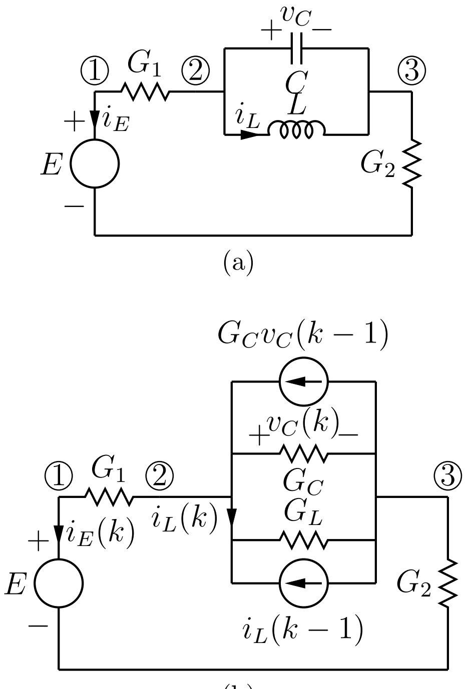


**Figure 2.3: A simple circuit and its discrete-time equivalent using the backward Euler approximation**

To present $H^{-1}$ in a reduced form, we can eliminate its first column since the multiplier (the first coefficient of the excitation vector) is equal to zero. Since we only need $v_2(k), v_3(k)$ and $i_L(k)$ to compute the value of the sources that will be used in the following step, we keep the corresponding rows while the others are also eliminated from the matrix. The final result is

$$H_R^{-1} = \frac{1}{\Delta}\begin{bmatrix} G_2+G_C+G_L & G_C+G_L & G_1(G_2+G_C+G_L) & G_2 \\ G_C+G_L & G_2+G_C+G_L & G_1(G_C+G_L) & -G_1 \\ G_2G_L & -G_1G_L & G_1G_2G_L & -G_1(G_2+G_C)-G_2G_C \end{bmatrix}$$

$$\begin{bmatrix} v_2(k) \\ v_3(k) \\ i_L(k) \end{bmatrix} = H_R^{-1}\begin{bmatrix} \frac{C}{T}v_C(k-1) \\ -\frac{C}{T}v_C(k-1) \\ E \\ -i_L(k-1) \end{bmatrix} \tag{2.13}$$

where $\Delta$ is the determinant of $H$. This equation, along with the definition of the current sources, contains the recurrence law that must be applied at each new computation step.

### Advantages and shortcomings of the MNA

While the Modified Nodal Analysis is very powerful and streamlined, it shows several shortcomings. The most crucial of these is that the circuit representation is not minimal [24]. Indeed, it seems unlikely that a matrix as large as the one described in (2.13) is required to represent the small circuit of figure 2.3. Considering that this large amount of operations must be applied to compute the new value of the signals at each new

simulation step, this complexity has a direct impact on the performance of any simulator. The additional overhead needed to update the controlled current sources must also be taken into account.
A minor problem lies in the discretization of the circuit. Since the reactive components must be converted to their discrete counterpart before the analysis, we can not use this method for variable-step simulations unless we compute $H$ and its inverse each time. We might be tempted to use more complex approximations such has the trapezoidal method like we did in (2.8). This should be done cautiously because it adds even more terms in the computed vector ($i_C$ is now explicitly computed) and in the expression of the current source.

### 2.2.3 The state-space analysis

**Introduction**

The state-space analysis (SSA) was developed in the early 60's to represent dynamic systems in an universal and compact form [29], and is based on the analysis of differential equations. The SSA is often used in control system design, and is the starting point of the so-called *Modern Control Theory.*
Given a linear, time-invariant, proper system, we can write a set of differential-algebraic equations called state-space equations describing its behavior:

$$\begin{aligned} \frac{\mathrm{d}x(t)}{\mathrm{d}t} &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{2.14}$$

The different terms appearing in these equations are as follow:

- $u(t)$ is a vector containing all independent excitations (sources)
- $y(t)$ is a vector containing all outputs (observations) of the system
- $x(t)$ is called the *state vector*, and contains the states of the system
- $(A, B, C, D)$ are constant matrices linking the vectors

The notion of state is somewhat nebulous, though it way be defined as the minimal amount of information that, along with the excitations, is needed to infer the behavior at any point in the future [5]. A state is typically the representation of an energy storage: for an electrical network, the state-vector generally contains all capacitor voltages and all inductor currents. Note that while the output signals are represented in (2.14), they are not needed to describe the behavior of the system : $y(t)$ is normally used to model sensors and other subsystems which allow to observe the system from an external point of view. The first equation of (2.14) is also called the state equation, while the second is the output equation.

### SSA and electrical networks

While this method originates from the field of mechanics, it was later applied to electrical engineering by Kuh and Rohrer in their seminal article called *The state-variable approach to network analysis* [4].
Just like the matrices and vectors used for the MNA, the matrices in (2.14) are obtained thanks to Kirchhoff's equations and the constitutive relations describing the elements of the circuit. Looking again at the network presented on figure 2.3a, we may write the following equations for the reactive components :

$$\begin{aligned} C\frac{\mathrm{d}v_C(t)}{\mathrm{d}t} &= i_C(t) = & \frac{E - v_C(t)}{R_1 + R_2} - i_L(t) \\ L\frac{\mathrm{d}i_L(t)}{\mathrm{d}t} &= v_L(t) = & v_C(t) \end{aligned} \tag{2.15}$$

The state-space equations are inferred by writing (2.15) in matrix form:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{L} \\ -\frac{1}{C} & -\frac{1}{(R_1+R_2)C} \end{bmatrix}\begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{(R_1+R_2)C} \end{bmatrix} E \tag{2.16}$$

This constitutes the minimal set of variables that may be used to infer all voltages and currents at a given time. If the state variables are also the observed variables (which is often the case), then the analysis is complete. If, for instance, we also want to observe the voltage at node 3 then the output equation is written as

$$y(t) = \begin{bmatrix} i_L(t) \\ v_C(t) \\ v_3(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & -\frac{R_2}{R_1+R_2} \end{bmatrix}\begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{R_2}{R_1+R_2} \end{bmatrix} E \tag{2.17}$$

The form of $C$ and $D$ is thus highly dependent on the measured variables, while the $A$ and $B$ matrices depend only on the circuit structure.
A few observations can be made from these equations. First, it is noticeable that the circuit topology does not appear explicitly : only the capacitor voltage and the inductor current are relevant to the analysis. Second, it is immediately evident that the state equation (2.16) is more compact than the equation obtained with the MNA for the same circuit in (2.13). Finally, another difference with the MNA is that the equations were obtained without discretizing the reactive elements. We are free to select the discretization method at any point in the future while the current model may still be used for continuous-time analysis (or with numerical tools that accept a continuous representation of differential equations, such as *Matlab*).

The steps used to obtain (2.16) were intuitive, but they do not constitute a general method for writing the state-space description. A generalized method based upon the graph theory was published in [4], and requires a lot of computations and network matrix manipulation. Alternative methods that use MNA as a starting point are presented in [30, 31], while [16] presents an algorithm that starts by writing the circuit equation using Kirchhoff's laws before eliminating and reordering the variables. While the last method is used with good performance in the offline simulator *PLECS*, the complexity of

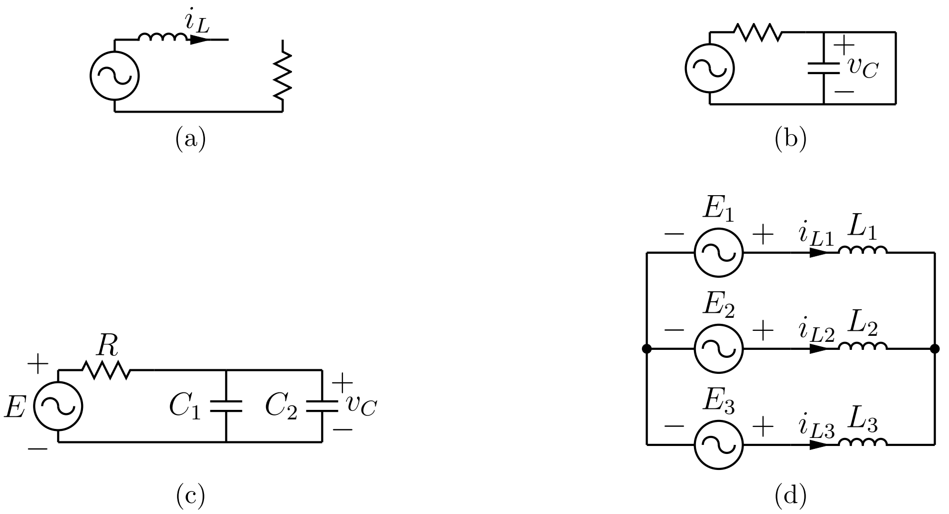


**Figure 2.4: Examples of circuits containing forced states**

the Gauss-Jordan elimination used to reorder the variables prohibits its use in real-time. As a direct consequence, we have to compute the state matrices before the real-time simulation. Since existing tools allow the user to obtain the state matrices for a given network, the development of such a tool falls outside of the scope of this thesis.

### Forced states

The number of states is typically equal to the amount of reactive components in the circuit, but it not always the case. A reduction of the number of states can occur when any of these statements are true :

- a loop containing only capacitors (and voltages sources) is present
- a cut-set containing only inductors (and current sources) is present

When a state reduction is performed, the eliminated states are characterized as *forced states*. Their value may be obtained by algebraic equations instead of differential equations.

The circuits of figure 2.4 all lead to state reduction:

For circuit (a) and (b), the state is respectively forced to zero because of the open-circuit or of the short-circuit:

$$\begin{aligned} \begin{bmatrix} i_L \\ v_C \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} i_L \\ v_C \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned} \tag{2.18}$$

The two capacitors of circuit (c) act as a single larger one, and the dynamics of their voltage $v_C$ is described by

$$\frac{\mathrm{d}v_C}{\mathrm{d}t} = -\frac{1}{R(C_1 + C_2)} v_C + E \tag{2.19}$$

Unless the individual currents of the capacitors must be known, this equation is sufficient to describe the circuit. For circuit (d), any of the three currents can be eliminated and written as a linear combination of the others. Assuming $i_{L3}$ is eliminated, we write it as a combination of the two other currents. This combination occurs either in the state equations

$$\begin{gathered}\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_{L1} \\ i_{L2} \\ i_{L3} \end{bmatrix} = \frac{1}{k}\begin{bmatrix} L_2+L_3 & -L_3 & -L_2 \\ -L_3 & L_1+L_3 & -L_1 \\ -L_2 & -L_1 & L_1+L_2 \end{bmatrix}\begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} \\ k = L_1L_2 + L_1L_3 + L_2L_3 \end{gathered} \tag{2.20}$$

or in the output equation

$$\begin{gathered}\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_{L1} \\ i_{L2} \end{bmatrix} = \frac{1}{k}\begin{bmatrix} L_2+L_3 & -L_3 & -L_2 \\ -L_3 & L_1+L_3 & -L_1 \end{bmatrix}\begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} \\ k = L_1L_2 + L_1L_3 + L_2L_3 \\ i_{L3} = -i_{L1} - i_{L2} \end{gathered} \tag{2.21}$$

Any of the two methods may be used to compute $i_{L3}$, but the second one is preferred to avoid any drift due to the integration of the rounding error.

A more general formulation is obtained by attempting to write the state vector as an output:

$$x(t) = C_x x(t) + D_x u(t) \tag{2.22}$$

In normal operation, the unique solution to this equation is to set $C_x$ as the identity matrix and to fill $D_x$ with zeroes. But, if some of the states are forced, they may also be computed using the unforced signals. By splitting the state vector $x(t)$ in its unforced part $x_u(t)$ and its forced part $x_f(t)$, we write

$$x(t) = \begin{bmatrix} x_u(t) \\ x_f(t) \end{bmatrix} = \begin{bmatrix} I & 0 \\ C_{xf}(t) & 0 \end{bmatrix}\begin{bmatrix} x_u(t) \\ x_f(t) \end{bmatrix} + \begin{bmatrix} 0 \\ D_{xf}(t) \end{bmatrix} u(t) \tag{2.23}$$

Where $C_{xf}$ and $D_{xf}$ are the submatrices of $C_x$ and $D_x$ related to the forced states. Note that, using this formulation, a forced state does not depend on other forced states.

### Discrete-time SSA

In contrast with the nodal analysis, the state-space method generates a set of equation that may be used for the continuous-time simulation of circuit transients. Furthermore, the form of the equations is a very generic linear differential and algebraic system. Solving differential equations numerically is a well-known problem, and many authors have presented multiple methods that range from the very simple Euler integration to the Runge-Kutta used in modern computing toolboxes [32–34]. The purpose of the discretization methods is to provide a numerical model of the form of a recurrence equation

$$x_d(k) = A_d x_d(k-1) + B_d u_d(k) \tag{2.24}$$

The $A_d$ and $B_d$ matrices depend on the discretization algorithm, $x_d(k)$ is the estimated value of $x(kT)$ obtained by the chosen method and $u_d(k) = u(kT)$. Among the properties found in the many existing numerical solvers for ordinary differential equations (ODEs), these are the more crucial:

- Precision : the solver must converge to a value very close to the one found in the real system. Higher order solvers are typically more precise. A solver of convergence order $n$ has an error proportional to $T^n$, where $T$ is the sampling period.
- Computing Effort : since all computations must be made in real-time, we cannot afford too many calculations if we aim at a low sampling period. This is in direct contradiction with the first point since higher order solvers are typically more complex.
- Stability : the method must not be unstable or present unwanted oscillations if the base system is stable

A few methods that may be used in real-time are described in the remainder of this section. Before going further, let us note that exact value of $x(kT)$ can be obtained by inverting (2.14)

$$x(kT) = x((k-1)T) + \int_{(k-1)T}^{kT} (Ax(t) + Bu(t))dt \tag{2.25}$$

The presented methods aim to find the best estimate of the integral in (2.25). To simplify all further developments, we shall assume that $u_d(k) \approx u_d(k-1)$. This approximation is fair because the sampling period is often much smaller than the time constants shown by the input signals (which are typically DC or low frequency AC signals). The simplest method is the backward Euler approximation (BEA), defined as

$$\begin{aligned} x(t) &\approx x(kT) \text{ for } t \text{ in } [(k-1)T, kT] \\ x_d(k) &= x_d(k-1) + T(Ax_d(k) + Bu(k)) \\ x_d(k) &= (I - TA)^{-1}(x_d(k-1) + TBu(k)) \end{aligned} \tag{2.26}$$

where $I$ is the identity matrix. The discrete-time state matrices are obtained by identification:

$$A_{d,BE} = (I - TA)^{-1}; B_{d,BE} = (I - TA)^{-1}TB \tag{2.27}$$

The BEA is very stable : if (and only if) the continuous time system in (2.14) is stable, then the sampled-time system is guaranteed to be stable. This property is known as *A-Stability*. The method has a convergence order of 1.

Another very popular method is the trapezoidal rule for integration, which estimates the integral in (2.25) with the approximation

$$x(t) \approx \frac{x(kT) + x((k-1)T)}{2} \text{ for } t \text{ in } [(k-1)T, kT] \tag{2.28}$$

Which allows us to write the recurrence

$$\begin{aligned} x_d(k) &= x_d(k-1) + \frac{T}{2}(A(x_d(k) + x_d(k-1)) + B(u_d(k) + u_d(k-1) \\ x_d(k) &= (I - \frac{T}{2}A)^{-1}((I + \frac{T}{2}A)x_d(k-1) + \frac{T}{2}Bu_d(k)) \\ A_{d,trap} &= (I - \frac{T}{2}A)^{-1}(I + \frac{T}{2}A) \\ B_{d,trap} &= (I - \frac{T}{2}A)^{-1}\frac{T}{2}B \end{aligned} \tag{2.29}$$

Like the BEA, the trapezoidal integration is A-Stable. However, the output may show oscillations that are not present in the continuous-time output [32], which is not the case for the BEA (the trapezoidal approximation is qualified as not *L-Stable*). The method has a convergence order of 2, which make it more accurate than the backward Euler. Moreover, a limit called the second Dahlquist barrier proves that no A-Stable method is more precise than the trapezoidal integration [33]. This property is the main reason of its widespread use. For example, SPICE uses the trapezoidal method as the default for its computations.
Finally, we also present the two-pass midpoint method, also called improved Euler approximation (IEA):

$$\begin{aligned} x^*(k) &= x_d(k-1) + T(Ax_d(k-1) + Bu_d(k)) \\ x_d(k) &= x_d(k-1) + TA\frac{x_d(k-1) + x^*(k)}{2} + TBu(k) \end{aligned} \tag{2.30}$$

The IEA is part of a larger class of solvers called predictor-corrector. A first rough value of $x_d(k)$ is computed first (using the simple forward Euler method), then corrected using a variant of the trapezoidal method. In contrast with the previous methods, the improved Euler is not A-Stable. Furthermore, it must be computed in two passes instead of one. Despite these problems, it also presents two interesting characteristics. First, it is easily adapted to variable-step discretization: the only modification is to multiply $A$ and $B$ by the variable $T$ before realizing the matrix products, whereas the other methods require an on-line matrix inversion. By not inverting any matrix, we also preserve the structure of the continuous-time system : if the $A$ matrix was mostly empty (ie. the circuit has a ladder structure), the number of elements in the matrix products is significantly reduced.

Any of these three algorithms is suitable for real-time simulation. In the remainder of this work, we will use the trapezoidal integration unless specified otherwise in the text.

### Advantages and shortcomings of the SSA

The main advantage of the SSA over the MNA lies in the size of the discrete matrices in (2.24) compared to those in (2.10). This is due to the independence of the equations from the topology of the circuit, whereas the MNA requires the calculation of all node voltages. As explained previously, the state vector is the minimal amount of information required to know the exact behavior of the circuit, and it makes sense that this leads to a lower number of computations. The differential-algebraic form of the equations allows us to use the results from the numerical analysis of differential equations, and we may freely choose a method which is either more precise (trapezoidal), more robust (backward Euler) or allowing to easily implement a variable step with minimal overhead (improved Euler).
This being said, the MNA is easier to use when dealing with non-linear circuits : in many cases, the non-linearity may be represented as a controlled source, just like we did with the reactive components (the source is then controlled by the non-linear law). The simulation algorithm then becomes

- perform a single-step simulation the circuit using a linear method

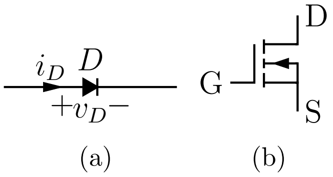


**Figure 2.5: Definition if the signals used to describe the diode and the MOS transistor**

- modify the source
- loop until convergence

The SSA does not implement controlled sources in the representations, which means that the whole matrices must be modified. This could lead to serious overhead even if the matrices are small.
At this point, the SSA seems a better fit for our problem, assuming we find a effective algorithm to modify the matrices on-the-fly. The next section, dealing with the modeling of power converters, will confirm this choice.

## 2.3 Power Converter Modeling

### 2.3.1 Introduction

One of the defining characteristics of power converters is their extensive use of semiconductor devices as electronic switches in order to control the power flow inside the circuit. These devices are generally described by a non-linear equation, which makes them impractical to use as-is in simulators. For example, the current-voltage characteristic of typical semiconductor diode is described by the Schockley equation

$$i_D = I_s(e^{v_D/v_T} - 1) \tag{2.31}$$

where $i_D$ and $v_D$ are the diode current and voltage defined in figure 2.5a, $v_T$ is the thermal voltage (a parameter that depends on the temperature) and $I_s$ is the diode specific saturation current. Similarly, a MOSFET transistor as shown in figure 2.5b is described by simplified equations that also include the gate voltage [35]:

$$i_D = \begin{cases} 0 & \text{if } v_{GS} \leq v_{th} \\ K((v_{GS} - v_{th})v_{DS} - \frac{v_{DS}^2}{2}) & \text{if } v_{GS} > v_{th}, v_{DS} \leq (v_{GS} - v_{th}) \\ \frac{K}{2}(v_{GS} - v_{th})^2 & \text{if } v_{GS} > v_{th}, v_{DS} > (v_{GS} - v_{th}) \end{cases} \tag{2.32}$$

where $K$ and $v_{th}$ are intrinsic properties of the transistor, $v_{GS}$ is the applied gate control voltage and $v_{DS}$ is the drain-to-source voltage. In power electronics, only the first two regions are exploited (the third one, called active region, is mainly used in analog amplification) and the behavior may be expressed as a function of a binary control signal $u$:

$$i_D = \begin{cases} 0 & \text{if } u = 0 \\ K((v_{GS} - v_{th})v_{DS} - \frac{v_{DS}^2}{2}) & \text{if } u = 1 \end{cases} \tag{2.33}$$

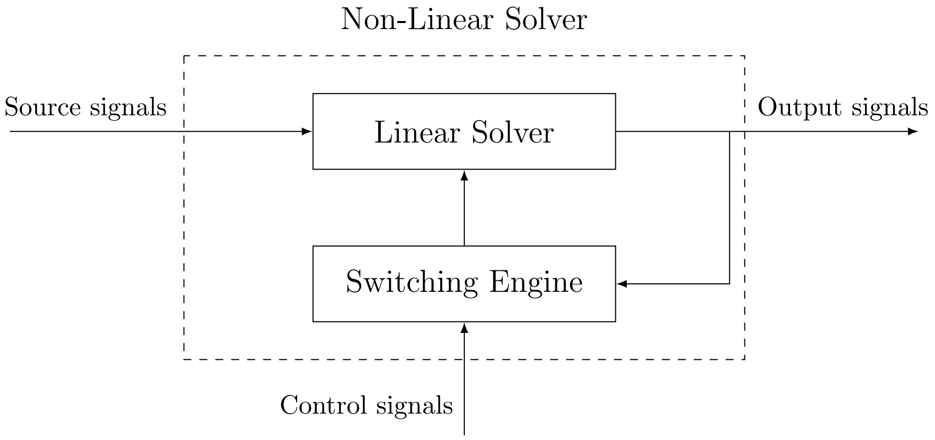


**Figure 2.6: Generic piecewise-linear solver, composed of a linear solver and a non-linear feedback that modifies the equations**

As explained in 2.2.3, the solving procedure is much more difficult if the circuit exhibits non-linear behavior (specially for the state-space analysis). While non real-time solvers such as Spice use iterative methods to find the precise value of the signals [25], we won't have the processing power required for such methods. To model the behavior of the switches in real-time, some simplifications have to be made. A popular method is the so-called *piecewise-linear approximation*, which follows these steps :

1. decompose the switch behavior into a set of linear models with their own validity domain

2. find the rules that control the transitions between these models

3. rewrite the circuit equations for each model

This process is described in the next sections, beginning with the different ways to model a generic binary switch before studying more complex power semiconductors. On the real-time platform, the procedure is implemented as shown in figure 2.6. At the core of the system, a linear solver computes the evolution of the signals by using the equations defined by the MNA or by the SSA. The non-linear part of the device is controlled by a second subsystem, called the switching engine, that modifies the linear description (by means of a change in the current sources of the MNA, or in the matrices of the SSA) according to the value of a subset of the output vector and/or additional control signals. The interaction of these two subsystems constitutes the non-linear solver.
Finding the right model for a simple binary switch is one of the most crucial steps in the design of our simulator. A badly chosen model may lead to added complexity, unwanted oscillations in the signals or hamper the convergence.

### 2.3.2 The binary switch

A binary switch is a dipole whose current-voltage law is modified according to a binary control signal. When a switch is in the *on-state* (also called *conducting state*), it offers little to no resistance to the current (the equivalent impedance is close to zero). Conversely, a switch in the *off-state* (or *blocking state*) presents a high equivalent impedance.

We will also use the term *configuration* (instead of state) of a switch whenever there is is a risk of confusion with the state variables. The exact representation of the switch has a deep impact on the general equation system of the circuit.

**Types of binary switches**

Among other characteristics, switches may be described by the way they are controlled. A switch is said to be *active* or *user-controlled* if its state is exclusively a function of an external stimulus. The best known example of an active switch is the transistor used as a switching device, whose state is given by the voltage or current applied to the input terminal (gate terminal for the MOS and the IGBT, base terminal for the BJT). An ideal (lossless) active switch is described by

$$\begin{aligned} v_D &= 0 \quad \text{if } u = 1 \\ i_D &= 0 \quad \text{if } u = 0 \end{aligned} \tag{2.34}$$

where $u$ is the binary control signal which is typically generated by the control system outside the circuit. With this model, a switch in the on-state is modeled as a short-circuit, while a switch on the off-state is represented by an open circuit At the opposite side of the spectrum, a *passive* or *circuit-controlled* switch is only controlled by currents and voltages inside the power converter and may not be directly set to either of its states by the user. The diode is by far the best-known example of a passive switch. Idealized piecewise-linear characteristics are derived from (2.31), and correspond to two different model levels. The first model is called the ideal diode, and is described by the following equations:

$$\begin{aligned} v_D &= 0 \quad \text{if } i_D > 0 \\ i_D &= 0 \quad \text{if } v_D \leq 0 \end{aligned} \tag{2.35}$$

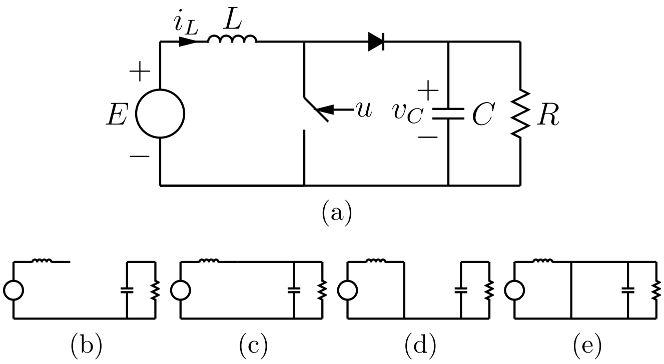
The half-line defined by the first equation-inequation pair corresponds to the diode in the on-state, and the second one to the off-state. The diode will remain in its current state until the circuit signals evolve to a point where the corresponding inequality is not verified anymore. At this point, the state of the diode changes and the circuit continues to evolve.

More complex semi-conductors may be modeled using these two ideal devices and additional elements, as will be shown in section 2.4.

We will now compare multiple methods that allow the use of ideal switches with a zero equivalent resistance in the on-state and an infinite resistance in the off-state, and select the most promising for our platform.

### 2.3.3 The open/short circuit model

This model is the simplest one, but also one of the most difficult to implement properly. Using this paradigm, the switch is modeled as a short circuit when in the on-state, and as an open circuit in the off-state. Once the state of all the switches, also called the *configuration* of the switches is known, the power circuit is reduced to an equivalent linear circuit called *topology*. A topology's equations may be written using the algorithms discussed in section 2.2.



(a)

(b) (c) (d) (e)

**Figure 2.7: Boost Converter and its four equivalent topologies**

As an example, let us look at the boost converter of figure 2.7a: this circuit contains two switches - one diode and one controlled switch (ie. a transistor) - and possesses four topologies as shown on figures 2.7b to 2.7e. Each of these topologies may be described by its state-space equations as follows :

$$\begin{cases} i_L(t) = 0 \\ \dfrac{\mathrm{d}}{\mathrm{d}t} v_C(t) = -\dfrac{1}{RC} v_C(t) \end{cases} \quad \text{when both switches are off} \tag{2.36}$$

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t} i_L(t) = \dfrac{1}{L}(E - v_C(t)) \\ \dfrac{\mathrm{d}}{\mathrm{d}t} v_C(t) = \dfrac{1}{C} i_L(t) - \dfrac{1}{RC} v_C(t) \end{cases} \quad \text{when the transistor is off and the diode is on} \tag{2.37}$$

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t} i_L(t) = \dfrac{1}{L} E \\ \dfrac{\mathrm{d}}{\mathrm{d}t} v_C(t) = -\dfrac{1}{RC} v_C(t) \end{cases} \quad \text{when the transistor is on and the diode is off} \tag{2.38}$$

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t} i_L(t) = \dfrac{1}{L} E \\ v_C(t) = 0 \end{cases} \quad \text{when both switches are on} \tag{2.39}$$

only one of these description is used at a given time, depending on the current configuration of the two switches. An inspection of these formulas reveals that some on the topologies lead to *dead states* as explained in section 2.2.3.

A circuit containing $n$ switches possesses a base number of $2^n$ independent topologies described by their own equations. Without any further simplification, this high number could quickly lead to complicated algorithms not only to find the correct topology, but also to store and change the equation sets without causing too much overhead. A basic example of simplification comes from noting that the fourth topology of Figure 2.7 never occurs in practice. Indeed, solving this circuit leads to a zero current across the conducting diode, which is incompatible with its constitutive law (2.35). Removing the topology from the list of available combinations further reduces the complexity of the selection algorithms.

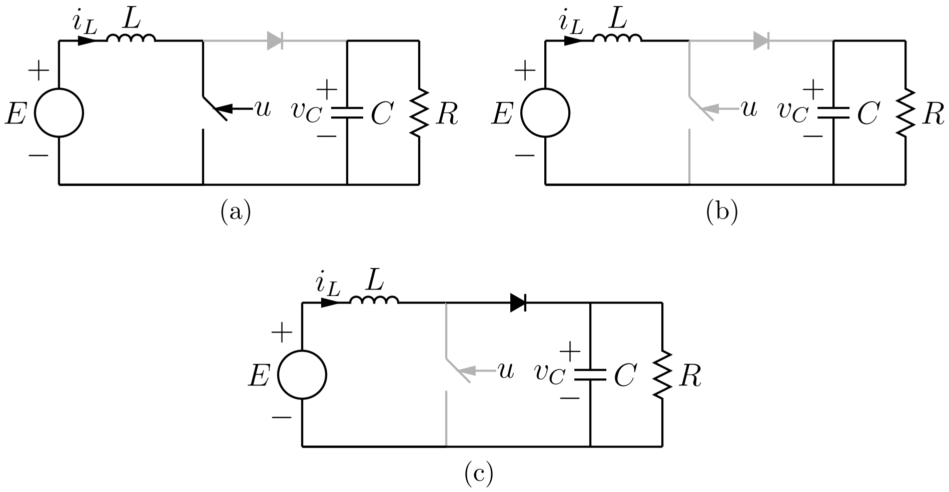


**Figure 2.8: Illustration of the state continuity problem. When the transistor switches from the on-state (a) to the off-state (b), the diode must be switched on (c) to allow $i_L$ to flow**

### Topology changes and state continuity

The open/short circuit model leads to a set of very simple circuits, but has one crucial problem: there is no easy method to track how and when the circuit switches from one topology to another. To be more precise, we have to write how the diodes will react to changes in

- the value of the states
- the control signals of the active switches

The first of these items is relatively intuitive : if the internal signals evolve in such a way that the inequalities defined in (2.35) are not respected anymore, then we have to change the state of the diode(s) accordingly. If, for instance, the current in the diode of the circuit 2.7a becomes negative, the diode is switched to the blocking state. The equations of the diodes' current and voltage, and hence the form of the inequalities to evaluate, is governed by the current topology.

The second item is a result of a property called *state continuity* that may be written as the following rule: the value of a state may never change instantaneously, even when the topology of the circuit is modified. This statement is a direct consequence of (2.14): if a state was allowed to change its value suddenly, it would generate Dirac pulses across the circuit, leading to the probable destruction of all components in its path. This property is the reason why freewheeling diodes are placed in circuits, as shown on the boost converter of figure 2.8. When the transistor is opened, the diode is forced to switch to the on-state to allow the inductor current to flow freely.

To find out how the diode reacts to changes, some algorithms actually simulate the effects of the Dirac pulses and interprets the results [36–38]. A variant of this method is implemented in the *PLECS* simulator [16].

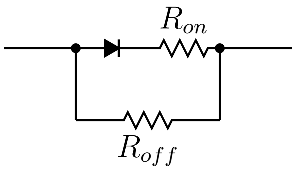


**Figure 2.9: A diode modeled as a bi-valued resistor thanks to an ideal diode and two resistors**

### Interaction with the linear solver

The non-linear part of the solver controls which topology is currently in use, following the rules defined in 2.3.3. Once the topology is chosen, the coefficients are passed to the linear solver for the next solving step. While this technique can be used by both the Modified Nodal Analysis and the State-Space Analysis, it is much more efficient when coupled with the latter one. The first advantage comes from the memory requirements: as we have shown in section 2.2.3, the simulation matrices obtained by the SSA contain fewer non-zero elements. This compactness allows us to design matrix selection algorithms that are both lighter and faster.

The second advantage comes from state continuity. Since not all states appear explicitly in MNA (2.10), we first have to compute them in addition to the node voltages and branch currents. While this additional overhead is not prohibitive, it leads to additional computations, and hence a reduced global performance.

## 2.3.4 The bi-valued resistor

With this model, the switch is modeled by a resistor whose value is a function of its state: the switch presents a resistance $R_{on}$ when in the on-state, and a resistance $R_{off}$ when in the off-state. These two resistances model the switch imperfections, with $R_{on} \gg R_{off}$.

The main advantage of this method is that the structure of the matrices stays the same for all topologies: the only effect is that some of the coefficients are modified by the value of the resistances. This guarantees state continuity, as we always provide a path for the currents, which simplifies the switching engine. However, we now have to deal with matrix coefficients with widely different values. Depending on the number representation in the processing platform, and especially when using fixed-point representation, we could observe matrices that are badly conditioned due to rounding. This method has also been shown to generate parasitic oscillations that may be partially compensated by adding a capacitor in parallel to the switch, such capacitors exist physically, but normally they should only be used for simulators aimed at accurate commutation waveforms, which is not our case [22].

Since this representation is actually a subcase of the on/off model (see figure 2.9), we will not use it explicitly in this work.

## 2.3.5 The controlled source switch

The popular controlled source model for ideal switch was first introduced to avoid changing the state matrices when the configuration of the switches is changed [12], and has

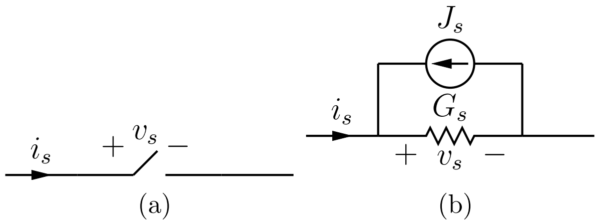


**Figure 2.10: An ideal switch and its controlled current source model**

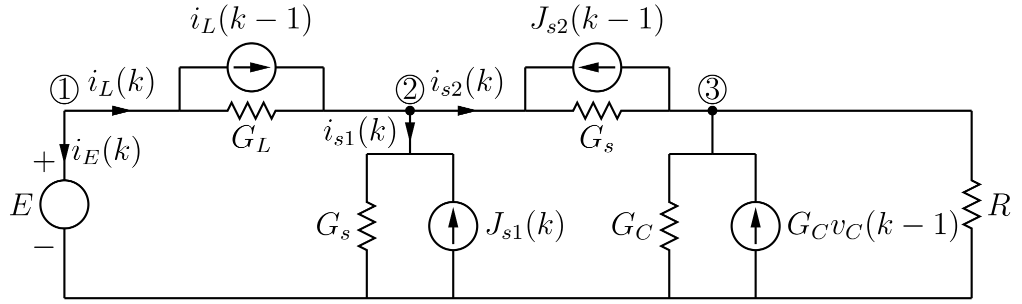


**Figure 2.11: The boost converter of figure 2.7a, with switches modeled by controlled current sources and reactive components replaced by their discrete-time equivalents**

since then been reimplemented in many MNA-based real-time power converter simulators [20, 39–43]. The procedure starts by replacing all switches by a variable gain controlled current source $J_s$ in parallel with a constant conductance $G_s$, as shown on figure 2.10b. To obtain the ideal switch law (2.34) or (2.35), the value of the current source is defined as

$$J_s(t) = \begin{cases} -i_s(t) & \text{if the switch is in the on-mode} \\ G_s v_s(t) & \text{if the switch is in the off-mode} \end{cases} \tag{2.40}$$

To translate this source to its discrete counterpart $J_s(k)$, we again have to choose the discretization method. If we assume that the signals evolve slowly compared to the sampling period, we write

$$i_s(k+1) \approx i_s(k), v_s(k+1) \approx v_s(k) \tag{2.41}$$

$$J_s(k+1) = \begin{cases} -i_s(k) & \text{if the switch is in the on-mode} \\ G_s v_s(k) & \text{if the switch is in the off-mode} \end{cases} \tag{2.42}$$

More complex forms obtained by identifying the switch as an $L-C$ component are given in [12]. This comparison provides an interesting point: with this model, the switch is represented as a small inductance when in the on-mode, and as a capacitor when in the off-state. This in turn implies that these parasitic elements may provoke additional oscillations in the signals that may degrade the performance if not properly damped by carefully choosing the value of $G_s$.

To give a better comparison with the open/short circuit model, we will again study the boost converter of figure 2.7a. We first translate the switches using (2.42) and the reactive components using the backward Euler approximation to obtain the circuit shown in figure

2.11. The modified nodal analysis of this circuit yields

$$\begin{bmatrix} 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 1 \\ 0 & 0 & \frac{1}{R}+G_C & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ G_L & -G_L & 0 & 0 & -1 & 0 & 0 \\ 0 & G_s & 0 & 0 & 0 & -1 & 0 \\ 0 & G_s & -G_s & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} v_1(k) \\ v_2(k) \\ v_3(k) \\ i_E(k) \\ i_L(k) \\ i_{s1}(k) \\ i_{s2}(k) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ G_C v_3(k-1) \\ E \\ -i_L(k-1) \\ J_{s1}(k-1) \\ J_{s2}(k-1) \end{bmatrix} \tag{2.43}$$

The complete evolution law is written by inverting the system matrix $H$. By removing the elements corresponding to $v_1$ and $i_E$ (which are not needed to solve the circuit) and those multiplied by the zero elements of the source vector, we obtain

$$\begin{aligned} \begin{bmatrix} v_2(k) \\ v_3(k) \\ i_L(k) \\ i_{s1}(k) \\ i_{s2}(k) \end{bmatrix} &= H_R^{-1} \begin{bmatrix} G_C v_3(k-1) \\ E \\ -i_L(k-1) \\ J_{s1}(k-1) \\ J_{s2}(k-1) \end{bmatrix} \\ J_{s1}(k) &= \begin{cases} -i_{s1}(k-1) & \text{if the transistor is in the on-mode} \\ G_s v_2(k-1) & \text{if the transistor is in the off-mode} \end{cases} \\ J_{s2}(k) &= \begin{cases} -i_{s2}(k-1) & \text{if the diode is in the on-mode} \\ G_s(v_2(k-1) - v_3(k-1)) & \text{if the diode is in the off-mode} \end{cases} \end{aligned} \tag{2.44}$$

where $H_R^{-1}$ is the $5 \times 5$ reduced inverse matrix. A study of this matrix shows that all elements are non-zero. Absent from (2.44) is the law controlling the state of the diode, which may be written as

$$\text{new state is} = \begin{cases} \begin{cases} \text{on} & \text{if } i_{s2} > 0 \\ \text{off} & \text{if } i_{s2} \leq 0 \end{cases} & \text{if previously on} \\ \begin{cases} \text{on} & \text{if } (v_2 - v_3) > 0 \\ \text{off} & \text{if } (v_2 - v_3) \leq 0 \end{cases} & \text{if previously off} \end{cases} \tag{2.45}$$

**Interaction with the linear solver**

Before computing a new linear step, the solver first evaluates the new configuration of the switches using (2.45) and computes the new value of the controlled sources. The new values are then passed to the linear solver for the new step.
Since the SSA does not implement controlled sources, the additional elements in the circuits are equivalent to new state variables requiring additional computations, limiting the use of this method to MNA only.

### 2.3.6 Comparison and algorithm selection

At this point, it becomes crucial to choose which of the two methods will be used in the real-time platform. Both methods have been used in many emulators with varying degrees of success, and both have a number of shortcomings that may sometimes be avoided. The transfer matrix $H$ used in the controlled-source model does not depend on the switch configuration, and only the sources need to be modified to take into account the state of the switches. This is a big advantage over the open/short circuit since fewer variables have to be adjusted, leading to a reduced memory usage as well as a smaller number of operations. The laws controlling the switches are also static and do not have to be modified when the topology changes, and all the variables needed to find the configuration of the switches are computed as part of the MNA process.

The biggest disadvantage of this method is the very high amount of computing resources required by the solver: more than 25 multiplications and 20 additions are needed to compute (2.44), while the open/short circuit method solves the same step with five multiplications and three additions.

For larger converters, the exact number of computations is a function of the number of elements in the circuit and of the circuit itself. In general, a MNA solver requires

- one equation and one source for each inductor current
- two equations and one source for each capacitor differential voltage
- three equations (one current and a differential voltage) and one source for each switch

Assuming a circuit composed of $c$ capacitors, $l$ inductors, $t$ active switches, $d$ diodes and $e$ independent sources, we have an unknown vector of $n_1 = (l + 2c + 3(t + d))$ elements, a source vector of $n_2 = (e + l + c + t + d)$ elements and a transition matrix of $n = n_1 * n_2$ coefficients. This is obviously a worst-case computation, since voltage nodes may be shared by multiple circuit components (as was the case of node 3 shared by the diode and the capacitor in the boost), but it gives a good first idea of the complexity.

For the state-space analysis, the state matrices $A$ and $B$ contain respectively $(l+c)*(l+c)$ and $(l+c)*e$ elements. This number is generally much lower than what we obtained with the MNA but does not include the additional equations required to control the diodes. The signal $y_d$ (current OR voltage, depending on the active topology) controlling each diode may be seen as measure and added to the output vector $y(k)$ of (2.14) :

$$\begin{bmatrix} y_{d1}(k) \\ \vdots \\ y_{dn}(k) \end{bmatrix} = C_d x(k) + D_d u(k) \tag{2.46}$$

Assuming one equation per diode as a base number, this adds $d*(l+c+e)$ multiplications for a total of $m = (l + c + d) * (l + c + e)$. Again, this is a upper bound that is seldom found in practice because the linear series-parallel structure of the power circuits tends to decouple the state variables, leading to matrices that are mostly empty.

### Example : AC/DC Converter

To put the previous developments into perspective, let us take a look at the two-stage AC/DC converter represented in figure 2.12. This circuit contains two inductors, two capacitors and twelve switches (reduced to eight by fusing each of the diode-transistor

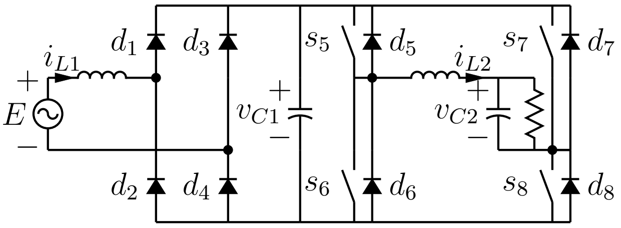


**Figure 2.12: Two stage AC/DC converter, illustrating state-decoupling**

pairs into a single switch). The modified nodal analysis requires the computation of 6 voltage nodes and 10 branch currents using a full 16-by-13 system matrix:

$$\begin{bmatrix} v_1(k) \\ \vdots \\ v_6(k) \\ i_{L1}(k) \\ i_{L2}(k) \\ i_{s1}(k) \\ \vdots \\ i_{s8}(k) \end{bmatrix} = H_R^{-1} \begin{bmatrix} E(k) \\ -i_{L1}(k-1) \\ -i_{L2}(k-1) \\ G_{C1}v_{C1}(k-1) \\ G_{C2}v_{C2}(k-1) \\ J_{s1}(k) \\ \vdots \\ J_{s8}(k) \end{bmatrix} \tag{2.47}$$

Each of the solver steps requires $13 * 16 = 208$ multiplications and $12 * 16 = 192$ additions to perform the dot products, and around 10 multiplications to compute the new value of the sources.

On the other hand, the state equations may be written as

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} i_{L1}(t) \\ v_{C1}(t) \\ i_{L2}(t) \\ v_{C2}(t) \end{bmatrix} = \begin{bmatrix} 0 & -k_1/L_1 & 0 & 0 \\ k_1/C_1 & 0 & -k_2/C_1 & 0 \\ 0 & k_2/L_2 & 0 & -k_3/C_2 \\ 0 & 0 & 1/C_2 & -1/R_2C_2 \end{bmatrix} \begin{bmatrix} i_{L1}(t) \\ v_{C1}(t) \\ i_{L2}(t) \\ v_{C2}(t) \end{bmatrix} + \begin{bmatrix} k_4/L_1 \\ 0 \\ 0 \\ 0 \end{bmatrix} E(t) \tag{2.48}$$

where $k_1, k_2 \in [-1, 0, 1]$ and $k_3, k_4 \in [0, 1]$ are coefficients that depend on the active topology. The front inductor $L_1$ effectively separates the input voltage from the decoupling capacitor $C_1$ while $L_1$ is independent from $L_2$ thanks to $C_1$, lowering the number of non-zero elements to less than half of the maximum. This zero/non-zero structure is kept when using the Improved Euler approximation, but not when using implicit solvers that force a matrix inversion, which tends to fill all the coefficients of $A$ and $B$ with non-zero values.

Even then, the maximum number of computations is limited to $4 * (4 + 1) = 20$ multiplications and 16 additions for the state equation. To compute the diode controlling signals, a theoritical maximum of $6 * (4 + 1) = 30$ multiplications are needed. Again, thanks to the decoupling provided by the reactive components, the majority of these coefficients will be equal to zero.

The results, rewritten on Table 2.1, show that using the state-space analysis instead of the MNA leads to a significant reduction of mathematical operations.

**Table 2.1: Comparison of the approximate amount of resources needed by the MNA and the SSA**

| | MNA | SSA |
|---|---|---|
| Multiplications | 220 | 50 |
| Additions | 200 | 40 |

### Selection

While modern digital platforms integrate powerful hardware units able to perform high-speed multiplications, low-cost FPGAs are still limited in their computational power while integrating a lot of high-speed general purpose logic [44]. This is the reason why we chose to use a state-space solver in our platform. However, this also forces us to develop highly optimized algorithms to quickly switch between the many topologies. To the best of our knowledge, no complete study of this subject has already been done.
Before developing any algorithm, we will first introduce a transitional model on the converter that includes both continuous changes and topology transitions: the hybrid automaton. This representation, described in section 2.5, will be used as a basis for the automated analysis and optimization algorithms presented in chapter 3.

## 2.4 Advanced device modeling

Many electronic switches are used in electronic circuits, from the simple diode to the IGBT and the thyristor. These devices may be modeled on the basis of the ideal on/off switch, by adding conductive losses or voltage thresholds. These imperfections may, in some cases, make the simulation more relevant to the test case. For example, adding imperfections to a transistor allow to simulate a cross conduction in a leg of an inverter.

### 2.4.1 Diode models

#### Piecewise approximation of a practical diode

The diode model introduced up to now is actually a lossless diode without any threshold or resistive behavior. These two parameters can easily be taken into account by adding components to the circuit (figure 2.14a). In this model the characteristic of the diode is linearized, transforming the exponential expression $i_D = I_s(e^{v_D/v_T} - 1)$ into a straight line taken between the threshold point and the nominal point, as shown in Figure 2.13. By definition, the slope of this line is equal to $\frac{1}{R_{on}}$, where $R_{on}$ in the on-state resistance
An even more complex model is obtained by adding the off-state resistance $R_{off}$ in parallel with the diode, and the reverse breakdown can be modeled by adding a second diode in anti-parallel, along with a source representing the breakdown voltage (Figure 2.14b). Since the typical value of $R_{off}$ lies between an hundred kilo-ohms and hundreds of mega-ohm (and both extremes can be measured on a single diode, depending on the temperature [45]), its effect is negligible and the resistance is not added to the circuit.

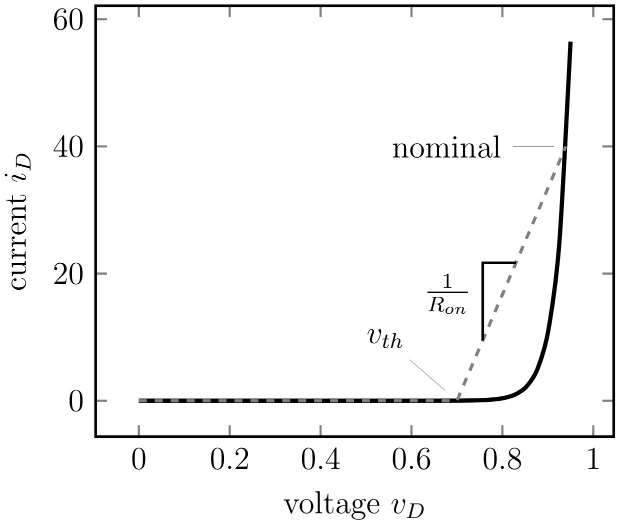


**Figure 2.13: piecewise linear approximation of a diode, with threshold voltage and on-state resistance**

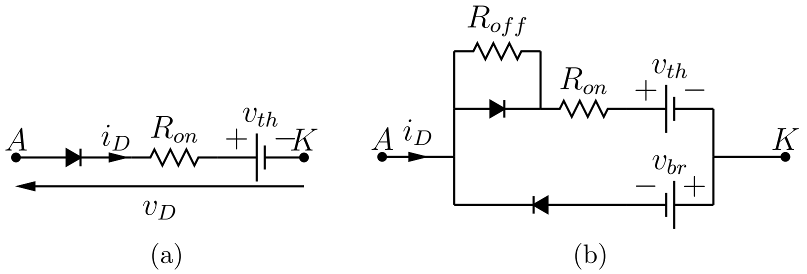


**Figure 2.14: Diode models with imperfections (a) with on-state resistance $R_{on}$ and threshold voltage $v_{th}$ (b) with breakdown state added**

### Reverse recovery

If the diode is placed in a switching circuit, a phenomenon known as *reverse recovery* may occur. When the diode is switched to the off-mode, a reverse current appears for a short time while the excess carrier charges are swept out of the device [46, 47], as shown in Figure 2.15a. The amplitude $I_{rr}$ and the duration $t_{rr}$ of this effect depends on the $\frac{\mathrm{d}i}{\mathrm{d}t}$ forced across the diode and on the quantity of charge $Q_{rr}$, which itself depends on the diode structure and on the current before the turn-off. So-called *fast recovery* diodes have a recovery time of a few tens of nanoseconds [48], while slower diodes may exhibit this behavior for a few microseconds. The underlying dynamics are not trivial, and multiple authors have demonstrated modeling methods for this effect [49–51]. One of the simplest, taken from [51], uses an inductor (to model the $\frac{\mathrm{d}i}{\mathrm{d}t}$ dependence) and a controlled current source (Figure 2.15b). This effect, while impressive, is generally not taken into account when performing system-level simulation since we expect the designer to choose diodes with a recovery time much smaller than the characteristic times (time constants and switching periods) of the circuit. Moreover, the reverse recovery can only be modeled

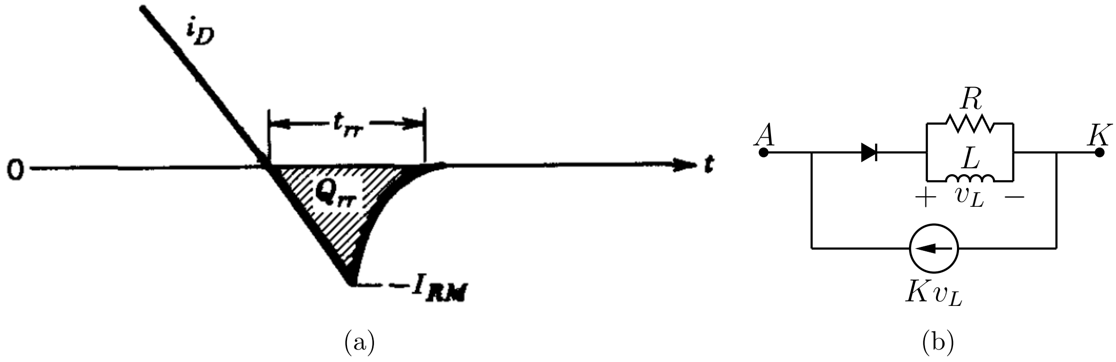


**Figure 2.15: Diode reverse recovery (a) evolution of $i_D$ (taken from [46]) (b) Example of equivalent circuit, where $R, L, K$ are parameters that depend on the diode and on the slope of the current**

when the fundamental time-step of the solver is short compared to the recovery time, which means the solver itself must very fast when modeling fast recovery diodes.

### 2.4.2 MOS transistor models

**Basic first quadrant model**

Since its creation in the 1970's, the MOS transistor (Figure 2.16a) has become the device of choice for switched-mode power supplies [52]. The main advantages of this transistor are its very high switching frequency capabilities and its low switching losses. Combined with the fact that the device is easily adapted to bidirectional operation, it is easy to see the reasons of its widespread use.

The transistor is controlled by its source-to-gate voltage $v_{GS}$, and measuring the output characteristics $i_D = f(v_{DS})$ for different values of the gate voltage produces the well known curves of Figure 2.17.

When $v_{GS}$ is below a (device-dependent) threshold, no current flows across the transistor. This corresponds to the off-state of the device, also called *cutoff region.* If the gate voltage voltage is high enough and the drain current is low enough, the transistor is placed in the on-mode, also called ohmic region: the characteristics may be approximated by a straight line whose slope depends on the gate voltage. The on-state resistance $R_{DSON}$ is defined as the inverse of the slope of the linear approximation, and corresponds to the resistance of the semiconductor channel inside the transistor (plus the resistance of the electrical contacts). The third region, called active region, is generally not used in power converters (but see *Transistor desaturation* in the following sections).

Assuming that the transistor operates either in cutoff region or in ohmic region, it can be modeled by an ideal controlled binary switch placed in series with a resistance whose value is equal to $R_{DSON}$ (Figure 2.16c). Note that, in many cases, the resistance is very small when compared to the other impedances of the circuit and may be neglected, as in Figure 2.16b.

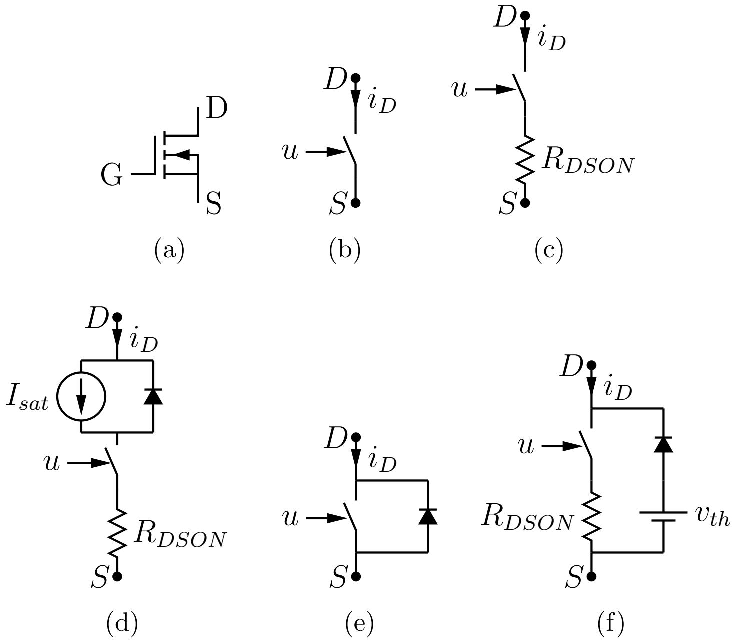


**Figure 2.16: Various models for the MOS transistor (a) Symbol of the MOS (b) ideal lossless MOS (c) with on-state resistance (d) with saturation current (e) ideal MOS with body diode (f) with body diode and on-state resistance**

### Third quadrant operation

The characteristics of the transistor are symmetrical with regards to the origin: the approximation $i_D = \frac{v_{DS}}{R_{DSON}}$ still holds for negative values of $V_{DS}$. However, thanks to their fabrication process, power MOSFETs include a body (parasitic) diode between the source and the drain. This diode is often used as a freewheeling device, allowing the designer to use fewer components in the circuit.

The diode is easily added to the model of the transistor, either in an idealized version (Figure 2.16e) or by taking losses into account (Figure 2.16f).

### Transistor desaturation

If, for a given $v_{GS}$, the current in the transistor becomes too high, the device switches from the ohmic region to the active region. In this region, the transistor behaves as a current sources whose value is imposed by the characteristics. This effect, known as *transistor desaturation* [1], can be taken into account by adding a current source in parallel with a diode to our model 2.16d. Note that, during normal operation, the transistor should stay in the ohmic region and not enter this state. There is no need to add this effect to the model, unless we want to simulate the limitation of accidental over currents by the MOSFET itself.

[1] The term *desaturation* may seem like as an odd choice at first, since the active region of a MOSFET is also known as the saturation region. This term actually originates from bipolar transistors, where the saturation corresponds to the on-state region

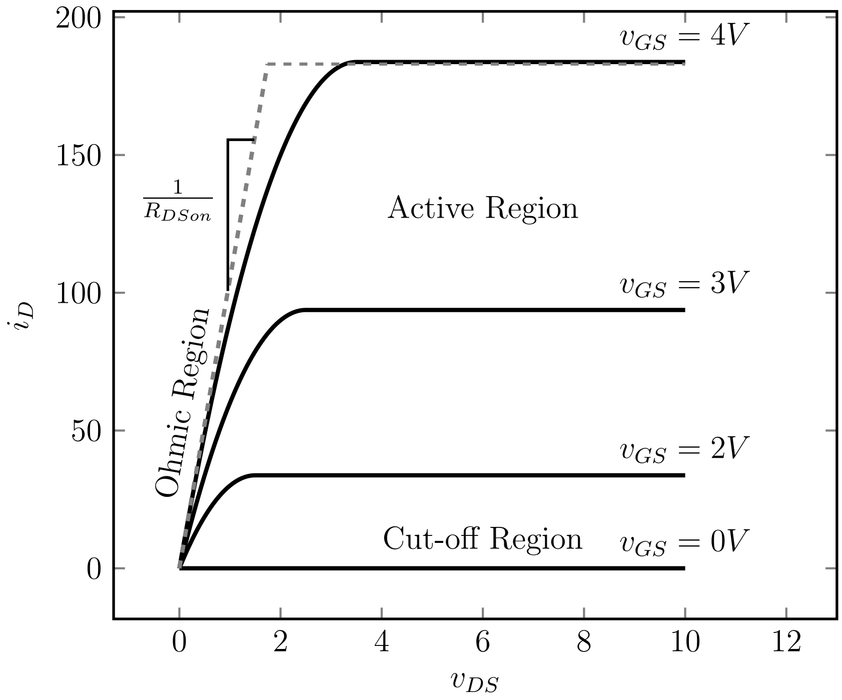


**Figure 2.17: Output characteristics of a typical MOS transistor for various values of $v_{GS}$, and its piecewise linear approximation for $v_{GS} = 4V$**

### 2.4.3 Bipolar transistors and IGBT models

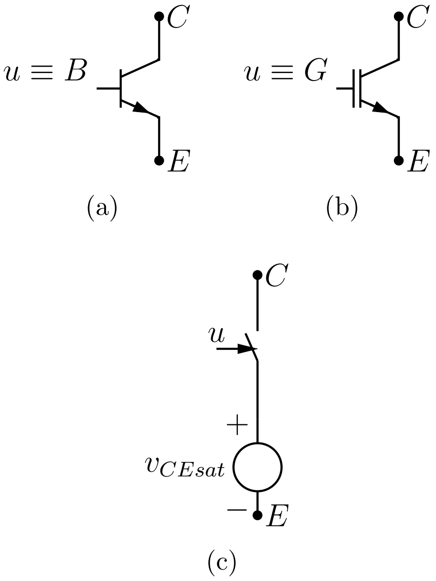
Since the development of the MOSFET and IGBT, power bipolar transistors have lost the major part of their market share; they (Figure 2.18a) are still used in applications requiring higher voltages and/or currents [46]. The typical bipolar transistor model in represented on Figure 2.18c, where a voltage source $V_{CEsat}$ is added in series with the ideal controlled switch to model the saturation voltage [35]. This model does not add any major difficulty to the simulation.
The Insulated Gate Bipolar Transistor (IGBT) is a very popular device combining the use of control of the MOSFET and the higher current density of the bipolar transistor. Since we do not model the control side of the switches (only the power ports are put into equations), the model for this device is the same as the one used for the bipolar transistor.

### 2.4.4 Thyristor models

Thyristors where one of the first (semi-) controlled devices, adding the ability to prevent a diode from switching-on until a positive current is applied to its gate terminal (Figure2.19). Hence, its on-off model is the same as the diode. Meanwhile, the conditions from switching the device to the on-state are:

- a positive cathode-to-anode voltage *and*
- a direct gate-cathode current

All the advanced models studied in section 2.4.1 can be used with the thyristor.



(a) (b)

(c)

**Figure 2.18: (a) bipolar transistor (b) Insulated Gate Bipolar Transistor (c) simple model for both transistors with the saturation voltage $V_{CEsat}$**

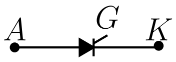


**Figure 2.19: A thyristor and its three terminals. The device acts as a diode whose off-to-on switching can be triggered by the gate (G) current**

### 2.4.5 Magnetically coupled circuits

#### Mutual inductors

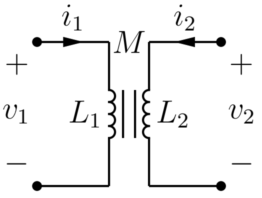


**Figure 2.20: Coupled inductors, with self-inductances $L_1, L_2$ and mutual inductance $M$. The currents correspond to the traditional convention : if a positive current enters the inductor according to the defined convention, it induces a positive voltage across the other coil.**

Multiple coils are said to be coupled when the magnetic flux generated by one of the coils is, at least partially, perceived by the others. Translated into an electrical circuit, this gives rise to the concept of mutual inductances, where the back emf of a coil is influenced by the current across coils. Let us take the basic two-winding mutual inductance of Figure

2.20: the equations governing the voltages and the currents are written as

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} L_1 & M \\ M & L_2 \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}i_1}{\mathrm{d}t} \\ \dfrac{\mathrm{d}i_2}{\mathrm{d}t} \end{bmatrix} \tag{2.49}$$

where $L_1$ and $L_2$ are the self-inductance of the primary and secondary sides, and $M$ is the mutual inductance linking the two windings (a higher $M$ increases the effect of the current in a winding on the voltage the other winding). Inverting (2.49) allows us to obtain the state-space equations of the inductor:

$$\begin{bmatrix} \dfrac{\mathrm{d}i_1}{\mathrm{d}t} \\ \dfrac{\mathrm{d}i_2}{\mathrm{d}t} \end{bmatrix} = \frac{1}{L_1L_2 - M^2} \begin{bmatrix} L_2 & -M \\ -M & L_1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{2.50}$$

This form assumes $L_1L_2 - M^2 \neq 0$. The coupling factor $k$ is defined as

$$M^2 = k^2 L_1 L_2 \tag{2.51}$$

and represents the efficiency of the magnetic coupling. For practical purposes, we have $0 \leq k < 1$, with $k = 0$ corresponding to the absence of coupling between the two windings. The concept of mutual inductance can be expanded to include circuits with three or more coupled inductors, leading to the more general form

$$\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} L_{11} & \dots & L_{1n} \\ \vdots & \vdots & \vdots \\ L_{n1} & \dots & L_{nn} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}i_1}{\mathrm{d}t} \\ \vdots \\ \dfrac{\mathrm{d}i_n}{\mathrm{d}t} \end{bmatrix} \tag{2.52}$$

### Ideal transformers

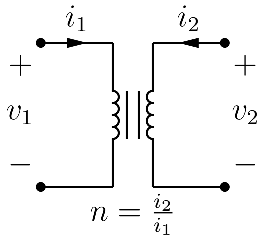


**Figure 2.21: An ideal transformer, with a transformation ratio $n = \frac{n_1}{n_2} = \frac{i_2}{i_1}$ where $n_1$ and $n_2$ are the number of turns at the primary and secondary windings**

An ideal transformer, as shown on Figure 2.21, is a special case of mutually coupled inductances with $k = 1$. It establish to following identities:

$$\begin{aligned} i_2 &= n i_1 \\ v_2 &= \frac{v_1}{n} \end{aligned} \tag{2.53}$$

where $n$ is the transformer ratio equal to $n = \frac{n_1}{n_2}$ where $n_1$ and $n_2$ are the number of turns at the primary and secondary windings. This definition, while useful, introduces multiple problems. First, the fact that this equation is only true for *non-continuous* signals is completely hidden. Secondly, it raises the question of *how* to introduce the equations into the state-space system. Should we represent the effect on one winding to the other as a current source or as a voltage source?

In practice, the answer depends on the circuit : if one winding is placed in series with a current source or an inductor, then we have to select the voltage source representation, while a current source representation is required when a voltage source or a capacitor is connected in parallel with a winding.

Because of these problems, we recommend the use of a model based on a practical transformer instead

### Practical transformers

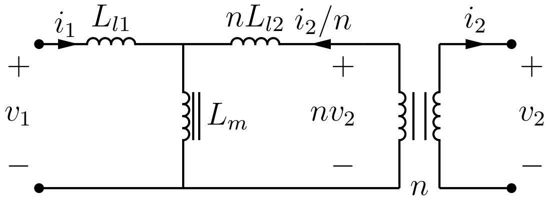


**Figure 2.22: Equivalent circuit of a non-ideal transformer without Joule losses seen from its primary side, with a transformation ratio $n = \frac{i_2}{i_1}$, leakage inductances $L_{l1}, L_{l2}$ and magnetization inductance $L_m$**

The classical representation of a non-ideal two-windings transformer is shown in Figure 2.22. This circuit includes all impedances as seen from the primary side, and contains an ideal transformer with a transformation ration $n$

- $L_{l1}$ and $L_{l2}$ are the leakage inductances, representing the flux generated from one winding and not seen by the others.
- $L_m$ is the magnetizing inductance, representing the couplings

The values of these inductances are, along with the transformation ratio, the main parameters describing the transformer and are typically found in the datasheets. To solve this circuit, let us apply Kirchhoff's laws to both sides :

$$v_1 = L_{l1}\frac{\mathrm{d}i_1}{\mathrm{d}t} + L_m\frac{\mathrm{d}i_1 + \frac{i_2}{n}}{\mathrm{d}t} nv_2 \quad = nL_{l2}\frac{\mathrm{d}i_2}{\mathrm{d}t} + L_m\frac{\mathrm{d}i_1 + \frac{i_2}{n}}{\mathrm{d}t} \tag{2.54}$$

After rearranging the terms:

$$\begin{bmatrix} v_1 \\ \\ v_2 \end{bmatrix} = \begin{bmatrix} L_m + L_{l1} & \frac{L_m}{n} \\ \\ \frac{L_m}{n} & L_{l2} + \frac{L_m}{n^2} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}i_1}{\mathrm{d}t} \\ \frac{\mathrm{d}i_2}{\mathrm{d}t} \end{bmatrix} \tag{2.55}$$

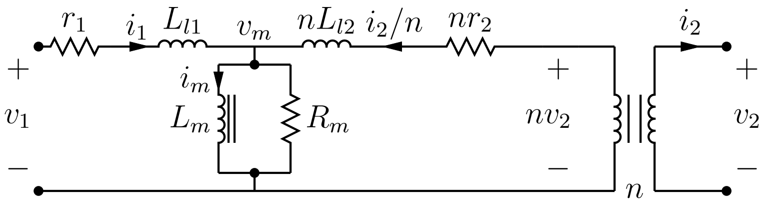


**Figure 2.23: Equivalent circuit of a non-ideal transformer seen from its primary side, with added resistors $r_1, r_2$ representing the resistance of the windings, and $R_m$ symbolizing losses due to eddy currents**

which is equivalent to the self- and mutual inductances described by (2.49). We may also introduce the equivalent resistances of the windings, as shown in Figure 2.23. To solve that circuit, we have to introduce $i_m$ as a third state variable (since $i_1 + i_2 = i_m$ is not true anymore). Defining $v_m$ as the voltage of the magnetizing branch, we have

$$\begin{aligned} v_1 &= L_{l1}\frac{\mathrm{d}i_1}{\mathrm{d}t} + r_1 i_i + v_m \\ nv_2 &= nL_{l2}\frac{\mathrm{d}i_2}{\mathrm{d}t} + nr_2 i_2 + v_m \\ v_m &= L_m\frac{\mathrm{d}i_m}{\mathrm{d}t} = R_m(i_1 + \frac{i_2}{n} - i_m) \end{aligned} \tag{2.56}$$

Which is put into state-space form by isolating the time differentials of the currents:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_1 \\ i_2 \\ i_m \end{bmatrix} = \begin{bmatrix} -\frac{r_1+R_m}{L_{l1}} & -\frac{R_m}{nL_{l1}} & \frac{R_m}{L_{l1}} \\ -\frac{R_m}{nL_{l2}} & -\frac{R_m+n^2r_2}{n^2L_{l1}} & \frac{R_m}{nL_{l2}} \\ \frac{R_m}{L_m} & \frac{R_m}{nL_m} & -\frac{R_m}{L_m} \end{bmatrix}\begin{bmatrix} i_1 \\ i_2 \\ i_m \end{bmatrix} + \begin{bmatrix} \frac{1}{L_{l1}} & 0 \\ 0 & \frac{1}{L_{l2}} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{2.57}$$

Bigger transformers (three phases or more) can be put into state-space equations in a similar fashion. In [53], the authors put a three-phase transformer into equations by converting its magnetic structure to an electrical circuit.

## 2.4.6 Electromechanical devices

### Introduction

Electrical motors and generators are an important part of power electronics. Indeed, due to their presence in electric cars, trains and other means of transportation, they are one of the typical electrical loads connected to the power conversion system. Hence, it seems logical to try to emulate their behavior as well. The biggest problem we will face is their non-linear behavior, due to the magnetic coupling between their windings, introducing the angle between the windings as an additional variable.

### DC motors and generators

The direct current drive does not contain non-linear terms when we use its most simple model. Hence, they may be modeled using the same sets of equations as the circuit itself.

The electrical (primary) port of the motor is described by

$$v_1 = L_1 \frac{\mathrm{d}i_1}{\mathrm{d}t} + r_1 i_1 + E \tag{2.58}$$

where $E$ is the back emf due to the rotation of the motor. If the excitation current of the motor is constant, or in the case of a permanent magnet motor, we have

$$E = k_\omega \omega \tag{2.59}$$

where $k_\omega$ is the speed constant of the motor. The mechanical side of the motor is described by the movement equation

$$J \frac{\mathrm{d}\omega}{\mathrm{d}t} = T_{em} - T_r = k_\omega i_1 - T_r \tag{2.60}$$

where $J$ is the rotational inertia of the motor and its mechanical load, $T_{em}$ is the electromechanical torque proportional to the armature current $i_1$ and $T_r$ is the resistive torque. The simplest model for $T_r$ is a torque linearly dependent of $\omega$, leading to

$$J \frac{\mathrm{d}\omega}{\mathrm{d}t} = k_\omega i_1 - k_r \omega \tag{2.61}$$

The complete drive may be modeled using controlled voltage sources and $RL$ circuits

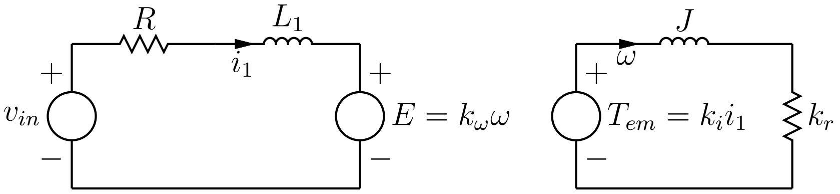


**Figure 2.24: Equivalent circuit of a DC drive, modeling the mechanical behavior as an electrical circuit**

as shown in Figure 2.24. If the drive is used as a generator, $T_r$ can be modeled using a voltage source instead, symbolizing the applied torque.

### AC drives

Three-phases AC motors are widely used thank to their higher power density when compared to DC drives. Thus, it makes sense to write the equations controlling these drives for real-time emulation purposes.
Before writing the dynamical equations governing the AC drive state-space according to its internal structure, we first introduce the following variables:

- $i_s = \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix}$ and $i_r = \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix}$ are respectively the stator and rotor currents
- $v_s = \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} v_a^+ - v_a^- \\ v_b^+ - v_b^- \\ v_c^+ - v_c^- \end{bmatrix}$ are the input voltages at the terminals of each of the three stator phases. If the phases are delta connected, we have $v_a^+ = v_b^-$, $v_b^+ = v_c^-$ and $v_c^+ = v_a^-$

- $v_r = \begin{bmatrix} v_A \\ v_B \\ v_C \end{bmatrix}$ are the voltages measured at the terminals of the rotor phases.
- $\Phi_s = \begin{bmatrix} \Phi_a \\ \Phi_b \\ \Phi_c \end{bmatrix}$ and $\Phi_r = \begin{bmatrix} \Phi_A \\ \Phi_B \\ \Phi_C \end{bmatrix}$ are the fluxes linkages of the stator and rotor windings

We also assume that the machine is properly balanced (i.e. all three stator phases have the same electrical behavior, same for the the three rotor phases) and the magnetic materials are linear (i.e. no saturation). Under these assumptions, the equations can be written as follows [54]:
We first write the equations linking the current and the fluxes:

$$\begin{aligned} \Phi_s &= L_s i_s + M_{rs} i_r \\ \Phi_r &= M_{sr} i_s + L_r i_r \end{aligned} \tag{2.62}$$

In these equations, $L_s$ and $L_r$ and the stator and rotor inductance matrices, defined as

$$\begin{aligned} L_s &= \begin{bmatrix} L_s & M_s & M_s \\ M_s & L_s & M_s \\ M_s & M_s & L_s \end{bmatrix} \\ L_r &= \begin{bmatrix} L_r & M_r & M_r \\ M_r & L_r & M_r \\ M_r & M_r & L_r \end{bmatrix} \end{aligned} \tag{2.63}$$

while $M_{sr} = M_{rs}^T$ represents the magnetic coupling between the stator and the rotor

$$M_{sr} = M_o \begin{bmatrix} \cos(\theta) & \cos(\theta + 2\frac{\pi}{3}) & \cos(\theta - 2\frac{\pi}{3}) \\ \cos(\theta - 2\frac{\pi}{3}) & \cos(\theta) & \cos(\theta + 2\frac{\pi}{3}) \\ \cos(\theta + 2\frac{\pi}{3}) & \cos(\theta - 2\frac{\pi}{3}) & \cos(\theta) \end{bmatrix} \tag{2.64}$$

where $\theta$ is the electrical phase shift between the static frame and the rotating frame. This matrix is the main source of non-linearity in the system, as it introduces trigonometric relations in the state-space. The fluxes are then eliminated by writing the voltage equations for the stator and the rotor

$$\begin{aligned} v_s &= R_s i_s + \frac{\mathrm{d}\Phi_s}{\mathrm{d}t} \\ v_r &= R_r i_r + \frac{\mathrm{d}\Phi_r}{\mathrm{d}t} \end{aligned} \tag{2.65}$$

where $R_s$ and $R_r$ are diagonal matrices containing the total resistance of the windings and the associated electrical connections. Assuming that the drive is fed electrically from the stator only, we have $v_r = 0$.
The electromechanical torque $T_{em}$ is given by

$$T_{em} = i_s^T \frac{\mathrm{d}M_{sr}}{\mathrm{d}\theta} i_r \tag{2.66}$$

Again, this equation is clearly non-linear. Finally, we have the load inertia equations

$$
\begin{aligned}
\frac{\mathrm{d}\theta_m}{\mathrm{d}t} &= \omega_m \\
J\frac{\mathrm{d}\omega_m}{\mathrm{d}t} &= T_{em} - T_r
\end{aligned}
\tag{2.67}
$$

where

- $\theta_m$ is the mechanical angle, related to the electrical angle by the relation $\theta = n\theta_m$, where $n$ is the number of pole pairs of the machine
- $\omega_m$ is the angular speed of the rotor
- $J$ is the rotating load inertia
- $T_r$ is the resistive torque of the load

To properly implement these equations, a non-linear module must be developed. In the current state of this work, we have chosen to avoid the implementation of non-linear systems.

However, this aspect could clearly be studied in the future. Since the signals inside electromechanical devices are relatively slow when compared to signals measured in the power conversion system (i.e. the mechanical time-constants are typically much larger than the electrical time-constants) , this non-linear module could be operated in parallel with the main solver at a lower refresh rate.

## 2.5 Transitional model of power converters using hybrid automata

### 2.5.1 Hybrid automata definition

At this point, it becomes clear that there are two types of dynamics inside a power circuit

- continuous dynamics, modeled by the state equations, describing the time-based evolution of the currents and voltages once the configuration of the switches (the state of all switches) is set
- discrete dynamics describe the sudden changes of topology according to the internal or external control laws of the switches

These two evolutions were kept separate until now, but we need to fuse them to provide an unified model of the converter. With this model, we obtain a mathematical representation that may be analyzed and compiled for the real-time platform.

The hybrid system paradigm is a very convenient approach, and was developed to model systems with continuously varying variables whose evolution laws depend on a number of discrete conditions [55,56]. In the past, multiple authors have suggested the use of hybrid systems to model power converters for the purpose of designing switched-mode controllers [57–60]. More crucially, a team from the *Massachusetts Institute of Technology* has also explicitly used hybrid systems systems in a real-time simulator of power converters [61,62], but the definition of an hybrid system is very generic and many emulator systems use a representation that is similar in every way, except for the name. An hybrid system is aptly described by a directed graph : the hybrid automaton representation. In this graph, each node (or vertex) corresponds to a given topology of the circuit, while the edges are the discrete transitions between the topologies due to a switch changing its state.

### 2.5.2 Circuit without discrete inputs

Mathematically, a hybrid automaton is defined by a collection $H = (S, x, f, D, E, G)$. The continuous state variables are represented by the $x = (x_1, \ldots, x_m) = \mathbb{R}^m$ vector and the continuous independent inputs (ie. the sources) by $u = (u_1, \ldots, u_n) = \mathbb{R}^n$. Meanwhile, $S = (s_1, \ldots, s_n)$ is a set of discrete variables describing the current topology of the system. These variables correspond to the binary state of the switches: each $s_i$ represents the state of a single switch (ie. $s_i = 1$ if the $i^{th}$ switch is currently *on*). Each combination (or configuration) of the switches defines a topology, and is represented by a node on the graph.

All these variables are linked by $f : S \times x \times u \to x$, a vector field describing the evolution of the state variables as a function of their current value, of the configuration of the switches and of the independent inputs:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(s, x, u, t) \tag{2.68}$$

For a given topology (i.e. for a given value of $s$), this equation corresponds to the state equation of the associated linear system. The validity domain of (2.68) for a given configuration of the switches is described by $D : S \to P(x)$, which corresponds to the conditions on the diodes, repeated here for convenience

$$\begin{aligned} i_D &> 0 \quad \text{if in the on-mode} \\ v_D &\leq 0 \quad \text{if in the off-mode} \end{aligned} \tag{2.69}$$

A crucial corollary of this formula is that the point $(v_D, i_D) = (0, 0)$ has been chosen to be part of the off-state. Each of the modes receives its own set on inequalities $D(s_i) \subset \mathbb{R}^m$ describing its domain as a polytope [2].

The links between the topologies are represented by $E \subseteq S \times S$, which is an $2^n$ by $2^n$ array that indicates if there is a possible direct transition from a given topology to another one. In the associated graph, each transition is represented by a directed edge between the nodes corresponding to the two topologies, and $G$ assigns a *guard* to each transition. This guard correspond to a condition that must be validated before crossing the transition.

When the system is in a given topology, it evolves according to its linear evolution laws until the state vector goes outside of the validity domain. At this point, the transition whose guard is validated is taken and the system jumps to a new topology with new dynamics.

[2] A polytope is a generalization of the concept of polygons and polyhedrons to an $n$ dimensional space. The convex polytope $P \subset \mathbb{R}^n$ is mathematically defined by the intersection of a number $m$ of half-spaces $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, where $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{n+m}$

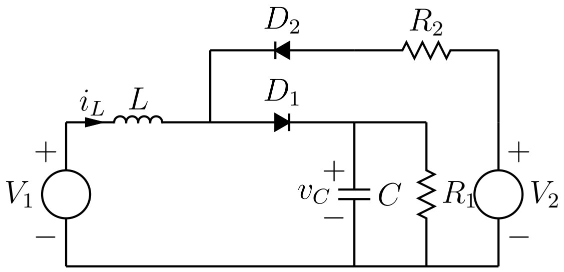


**Figure 2.25: Example circuit containing two diodes**

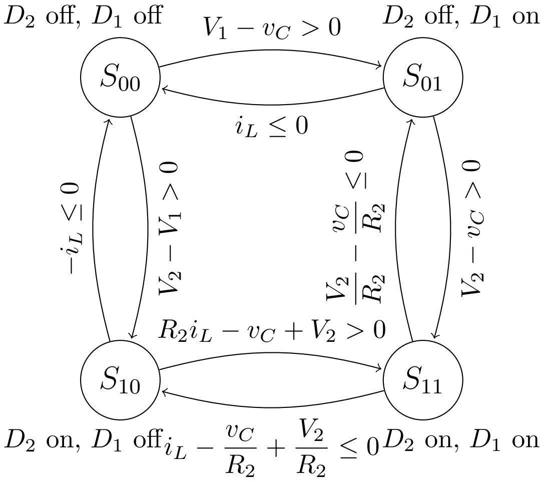


**Figure 2.26: Topology graph for the example circuit of figure 2.25**

**Example 2.** To illustrate these concepts, we will use the circuit of figure 2.25. This circuit contains two diodes, and hence four different topologies with their own dynamics and validity domain: $S = (d_1, d_2) = \{(0,0), (0,1), (1,0), (1,1)\}$ where $(d_1, d_2)$ contains the current state of each of the two diodes. The two state variables $i_L$ and $v_C$ are described by :

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} = A_{(d_1,d_2)} \begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} + B_{(d_1,d_2)} \begin{bmatrix} V_1(t) \\ V_2(t) \end{bmatrix} \tag{2.70}$$

where $(A_{(d_1,d_2)}, B_{(d_1,d_2)}) = f(d_1, d_2)$ controls the dynamics of the circuit depending on the topology. The validity domain of each topology is obtained by the calculation of (2.69) :

$$\begin{array}{llr} D(0,0) = v_{d1} \leq 0 & \cap\, v_{d2} \leq 0 & = (E_1 - v_C \leq 0) \cap (E_2 - E_1 \leq 0) \\ D(1,0) = i_{d1} > 0 & \cap\, v_{d2} \leq 0 & = (i_L > 0) \cap (E_2 - v_C \leq 0) \\ D(0,1) = v_{d1} \leq 0 & \cap\, i_{d2} > 0 & = (E_2 - R_2 i_L - v_C \leq 0) \cap (-i_L > 0) \\ D(1,1) = i_{d1} > 0 & \cap\, i_{d2} > 0 & = (i_L - \dfrac{E_2 - v_C}{R_2} > 0) \cap (\dfrac{v_C - E_2}{R_2} > 0) \end{array} \tag{2.71}$$

The graph corresponding to the hybrid automaton is constructed by representing each topology as a node, as shown in figure 2.26 where $S_{01}$ is the topology corresponding to $(d_1, d_2) = (0, 1)$. Each transition (i.e. each edge connecting two nodes) corresponds to the change of state of a single diode. The conditions (guard) attached to the edges are obtained by inverting the inequality imposed by the corresponding diode in (2.71). For

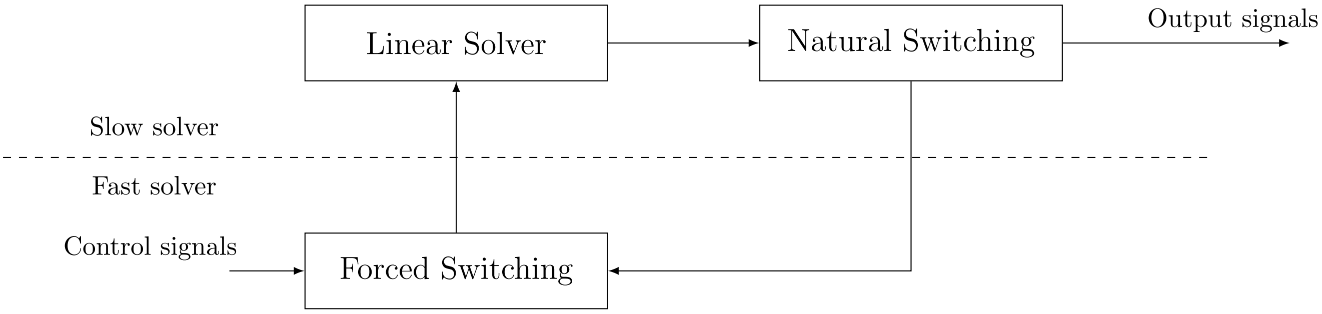


**Figure 2.27: Separation of the study of the two types of transitions, which allows for two different sampling times**

instance the transition going from $S_{00}$ to $S_{01}$ is taken when the system goes outside of the boundary defined by $(V_2 - V_1 \leq 0)$, hence the associated condition is $(V_2 - V_1 > 0)$.

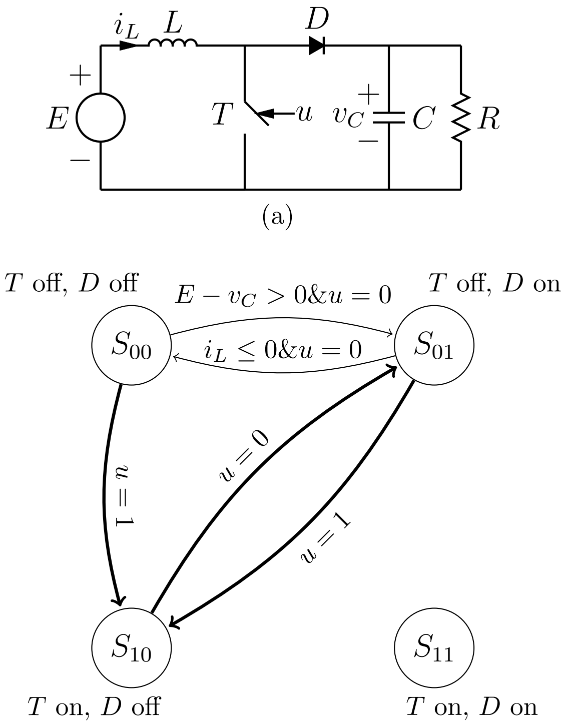
Describing a power converter by an hybrid system not only provides us with a formal mathematical model which is easily ported to a computer program, but also allow us to benefit from the studies that have been made on this subject in the past. With these tools in hand, we can study how to build the automaton and what kind of information can be extracted from it.

### 2.5.3 Circuit with discrete inputs

A limitation of this model is that, under the chosen definition, only topology changes due to the continuous evolution of the state variables are tracked (ie. the guards are a function of the states and/or of the inputs). A more general description allows discrete input signals, such as those used to control the transistors and the other active switches, to be included in the domain definition and the guards condition [63]. The transitions are now classified either as

- *natural* if they correspond to a change in the continuous signals
- *forced* if they correspond to a change in a discrete control signal

The way these two types are handled depends on the chosen simulation procedure. Almost all real-time solvers based on the hybrid system approach evaluate both types at the same time [21, 61, 62, 64] and find the correct topology in a single step. Instead, we propose the usage of two different modules as shown on figure 2.27. This separation allows the individual exploration and optimization of the two phenomena, which may (hopefully) reduce the overall resource usage. Furthermore, different sampling times may be used for the two parts. This second point is particularly interesting for circuits whose switches are driven by high-frequency control signals (ie. signals whose frequency is not significantly lower than the sampling rate). For instance, let us consider a circuit sampled at $1MHz$ and controlled by a $100kHz$ PWM. If the control signals are sampled at $1MHz$, we only have a precision of 10% on the duty cycle. On the other hand, sampling the control signals at a higher rate provides a much better resolution and a more accurate representation of the input.



(a)

(b)

**Figure 2.28: Boost converter and its full transition graph. The bold arrows correspond to forced transitions and the thinner arrows to natural transitions**

A more precise study will be carried in section 5.4.6, during the design of the real-time platform.

**Example 3.** The graph of figure 2.28b represents the boost converter. Each topology is a node of the graph noted $S_{ud}$, where $u$ is the state of the transistor and $d$ the state of the diode. The forced transitions (i.e. the edges of the graph), represented by bold arrows, carry a guard condition that only includes the switch control signal $u$. Both natural transitions carry a guard composed of a condition on the state variables and on the value of the control signal ($u = 0$). This graph may be described in a more explicit manner :

- if the controlled switch is closed, the automaton goes to topology $S_{10}$
- if the controlled switch is opened, the automaton goes to topology $S_{01}$
- while the controlled switch remains open, the system is controlled by the natural transitions

The topology $S_{11}$ is not reached by any transition and hence is said to be *unreachable.* This is to be expected since the transition from $S_{10}$ to $S_{11}$ would have a condition $v_C < 0$, which is impossible since the capacitor voltage is described by $C\dfrac{\mathrm{d}v_C}{\mathrm{d}t} = i_D - v_C/R$, where the diode current $i_D \in \mathbb{R}^+$. In the event where $i_D = 0$, the voltage voltage will converge exponentially to zero without taking a negative value. Furthermore the diode current is equal to zero when in $S_{11}$, which means that diode would immediately be switched off again.

This example, while not complete, illustrates the separation property mentioned previously: the two types of transitions are independent and may be evaluated by two separate entities.
As a conclusion, we have shown that the hybrid system description is very useful to translate the circuit into a mathematical object, and we will use it extensively in the next chapters.

## 2.6 Simulating the power converter

### 2.6.1 Introduction

The concepts introduced in section 2.5 provide us with a powerful description of the power converter dynamics. Using this tool, we can track how the circuit reacts to the evolution of the internal signals or to changes in the control signals of the switches. If we write these rules as a program, we obtain a simulator of the circuit.
The framework provided by automata gives us a natural separation between

- the linear subsystem, simulating the evolution of the circuit while it stays in the same topology
- the switching module that controls the topology-to-topology evolution (and hence the state matrices) according to a set of rules based on the diode voltages and currents as well as the control signals of the active switches

The role of the linear subsystem is to solve the state-space equations

$$\begin{aligned} \frac{\mathrm{d}x(t)}{\mathrm{d}t} &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{2.72}$$

using the chosen discrete-time solver. While the system stays in its current node, the state matrices are constant and allow us to simulate both the states and the outputs. These two equations may be written in the serial form

$$\begin{aligned} x(k+1) &= f(x(k), u(k)) \\ y(k+1) &= Cx(k+1) + Du(k) \end{aligned} \tag{2.73}$$

or in the parallel form

$$\begin{aligned} x(k+1) &= f(x(k), u(k)) \\ y(k+1) &= Cx(k) + Du(k) \end{aligned} \tag{2.74}$$

where $f(x, u)$ is a linear function that depends on the solver and on the state matrices. These two forms are represented in Figure 2.29a and 2.29b, where $z^{-1}$ is the unit delay (i.e. a memory whose output is equal to the input present during the previous time-step). The parallel solver introduces a lag of one time step between the update of a state and its effect on the outputs, but on the other hand provides a means to compute both

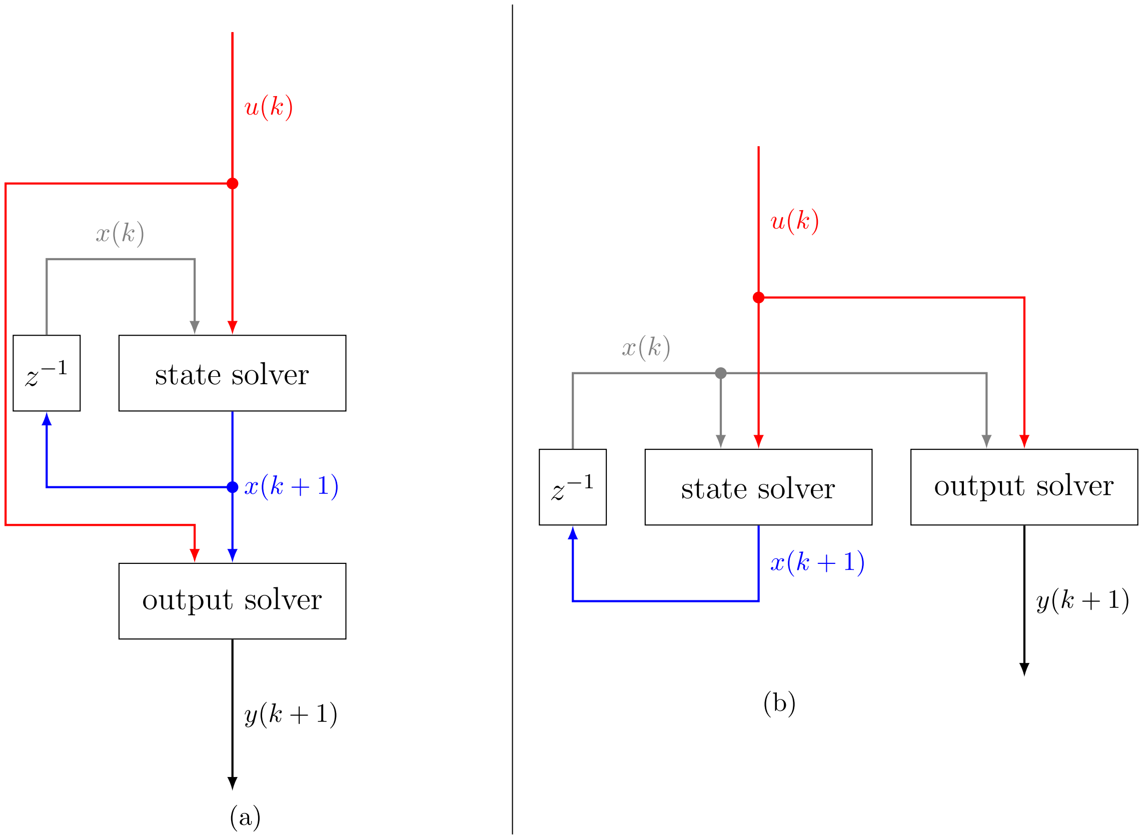


**Figure 2.29: Two versions of the linear solver (a) Serial solver (b) Parallel solver**

sets of signals at the same time using parallel computing methods, reducing the total computation time.

The switching module uses the states and inputs signals to control the diodes and to select the new state matrices to be used in the next iteration. As described in 2.5.3, the circuit will move from the current node of the automaton to another one when the diodes or the transistors change their configuration. To verify if a diode must be switched or not, the switching module must evaluate a set of diode currents and voltages. These signals are written in the same form as the output variables

$$\begin{bmatrix} i_D(t) \\ v_D(t) \end{bmatrix} = Ex(t) + Fu(t) \tag{2.75}$$

Again, the $E$ and $F$ matrices depend on the current topology. The switching module is also able to control the diode in reaction to a change in the state of an active switches (eg. to allow an inductive current to flow).

### 2.6.2 Structure selection

The sub-modules may be arranged in many configurations to obtain the global structure of the real-time simulator. Each of the configuration must include

- the state variable solver

- the output variable solver
- the switching module, which may be separated in its two sub-modules:
  - the natural switching module, which controls the diodes according to the evolution of the linear signals
  - the forced switching module, which controls the diodes according to the evolution of the active switches

### Split structure

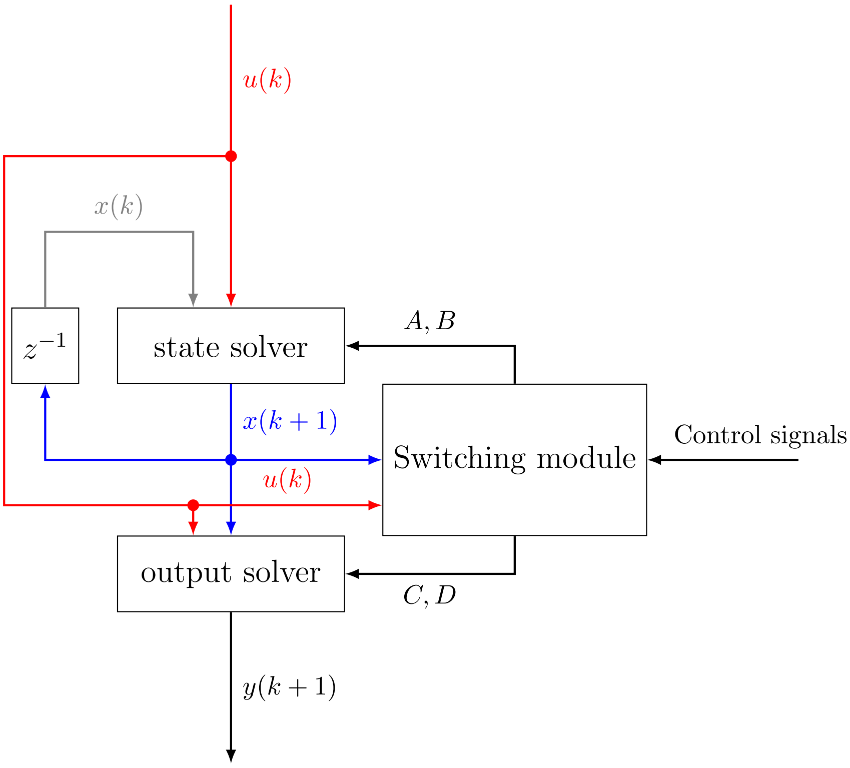
A first example of structure is provided on Figure 2.30a. At the start of each step the state solver uses the previous states $x(k)$ and sources $u(k)$ to compute the new states $x(k+1)$. Once $x(k+1)$ is known, it is fed to the output solver and to the switching engine. Since these modules are independent, they are easily run in parallel to reduce the total latency. Alternatively, the outputs may be computed in parallel with the states if we use their previous value instead of the current one, as shown in Figure (b).
This latter structure is clearly the simplest one, and allows a rather high degree of parallelism. However, it may lead to incoherent signal values. Let us show it using the simple boost converter of figure 2.28a, with the diode in the on-state while the transistor is in the off-state. Let us assume that the inductor current becomes negative, according the state solver. In reaction, the switching module will force the diode into the off-mode, the boost operates in discontinuous conduction-mode. Unfortunately only the matrices $A, B, C, D$ are updated by the switching module for the next step; the states $x(k+1)$ and the outputs $y(k+1)$ remain the same hence the current will stay negative, which may disrupt the switching engine or the external circuits connected to the outputs signals.

### Interleaved structure

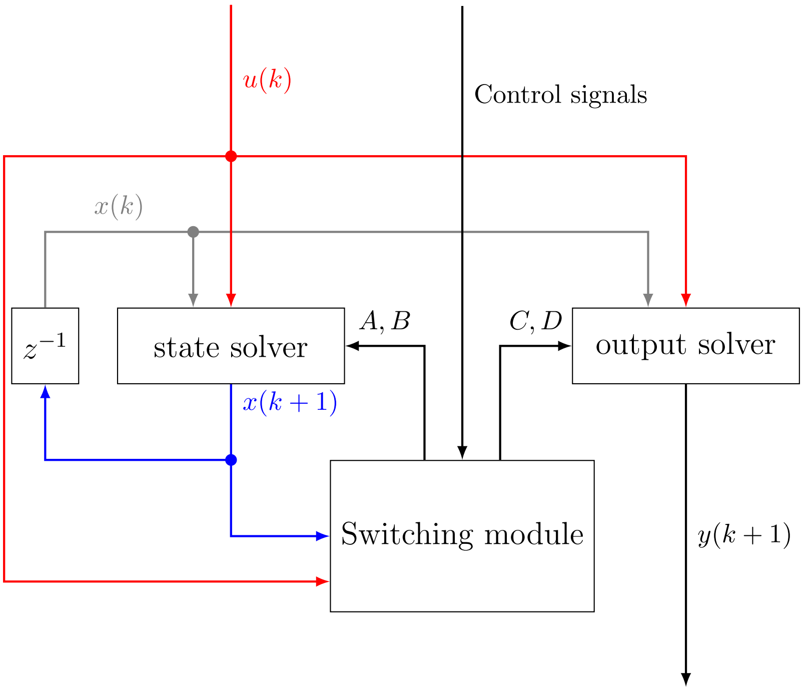
This problem is alleviated by the interleaved structure represented in Figure 2.31. The first difference with the previous structure consists in splitting the switching module in two steps (forced switching and natural switching) interleaved with the state solver. The second difference is the addition of a state adjustment in the output solver, according to the new topology. This allows, for example, to clamp the inductor current to zero should it become negative. Assuming the state solver now computes a *temporary* value $x_T(k+1)$, we adjust the state value $x(k+1)$ using

$$x(k+1) = C_x x_T(k+1) + D_x u(k) \tag{2.76}$$

If the states must not be adjusted, then $C_x$ is the identity matrix while $D_x$ is filled with zeros. Otherwise, these matrices contain the updated coefficients allowing to recompute the states using the other signals as reference. The states modified by this operation are effectively forced states, as described in section 2.2.3 and the $C_x$ and $D_x$ matrices were already introduced in (2.23). Since we need to compute (2.76) using this system, it makes sense to run the output solver in parallel to keep the latency as low as possible.
This process effectively uses the state solver of the previous topology while using the output solver of the new one. The performance of the two methods will be assessed in section 5.2, dedicated to the real-time platform implementation. We will now take a closer look at the switching modules, and study their internal structure.



(a)



(b)

**Figure 2.30: Solver with split structure, where the switching module takes effect on the new time step (a) Serial computation of outputs (b) Parallel computation of the outputs**

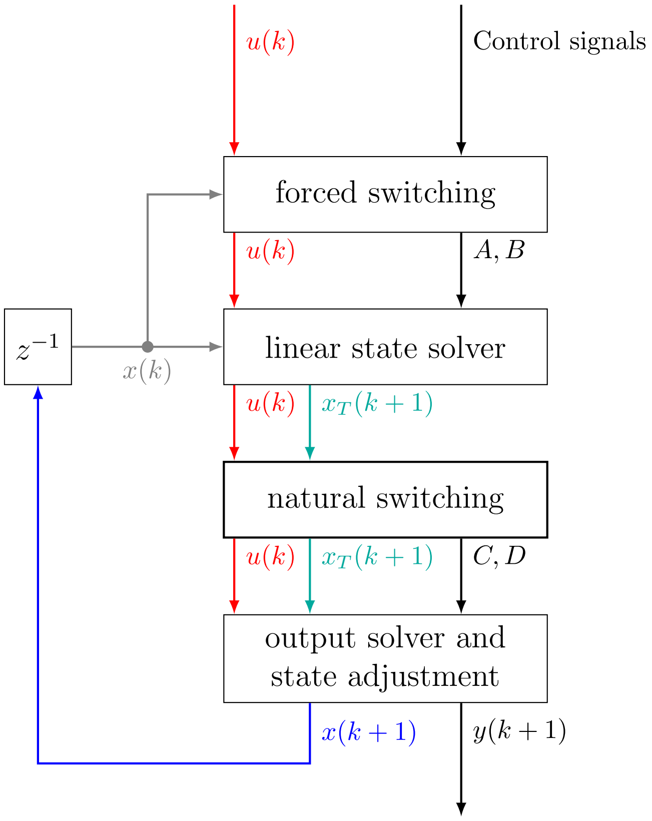


**Figure 2.31: Solver with interleaved structure, where the changes are taken into account during the same time step. The natural switching module (in bold) is shown in details on Figure 2.32**

### 2.6.3 The natural switching module

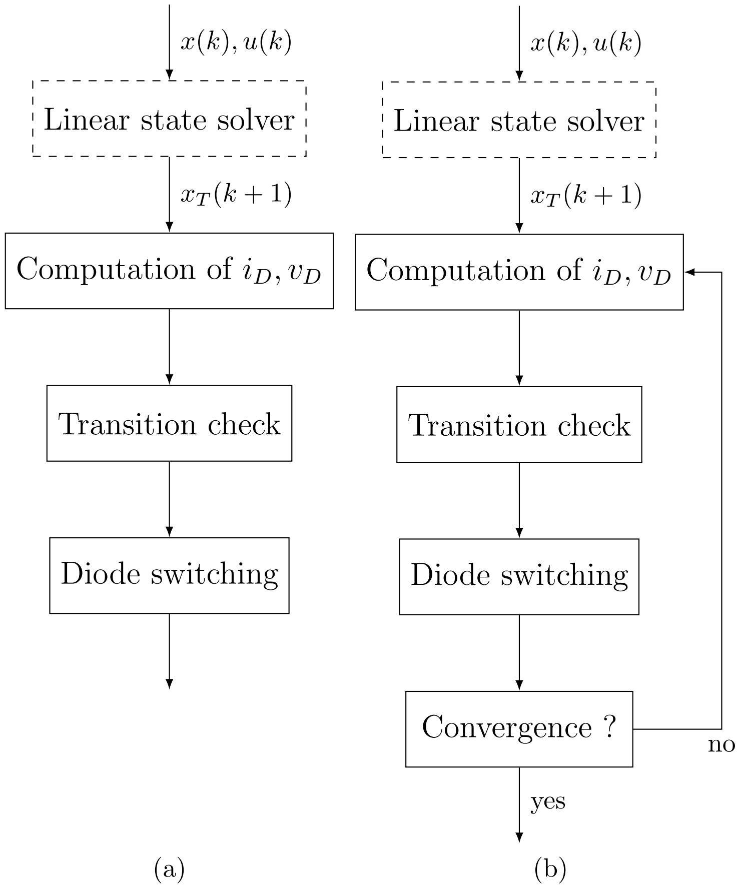
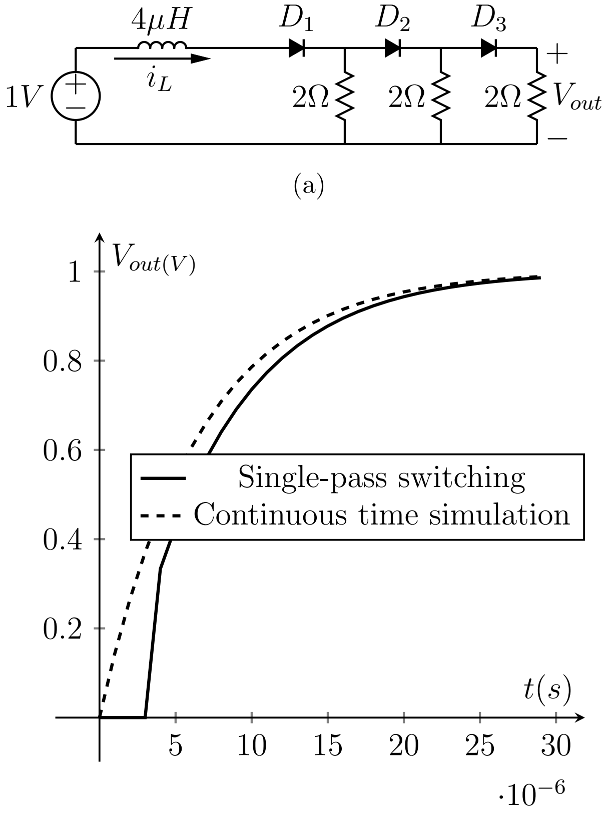


**Figure 2.32: Examples of simulation procedures for the natural switching step of Figure 2.31**

The switching modules are the most important contributors to the total latency of the system. Indeed, we not only need to compute a set of conditions to verify if a guard leading out of the current node is active using (2.75), but we also have to infer the new state of the diodes based on the evaluation. Hence, it makes sense to optimize these modules first. A basic algorithm for the natural switching module is represented on figure 2.32a and contains four successive operations that are performed once at each time step:

1. the linear state solver computes the new value of the currents and voltages
2. the diode currents and voltages are computed
3. a test is made to check if any transition leading out of the current topology is active
4. if a transition is taken, the corresponding diode is switched

Once these operations are performed, the system then waits for the next time step. Since a transition corresponds to a single diode (see section 2.5), this algorithm will detect only



(a)

(b)

**Figure 2.33: (a) Example circuit with multiple diodes switching in series (b) simulation of the circuit using the basic diode switching algorithm of Figure 2.32a, highlighting the high error in the first few time steps**

one switching event during each time step. If $n$ diodes should switch at the same time, the final topology will only be reached after $n$ time steps. In some cases, this lag could seriously reduce the quality of the waveform and reduce the perceived performance of the control loop by introducing errors between the expected waveform and the result of the emulation.

This is better illustrated using the circuit of figure 2.33a, which contains three diodes in series. Assuming the inductor current in initially equal to zero, it will take three time steps for all diodes to switch on. The resulting effects on the evolution of the output voltage $V_{out}$ is shown on figure 2.33b. The first cause of degradation results from the fact that the voltage is equal to zero until the third diode is switched on even if the current $i_L$ is rising, and the second from the equivalent resistance perceived by the inductor which is equal to $R_{eq} = \frac{2\Omega}{\text{number of on-state diodes}}$ instead of $\frac{2\Omega}{3}$. This equivalent resistance eventually converges to the correct value, but in the mean time the value computed by the solver is prone to errors, especially when the circuit reacts quickly (i.e. when the time constants are not significantly larger than the solver time step).

For instance, very fast circuits such as discontinuous conduction-mode flyback converters with time constants under $10\mu s$ have been reported [65, 66]. Circuits may also exhibit very fast transients when some power switches are dysfunctional and short a capacitor with a low impedance path.

To avoid this effect, a multiple-pass switching algorithm may be used instead, as shown in figure 2.32b. In this version, the switching algorithm is called repeatedly until a stable topology is found. The correct state of all diodes will be found in the current time-step, but more time is spent in the natural switching module, which could unfortunately lead to increasing the time step. Furthermore, the time spent to obtain the topology is not constant since the number of executions inside the loop depends on the previous topology (though a worst case can be estimated), which results in a non-constant loop delay that could reduce the phase margin of the controllers.

Therefore, we propose a new procedure, shown in figure 2.34. The main difference lies in the insertion of an additional *Conditions selection* step during which we decide which conditions must be evaluated: instead of selecting the diode currents and voltages of the current topology, we evaluate simultaneously all transitions of the graph that are part of a path starting from the current topology. By evaluating all these transitions at the same time, we are able to select the correct solution in a single step.

**Example 4.** Let us use the example circuit of figure 2.25 as an example of our algorithm. Starting from the complete transition graph of figure 2.26, we select for each topology the shortest path (or paths in the case of identical lengths) leading to all other topologies. The resulting sub-graph obtained when starting from topologies $S_{00}$ and $S_{10}$ are shown on figure 2.35. The reason why we keep only the shortest path is to prevent the creation of loops in the reduced graph, which would correspond to a single diode switching multiple times.

Once the graph has been reduced, it is easily translated to a table linking the result of the evaluation to the final topology. When starting from $S_{00}$, the result is Table 2.2, where a '1' implies the condition has been evaluated as true, a '0' that it has been evaluated as false, and a '-' means that the evaluation has no impact on the final result.

On first examination, this procedure seems to add a lot of computations to the switch-

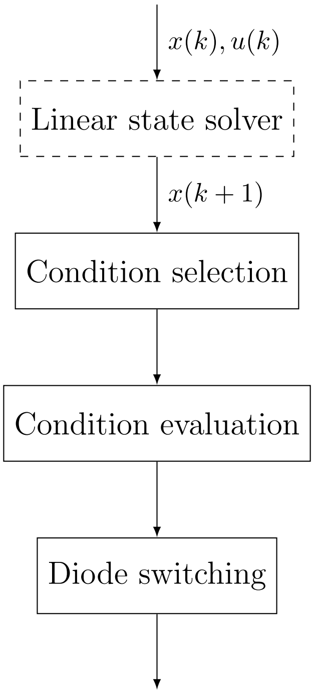


**Figure 2.34: Simulation procedure with optimized condition selection**

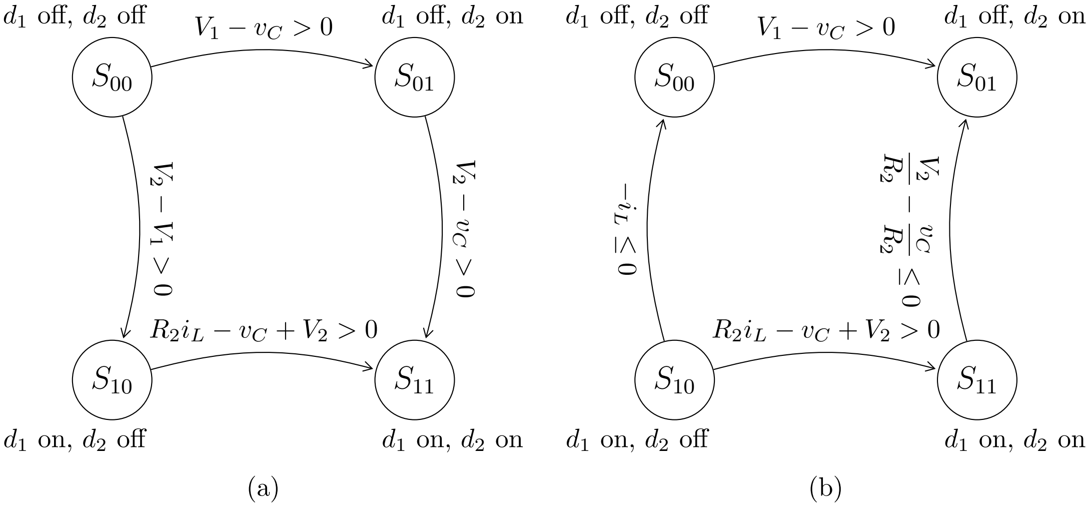


**Figure 2.35: Simplified transition graph when starting from topology (a) $S_{00}$ and (b) $S_{10}$**

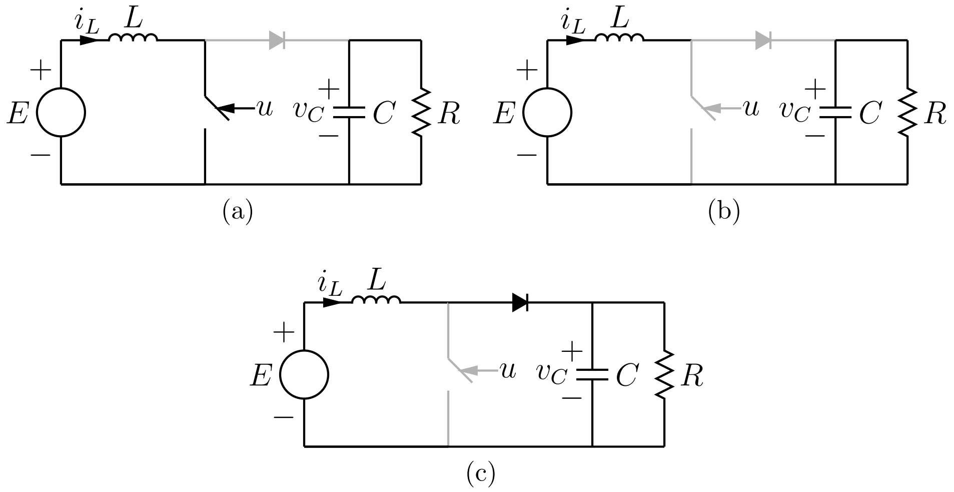
ing module. It may be shown that this is not the case, and a deeper analysis carried out in section 3.3 will introduce multiple methods to reduce the graph to a minimum, hence reducing the load on the real-time platform.

### 2.6.4 Forced switching module

The goal of the forced switching analysis is to envisage all possible changes in the circuit when the state of an active switch is modified. At first glance, this seems to be easy:

**Table 2.2: Natural commutation table for the graph of figure 2.35a**

| $V_1 - v_C > 0$ | $V_2 - v_C > 0$ | $V_2 - V_1 > 0$ | $R_2 i_L - v_C + V_2 > 0$ | Topology |
|---|---|---|---|---|
| 1 | 0 | - | - | $S_{01}$ |
| 1 | 1 | - | - | $S_{11}$ |
| 0 | - | 1 | 0 | $S_{10}$ |
| 0 | - | 1 | 1 | $S_{11}$ |
| 0 | - | 0 | - | $S_{00}$ |



(a)

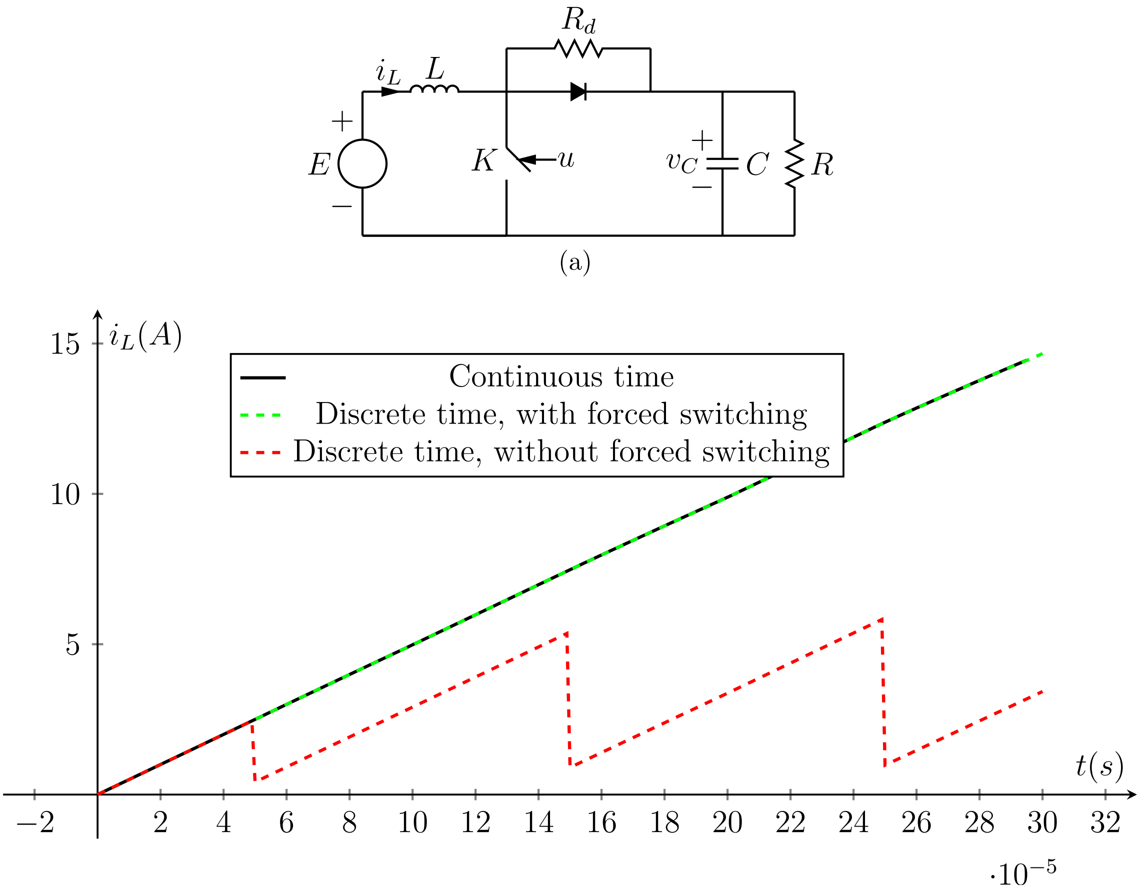
(b)

(c)

**Figure 2.36: Illustration of the state continuity problem. When the transistor switches from the on-state (a) to the off-state (b), the diode must be switched on (c) to allow $i_L$ to flow**

change the topology according to the new state of the active switches, and proceed with the simulator as it was described in the previous chapters. Unfortunately, the behavior of the circuit is not that simple. Let us take a look at the boost converter and assume that the ideal active switch is *on* while the diode is in blocking mode, as shown of figure 2.36a. The positive voltage on $L$ makes the current $i_L$ rise, which accumulates energy in the inductor. Let us now assume that the active switch is suddenly forced to its off state. If we keep the diode in its previous configuration, the system should end in topology 2.36b where the current $i_L$ is instantaneously reduced to zero (assuming ideal semiconductors). Of course, in the real-world, the diode naturally switches on (topology 2.36c) to keep the current flowing freely. The main problem faced by the forced switching module is to efficiently and correctly identify which diodes have to be switched when the state of the active switches is changed.
This problem is due to our use of ideal switches. If, for example, the diode is modeled by a bi-valued resistor or by a resistor in parallel with a source (see section 2.3.2), a path for the current always exists. Unfortunately this resistor introduces its own problems, as we will show now.

**Example 5.** Let us consider the boost converter of figure 2.37a, showing the resistor $R_d$



(a)

(b)

**Figure 2.37: Boost converter with an additional resistance, and its simulation with and without the forced switching module. A noticeable drop occurs when no forced switching module is implemented (red line) and the transistor is switched off (at $t = 50\mu s$, $t = 150\mu s$ and $t = 250\mu s$)**

in parallel with the diode. The evolution of the states variables is described by

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{dt}}\begin{bmatrix} i_L \\ v_C \end{bmatrix} &= A\begin{bmatrix} i_L \\ v_C \end{bmatrix} + Bu \\
A &= \begin{cases}
\begin{bmatrix} 0 & 0 \\ 0 & -\frac{R+R_d}{RR_dC} \end{bmatrix} & \text{diode } \textit{off}\text{, transistor } \textit{on} \\
\begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} & \text{diode } \textit{on}\text{, transistor } \textit{off} \\
\begin{bmatrix} \frac{R_d}{L} & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} & \text{diode } \textit{off}\text{, transistor } \textit{off}
\end{cases} \\
B &= \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix}
\end{aligned} \tag{2.77}$$

We shall now compare the continuous-time signals with their discrete-time counterparts. We first assume a solver able to perform forced diode control (i.e. able to change the state of the diode according to a change in the control signal of the active switch) and based on a backward Euler solver. The solver will go through the following steps after each sampling period:

1. Control the diode according to these rules:
   - If the transistor is closed, the diode is switched *off*
   - If the transistor is opened, the diode is switched *on*
2. Select the state matrices according to the state of the switches
3. Perform a linear step using (2.27)
4. Control the diode according to these rules:
   - If the diode is *on* and $i_L \leq 0$, the diode is switched *off*
   - If the diode is *off* and the transistor is *off* and $R_d i_L > 0$, the diode is switched *on*

The evolution of $i_L$ is shown in figure 2.37b for the following parameters :

- $L = 20mH$, $C = 100\mu F$, $R = 10\Omega$, $R_d = 10k\Omega$, $E = 100V$
- sampling period $T = 1\mu s$
- discretization method : backward Euler
- transistor control signal: $10kHz$, 0.5 duty cycle PWM
- circuit operating in continuous conduction mode

When the transistor is active, the current $i_L$ increases according to $L\frac{\mathrm{d}i_L}{\mathrm{d}t} = E$. When the transistor is switched off, the diode starts immediately to conduct, changing the control law to $L\frac{\mathrm{d}i_L}{\mathrm{d}t} = E - v_C$. Since the simulation of Figure 2.37b corresponds to the start-up of the circuit, the output voltage $v_C$ is still low; implying that $i_L$ continues to increase even when the transistor is *off*. To provide an accurate representation of the continuous-time evolution of the current, we use the offline simulator *Plecs* with a time-step of $10ns$. The signal obtained by this simulation is represented on the figure (in black), while the signal obtained our emulation procedure is plotted is green. Clearly, both plots are very close to each other. Let us now consider a solver without forced diode control, and based on the following steps :

1. Select the state matrices according to the configuration of the switches
2. Perform a linear step using (2.27)
3. Control the diode according to these rules:
   - If the diode is *on* and $i_L \leq 0$, the diode is switched *off*

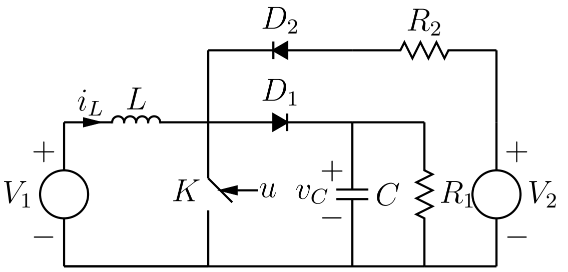


**Figure 2.38: Modified boost circuit used to illustrate state discontinuity**

- If the diode is *off* and the transistor is *off* and $R_d i_L > 0$, the diode is switched *on*

During a single time step, both switches are in the *off* state. While this may seem insignificant, the effects on the signals are profound. With the designated values for the components, the discrete state matrices for the topology are

$$\begin{aligned} A_d = (I - TA)^{-1} &= \begin{bmatrix} 0.17 & -1 \times 10^{-4} \\ 17 \times 10^{-4} & 1 \end{bmatrix} \\ B_d = (I - TA)^{-1} TB &= \begin{bmatrix} 0.83 \times 10^{-4} \\ 0.83 \times 10^{-6} \end{bmatrix} \end{aligned} \tag{2.78}$$

The first coefficient of $A_d$ is the most crucial here, as it implies that $i_L$ is reduced to 17% of its value in a single step (neglecting the effects of all other signals). This drop heavily disturbs the signals, as shown by the dashed red plot on figure 2.37b for $t = 50\mu s, t = 150\mu s$ and $t = 250\mu s$. Due to the drop, the emulated current cannot be considered as an accurate representation of the real behavior of the circuit. If we had used the forward Euler or the improved Euler (defined by (2.30)) approximation instead, the system would have become unstable. This effect can be partially counteracted by using a smaller time step or a more precise solver (i.e. the trapezoidal solver), but cannot be completely suppressed. During normal operation, this topology is used when the circuit is in discontinuous conduction mode.

This simple example shows that a module able to modify the state of the diodes as soon as the active switches are changed is absolutely required to keep a high precision, no matter which solver is used. When using ideal switches, this module is mandatory, otherwise the circuit could end in a topology that would force sudden changes in the signals, such as dropping a current to zero instead of switching a diode, or even in a topology whose equations cannot be written (shorting a voltage source, for example). More precisely, this module must be able to select the correct state of the diode taking into account

- the current topology of the switches
- the new state of the active switches
- the current value of the states and inputs

This dependance is easily shown using the circuit of figure 2.38. We first assume that the active switch is closed, allowing the inductor current to increase or decrease depending on the sign of $V_{in}$. When the transistor is switched off, one of the two diodes will be forced to conduct depending on the value of $i_L$. Similarly, $D_1$ must be switched off whenever $K$ is closed to avoid any discontinuity in $v_C$.

The selection of the conditions that must be evaluated in real-time is a complex problem, studied in detail in section 3.4.

### 2.6.5 Where to go next

This section introduced the basic modules of the real-time platform, as well as their interactions. We gave a broad view of the switching modules without going into details. In chapter 3, we shall perform a much deeper study of the circuit dynamics in order to extract all signals that we need to evaluate to control the diodes correctly.

## 2.7 Conclusions

In this chapter, we have studied the multiple options at our disposal to represent a power converter and made choices based on the expected performance on the real-time platform. We have first studied how to describe a circuit controlled by *linear* laws by means of the Modified Nodal Analysis (MNA) and the State Space Analysis (SSA). The comparison shows that while the MNA is easier to obtain and provides an easy framework to integrate non-linear laws, the equations obtained by the SSA are much more compact and do not depend on the algorithm used to convert the differential equation to a discrete-time system.

The choice of the SSA has been confirmed later, when we studied different means of representing the *non-linear* behavior introduced by the switches (diodes and transistors). We have shown that the most popular method, which uses MNA and controlled current sources to model the switches, gives a set of laws that are very easy to implement but makes the system much larger that it should be. In contrast, the alternative of using SSA and switches modeled by open/short circuits yields matrix products that are much smaller at the cost of more complex laws to control the diodes. Finally, we opted to use the SSA because of its lighter burden on the real-time platform, forcing us to develop new algorithms to automate the analysis of the circuit in order to obtain and optimize these diode control laws.

A further step in this basic modeling has been reached thanks to the introduction of hybrid automata in section 2.5. Thanks to a powerful graph-based representation, this tool allows to easily track how the circuit reacts to external stimuli and to the evolution of the state variables. Once implemented as a program, an hybrid automaton becomes a convenient high level simulator, as illustrated in section 2.6. We have compared different implementation of the simulator and introduced the natural and forced switching modules whose job is to control the diodes.

Until now, all the calculations and analyses were done by hand. For larger circuits, this process is tedious and prone to error. For this reason, the next chapter will be focused on the systematic analysis of the circuit in order to automatically derive all information needed to infer the hybrid automaton associated with the target power converter, as well as various tools to infer the rules governing the switching modules.

# Chapter 3

# Automated circuit analysis and reduction

## 3.1 Introduction

### 3.1.1 The need for automated tools

The traditional design flow for embedded digital systems is composed of four main steps, represented on figure 3.1

1. Design entry: high-level description of the application (in this case, a topological description of the circuit)
2. Analysis: conversion of the human-readable design to a mathematical/algorithmic representation of the real-time system
3. Compilation: conversion of the algorithm to a machine-readable binary format
4. Application: the design is ported on the embedded system and is ready for practical use

While a better design entry tool has a general impact on user experience thanks to better ergonomics, the analysis and compilation steps have a profound effect on the final performance of the design. A better analysis method leads to simpler, more compact algorithms while state-of-the-art compilers use their knowledge of the processing platform to provide binary code that use the platform capabilities to their best. The analysis step will be the main focus of this chapter, as we will propose a method that allows a seamless transition from a circuit drawn in a computer-aided design (CAD) program to the programming files required by the compiler.

In section 2.3, we explained how to model a power converter and its different topologies, depending on the state of the switches. We also introduced basic concepts about how the transitions between the topologies are handled by looking at the currents and voltages across the diodes. For the small circuits used in the examples of section 2.3.6, a simple study of the structure allows a trained engineer to write the equations controlling the diodes quite easily. As the number of elements increases, such a study may be not be possible without spending a lot of time writing equations for all topologies and all possible

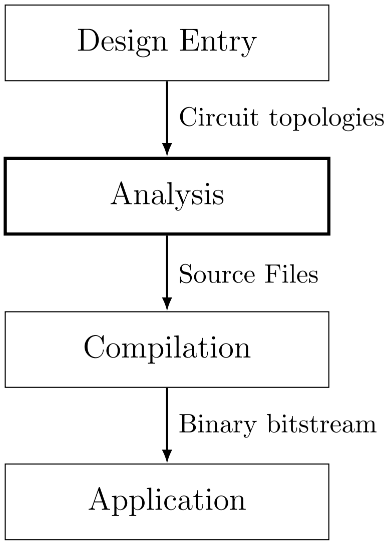


**Figure 3.1: Four-step design flow. The *Analysis* step is the focus of this chapter**

transitions between these topologies.

In section 2.3.6, we compared different simulation algorithms before settling on a state-space solver whose matrices are controlled in real-time by a non-linear switching engine, and the hybrid automaton paradigm introduced in section 2.5 provides a straightforward representation of the dynamics due to the discrete changes of the power switches. We introduced the basic rules of diode control in section 2.6, showing how to switch the diodes in reaction to a change in the state variables or in the switch control signals.

At this point, we have defined all the basic elements that will be used to provide the user with an automated analysis tool able to draw the hybrid automaton transition graph. More over, we will also propose methods to optimize and simplify the resulting switching algorithm. We first introduce in section 3.2 a method to draw the transition graph based on the individual equations of each topology and eliminate topologies that will never be used. Afterward, we will propose in sections 3.3 and 3.4 a new set of tools to automatically derive all rules governing topology changes and optimize the results to minimize the computational impact on the real-time platform.

It should be noted that we first presented these methods in two conference articles. While the analysis presented in [67] corresponds to a rough first draft of the complete method presented here, the algorithm is present in a near-final state in [68].

### 3.1.2 Real-time/Offline separation

At this point, it seems useful to remind the reader that any automated real-time tool also contains an offline part dedicated to the preparation of the information before porting the design on the real-time processing unit. This offline section takes the raw information provided by the user (eg. the electronic circuit in netlist or graphical form), converts it to a format readable by the platform and compiles it to a binary file which is used to program or configure the platform. Depending on the offline tools used, additional computing can be provided to optimize the results in multiple ways in order to reduce the load on the platform at the cost of additional preparation time. This is generally not

a problem, since this task can be run on powerful decentralized servers. If done correctly, these optimization steps allow to use fewer of the limited resources provided by the real-time platform and raises the operating frequency of the platform.
To select whether the offline or the real-time part should execute a given operation, the basic rule is that no new information should be discovered during the real-time execution and that the platform should be as simple as possible. Keeping this simple rule in mind, we choose to force the entirety of the circuit analysis on the offline program in order to reduce the real-time unit to a set of linear solvers and a list of if-then-else statements to allow topology changes. This is in stark contrast with offline, computer-based power electronics simulator that can afford to take a few milliseconds to establish the equations of a newly reached topology that had not been used before or to build the transition graph during the execution.
The off-hand study we did for the boost converter may be applied to any converter. However, as the number of switching elements rises, it becomes exponentially more complex to draw the transition graph and extract the necessary information to compute the guards. While obtaining the transition graph is a relatively easy task, properly parsing it while simplifying and optimizing the output is much more difficult. At this point in time and to the best of our knowledge, no study has been published on the matter. In the remainder of this chapter we will introduce and develop a new original set of tools to automate the study and the optimization of power converter circuits (or, more generally, any piecewise linear circuit).

## 3.2 Hybrid automaton derivation

### 3.2.1 Graph separation

The hybrid automaton paradigm introduced in section 2.5.1 is represented as a directed graph where each node corresponds to a particular configuration of the switches called topology. A topology is defined by

- a switch configuration $S$
- state equations controlling the internal dynamics, by way of the state matrices $A, B, C, D$
- a definition domain described by a set of conditions, each written in one of these two forms corresponding respectively to a condition on the diode current of the diode voltage

$$\begin{aligned} i_d(k) = Ex(k) + Fu(k) > 0 \\ v_d(k) = Ex(k) + Fu(k) \leq 0 \end{aligned} \tag{3.1}$$

As explained in section 2.5.3, the transitions between the nodes may correspond to the natural commutation of a diode or the forced commutation due to a change in the control signal of an active switch. We also suggested the use of separated switching module in the real-time platform. Because of this, the transition graph may be separated in multiple independent parts (subgraphs), where each subgraph corresponds to a particular configuration of the active switches. In a given subgraph, the transitions correspond to

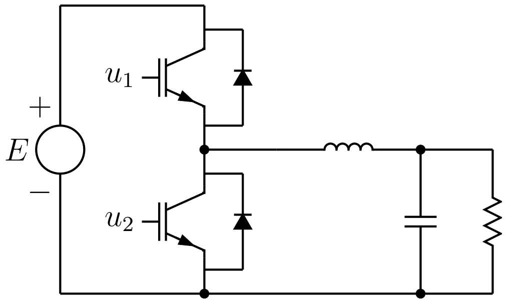


**Figure 3.2: Half-bridge converter : the case where both $u_1$ and $u_2$ are active leads to an impossibility to write the equations and must be rejected**

the diodes switching in reaction to the circuit reaching a state outside of the definition domain of its current topology. In the remainder of this section, we will show how each subgraph is derived by studying the topologies individually. Since the method to obtain the conditions required to jump between two discrete subgraphs is completely different from the one used to describe the jumps inside a given subgraph, these will be studied separately in section 3.4.

### 3.2.2 Drawing the transition graphs

Obtaining the transition graph for the natural switching events is quite straightforward. The first step is obtaining a list of the nodes of the graph. For a circuit containing $n_d$ diodes, we may define $2^{n_d}$ individual configurations for the state of the diodes. In practice, the amount of topologies that may be physically reached by the circuit may be much lower than that. Some of the switch configurations may lead to equations where Kirchhoff's laws cannot be applied without having an infinite current or voltage. These topologies are against the base principles of physics and are eliminated from the list. This is the case with the half-bridge configuration of figure 3.2 when both transistors are in the on-state: if both switches are ideal, the voltage source is shorted, resulting in an infinite current and in the impossibility of writing the equations. These topologies are qualified as being *unreachable*. By removing all unreachable topologies from the list, we ensure that that platform will not end in an unknown state. However, since the control signals come from an external source, a remapping must be made to select another topology when both control signals are active.

Normally, this event is prevented by the controller and the transistor drivers [69, 70], but this is typically a case where we might want a cross conduction to occur in order to test a protection system. In that case, the addition of a small resistor or inductor in series or in parallel with the switches ensures the compatibility with the Kirchhoff's equation for all topologies. If the converter is based on MOSFETS, the equivalent resistance may correspond to the $R_{DSON}$ of the transistor.

To draw the transitions corresponding to natural commutations, we will first make two assumptions:

- the state of the active switches does not change between the two connected topologies
- a transition always corresponds to the change of state of a single diode

Under these assumptions, obtaining the transitions becomes straightforward. The following steps are executed for each admissible topology of the graph :

1. select the first diode
2. invert the state of the diode : the endpoint of the transition is the topology with the corresponding modified switch configuration unless this topology was proven to be unreachable during the initial enumeration of all topologies, in which case the transition is not added to the graph
3. obtain the current condition $i_d(k) > 0$ or the voltage condition $v_d(k) \leq 0$ on the diode in the initial topology by derivating its state-space equations and by isolating the corresponding signal in the output equation $y(k) = Cx(k) + Du(k)$
4. the guard on the transition is the opposite of this relationship : if the diode was conducting in the starting topology, then the guard is $i_d(k) \leq 0$ (and $v_d(k) > 0$ if it was blocking)
5. select the next diode, and return to the second step

As an example, let us take the circuit represented in Figure 2.25 and assume that we are studying the transitions leading out of the topology $S_{00}$ with both diodes off. If we change the state of the first diode, we end up in topology $S_{10}$ and, since this topology in reachable, we add the transition to the graph. To write the guard associated with the transition, we first obtain the diode voltage $v_{d1}$ in the initial topology

$$v_{d1} = V_1 - v_C \tag{3.2}$$

The circuit will remain in $S_{00}$ until $v_{d1} > 0$, at which point the newly defined transition will be taken. Since the state of the $n_a$ active switches does not change, we obtain $2^{n_a}$ disjoint graphs. The circuit cannot jump naturally from one of the graphs to another, and a forced transition must be taken. These transitions are not as easy to obtain, and will be computed in section 3.4 thanks to the introduction of a new tool.
The second assumption begets the question : what if multiple transitions are validated at the same time? As it turns out, we can evaluate the diodes in any order and the circuit will always end in the correct topology. This stems from the fact that, in a real circuit operating in continuous time, the probability of two switching events happening at the exact same time is zero, and a diode will always switch right before the other. However, the end result is the same and both diodes will have switched after the two events have passed. In discrete time, this means that no matter which path is taken, the correct state will eventually be reached. This important result also means that we can study each diode separately while assuming the other diodes stay in their current state.

**Example 6.** To illustrate this, let us take the graph of figure 2.26, corresponding to the circuit of figure 2.25, and let us suppose the circuits starts in configuration $S_{00}$. After a new linear step is made, we assume that both outbound guards $V_1 - v_C > 0$ and $V_2 - V_1 > 0$ are validated, meaning that both diodes should switch and that the circuit should end up in $S_{11}$. It may easily be shown that asserting either of these transitions first will result in the correct transition:

- if $V_1 - v_C > 0$ is asserted first
    - the circuit jumps to $S_{01}$
    - since $V_1 - v_C > 0$ and $V_2 - V_1 > 0$, then $V_2 - v_C > 0$ is validated
    - the circuit jumps to $S_{11}$
- if $V_2 - V_1 > 0$ is asserted first
    - the circuit jumps to $S_{10}$
    - since $V_1 - v_C > 0$ and $V_2 - V_1 > 0$ *and* $i_L = 0$ *in the starting topology*, $R_2 i_L - v_C + V_2 > 0$ is validated
    - the circuit jumps to $S_{11}$

This procedure actually covers more cases, since $V_1 > V_2 > v_C$ and $V_2 > v_C > V_1$ also lead to $S_{11}$ (as they should).

Once the base graph is obtained, our new tool allows us to simplify the transitions by removing as many unneeded transitions as possible.

## 3.3 Automated Natural switching exploration and optimization

### 3.3.1 The need for an automated analysis tool

In section 2.6.3, we introduced a new natural switching method to avoid a delay of multiple time steps when many diodes are switched in sequence. This method is based on the extraction of a subgraph linking the current node to each other available node using only the shortest path. For example, the circuit of figure 3.3a leads to the transition graph 3.3b; which is then simplified to the subgraphs of figures 3.3c and 3.3d when starting respectively from node $S_{00}$ and $S_{10}$. The evaluation of all conditions present in the subgraph allow us to pinpoint the correct topology in a single step, at the price of additional computations. While this may add a lot of unneccessary evaluations, we will show that this is not neccessarily the case. For example, the following cases lead to a reduction of the graph, reducing the total load:

- the guard linked to a transition may be repeated in multiple point of the circuit, in which case only one evaluation is made
- a guard may be prevented from being active by other guards
- a guard may be always active

The remainder of this section will consist in a deep study of the transition graphs and the development of optimization techniques that will allow us to remove as many unneeded transitions and nodes as possible. We will start by first defining the reachable set of nodes. Afterward, we will describe two types of nodes, unreachable and unstable nodes,

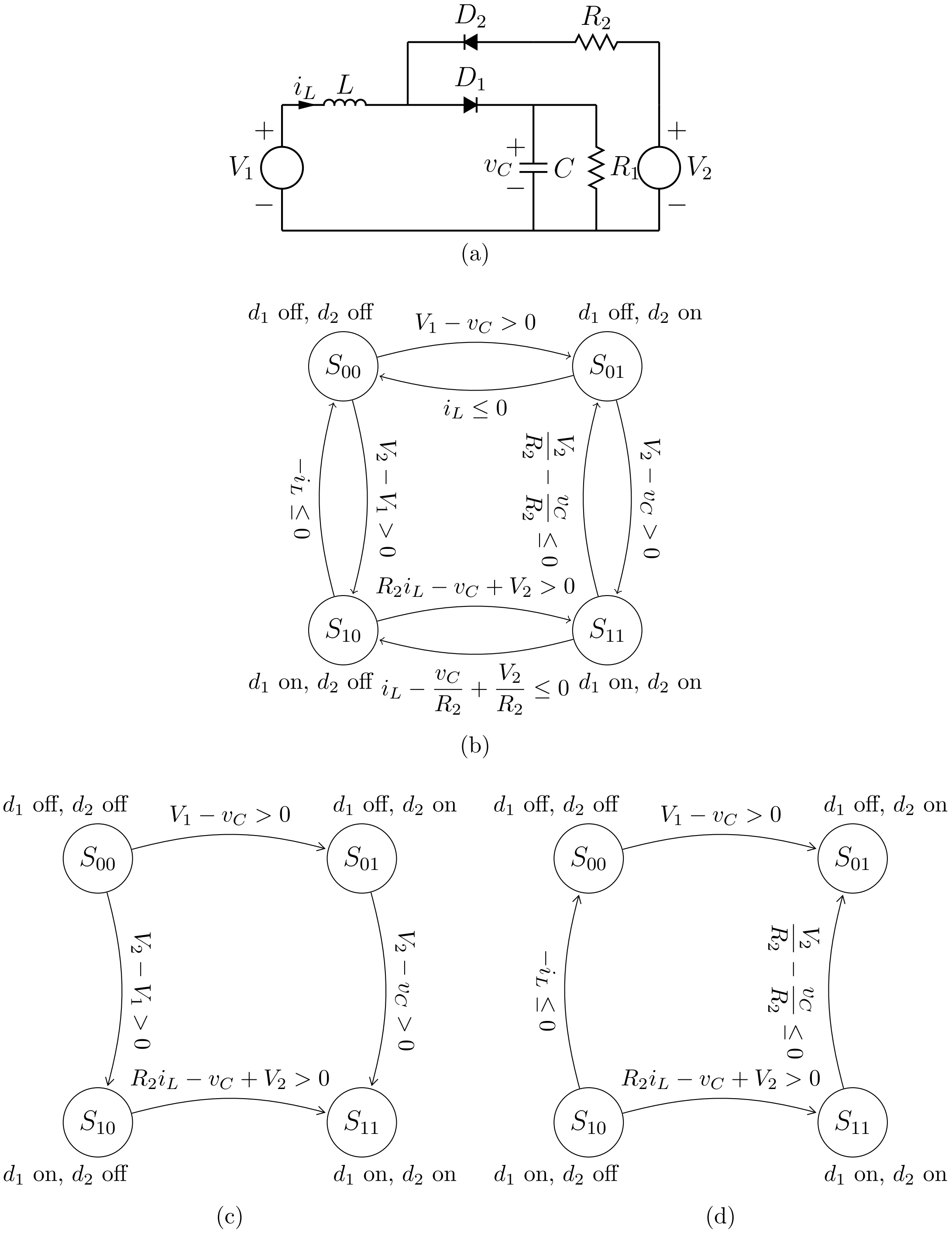


**Figure 3.3: Example circuit and its simplified transition graphs when starting from topology (c) $S_{00}$ and (d) $S_{10}$**

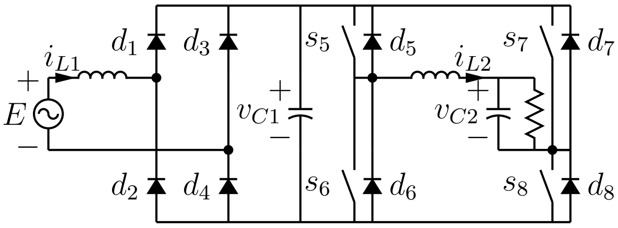


**Figure 3.4: Two stage AC/DC converter**

whose properties allow a reduction of the graph. Finally, we introduce a procedure to extract all logic conditions necessary to evaluate the new state of the circuit and link the result of the evaluation to the corresponding topology.

### 3.3.2 The reachability problem

To extract the list of conditions that we need to evaluate when the system starts from a given node, we must know all nodes that may be reached in a finite time starting from $x(k)$. This property, known as *reachability*, is often studied when dealing with system safety, to ensure that the system stays away from an unsafe state [71]. When dealing with hybrid systems, two different types of reachability may be defined:

- the state-to-state reachability defines the set of all reachable state points $x(k)$ starting from the current state
- the node-to-node reachability defines the set of all reachable nodes in the transition graph, starting from the current node

Among those two, the second one is obviously the more interesting for our application. Unfortunately, this problem has been proven to be undecidable for systems with more than two state variables, which means that the set of reachable nodes cannot be formally computed [63,72]. To ensure safety, the reachable set is generally over-estimated by propagating an approximation over successive time-steps [71,73]. In our case, this procedure is not as useful since it does not guarantee a minimal set of conditions to be evaluated.

**Example : AC/DC Converter**

The circuit of figure 3.4 provides a clear example of this problem. Let us assume that all switches and diodes are in the *off* state: $d_5, d_8$ should become conducting as soon as $v_{C1} < v_{C2}$. However, we know that $v_{C1}$ will not change its value as long as the diodes stay in the off state and that $v_{C2}$ will keep on decreasing as the capacitor is discharged by the resistor. The switching condition will never be true, and the same reasoning can be made for the $d_6, d_7$ pair. This rule is easy to infer using a simple reasoning, but is very difficult to translate into an algorithm. While we could have studied the derivative of both variables, this method is far from being universal and should be avoided.

The circuit of figure 3.4 provides a clear example of this problem. Let us assume that all switches and diodes are in the *off* state: $d_5, d_8$ should become conducting as soon as $v_{C1} < v_{C2}$. However, we know that $v_{C1}$ will not change its value as long as the diodes stays in the off state and that $v_{C2}$ will keep on decreasing as the capacitor is discharged

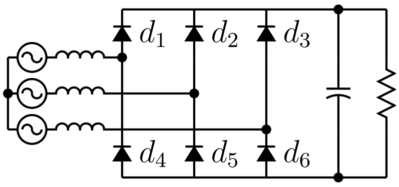


**Figure 3.5: Three phase rectifier: the topology where only $(d_1, d_2, d_3)$ are on and $(d_4, d_5, d_6)$ are off is unstable since the sum of their current is equal to zero**

by the resistor. The switching condition will never be true, and the same reasoning can be made for the $d_6, d_7$ pair. This rule is easy to infer using a simple reasoning, but is very difficult to translate into an algorithm. While we could have studied the derivative of both variables, this method is far from being universal and should be avoided.

Instead, we propose the use of a new algorithm that we build specifically to simplify graphs corresponding to power converters based on topological and electrical rules. Our goal is to reduce the amount of admissible nodes to a minimum, which should limit the amount of calculations needed to select the correct node as the circuit evolves.

### Unreachable and unstable nodes

Not every switch configuration is admissible as a node in the graph: some of the topologies must be rejected based on electrical rules. The rejected nodes may be put in one of two categories. We already introduced *unreachable* nodes in example 3. These nodes correspond to topologies whose state equations cannot be written because of electrical inaccuracies, and may be safely removed from the graph.

A second class of inadmissible nodes are *unstable* topologies. In opposition with unreachable topologies, the state equations of these topologies may be written using the classical procedure. However, they will never be observed during actual use of the converter because their definition domain is reduced to the null set. This means that when a transition leads to an unstable topology, another transition leading out of it must be active at the same time. These topologies are kept in the graph as transient points between two stable topologies, but are removed from the list of possible endpoints. These special types of topologies are studied in the next two sections

## 3.3.3 A method for detecting unstable topologies

Unstable nodes are detected by studying if their definition domain is reduced to the null set. The circuit of figure 2.33a that we used to illustrate the disadvantages of a single pass switching engine is a simple example of circuit containing unstable topologies. If we take the topology where $D_1$ is in the on-state while $D_2$ and $D_3$ are in the off-state, the domain is defined by

$$\begin{aligned} i_{D1} &= i_L && > 0 \\ v_{D2} &= Ri_L && \leq 0 \\ v_{D3} &= 0 && \leq 0 \end{aligned} \tag{3.3}$$

The first two inequalities are mutually exclusive, hence this topology is unstable.

Topology instability also happens when there are incompatibilities between two or more diode currents. This is easily shown on the three-phase rectifier of figure 3.5 : assuming the three upper diodes are in the on-state while the lower three are in the off-state, the domain is defined by $i_{d_{(1,2,3)}} > 0, v_{d_{(4,5,6)}} \leq 0$. By applying Kirchhoff's current law at the cathode of $d_{(1,2,3)}$, we obtain

$$i_{d1} + i_{d2} + i_{d3} = 0 \tag{3.4}$$

At least one of the three diode currents must be negative for this equality to hold, making this topology unstable.

We will now propose a general method to detect unstable topologies. For a topology to be stable, all of its diode signals must remain in their definition domain. Assuming the conditions are separated into a diode current set $T_i = \{i_{d0}, \dots, i_{dn_i}\}$ and a diode voltage set $T_v = \{v_{d0}, \dots, v_{dn_v}\}$, then any positive linear combinations of the elements of these sets must respect the following laws

$$\begin{aligned} i_{dx} + \sum_{j=0}^{n_i} k_j i_{dj} &> 0 \forall i_{dx} \in T_i \\ \sum_{i=0}^{n_v} l_i v_{di} &\leq 0 \\ -i_{dx} + \sum_{j=0}^{n_j} k_j i_{dj} + \sum_{j=0}^{n_v} l_j v_{dj} &> 0 \forall i_{dx} \in T_i \\ k_i &\geq 0 \forall i \\ l_i &\geq 0 \forall i \end{aligned} \tag{3.5}$$

A topology is unstable if we are able to prove that at least one of these inequalities is not respected. To this end, we will establish an algorithm to verify if any diode current $i_{dx}$ may be written in the form

$$\sum_{j=0}^{n_j} k_j i_{dj} - \sum_{j=0}^{n_v} l_j v_{dj} = -i_{dx} \tag{3.6}$$

If it is the case, then the strict inequality on the diode currents in equation (3.5) cannot be respected for every element of $T_i$, and the topology is qualified as being unstable. Since the diode signals are part of the circuit, they may be expressed in terms of states and inputs:

$$\begin{bmatrix} i_{d_0} \\ \vdots \\ i_{d_{ni}} \\ v_{d_0} \\ \vdots \\ v_{d_{nv}} \end{bmatrix} = Ex + Fu = \begin{bmatrix} E_I & F_I \\ E_V & F_V \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{3.7}$$

where $[E_I \quad F_I]$ and $[E_V \quad F_V]$ are the submatrices of $E$ and $F$ containing only the lines corresponding to diodes currents and voltages respectively. Since (3.6) must be true for every $x$ and $u$, it may be restated as finding equalities of the form

$$\begin{bmatrix} E^T_{I,0} & \dots & E^T_{I,n_i} & -E^T_{V,0} & \dots & -E^T_{V,n_v} \\ F^T_{I,0} & \dots & F^T_{I,n_i} & -F^T_{V,0} & \dots & -F^T_{V,n_v} \end{bmatrix} \begin{bmatrix} k_0 \\ \vdots \\ k_{n_i} \\ l_0 \\ \vdots \\ l_{n_v} \end{bmatrix} = \begin{bmatrix} -E^T_{I,x} \\ -F^T_{I,x} \end{bmatrix} \tag{3.8}$$

$$k_i \geq 0 \forall i$$

$$l_i \geq 0 \forall i$$

where $E^T$ is the transpose vector of $E$. To solve this problem, we suggest the use of the method described in the following section. This method is based on a classical system solver combined with a linear programming approach.

### Solving a positive linear combination problem

At many points in this chapter, we will be faced with a similar problem : is a constant vector $b = (b_1, \dots, b_m)$ a positive linear combination of constant vectors contained in a set $T = (y_1, \dots, y_n)$ ? In mathematical terms, the goal is to find a set of coefficients $k = (k_1, \dots, k_n)$ such that

$$\sum_{i=1}^{n} k_i y_i = b \tag{3.9}$$

$$k_i \geq 0 \; \forall i \tag{3.10}$$

Or, in matrix form:

$$Yk = b$$

$$\begin{bmatrix} y_{1,1} & \cdots & y_{1,m} \\ \vdots & \vdots & \vdots \\ y_{n,1} & \cdots & y_{n,m} \end{bmatrix} \begin{bmatrix} k_1 \\ \vdots \\ k_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \tag{3.11}$$

$$k_i \geq 0 \; \forall i$$

Equation (3.9) is called the *unconstrained problem* (i.e., we do not impose any limitations on $k_i$), which is later bounded by (3.10) to form the *constrained problem.* Depending on the $b$ and $y$ vectors, the unconstrained problem may have

- a single solution
- no solution at all
- an infinite amount of solutions

If the solution is unique, and if all elements of $k$ are characterized by $k_i \geq 0$, then a positive linear dependency is found between $x$ and the elements of $T$. If at least one coefficient is strictly negative, $x$ cannot be written as a positive linear combination. The second case happens when the system is inconsistent (ex $\begin{bmatrix}1\\0\end{bmatrix} k = \begin{bmatrix}1\\1\end{bmatrix}$), and also results in the impossibility of writing a positive linear dependence. These two cases are studied by solving the equation system corresponding to the unconstrained problem. This is done using well known linear algebra methods such as Gauss-Jordan elimination [74].

At this point, the only remaining problem is the situation where the linear system is under-constrained, leading to an infinite number of solutions for $k$. In this case, (3.11) defines a convex polytope in $\mathbb{R}^m$. If the definition domain of this polytope includes any point where $k \geq 0$, then a positive linear dependence exists. Note that we do not need to know the exact value of $k$: the knowledge that such a solution exists (or does not exist) is enough.

To this end, we suggest the use of a method based on Danzig's *simplex* algorithm. The simplex algorithm is a well known linear programming method used to optimize (ie. minimize or maximize) a linear cost function

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} c_i x_i \tag{3.12}$$

Under a set of linear constraints

$$\begin{gathered} Ax = b \\ \begin{bmatrix} a_{1,1} & \ldots & a_{1,n} \\ \vdots & \vdots & \vdots \\ a_{m,1} & \ldots & a_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \\ x_1 \ldots x_n \geq 0 \end{gathered} \tag{3.13}$$

The algorithm works by starting from an initial value of $x$ that respects all constraints but does not provide the optimal value for $f$, and then moving along the polytope defined by (3.13) until the optimum is reached.

Comparing (3.11) and (3.13), we remark that our problem of searching positive values for the elements of the $k$ and $l$ vectors is similar to the search of the starting point of the associated simplex. If such a point exists, then it is necessarily satisfies all constraints defined by (3.11) and proves the existence of a positive linear dependence. In linear optimization, this starting point is knows as the *initial basic feasible solution*, and is found using one of the multiple algorithms developed to solve the problem. Among these algorithms, the dual phase simplex is one of the easiest to implement.

To use the dual phase simplex, we first introduce a vector $k' = \begin{bmatrix} k'_1, \ldots, k'_m \end{bmatrix}$ of artificial variables such that

$$\begin{gathered} Yk + k' = x \\ \begin{bmatrix} y_{1,1} & \cdots & y_{1,n} \\ \vdots & \vdots & \vdots \\ y_{m,1} & \cdots & y_{m,n} \end{bmatrix} \begin{bmatrix} k_1 \\ \vdots \\ k_n \end{bmatrix} + \begin{bmatrix} k'_1 \\ \vdots \\ k'_m \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \\ k_i \geq 0 \forall i \\ k'_i \geq 0 \forall i \end{gathered} \tag{3.14}$$

Without loss of generality, we assume $b_i \geq 0 \ \forall i$. This is not a problem since we can multiply any line by $-1$ beforehand. We also introduce the cost function

$$f(k') = \sum_{i=1}^{m} k'_i \tag{3.15}$$

The introduction of these variables allows us to clearly identify $k' = b$ as an initial feasible solution. The traditional simplex algorithm is then used to minimize $f(k')$. If we are able to reduce the value of this function to $f(k') = 0$, implying that $k'_i = 0 \forall i$, then the corresponding $k \geq 0$ is also a solution to the starting problem and a positive linear dependency exists between $T$ and $b$.

**Example 7.** As an example, we want to assert if the current $i_d = -2i_{L1} + 2i_{L2}$ is a positive linear combination of the members of the set $T = \{i_{d1}, i_{d2}, i_{d3}\}$ where

$$\begin{aligned} i_{d1} &= i_{L1} \\ i_{d2} &= i_{L2} \\ i_{d3} &= -2i_{L1} + i_{L2} \end{aligned} \tag{3.16}$$

which maps to the system

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix} \tag{3.17}$$

This system in under-constrained and admits an infinite number of solutions for $k$. To verify if any of the solutions is admissible, we first multiply the first line by $-1$ and introduce the artificial vector $k' = [k'_1, k'_2]$ in order to setup the linear programming problem

$$\begin{gathered} \begin{bmatrix} -1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} + \begin{bmatrix} k'_1 \\ k'_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \\ k_1, k_2, k_3, k'_1, k'_2 \geq 0 \\ \min k'_1 + k'_2 \end{gathered} \tag{3.18}$$

The simplex may now be started, using $k' = [2, 2]$ as the initial feasible solution. After a few iterations, the algorithm converges to

$$\begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k'_1 \\ k'_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.19}$$

Since $k' = 0$, the coefficients of $k$ are also a solution to the original system and show a positive linear dependency between $T$ and $i_d$. Note that there are multiple solutions to this problem, and the exact value depends on the order in which the variables are eliminated by the simplex. More precisely, any triplet such that

$$\begin{aligned} k_1 &= -2 + 2k_3 \\ k_2 &= 1 - k_3 \\ -1 \leq k_3 &\leq 1 \end{aligned} \tag{3.20}$$

is a feasible solution.
We will now apply the same procedure to the following system

$$\begin{gathered}\begin{bmatrix}-1 & 0 & -2\\ 0 & 1 & 1\end{bmatrix}\begin{bmatrix}k_1\\ k_2\\ k_3\end{bmatrix}+\begin{bmatrix}k_1'\\ k_2'\end{bmatrix}=\begin{bmatrix}2\\ 2\end{bmatrix}\\ k_1, k_2, k_3, k_1', k_2' \geq 0\\ \min k_1' + k_2'\end{gathered} \tag{3.21}$$

The simplex converges to

$$\begin{bmatrix}k_1\\ k_2\\ k_3\\ k_1'\\ k_2'\end{bmatrix}=\begin{bmatrix}0\\ 2\\ 0\\ 2\\ 0\end{bmatrix} \tag{3.22}$$

The minimal value obtained by the simplex is greater than zero, hence no positive linear dependence exists.

**Example 8.** Let us return to the three-phase circuit of figure 3.5. To study the presence of linear dependence, we first need to write the measurement equation 3.7 for the diode signals. For the topology where $(d_1, d_2, d_3)$ are on and $(d_4, d_5, d_6)$ are off, these are

$$\begin{bmatrix}i_{d1}\\ i_{d2}\\ i_{d3}\\ v_{d4}\\ v_{d5}\\ v_{d6}\end{bmatrix}=\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & -1 & 0 & 0 & 0\\ 0 & 0 & 0 & -1 & 0 & 0 & 0\\ 0 & 0 & 0 & -1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}i_{L1}\\ i_{L2}\\ i_{L3}\\ v_C\\ E_1\\ E_2\\ E_3\end{bmatrix}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ -1 & -1 & 0\\ 0 & 0 & -1\\ 0 & 0 & -1\\ 0 & 0 & -1\end{bmatrix}\begin{bmatrix}i_{L1}\\ i_{L2}\\ v_C\end{bmatrix}=H\begin{bmatrix}i_{L1}\\ i_{L2}\\ v_C\end{bmatrix} \tag{3.23}$$

The second equality is obtained by noting that $i_{L1}+i_{L3}+i_{L3}=0$, and by deleting columns without non-zero elements. To assert if the topology is unstable, we write the problem in the form of (3.11) with $i_{d1}$ at the right hand side of the equations:

$$\begin{bmatrix}0 & -1 & 0 & 0 & 0\\ 1 & -1 & 0 & 0 & 0\\ 0 & 0 & 1 & 1 & 1\end{bmatrix}\begin{bmatrix}k_1\\ k_2\\ l_1\\ l_2\\ l_3\end{bmatrix}=\begin{bmatrix}-1\\ 0\\ 0\end{bmatrix} \tag{3.24}$$

solving the system leads to $k_1 = k_2 = 1$ and $l_1 + l_2 + l_3 = 0$. The second equation is solved using the simplex

$$\begin{gathered}\begin{bmatrix}1 & 1 & 1\end{bmatrix}\begin{bmatrix}l_1\\ l_2\\ l_3\end{bmatrix}=1\\ l_i \geq 0\\ \min l_1 + l_2 + l_3\end{gathered} \tag{3.25}$$

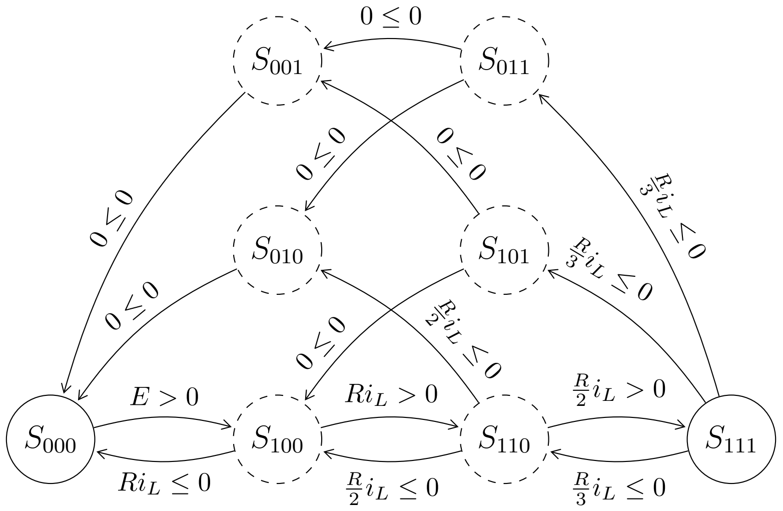


**Figure 3.6: Transition graph of the circuit of figure 2.33a, where dashed nodes are unstable. Transition guards of the form $0 > 0$ have been removed for clarity.**

which has an immediate solution in $l_1 = l_2 = l_3 = 0$. A positive linear dependence is established between $-i_1$ and the other vectors, and the topology is unstable. The result would have been the same if we had taken $i_2$ or $i_3$ as the right hand side variable.

**Example 9.** A second example of unstable topologies may be shown using the circuit of figure 2.33a. This circuit was used previously to show the impact of the diode switching module on the system, and we will now show that deleting unstable topologies allows us to find the correct node in one step without adding any complexity. We will first show that any topology where one or two of the diodes are in the on-state is unstable.
For the topology where $(d_1, d_2, d_3) = (1, 0, 0)$, the diode signals are

$$\begin{bmatrix} v_{d2} \\ v_{d_3} \\ i_{d1} \end{bmatrix} = \begin{bmatrix} R & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i_L \\ E \end{bmatrix} = H \begin{bmatrix} i_L \\ E \end{bmatrix} \tag{3.26}$$

Selecting $i_{d1}$ as the tested variable, we obtain the system

$$\begin{bmatrix} -R & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \tag{3.27}$$

which is easily solved and results in $l_1 = 1/R, l_2 = 0$. Again, the corresponding topology in unstable The topology $(d_1, d_2, d_3) = (1, 1, 0)$ in unstable for the same reason; and all topologies where $d_1$ is off while either $d_2$ or $d_3$ is on is unstable because the current across the diode is equal to zero, ie. the system

$$\begin{bmatrix} i_{d2} \\ -v_{d1} \end{bmatrix} \begin{bmatrix} k_1 \\ l_1 \end{bmatrix} = -i_{d3} \tag{3.28}$$

is solved for $k_1 = l_1 = 0$. The transition graph of this circuit is drawn in figure 3.6, where the unstable topologies are represented by dashed nodes. A crucial information to take away from this graph is that almost all topologies are unstable, which means that

**Table 3.1: Natural commutation table for the graph of figure 3.6**

| Starting Topology | $E > 0$ | $i_L \leq 0$ | New Topology |
|---|---|---|---|
| $S_{000}$ | 1 | - | $S_{111}$ |
| $S_{000}$ | 0 | - | $S_{000}$ |
| $S_{111}$ | - | 1 | $S_{000}$ |
| $S_{111}$ | - | 0 | $S_{111}$ |

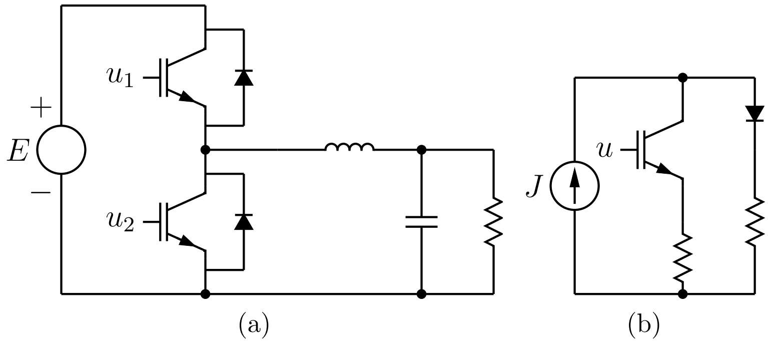


**Figure 3.7: Examples of circuits containing at least one unreachable topology**

it may be greatly simplified. If we start in $S_{000}$ and the input voltage becomes negative, we should switch to $S_{100}$. But, since this topology is unstable, it will instantly switch to $S_{110}$ and then to $S_{111}$. Similarly, if we start in topology $S_{111}$ and the inductor current becomes positive, the chain of unstable topologies will ultimately lead to $S_{000}$.
Thus, the whole graph may be simplified to Table 3.1. At any time, the evaluation of a single transition gives the final result in one step. This simple graph could be simplified just by looking at it, but bigger circuits may lead to large graphs that necessitate the help of a computer to be parsed.

The study of unstable topologies allows us to reduce the system complexity by removing the unnecessary transitions from the graph, leading to fewer signals to compute in real-time. In some cases, like the circuit studied in the second example, the reduction removes the majority of the available topologies and reduces the graph to a single if-else statement.

### 3.3.4 Unreachable topologies

The detection of unreachable topologies happens early in the compilation process, when the equations for each switch combination are extracted. When faced with a situation where Kirchhoff's equations cannot be written, the topology is marked as unreachable before even drawing the transition graph. The most common cause is the shorting of a voltage source like in figure 3.7a with both transistors on or putting a current source in series with an open circuit like in figure 3.7b with both the transistors and the diode off. Those topologies are unreachable because the switches annd the sources are ideal. This behavior may be avoided by modeling the switches by their non-ideal form, or by adding stray inductors and resistors.
Note that shorting a capacitor or opening an inductive branch does not generate an

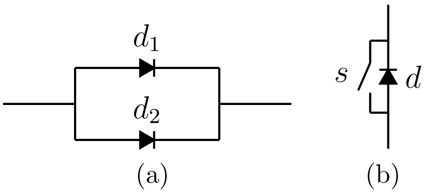


**Figure 3.8: Examples of circuits with conditional unreachability. (a) Only the topologies where $d_1$ and $d_2$ are in the same state are kept (b) The topology with $s$ and $d$ on is marked as unreachable**

unreachable topology, but rather forces the associated state variable to zero. This happens when a converter is used in discontinuous conduction mode.

### Conditional unreachability

The previous section dealt with topologies that were intrinsically unreachable, meaning that they were rejected because of their own equations. We will now study topologies that are unreachable because they are always superseded by another one. Let us start with the example of the two-diodes circuit of figure 3.8a. If a single diode is in the on state, it carries a current equal to $I$ while the other one perceives a zero voltage. If, on the other hand, the two diodes are conducting they both carry a current equal to $\frac{I}{2}$. These three topologies are equivalent for all purposes, and we are free to select of them while marking the other two as unreachable. The most natural choice is to keep the topology were both diodes are conducting, since it corresponds to the behavior the would be seen on the real circuit equivalent.

A second example of conditional unreachability is illustrated by the circuit of figure 3.8b containing an ideal controlled switch in parallel with a diode, which could be seen as an idealized MOS transistor. If the controlled switch is put in the on state and the diode is blocking, the voltage across the diode is again equal to zero, meaning that it will never switch to the on state. Thus, the topology where both switches are active may be safely marked as unreachable, even if it was technically stable.

## 3.3.5 Graph exploration and simplification

Removing all unreachable nodes from the graph provides us with a reduced (but valid) representation of the circuit behavior. We may now parse it in order to extract the transitions that must be evaluated and the link between the result of the evaluation and the final topology while removing all unneeded computations.

This process consists in three main steps that are repeated for each starting topology: first, we find the graph that contains the shortest path from the starting topology to each other stable node. Then, this graph is parsed a first time to remove all unstable nodes and associated transitions. Finally, we remove all unneeded transitions and build the final logic equations linking the nodes.

These steps are described individually in the next sections, starting with the extraction of the minimal graph from the complete system.

### Drawing the reduced graph

As was shown in example 4, each starting topology possesses its own switching graph, extracted from the main transition graph. This subgraph is obtained by keeping the shortest path (or paths) leading from the starting topology to each of the other stable topologies in the graph. The reasoning behind this is that a single diode should not switch twice in response to a single event. At this point, the graph may still contain unstable nodes as intermediary topologies along the path.

This process is implemented using a variation of Dijkstra's algorithm for finding the shortest path between two nodes of a directed graph [75]. While this algorithm is neither the fastest nor the most memory-efficient compared to other pathfinding approaches like the Bellman-Ford algorithm [76] or Yen algorithm [77], it is very easy to implement and customize to our needs. Furthermore, the graphs are very small compared to the computer networks, that may contain tens of thousands of nodes, found in the classical use of these algorithms. Before explaining our approach, we first define the weight of a node as the amount of discrete transitions needed to go from the starting node (called the root) to this node. The reduced graph is then built using an iterative approach.

1. The current minimal graph is set to contain the root only.

2. The weight of each node not in the minimal graph is set to an arbitrarily large value (any number larger than the number of nodes) .

3. For each transition leading out of the nodes contained in the current minimal graph, we compute the weight of the node at the other extremity. Transitions between two nodes of the minimal graph are kept only if the computed weight is equal to the previous weight (corresponding to different minimal paths between two nodes).

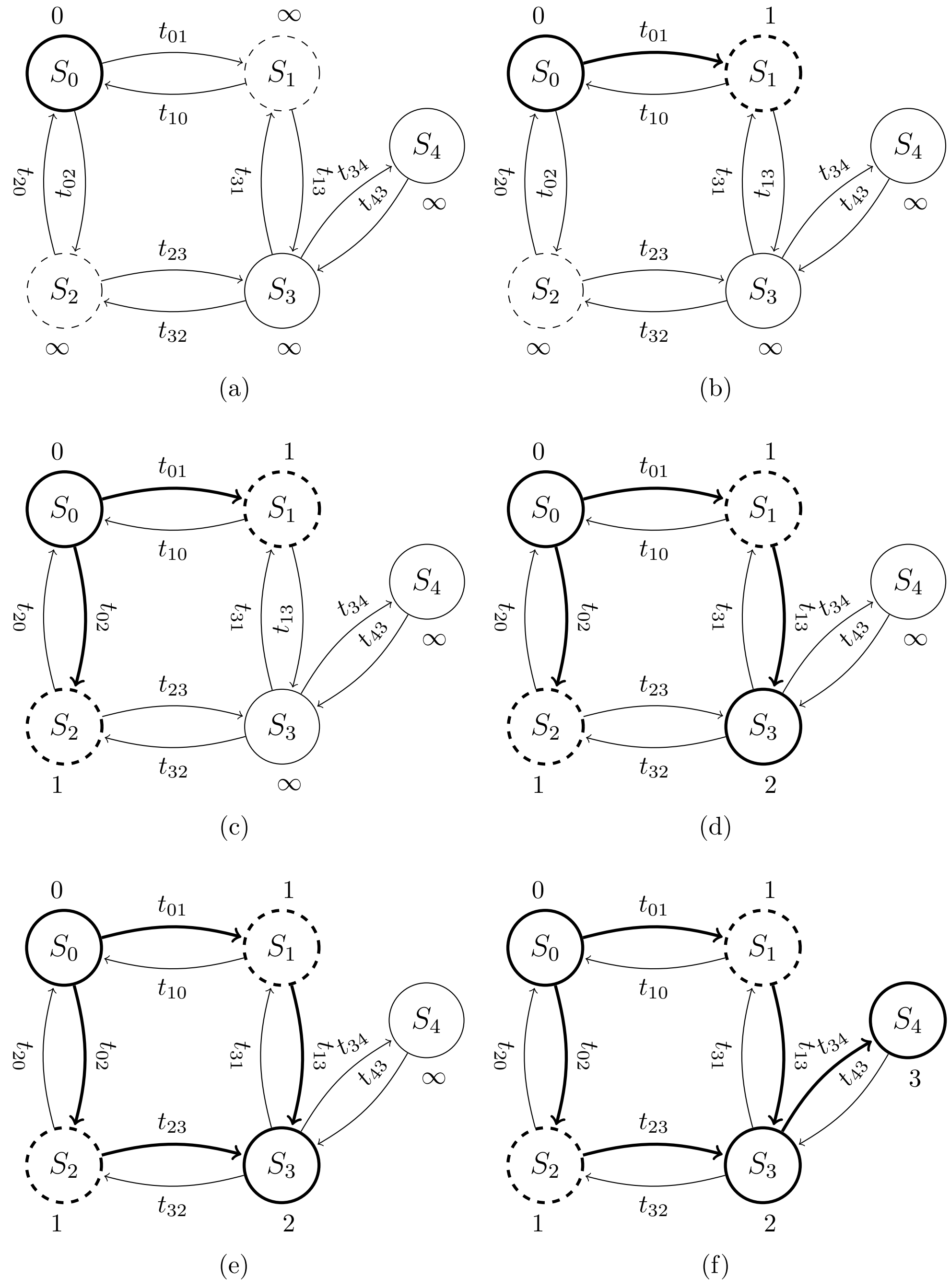
4. The node with the minimal weight is selected. If this weight is equal to the previously set large value (ie. no new transition may be added), then the end of the algorithm is reached. If this is not the case then both the node and the corresponding transition are added to the minimal graph, and the algorithm returns to the second step.

The output of this algorithm is the incidence matrix $A$ whose elements are defined by as $A_{ij} = 1$ if a transition from node $i$ to node $j$ is found in the reduced graph, otherwise $A_{ij} = 0$.

**Example 10.** Our algorithm will now be used to parse the graph of figure 3.9a, with node $S_0$ as the root. The figures 3.9b through 3.9f show the step by step progress of the pathfinding algorithm. There are two independent paths of equal length leading to $S_2$, and both are kept by the shortest path algorithm. The corresponding incidence matrix is

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.29}$$

The subgraph is now ready. The next section will introduce a set of properties associated with the transition, before using these results to further reduce the graph.



(a) (b) (c) (d) (e) (f)

**Figure 3.9: Usage of the pathfinding algorithm, where the bold items are part of the reduced graph**

### Some properties of the transitions

The nodes of the graph are linked by the transitions, each associated to a guard carrying a condition. These conditions are naturally the exact opposite of those used to define the definition domain of the node at the origin of the transition. A transition is *validated* if the corresponding condition is *True*, otherwise it is invalid. If, for instance, the transition corresponds to a diode switching from the on state to the off state, the guard carries a condition based on a current. More precisely, it corresponds to the diode current becoming negative, and hence the condition is of the form

$$i_d = \begin{bmatrix} E & F \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \leq 0 \tag{3.30}$$

On the other hand, a transition corresponding to a diode switching to the on state will carry a condition based on a voltage of the form

$$v_d = \begin{bmatrix} E & F \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} > 0 \tag{3.31}$$

The various transitions are separated into four sets :

- $T_I$ is the set of *valid* transitions whose condition is based on a *current*
- $T_I'$ is the set of *invalid* transitions whose condition is based on a *current*
- $T_V$ is the set of *valid* transitions whose condition is based on a *voltage*
- $T_V'$ is the set of *invalid* transitions whose condition is based on a *voltage*

The study of these sets provides us with interesting properties. Let us take $T_I = \{i_{d1}, \ldots, i_{dn}\}$ as an example. Each element of the set must obey (3.30). The same is true for any positive linear combination of the included diode currents :

$$\begin{gathered} \sum_{j=1}^{n} k_j i_{dj} \leq 0 \\ k_j \geq 0 \forall j \end{gathered} \tag{3.32}$$

Since this condition must be true for all values of $x$ and $u$, we can also write

$$\begin{bmatrix} E_1^T & \ldots & E_n^T \\ F_1^T & \ldots & F_n^T \end{bmatrix} \begin{bmatrix} k_1 \\ \vdots \\ k_n \end{bmatrix} \leq \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.33}$$

where $E_j^T$ is the transpose of the $E$ vector of the $j^{th}$ current of the set. The vector inequality means that each row, taken individually, is equal to or less than zero.
The same reasoning may be applied for the currents included in the $T_I'$ set. Since these currents are supposed to be strictly positive, we may write

$$\begin{gathered} \sum_{i_{dj} \in T_I'} k_j' i_{dj} > 0 \\ k_j' \geq 0 \forall j \end{gathered} \tag{3.34}$$

with the additional constraint that at least one of the $k'_j$ must be strictly positive. This leads to

$$\begin{bmatrix} E_1^T & \dots & E_n^T \\ F_1^T & \dots & F_n^T \end{bmatrix} \begin{bmatrix} k'_1 \\ \vdots \\ k'_n \end{bmatrix} > \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.35}$$

By repeating the same process to the two other sets, we obtain :

$$\begin{aligned} \sum_{i_{dj} \in T_I} k_j i_{dj} &\leq 0 \\ \sum_{i_{dj} \in T'_I} k'_j i_{dj} &> 0 \\ \sum_{v_{dj} \in T_V} l_j v_{dj} &> 0 \\ \sum_{v_{dj} \in T'_V} l'_j v_{dj} &\leq 0 \\ k_j, k'_j, l_j, l'_j &\geq 0 \forall j \end{aligned} \tag{3.36}$$

Furthermore, we may also combine the two valid sets $T_I$ and $T_V$ to write

$$\begin{aligned} \sum_{i_{dj} \in T_I} k_j i_{dj} - \sum_{v_{dj} \in T_V} l_j v_{dj} &\leq 0 \\ k_j, l_j &\geq 0 \forall j \end{aligned} \tag{3.37}$$

and, of course

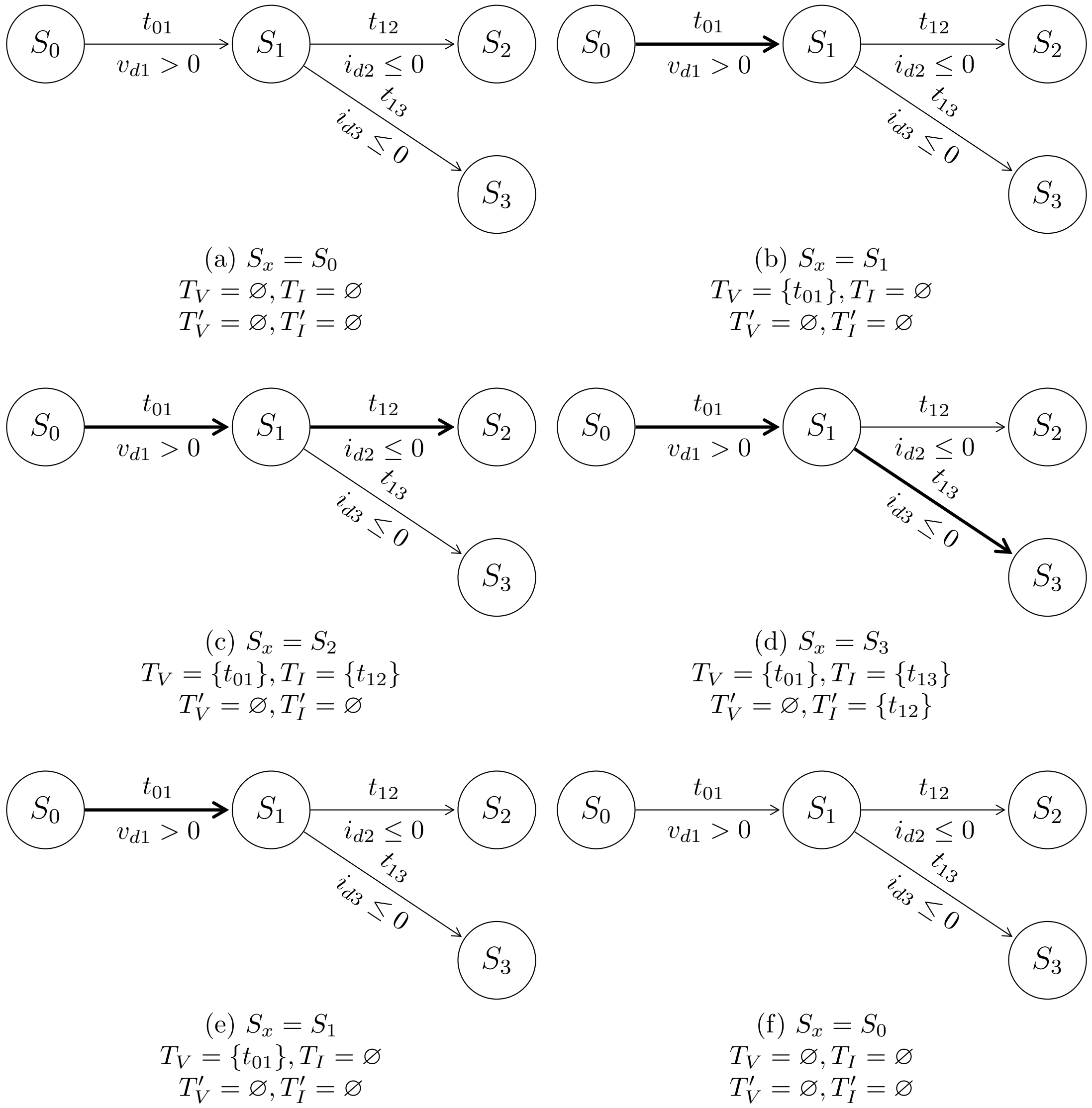
$$\begin{aligned} \sum_{v_{dj} \in T'_V} l'_j v_{dj} - \sum_{i_{dj} \in T'_I} k'_j i_{dj} &\leq 0 \\ k'_j, l'_j &\geq 0 \forall j \end{aligned} \tag{3.38}$$

These relations will prove to be very helpful when attempting to reduce the graph.

### Parsing of the graph

The next section will introduce reduction techniques based on the knowledge of the valid sets $T_V, T_I$ and the invalid sets $T'_V, T'_I$. These sets are completed during the parsing of the graph using the following recursive algorithm :

1. All sets are assumed to be empty, and the current node $S_x$ is set to the root node
2. For each transition leading out of $S_x$, repeat these steps :
   (a) The transition is added to the valid set $T_V$ or $T_I$
   (b) Change $S_x$ to the node at the end of the transition and return to step 2 with the new node
   (c) When the algorithm returns, remove the transition from the valid set and add it to the corresponding invalid set $T'_V$ or $T'_I$



(a) $S_x = S_0$
$T_V = \varnothing, T_I = \varnothing$
$T'_V = \varnothing, T'_I = \varnothing$

(b) $S_x = S_1$
$T_V = \{t_{01}\}, T_I = \varnothing$
$T'_V = \varnothing, T'_I = \varnothing$

(c) $S_x = S_2$
$T_V = \{t_{01}\}, T_I = \{t_{12}\}$
$T'_V = \varnothing, T'_I = \varnothing$

(d) $S_x = S_3$
$T_V = \{t_{01}\}, T_I = \{t_{13}\}$
$T'_V = \varnothing, T'_I = \{t_{12}\}$

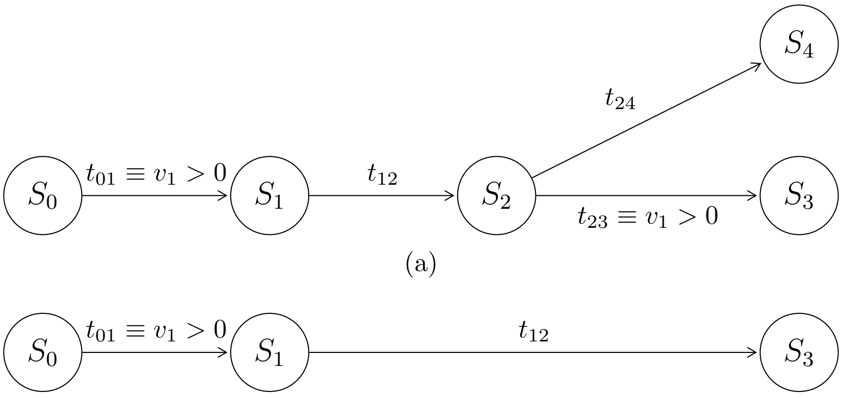
(e) $S_x = S_1$
$T_V = \{t_{01}\}, T_I = \varnothing$
$T'_V = \varnothing, T'_I = \varnothing$

(f) $S_x = S_0$
$T_V = \varnothing, T_I = \varnothing$
$T'_V = \varnothing, T'_I = \varnothing$

**Figure 3.10: Example of parsing a graph, showing how the various sets are completed and depleted as the algorithm progresses**

3. remove all transitions leading out of $S_x$ from the valid and invalid sets
4. If the current node is the root node, the parsing is complete. Else, return to the previous node

This algorithm is illustrated by the steps shown on figure 3.10. We first assume that we start in topology $S_0$ with all the $T$ sets initially empty. The first step is to use the transition $t_{01}$ to change from $S_0$ to $S_1$. By doing so, we add $t_{01}$ to $T_V$ since the guard associated to the transition corresponds to a voltage. If we were to study the stability or the reachability of topology $S_1$ (see next sections), we would use the current value of the $T$ sets. This process is repeated to go from $S_1$ to $S_2$ : the transition $t_{12}$ is added to $T_I$ since it carries a current-based condition while $T_V$ in unchaged. These new sets would be used if topology $S_2$ was studied. Since $S_2$ is the end of the path, we return to $S_1$ to study the second branch. Before going from $S_1$ to $S_3$ , we first add $t_{12}$ to $T'_I$ since



(a)

(b)

**Figure 3.11: Example of a path with an conditionally unstable node ($S_2$ in (a)) : the graph may be reduced to the one shown in (b)**

this transition corresponds to a path which was already studied. Then, we add $t_{13}$ to $T_I$ and move to $S_3$. Since the $T_V, T_I, T_I', T_V'$ set are now obtained for all the topologies, the parsing is complete.

While this parsing does not modify the graph by itself (and indeed, the first and last steps of Figure 3.10 are exactly the same), this method provides us with a standard way to recursively study the graph. The reduction methods introduced in the next sections will call various functions at each step of the parsing to eliminate of fuse some of the transitions based on the $T$ sets of the current topology.

### Conditional instability

The first step taken to reduce the complexity the graph is to detect all transitions that are always validated when the system starts from a given node. The identification of these transitions allows us to bypass some nodes completely and to remove many transitions along the way.

The graph presented in figure 3.11 provides a simple example : both $t_{01}$ and $t_{23}$ carry the condition $v_1 > 0$. If the process jumps from $S_0$ to $S_1$ and then to $S_2$, which means that both $t_{01}$ and $t_{12}$ are valid, we are certain that $t_{23}$ will be validated as well. In response, $S_2$ may be treated as an unstable node. Furthermore, all other transitions leading out of $S_2$ are removed since the system is guaranteed to take the transition $t_{23}$. This behavior is called conditional instability.

To detect conditionally unstable nodes, we use an algorithm similar to the one used to detect unstable nodes in section 3.3.3. As a reminder, the transitions correspond to a single diode changing its state either because the current becomes negative, or because the voltage becomes strictly positive.

The transitions contained in the path leading from the root node to the current node $S_x$ are, by definition, already validated. It follows that the sets $T_V$ and $T_I$ are composed of the conditions corresponding to these transitions. We now use the relations defined in (3.36) and (3.37). We assume the valid sets are defined as $T_V = \{v_{d1}, \dots, v_{dn_v}\}$ and $T_I = \{i_{d1}, \dots, i_{dn_i}\}$. Let us examine first the case of a current-based transition $i_{dx}$ leading out of the node $S_x$. We verify if we can write an identity of the form

$$\begin{gathered}\sum_{i_{dj}\in T_i} k_j i_{dj} - \sum_{v_{dj}\in T_v} l_j v_{dj} = i_{dx} \\ k_j, l_j \geq 0 \forall j\end{gathered} \tag{3.39}$$

Since the left side of this equation is always less or equal to zero, this identity implies that the transition will always be valid. At that point, $S_x$ is considered as conditionally unstable: the system cannot remain in this node and will jump to another node using the studied transition.

All the transitions correspond to diode signals, which we write in state-space form :

$$\begin{bmatrix} i_d \\ v_d \end{bmatrix} = \begin{bmatrix} E_I & F_I \\ E_V & F_V \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} i_{dx} \quad = \begin{bmatrix} E_{I,x} & F_{I,x} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{3.40}$$

Since (3.39) must hold for all $x$ and $u$, the problem is restated as finding positive $k_j, l_j$ coefficients such that

$$\begin{gathered}\begin{bmatrix} E_{I,1}^T & \dots & E_{I,n_i}^T & -E_{V,1}^T & \dots & -E_{V,n_v}^T \\ F_{I,1}^T & \dots & F_{I,n_i}^T & -F_{V,1}^T & \dots & -F_{V,n_v}^T \end{bmatrix} \begin{bmatrix} k^T \\ l^T \end{bmatrix} = \begin{bmatrix} E_{I,x}^T \\ F_{I,x}^T \end{bmatrix} \\ k_j, l_j \geq 0 \forall j\end{gathered} \tag{3.41}$$

This problem is solved using the simplex procedure described in section 3.3.3. If a set of positive coefficients is found, the graph is simplified as shown in figure 3.11b : the node at the origin of the studied transition is completely bypassed since we know that the transition will be valid. Furthermore, we do not need to keep the transition in the graph anymore.

The same reasoning is made if the guard carries a voltage-based condition $v_{dx} > 0$: we try to find an identity of the form

$$\begin{gathered}\sum_{v_{dj}\in T_v} l_j v_{dj} = v_{dx} \\ l_j \geq 0 \forall j\end{gathered} \tag{3.42}$$

The left side is strictly positive, and this relation insures it is the same for $v_{dx}$. As an additional constraint, at least one of the coefficients must be strictly positive (ie. $v_{dx} \neq \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$). This is equivalent to solving the following problem :

$$\begin{gathered}\begin{bmatrix} E_{V,1}^T & \dots & -E_{V,n_v}^T \\ F_{V,1}^T & \dots & -F_{V,n_v}^T \end{bmatrix} \begin{bmatrix} l_1 \\ \vdots \\ l_{n_v} \end{bmatrix} = \begin{bmatrix} E_{V,x}^T \\ F_{V,x}^T \end{bmatrix} \\ l_j \geq 0 \forall j\end{gathered} \tag{3.43}$$

The result is the same : if we are able to find a set of positive coefficients, the topology is bypassed.

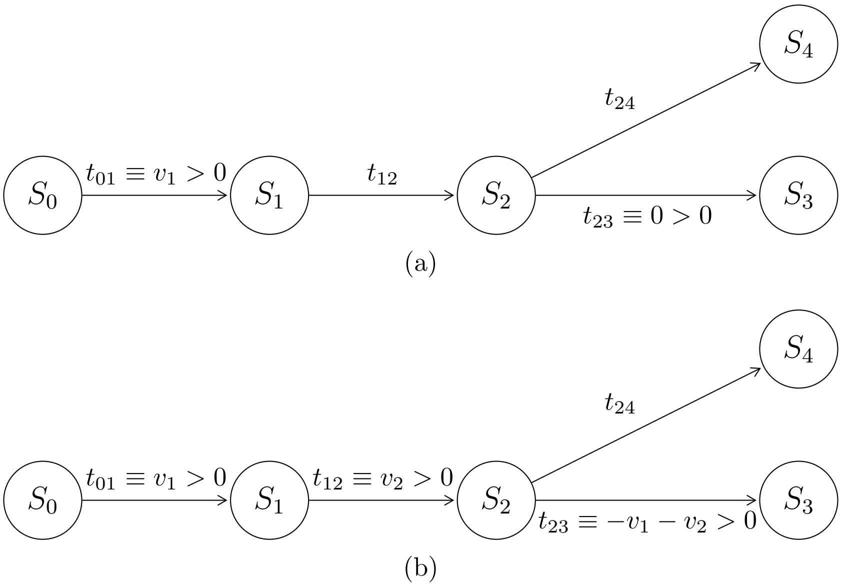


**Figure 3.12: Examples of paths with conditionally unreachable nodes ($S_3$ in this case)**

### Conditional unreachability

In contrast with the previous section, we will now study transitions that are never validated. Since these transitions are never crossed they are simply removed from the graph. Two basic examples are given in figure 3.12. In the case of Figure 3.12a, the transition $t_{23}$ has the guard $0 > 0$, which cannot be validated. In figure 3.12b, the transition $t_{23}$ will never be valid because its guard is in direct contradiction with $t_{01}$ and $t_{12}$ that are earlier in the path and are supposed true when the path reaches $S_2$. In both cases, the node $S_3$ may be removed from the path.
This behavior, known as conditional unreachability, is identified by comparing a transition and all its predecessors currently in the path. The predecessors must obey (3.36), and these rules will be used to test the compatibility of the studied transition. Assuming a transition corresponding to a diode voltage and carrying a condition of the form $v_{dx} > 0$, we try to find $k = \left[k_1, \dots, k_{nv}\right]$ and $l = \left[l_1, \dots, l_{n_i}\right]$ such that

$$\begin{gathered}\sum_{v_{dj} \in T_V} k_j v_{dj} - \sum_{i_{dj} \in T_I} l_j i_{dj} = -v_{dx} \\ k_j, l_j \geq 0 \forall j\end{gathered} \tag{3.44}$$

According to (3.37), $v_{dx} \leq 0$ and the transition will never be validated. This is equivalent to verifying if the following system possesses a valid solution

$$\begin{gathered}\begin{bmatrix} E_{V,1}^T & \dots & E_{V,n_v}^T & -E_{I,1}^T & \dots & -E_{I,n_i}^T \\ F_{V,1}^T & \dots & F_{V,n_v}^T & -F_{I,1}^T & \dots & -F_{I,n_i}^T \end{bmatrix} \begin{bmatrix} k^T \\ l^T \end{bmatrix} = \begin{bmatrix} -E_{V,x}^T \\ -F_{V,x}^T \end{bmatrix} \\ k_j, l_j \geq 0 \forall j\end{gathered} \tag{3.45}$$

The figure 3.13 shows another cause for the conditional unreachability of a transition. During the parsing of the graph, we will have at one point $T_I' = \{t_{12}, t_{13}\}$ while studying

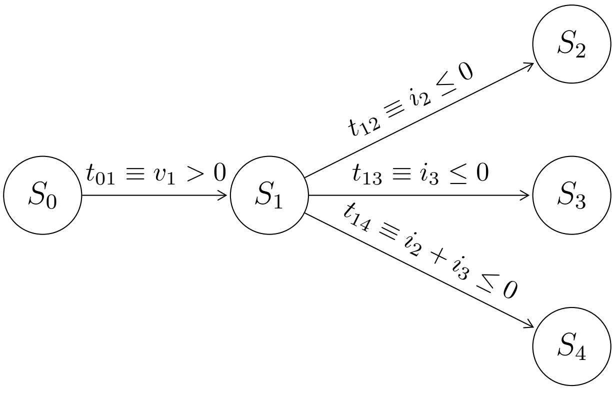

$S_0$
$t_{01} \equiv v_1 > 0$
$S_1$
$t_{12} \equiv i_2 \leq 0$
$t_{13} \equiv i_3 \leq 0$
$t_{14} \equiv i_2 + i_3 \leq 0$
$S_2$
$S_3$
$S_4$

**Figure 3.13: Another example where $S_4$ is conditionally unreachable, this time because of the invalid set**

$t_{14}$. Using (3.36), we obtain

$$i_2 + i_3 \leq 0 \tag{3.46}$$

which means that the condition associated with $t_{14}$ will never be true and the transition may be safely removed from the graph.

Again, this study is easily put into more general terms, as any guard that may be written as a positive linear combination of transition guards included in $T'_V$ or $T'_I$ will never be validated. If the transition carries a condition based on a voltage $v_{dx} > 0$, we try to find positive coefficients such that

$$\begin{gathered}\sum_{v_{dj} \in T'_V} l'_j v_{dj} - \sum_{i_{dj} \in T'_I} k'_j i_{dj} = v_{dx} \\ k'_j, l'_j \geq 0 \forall j\end{gathered} \tag{3.47}$$

According to (3.38), the left side is negative, which means that the condition is never true and the transition is removed. A transition based on a current $i_{dx} \leq 0$ results in solving

$$\begin{gathered}\sum_{i_{dj} \in T'_I} k'_j i_{dj} = i_{dx} \\ k'_j \geq 0 \forall j\end{gathered} \tag{3.48}$$

The results of this method are highly dependent on the order in which the transitions are studied. If we had studied $t_{14}$ first, then $t_{13}$ and finally $t_{12}$ in 3.13, we would not have detected any identity of the form defined by (3.48) and no transition would have been removed. This issue is easily solved by slightly modifying the parsing algorithm : for each transition leading out of $S_x$, do the following :

1. add all other transitions leading out of $S_x$ to the invalid sets $T'_V$ or $T'_I$
2. attempt to find an identity of the form (3.47) or (3.48)
3. if an identity is found, we modify the parsing order so that the transition is now studied last. It will then be removed by the normal process

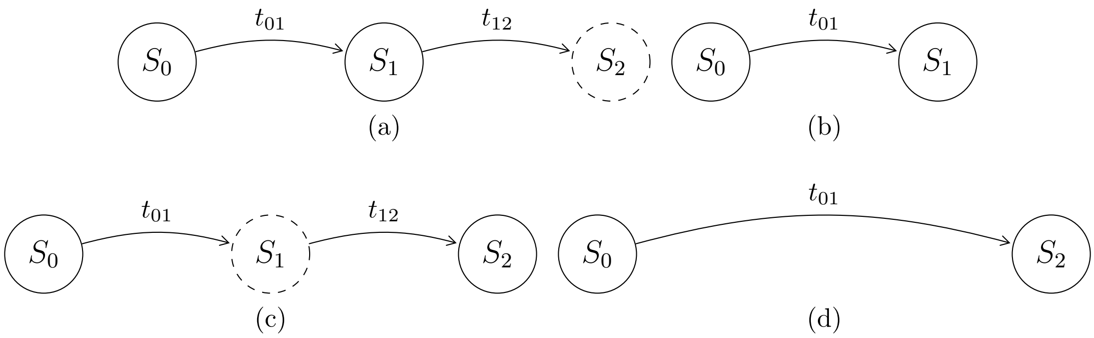


**Figure 3.14: Basic example of removal of an unstable node if the graph**

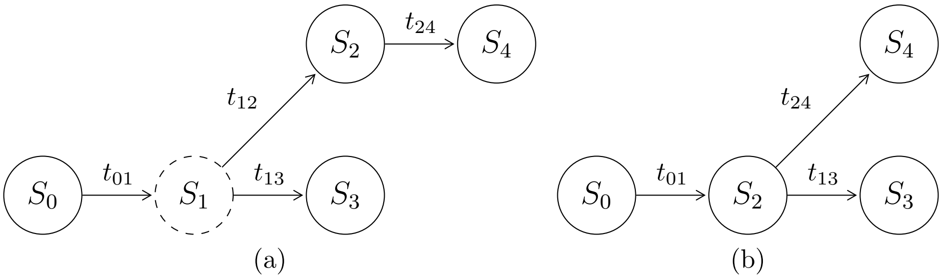


**Figure 3.15: Removal of an unstable node with multiple choices : $S_1$ is replaced by $S_2$ and the transition $t_{12}$ is removed**

### Unstable nodes removal

The final simplification step consists in the removal of unstable topologies from the graph. The core principle of this process is that if an active transition leads to an unstable topology, there must be at least one active transition leading out of it. We use this fact to our advantage to reduce the number of evaluations, as we will show with a couple examples. The first case is represented in figure 3.14a, with an unstable topology placed at the end of the path. This is the easiest case to deal with, and the topology is removed from the graph along with the corresponding transition, obtaining the graph of figure 3.14b.
For the second case, let us assume that the transition graph for the topology $S_0$ is represented on figure 3.14c: the path leading out from $S_0$, assumed to be the starting topology, first goes to the unstable node $S_1$ before reaching $S_2$. If $S_1$ was stable, we would have to evaluate both $t_{01}$ and $t_{12}$ and select the correct topology accordingly. However, since $S_1$ in unstable, there is no need to evaluate $t_{12}$ : if $t_{01}$ is validated, the circuit jumps to topology $S_1$, and next instantly to $S_2$. This effectively means that $S_2$ is put in place of $S_1$, as shown in figure 3.14d. A similar reasoning is made for the graph of figure 3.15a: assuming $t_{01}$ is validated, we know that either $t_{12}$ or $t_{23}$ is valid. This means that we only need to evaluate one of these conditions to know whether the graph jumps to $S_2$ or $S_3$. Again, this is represented by putting one of these nodes in place of $S_1$, removing a transition in the process. The result is shown in figure 3.15b, where we chose to move $S_2$. With this change, we were able to remove $t_{12}$.
This modification is performed during a parsing of the graph. Assuming that a path leads from a stable topology $S_x$ to an unstable topology $S_y$ using the transition $t_{xy}$, the

following steps are followed: if $S_y$ has no outbound transition, remove $t_{xy}$ from the graph. If $S_y$ possesses outbound transitions, perform the following:

1. select the first transition, assumed to lead to node $S_z$
2. remove this transition from the graph
3. modify the graph so that $t_{xy}$ leads to $S_z$ instead
4. add all other transitions leading out of $S_y$ to $S_z$

The graph is now fully optimized, and is ready for its implementation.

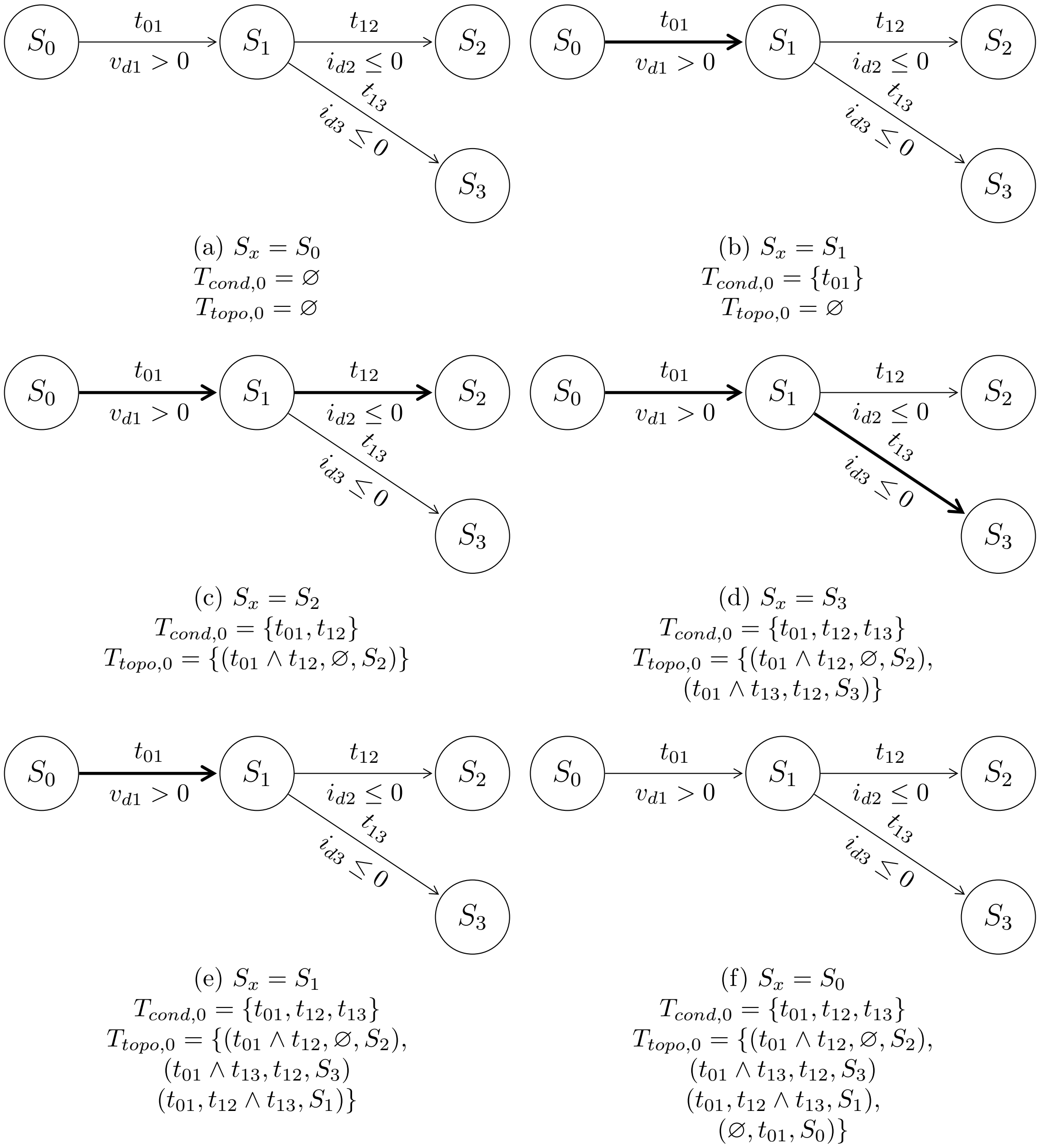
### 3.3.6 Writing the results in tables

Once the steps described in the previous section are performed, the reduced subgraph is finally ready to be converted to two tables describing the behavior of the system:

- the first table $T_{cond}$ links the current topology to the list of guard conditions to be evaluated
- the second table $T_{topo}$ links the result of the evaluation to the final topology

These two tables are easily obtained during a final parsing of the graph. Each root topology $S_i$ (and hence each subgraph) is represented by its own entries $T_{cond,i}, T_{topo,i}$ in the tables. The procedure from section 3.3.5 is rewritten here in its completed version; the new steps are highlighted in bold.

1. All sets are assumed are empty, and the current node $S_x$ is set to $S_i$
2. **Initialize $T_{cond,i}, T_{topo,i}$ to an empty set, and start parsing the graph normally**
3. For each transition leading out of $S_x$, repeat these steps :
   (a) The transition is added to the valid set $T_V$ or $T_I$
   (b) **Add the transition to $T_{cond,i}$**
   (c) Change $S_x$ to the node at the end of the transition and return to step 2 with the new node
   (d) When the algorithm returns, remove the current transition from the valid set and add it to the corresponding invalid set $T'_V$ or $T'_I$
4. **create an entry in $T_{topo,i}$ of the form $[valid, invalid, S_x]$ where $valid$ and $invalid$ are respectively the list of transitions in the valid and the invalid sets at the moment**
5. remove all transitions leading out of $S_x$ from the valid and invalid sets
6. If the current node is the root node, the parsing is complete. Else, return to the previous node



(a) $S_x = S_0$
$T_{cond,0} = \varnothing$
$T_{topo,0} = \varnothing$

(b) $S_x = S_1$
$T_{cond,0} = \{t_{01}\}$
$T_{topo,0} = \varnothing$

(c) $S_x = S_2$
$T_{cond,0} = \{t_{01}, t_{12}\}$
$T_{topo,0} = \{(t_{01} \wedge t_{12}, \varnothing, S_2)\}$

(d) $S_x = S_3$
$T_{cond,0} = \{t_{01}, t_{12}, t_{13}\}$
$T_{topo,0} = \{(t_{01} \wedge t_{12}, \varnothing, S_2),$
$(t_{01} \wedge t_{13}, t_{12}, S_3)\}$

(e) $S_x = S_1$
$T_{cond,0} = \{t_{01}, t_{12}, t_{13}\}$
$T_{topo,0} = \{(t_{01} \wedge t_{12}, \varnothing, S_2),$
$(t_{01} \wedge t_{13}, t_{12}, S_3)$
$(t_{01}, t_{12} \wedge t_{13}, S_1)\}$

(f) $S_x = S_0$
$T_{cond,0} = \{t_{01}, t_{12}, t_{13}\}$
$T_{topo,0} = \{(t_{01} \wedge t_{12}, \varnothing, S_2),$
$(t_{01} \wedge t_{13}, t_{12}, S_3)$
$(t_{01}, t_{12} \wedge t_{13}, S_1),$
$(\varnothing, t_{01}, S_0)\}$

**Figure 3.16: Writing the switching tables for the graph of Figure 3.10. The $\wedge$ operator is the logical *and*, $\varnothing$ is the null set. The bold arrows represent the evolution of the parsing of the graph**

**Table 3.2: Binary commutation table for the graph of figure 3.16, starting from $S_0$**

| $t_{01}$ | $t_{12}$ | $t_{13}$ | New Topology |
|---|---|---|---|
| 1 | 1 | - | $S_2$ |
| 1 | 0 | 1 | $S_3$ |
| 1 | 0 | 0 | $S_1$ |
| 0 | - | - | $S_0$ |

The steps are illustrated in figure 3.16, showing how the graph of figure 3.10 is converted to the two tables $T_{cond,0}$ and $T_{topo,0}$. The table $T_{topo,0}$ starts to be filled during step (d). When the design is ported on the real-time platform, the result of the evaluation of the conditions is directly linked to the new topology using the entries $[valid, invalid, S_x]$ : if all conditions in the *valid* are evaluated as true and all conditions in *invalid* are evaluated as false, $S_x$ is selected as the new topology. For example the entry $(t_{01} \wedge t_{13}, t_{12}, S_3)$ means that $S_3$ is selected when the transitions $t_{01}$ and $t_{13}$ are validated *and* $t_{12}$ is not validated.

These results are easily translated to a binary table, as shown on Figure 3.2 where $'-'$ corresponds to a *don't care*, which means that the result of the evaluation of the transition has no impact on the selection. On the platform, we will have to assign 1 if a condition is evaluated as true, and 0 otherwise. By comparing the resulting binary vector with the table, we are able to select the correct topology.

The two tables conclude the natural switching analysis. Additional processing is needed to properly convert the tables to a format understood by the real-time platform. These steps are described in section 5.3.4.

## 3.4 Automated Forced switching exploration and optimization

### 3.4.1 Introduction

In section 2.6.4, we introduced the forced switching module and gave a brief explanation of its inner workings. When the state of an active switch is changed, we must control all diodes to ensure that the circuit lies in a topology whose diode signals are within their specified bounds. This means that we have to test all topologies whose active switches are in the correct state to verify if the current value of the state and input variables lies in the definition domain, and select the topology for which this test is verified. Moreover, this topology must guarantee the continuity of all state variables.
The selection process is similar to the one used for the natural switching module: we select the set of conditions that have to be evaluated, depending on the current topology as well as on the new state of the active switches. The state of the diodes is then deduced from the result of the evaluation.

**Example 11.** We shall provide a simple example, using the circuit of figure 3.17. Let us assume that the active switch $K$ commutates from *on* to *off*: $D_1$ and/or $D_2$ will have

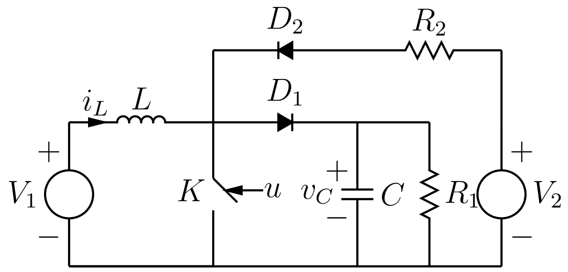


**Figure 3.17: Modified boost circuit used to illustrate the topology selection process**

to be switched, leading to a new topology whose diode voltages and/or currents are all within their bounds. The case where both diodes are *off* is ignored, since this would force the inductor current $i_L$ to zero. The definition domain of the other topologies are:

- for $D_1$ on, $D_2$ off:

$$\begin{aligned} i_{D1} &= i_L && > 0 \\ v_{D2} &= V_2 - v_C && \leq 0 \end{aligned} \tag{3.49}$$

- for $D_1$ off, $D_2$ on:

$$\begin{aligned} v_{D1} &= V_2 + R_2 i_L - v_C && \leq 0 \\ i_{D2} &= -i_L && > 0 \end{aligned} \tag{3.50}$$

- for $D_1$ on, $D_2$ on:

$$\begin{aligned} i_{D1} &= i_L + \frac{V_2 - v_C}{R_2} && > 0 \\ i_{D2} &= \frac{V_2 - v_C}{R_2} && > 0 \end{aligned} \tag{3.51}$$

The correct topology is then selected based on the evaluation of the conditions. Since the value of the signals change over time, this test must be made in real-time.

The next section will introduce a method to automatically find the set of conditions to evaluate. Furthermore, and just like we did during the natural switching analysis, we will show that this set may be largely reduced (in some cases to nothing) thanks to some interesting properties described in section 3.4.2.
Let us give an example by assuming that the circuit of Figure 3.17 starts in the topology $(K, D_1, D_2) = (on, off, off)$ and that the switch $K$ is suddenly turned off. The definition domain of the starting topology is

$$\begin{aligned} v_C &\geq 0 \\ V_2 &\leq 0 \end{aligned} \tag{3.52}$$

Hence, the second condition of (3.51) can not be true and the corresponding topology does not have to be considered as an acceptable choice of topology. Furthermore, the second condition of (3.49) is always true and must not be evaluated. The same reasoning is made for (3.50) : assuming $-i_L < 0$, the condition on $v_{D1}$ is also verified. As a consequence, the evaluation of $v_{D1}$ is not necessary. After these optimizations, only two conditions remain: $i_L > 0$ and $-i_L > 0$.

### 3.4.2 Analysis

#### General principles

When the state of an active switch is changed, the system goes from one subgraph to another, and our goal is to find which node is the correct one in the new subgraph. The graph parsing algorithm introduced in section 3.3 cannot be used directly, since it requires the knowledge of a starting node inside the new subgraph. As discussed previously, we cannot simply change the value of the active switches and use the resulting topology as the starting node since it might be unreachable, meaning that its equations cannot be written. Even if the topology is reachable, an incorrect choice could well lead to incoherent signals in the circuit. Indeed, any instantaneous modification of the states would lead to virtual Dirac pulses across the circuit according the state evolution equation

$$\frac{\mathrm{d}x}{\mathrm{d}t} = Ax + Bu \Rightarrow Ax + Bu = \delta(t) \tag{3.53}$$

These pulses do not bear any physical meaning and do not happen in the real circuit. They may however be used as a tool to converge to the correct topology, as described in multiple articles (to cite a few: [16, 36, 38, 78]). Let us look at the circuit of figure 3.17. We assume that the active switch $K$ is suddenly opened at time $t_{0^-}$. We have to find the correct configuration of the diodes. One of the choices is the topology with all switches *off*. For this topology, we write

$$\begin{aligned} v_{d1} &= V_1 - v_L - v_C \\ v_{d2} &= -V_1 + v_L + V_2 \end{aligned} \tag{3.54}$$

Opening all switches does not leave any path for the inductor current, forcing the creation of Dirac pulses in the circuit:

$$\begin{aligned} v_{d1} &= V_1 + Li_L(t_{0^-})\delta - v_C \\ v_{d2} &= -V_1 - Li_L(t_{0^-})\delta + V_2 \end{aligned} \tag{3.55}$$

Since Dirac pulses are supposed to have an infinite value, we only need to keep the impulsive part of these equations. The voltage $v_{d1}$ becomes infinitely positive (for an infinitely small period of time), forcing $d_1$ to switch to the on state.

This method effectively implements the forced switching module using an approach similar to the natural switching module, jumping from node to node until all diode signals are within their designated limits. Its biggest difficulty lies in writing of (3.55), relying on a modified state-space of the form

$$y(t) = Cx(t) + Du(t) + E\frac{\mathrm{d}x}{\mathrm{d}t} \tag{3.56}$$

This form is not compatible with the other parts of our system, and requires a modification of the state-space solver. Furthermore, it does not solve the problem of starting from an unreachable topology. That is why we have developed another method.

An alternative lies in the systematic study of *all* topologies whose active switches are in the correct state. The correct topology is the one whose diode currents and voltages are all within their limits, which implies that the current state of the system is inside

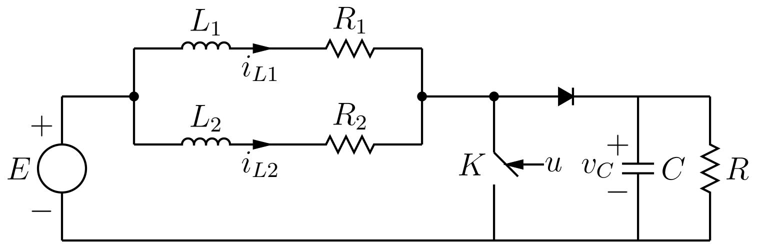


**Figure 3.18: Circuit where the state $i_{L2}$ is not forced to a constant, but to an expression $i_{L2} = -i_{L1}$**

the definition domain of the topology. Assuming a circuit with $n_d$ diode, we have to evaluate at most the $n_d$ conditions of each of the $2^{n_d}$ independent topologies. This number grows quickly (eg. $6 * 2^6 = 384$ conditions for a circuit containing 6 diodes) and becomes impractical unless we develop reduction algorithms to minimize the set of reachable topologies.
The study of Dirac pulses carried out previously may be interpreted in another way: since changing the value of the states during the transition induces unphysical signals in the system, we only need look at topologies that preserve the value of the state variables. Or, otherwise said, we can eliminate any topology that would introduce a discontinuity in the states.
These discontinuities only happen if the state becomes a *forced state* (see section 2.2.3) whose value is no longer controlled by a differential equation, but may be computed using the instantaneous value of the other states and inputs. Mathematically, a state $x^*(t)$ is forced if

$$x_i^*(t) = \sum_{j \neq i} c_j x_j(t) + \sum_j d_j u_j(t) \tag{3.57}$$

where $c_j, d_j$ are constant coefficients.
The boost converter provides a simple example of forced states. If both switches are open, the inductor current $i_L(t)$ is forced to zero, and is no longer controlled by a differential equation. Similarly, $v_C(t)$ is forced to zero when both switches are closed. Since the states were not forced before the topology change, we do not keep these two topologies in our list of available choices. Note however that when a state variable becomes unforced (i.e. $x_i^* = x_i$), this event neither changes its value, nor creates a Dirac pulse.

In the boost circuit, the signals were forced to a zero value. This is not always the case, as illustrated by the circuit of figure 3.18. Assuming the transistor starts in the on state, the two currents are controlled individually by their respective state space equations

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_{L1} \\ i_{L2} \end{bmatrix} = \begin{bmatrix} -\frac{R_1}{L_1} & 0 \\ 0 & -\frac{R_2}{L_2} \end{bmatrix} \begin{bmatrix} i_{L1} \\ i_{L2} \end{bmatrix} + \begin{bmatrix} \frac{1}{L_1} \\ \frac{1}{L_2} \end{bmatrix} E \tag{3.58}$$

When both switches are open, the states are linked through Kirchhoff's current law. Selecting $i_{L1}$ as the independent variable, we obtain

$$\begin{aligned} \frac{\mathrm{d}i_{L1}}{\mathrm{d}t} &= -\frac{R_1 + R_2}{L_1 + L_2} i_{L1} \\ i_{L2} &= -i_{L1} \end{aligned} \tag{3.59}$$

If the transistor is switched off, keeping the diode in the off state would force $i_{L2}$ to change its value. The only remaining choice is to switch on the diode to allow both states $i_{Li}$ to be controlled separately; i.e.

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_{L1} \\ i_{L2} \end{bmatrix} = \begin{bmatrix} -\frac{R_1+R_3}{L_1} & -\frac{R_2}{L_1} \\ -\frac{R_3}{L_2} & -\frac{R_2+R_3}{L_2} \end{bmatrix}\begin{bmatrix} i_{L1} \\ i_{L2} \end{bmatrix} + \begin{bmatrix} \frac{1}{L_1} \\ \frac{1}{L_2} \end{bmatrix} E \tag{3.60}$$

Since the states and the inputs must keep their value after the topology change, we deduce the following corollary : *the current system state is part of the definition domain of both the starting and the ending topology*, and all conditions defined by the diode signals of the starting topology should still be verified. This very important result will be used during the reduction steps carried out in the next sections.

The analysis tool works by performing the following step for each stable starting topology and for each available configuration of active switches:

1. build a list of all reachable topologies according to the current configuration of the active switches. As a reminder, a topology is reachable if its state equations can be written (which is not the case if, for instance, a voltage source is shorted by a switch, see Section 3.3.2)
2. remove all topologies that would modify a state to a forced value
3. remove all topologies that are incompatible with the starting topology
4. simplify the conditions of the remaining elements in the list

### Forced states detection

While the evolution of a forced state is given by the associated differential equation, its instantaneous value may also be computed using the other states as well as the input signals. We repeat here the equation first encountered in section 2.2.3:

$$x(t) = C_x x(t) + D_x u(t) \tag{3.61}$$

If none of the states are forced then this equation has got a unique solution: $C_x$ is the identity matrix $I$ while $D_x$ is filled with zeros. Any other value means that the value of a state is forced by the other signals.

With this form, detecting if a state is forced is an immediate task: we only have to look at the corresponding line in the $C_x$ and $D_x$ matrices. These matrices are obtained by adding the states to the output vector (i.e. by *measuring* the states) before performing the state-matrices extraction. If we use *Plecs*, this is be done by adding ampere-meters in series with inductors and voltmeters in parallel with the capacitors. If the states are forced to a set value, this value is computed as part of the Gauss elimination performed during the circuit compilation [16].

This same procedure allows to detect if a state jumps from a forced value to *another* forced value.

### Incompatible topologies

The first major step of our search algorithm is to remove all topologies whose definition domain is incompatible with the definition domain of the starting topology $S_0$. Since the system lied in the definition domain of the starting topology, all of its stability conditions are respected

$$\begin{aligned} v_{dj} &\leq 0 \ \forall v_{dj} \in D(S_0) \\ i_{dh} &> 0 \ \forall i_{dh} \in D(S_0) \end{aligned} \tag{3.62}$$

where $D(S_0)$ is the domain of $S_0$, as defined in section 2.5.1. The first line corresponds to the voltage of the diodes in the off-state while the second line corresponds to the current across the diodes in the on-state. This is also true for any positive linear combination of these signals

$$\begin{aligned} \sum_{v_{dj} \in D(S_0)} k_j v_{dj} \leq 0 \\ \sum_{i_{dh} \in D(S_0)} l_h i_{dh} > 0 \\ k_j, l_h \geq 0 \forall j, h \end{aligned} \tag{3.63}$$

where at least one $l_h$ is strictly positive. Finally, we can combine these two expressions into

$$\sum_{v_{dj} \in D(S_0)} k_j v_{dj} - \sum_{i_{dh} \in D(S_0)} l_h i_{dh} \leq 0 \tag{3.64}$$

Meanwhile, a topology $S_x$ is compatible with the starting topology if there exists some combination of states and inputs such that

$$\begin{aligned} v'_{dj} &\leq 0 \forall v'_{dj} \in D(S_x) \\ i'_{dh} &> 0 \forall i'_{dh} \in D(S_x) \end{aligned} \tag{3.65}$$

Looking at (3.64) and (3.65), we notice that $S_x$ will be incompatible with $S_0$ if we are able to find a diode current $i'_{dx} \in D(S_x)$ respecting the following condition

$$\begin{aligned} \sum_{v_{dj} \in D(S_0)} k_j v_{dj} - \sum_{i_{dh} \in D(S_0)} l_h i_{dh} = i'_{dx} \\ k_j, l_h \geq 0 \forall j, h \end{aligned} \tag{3.66}$$

In this case, (3.65) will never be true and $S_x$ is safely removed from the list of candidates. To write this in matrix form, we first write the diode currents and voltages as

$$\begin{bmatrix} i_d \\ v_d \end{bmatrix} = \begin{bmatrix} E_I & F_I \\ E_V & F_V \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{3.67}$$

This form, that we already introduced in section 3.3.5, uses the following variables

- $v_d = \begin{bmatrix} v_{d1} \\ \vdots \\ v_{dn_{off}} \end{bmatrix}$ is a column vector containing the voltages of all diodes in the off-state

- $i_d = \begin{bmatrix} i_{d1} \\ \vdots \\ i_{dn_{on}} \end{bmatrix}$ is a column vector containing the currents of all diodes in the on-state

- $E_V, E_I, F_I, F_V$ are the state matrices linking $x$ and $u$ to the diode signals

Similarly, we write $i'_{dx}$ using its output equation:

$$i'_{dx} = \begin{bmatrix} E_{i'_{dx}} & F_{i'_{dx}} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{3.68}$$

where $E_{i'_{dx}}$ and $F_{i'_{dx}}$ are the lines of $E'_I$ and $F'_I$ (the output matrices for topology $S_x$) corresponding to $i'_{dx}$ With these notations, and by stating that (3.66) must be true for all $x$ and $u$, we restate the problem as finding $k, l$ to satisfy

$$\begin{bmatrix} E_V^T & -E_I^T \\ F_V^T & -F_I^T \end{bmatrix} \begin{bmatrix} k \\ l \end{bmatrix} = \begin{bmatrix} E_{V,1}^T & \dots & E_{V,n_{off}}^T & -E_{I,1}^T & \dots & -E_{I,n_{on}}^T \\ F_{V,1}^T & \dots & F_{V,n_{off}}^T & -F_{I,1}^T & \dots & -F_{I,n_{on}}^T \end{bmatrix} \begin{bmatrix} k_1 \\ \vdots \\ k_{n_{off}} \\ l_1 \\ \vdots \\ l_{n_{on}} \end{bmatrix} = \begin{bmatrix} E_{i'_{dx}}^T \\ F_{i'_{dx}}^T \end{bmatrix}$$
$$k_j, l_h \geq 0 \forall j, h \tag{3.69}$$

where $E_I^T$ is the transpose of $E_I$.
A similar condition exists on the diode voltages:

$$\sum_{i_{dh} \in D(S_0)} l_h i_{dh} = v'_{dx}$$
$$l_h \geq 0 \forall h \tag{3.70}$$

where at least one coefficient is different from zero (i.e. $v'_{dx}$ is not the null vector). In matrix form:

$$\begin{bmatrix} E_I^T \\ F_I^T \end{bmatrix} \begin{bmatrix} l \end{bmatrix} = \begin{bmatrix} E_{I,1}^T & \dots & E_{I,n_{on}}^T \\ F_{I,1}^T & \dots & F_{I,n_{on}}^T \end{bmatrix} \begin{bmatrix} l_1 \\ \vdots \\ l_{n_{on}} \end{bmatrix} = \begin{bmatrix} E_{v'_{dx}}^T \\ F_{v'_{dx}}^T \end{bmatrix}$$
$$l_h \geq 0 \forall h \tag{3.71}$$

To find these identities, we apply the two-step simplex algorithm (see section 3.3.3) to each diode voltage and current of all potential target topologies.

**Example 12.** We will use the circuit of figure 3.19 to illustrate the procedure which we have just described. We assume the circuit starts with $K, D_1, D_2$ on and $D_3, D_4$ off.
When $K$ is switched off, we have to select the correct state for the four diodes. It is easy to see that $D_1$ and $D_2$ must remain in the on-state to allow $i_{L1}$ and $i_{L2}$ to keep their value.

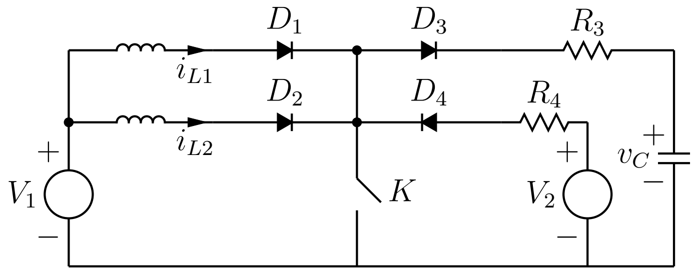


**Figure 3.19: Circuit used to illustrate incompatible topologies in the hard switching module**

For the same reason, we eliminate the topology with $D_3$ and $D_4$ in the off-state, leaving us only with $(D_1, D_2, D_3, D_4) = (1,1,0,1), (1,1,1,0), (1,1,1,1)$ as the only remaining choices. We now apply our procedure, starting with the domain of the starting topology $S_0$: the diode signals are given by

$$\begin{bmatrix} i_{D1} \\ i_{D2} \\ v_{D3} \\ v_{D4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} i_{L1} \\ i_{L2} \\ v_C \\ V_1 \\ V_2 \end{bmatrix} \tag{3.72}$$

While the diode signals for $(D_1, D_2, D_3, D_4) = (1,1,0,1)$ are

$$\begin{bmatrix} i'_{D1} \\ i'_{D2} \\ v'_{D3} \\ i'_{D4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{L1} \\ i_{L2} \\ v_C \\ V_1 \\ V_2 \end{bmatrix} \tag{3.73}$$

We will try to find a combination of the form (3.66) or (3.70). For diode current $i'_{D4}$, we attempt to find $(k, l)$ such that

$$\begin{bmatrix} -i_{D1} & -i_{D2} & v_{D3} & v_{D4} \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ k_3 \\ k_4 \end{bmatrix} = i'_{D4} \tag{3.74}$$
$$l_1, l_2, k_3, k_4 \geq 0$$

Using (3.72) and (3.73), and by stating that this equality must be true for all values of the states and inputs:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ k_3 \\ k_4 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.75}$$
$$l_1, l_2, k_3, k_4 \geq 0$$

One of the possible solutions is $(l_1, l_2, k_3, k_4) = (1, 1, 0, 0)$. Since all coefficients are positive, this topology is incompatible with the previous one. If we apply the same procedure to $(D_1, D_2, D_3, D_4) = (1, 1, 1, 1)$, we obtain

$$\begin{bmatrix} i'_{D1} \\ i'_{D2} \\ i'_{D3} \\ i'_{D4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \frac{R_4}{R_3+R_4} & \frac{R_4}{R_3+R_4} & -\frac{1}{R_3+R_4} & 0 & \frac{1}{R_3+R_4} \\ -\frac{R_3}{R_3+R_4} & -\frac{R_3}{R_3+R_4} & -\frac{1}{R_3+R_4} & 0 & \frac{1}{R_3+R_4} \end{bmatrix} \begin{bmatrix} i_{L1} \\ i_{L2} \\ v_C \\ V_1 \\ V_2 \end{bmatrix} \tag{3.76}$$

Writing (3.66) for $i'_{D4}$:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ k_3 \\ k_4 \end{bmatrix} = \begin{bmatrix} -\frac{R_3}{R_3+R_4} \\ -\frac{R_3}{R_3+R_4} \\ -\frac{1}{R_3+R_4} \\ 0 \\ \frac{1}{R_3+R_4} \end{bmatrix} \tag{3.77}$$
$$l_1, l_2, k_3, k_4 \geq 0$$

This system is solved for $(l_1, l_2, k_3, k_4) = \left(\frac{R_3}{R_3+R_4}, \frac{R_3}{R_3+R_4}, \frac{1}{R_3+R_4}, \frac{1}{R_3+R_4}\right)$. Again, the coefficients are positive and the topology is incompatible.
Finally, we write the diode signals for $(D_1, D_2, D_3, D_4) = (1, 1, 1, 0)$

$$\begin{bmatrix} i'_{D1} \\ i'_{D2} \\ i'_{D3} \\ v'_{D4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ R_3 & R_3 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} i_{L1} \\ i_{L2} \\ v_C \\ V_1 \\ V_2 \end{bmatrix} \tag{3.78}$$

Clearly, the signals for $i'_{D1}$, $i'_{D2}$ and $i'_{D3}$ are compatible with the previous topology. If we look at $v'_{D4}$, we write the

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ k_3 \\ k_4 \end{bmatrix} = \begin{bmatrix} R_3 \\ R_3 \\ 1 \\ 0 \\ -1 \end{bmatrix} \tag{3.79}$$
$$l_1, l_2, k_3, k_4 \geq 0$$

which is solved for $(l_1, l_2, k_3, k_4) = (-R_3, -R_3, -1, -1)$. This time, the coefficients are negative and the topology is compatible.

Once the topologies are eliminated, we proceed to the next step : the elimination of useless conditions.

### Condition elimination

While some of the conditions may preclude a topology from being selected because it is incompatible with the starting topology, another group of conditions may be automatically true because of their relation with the conditions of the starting topology. From (3.64) and (3.65), we notice that if we are able to write one of the diode signals of a topology in one of the forms

$$\begin{aligned} \sum_{v_{dj}\in D(S_0)} k_j v_{dj} - \sum_{i_{dj}\in D(S_0)} l_j i_{dj} &= v'_{dx} \\ \sum_{i_{dj}\in D(S_0)} l_j i_{dj} &= i'_{dx} k_j, l_j \quad \geq 0 \forall j \end{aligned} \tag{3.80}$$

then the associated condition defined by (3.65) is automatically verified. Hence, there is no need to evaluate the condition. The elimination process leads to a reduction of the logic behind the forced switching module.
A subtle variant consists in adding the signals from the topology $S_x$ except $i'_{dx}$ or $v'_{dx}$ to (3.80):

$$\begin{aligned} \sum_{v_{dj}\in D(S_0)} k_j v_{dj} - \sum_{i_{dj}\in D(S_0)} l_j i_{dj} + \sum_{v'_{dj}\in D(S_x)\backslash v'_{dx}} k'_j v'_{dj} - \sum_{i'_{dj}\in D(S_x)} l'_j i'_{dj} &= v'_{dx} \\ \sum_{i_{dj}\in D(S_0)} l_j i_{dj} + \sum_{i'_{dj}\in D(S_x)\backslash i'_{dx}} l'_j i'_{dj} &= i'_{dx} \\ k_j, l_j, k'_j, l'_j &\geq 0 \forall j \end{aligned} \tag{3.81}$$

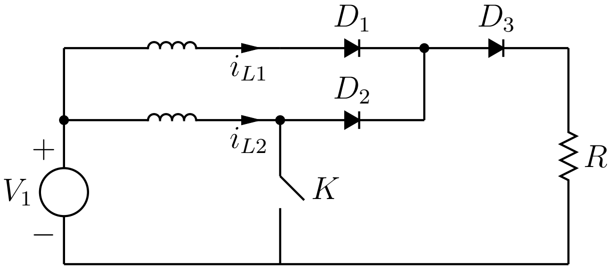


**Figure 3.20: Circuit showing the use of** (3.81)

**Example 13.** This second variant is not used much, and only appears in some circuits. As an example, let us look to the circuit of figure 3.20, and let us assume that the starting switch configuration is $(K, d_1, d_2, d_3) = (on, on, off, on)$. The domain of this topology is defined by

$$\begin{aligned} i_{d1} &= i_{L1} & &> 0 \\ v_{d2} &= -Ri_{L1} & &\leq 0 \\ i_{d3} &= i_{L1} & &> 0 \end{aligned} \tag{3.82}$$

We now assume that the transistor is switched off. We want to write the conditions leading to the selection of $(K, d_1, d_2, d_3) = (off, on, on, on)$ as the new topology. The definition domain is

$$\begin{aligned} i'_{d1} &= i_{L1} & > 0 \\ i'_{d2} &= i_{L2} & > 0 \\ i'_{d3} &= i_{L1} + i_{L2} & > 0 \end{aligned} \tag{3.83}$$

In this case, we easily see that $i'_{d3} = i_{d1} + i'_{d2}$, and there is no need the evaluate $i'_{d3}$ to select the correct topology. Furthermore, there is also no need to look at $i'_{d1}$ since it is equal to $i_{d1}$, which means that the condition $i'_{d1} > 0$ is true by virtue of (3.82).

The algorithm used to detect signals that can be written in the form of (3.81) is the same as the one used to detect incompatible topologies, and will not be detailed further. If, at any point, all conditions of a topology $S_x$ are detected as being always true, there is no need to look any further: this topology will always be selected when the circuit starts in the correct topology and the active switches are modified to the configuration of $S_x$. An immediate corollary is that the diodes are never modified unless the configuration of the active switches changes. This process is repeated for all combinations of starting topologies and configuration of active switches.
Once the elimination is complete, the last step is to establish a correspondence between the result of the evaluations and the correct topology.

### 3.4.3 Writing the results in tables

Like we did for the natural switching module in section 3.3.6, the final part of the forced switching exploration is the creation of tables to match the results to the new configuration of the diode. The first table, called the *condition table* hereafter, takes the current topology and the new configuration of the active switches as its input and outputs the list of conditions to evaluate.
A second table, the *switching table*, uses the result of the evaluation to find the new state of the diodes. Assuming a circuit with $n_a$ active switches and $n_d$ diodes, we have a condition table containing at most $2^{n_a+n_d}2^{n_a}$ entries (since there are $2^{n_a+n_d}$ possible starting topologies, and $2^{n_a}$ new configurations of transistors. Each entry leads to a switching table containing at most $2^{n_d} * n_d$ entries, since we have to select among $2^{n_d}$ configurations of diodes, each possessing up to $n_d$ conditions.
Note that that the actual size of the tables is likely to be much smaller, because all our previous operations have removed topologies and conditions for the lists. Furthermore, we do not study topologies that are unstable or unreachable, because the circuit cannot start or end in any of these configurations.
Since the exploration algorithm is much simpler than the one we used for the natural switching module, the tables are much easier to compile. For each starting configuration, we do the following for all configurations of diodes that were not marked as incompatible:

1. add all conditions corresponding to the domain of the configuration and that were not eliminated during a previous step to the current entry of the condition table. Conditions already in the table are discarded

2. add an entry to the switching table, with the expected result of the evaluation of the conditions as the input, and the diode configuration as the output

**Example 14.** We will use the circuit of figure 3.17 as an example. The circuit contains 8 different topologies, 2 of which are unstable and hence ignored in this analysis: $(K, D_1, D_2) = (on, on, off)$ and $(K, D_1, D_2) = (on, on, on)$. Indeed, they lead to $i_{D1} = 0$ which is incompatible with $D_1$ on. The condition table is written in Table 3.3, where

| $K$ | $D_1$ | $D_2$ | $K'$ | Conditions |
|---|---|---|---|---|
| off | off | off | off | $\varnothing$ |
| on | off | off | on | $\varnothing$ |
| off | on | off | off | $\varnothing$ |
| off | off | on | off | $\varnothing$ |
| on | off | on | on | $\varnothing$ |
| off | on | on | off | $\varnothing$ |
| off | off | off | on | $V_2 \leq 0, \frac{V_2}{R_2} > 0$ |
| on | off | off | off | $i_L > 0,\ -i_L > 0$ |
| off | on | off | on | $V_2 \leq 0, \frac{V_2}{R_2} > 0$ |
| off | off | on | on | $V_2 \leq 0, \frac{V_2}{R_2} > 0$ |
| on | off | on | off | (3.49)(3.50)(3.51) |
| off | on | on | on | $V_2 \leq 0, \frac{V_2}{R_2} > 0$ |
| on | on | off | – | unstable |
| on | on | on | – | unstable |

**Table 3.3: Condition Table for the circuit of Figure 3.17**

$K, D_1, D_2$ is the configuration of the switches in the previous topology and $K'$ is the new state of the active switch. As expected, we only need to evaluate some conditions if $K' \neq K$ : the diodes may only change if at least one active switch is modified. When the transistor is switched on, the conditions are always the same and correspond to the test of $v_{D2}$ and $i_{D2}$.

The study of $(K, D_1, D_2, K') = (on, off, off, off)$ was carried in section 3.4.1. A similar process was applied to $(K, D_1, D_2, K') = (on, off, on, off)$. Each of these combinations also possesses its own switching table to interpret the results of the evaluation of

| $V_2 \leq 0$ | $\frac{V_2}{R_2} > 0$ | $D_1$ | $D_2$ |
|---|---|---|---|
| 1 | - | off | off |
| 0 | 1 | off | on |

**Table 3.4: Switching Table for** $(K, K') = (off, on)$

| $i_L > 0$ | $-i_L > 0$ | $D_1$ | $D_2$ |
|---|---|---|---|
| 1 | - | on | off |
| 0 | 1 | off | on |

**Table 3.5: Switching Table for** $(K, D_1, D_2, K') = (on, off, off, off)$

the condition (Table 3.4 to 3.6). These tables show that if all conditions associated with a topology are met (a '1' in the corresponding column), then this topology is selected. We arbitrarily selected a preference order for the topologies, this order is not crucial. Once the tables are known, the diodes are controlled in real-time by following these steps:

1. Select the conditions to evaluate using the condition table
2. Compute the signals and compare the result to zero
3. Control the diodes according to the comparison and to the switching table

### Improvements

A few enhancements can be made to further reduce the amount of conditions to evaluate.

Since we must select one of the $n$ topologies from the switching table, it follows that we only need to evaluate the conditions associated with $n-1$ topologies. If the result of the comparison yields that none of these topologies should be selected, then the last one is selected. The conditions associated with the excluded topology are eliminated, effectively removing a column from the table. For example, the switching table 3.6 becomes table 3.7 once the last column is eliminated and the columns are expanded to show the individual conditions.
Another improvement is made by finding conditions that are mutually exclusive, in which case one of the two may be eliminated. For example, Table 3.7 is further reduced to 3.8 by noting that $\frac{V2 - v_C}{R_2} > 0$ and $V_2 - v_C \leq 0$ are mutually exclusive.
The creation of the tables concludes the forced switching exploration. We have now extracted all the information needed by the real-time platform, which will be designed in the next chapter. In the next section, we will apply the complete offline tool to a number of test cases, allowing us to better show the advantages and limitations of our system.

| $\frac{V2-v_C}{R_2} > 0$ and | $i_L > 0$ and | $-i_L > 0$ and | | |
|---|---|---|---|---|
| $i_L + \frac{V2-v_C}{R_2} > 0$ | $V_2 - v_C \leq 0$ | $V_2 + R_2 i_L - v_C \leq 0$ | $D_1$ | $D_2$ |
| 1 | - | - | on | on |
| 0 | 1 | - | on | off |
| 0 | 0 | 1 | off | on |

**Table 3.6: Switching Table for** $(K, D_1, D_2, K') = (on, off, on, off)$

| $i_L + \frac{V2-v_C}{R_2} > 0$ | $\frac{V2-v_C}{R_2} > 0$ | $i_L > 0$ | $V_2 - v_C \leq 0$ | $D_1$ | $D_2$ |
|---|---|---|---|---|---|
| 1 | 1 | - | - | on | on |
| 0 | - | 1 | 1 | on | off |
| - | 0 | 1 | 1 | on | off |
| All other combinations | | | | off | on |

**Table 3.7: Switching Table for** $(K, D_1, D_2, K') = (on, off, on, off)$**, modified by removing the conditions associated with the last topology. The columns from Table 3.6 are expanded to show the individual conditions.**

## 3.5 Test Cases

### 3.5.1 Introduction

In this section, we will apply our offline algorithms to a wide range on power converters. The results of these analyses, implemented in *Python*, will be interpreted in order to highlight the strengths and the shortcoming of our methodology. We will also assess the accuracy of the waveforms obtained using the simulation algorithm described in section 2.6 by comparing them to their continuous-time equivalents obtained by using the offline simulator *PLECS*® with a time step of 50ns. The accuracy will be judged by evaluating the *relative error*, defined here as the instantaneous difference between the emulated and its expected value, divided by the typical (i.e. the average) value of the signal. We have selected to use the interleaved algorithm to ensure that the diode currents and voltages stay within their bounds.

At this point, the simulations are not made on the real-time platform but on a computer: the only goal is to judge the accuracy and not the computing performance. The results on the real-time platform should be very similar as long as there are enough resources to hold the design on the platform and the total duration of the computations is shorter that the selected time-step.

Unless stated otherwise, the discrete time-step for the simulations is equal to $1\mu s$ and the solver is based on the Trapezoidal integration.

| $i_L + \dfrac{V2 - v_C}{R_2} > 0$ | $\dfrac{V2 - v_C}{R_2} > 0$ | $i_L > 0$ | $D_1$ | $D_2$ |
|---|---|---|---|---|
| 1 | 1 | - | on | on |
| - | 0 | 1 | on | off |
| All other combinations | | | off | on |

**Table 3.8: switching table obtained after removing conditions that are mutually exclusive from Table 3.7**

## 3.5.2 The ideal boost converter

### Circuit description

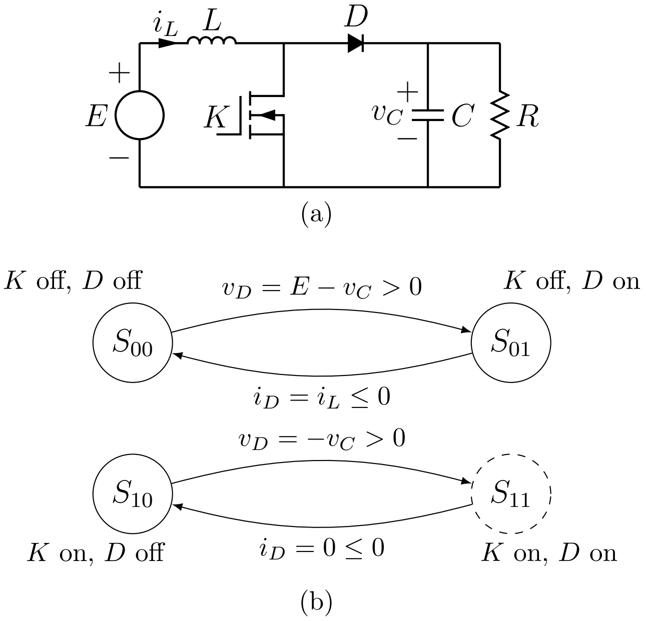


**Figure 3.21: Boost converter and its transition graph, separated in two subgraphs according to the state of the active switch**

The boost converter (Figure 3.21a) is, along with the buck and the buck-boost converters, one of the best known power electronics circuits. Indeed, this DC-to-DC converter is able to raise a DC voltage with a high efficiency (90% or more depending on the power rating and the quality of the components) [46] which is very useful when operating from low voltages like a single cell battery. The size is another advantage of this converter, and it is easily integrated in small circuits such as battery chargers. Since this circuit is so ubiquitous, it is a natural target for simulators. In this section, we will study the ideal (lossless) boost converter.

### Topological study and automaton extraction

The boost converter is composed of

- two independent reactive components ($L$ and $C$)
- one voltage source $E$
- one active switch $K$, assumed ideal and bidirectional
- one diode $D$, assumed as ideal

The 2 switches lead to a maximum of $2^2$=4 topologies, each of which described by a 2-by-2 $A$ matrix and a 2-by-1 $B$ matrix. Among those four topologies, only three are stable and reachable. Indeed, the topology were both switches are in the on-state leads to $v_C = 0$ and, in turn, $i_D = 0$ which means that D cannot be on. This topology is thus unstable according to the definitions stated in section 3.3.3.
In order to draw the automaton, the topologies are separated in two subgraphs – one for each configuration of $K$ – as represented in Figure 3.21b. The transitions between two nodes of the same subgraph correspond to the natural commutation of the diode. The condition associated with the transition is obtained by computing the diode voltage or current according to the method described in section 3.2.

### Natural switching analysis

The natural switching analysis of the automaton is very simple and straightforward. The analysis, described in section 3.3, is applied to each subgraph of Figure 3.21b. When $K$ is off, the circuit may switch back and forth between the nodes of the upper graph. The path linking the two nodes contains a single transition, whose guard depends on the current state of the diode (note: the inductor voltage $v_L = L\dfrac{\mathrm{d}i_L}{\mathrm{d}t}$ is equal to zero when the diode is off, hence the diode voltage is equal to $E - v_C$). The analysis is even simpler when $K$ is on, since there is only one available transition (i.e. the only path out of topology $(K, D) = (on, off)$ leads to an unstable topology). As such, we do not need to evaluate any signal to control the diode. A placeholder condition of the form $0 \leq 0$ is added instead to insure that all topologies have the same number of conditions to evaluate.

| $K$ | $D$ | Signal to evaluate |
|---|---|---|
| off | off | $c \equiv E - v_C > 0$ |
| off | on | $c \equiv i_L \leq 0$ |
| on | off | $c \equiv 0 \leq 0$ |

**Table 3.9: Condition table for the natural switching module of the ideal boost converter**

These results are written in Table 3.9, indicating the condition to evaluate, and on table 3.10 allowing us to find the new state of the diode according to the previous topology and to the result of the evaluation of the condition.

### Forced switching analysis

The simplicity of the circuit translates into a very short forced switching analysis. When $K$ is switched on, then the circuit must jump to the topology $(K, D) = (on, off)$. It

| $K$ | $D$ | $c$ | new $D$ |
|---|---|---|---|
| off | off | $E - v_C \leq 0$ | off |
| off | off | $E - v_C > 0$ | on |
| off | on | $i_L \leq 0$ | off |
| off | on | $i_L > 0$ | on |
| on | off | $-$ | off |

**Table 3.10: Switching Table for the natural switching module of the ideal boost converter**

remains in this configuration until $K$ is switched off. When this change occurs, we have to select the correct topology among the two upper nodes of Figure 3.21b. Looking at the states matrices of topology $(K, D) = (off, off)$, we write the state variables in the form of (2.22) :

$$\begin{bmatrix} v_C(t) \\ i_L(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_C(t) \\ i_L(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} E \tag{3.84}$$

This topology defines $i_L(t)$ as a forced state, while it is not the case when $K$ is on. This leads us to remove this topology from the list of available choices, leaving $(K, D) = (off, on)$ as the only remaining configuration.

| $K$ | $D$ | $K'$ | new $D$ |
|---|---|---|---|
| $-$ | $-$ | off | on |
| $-$ | $-$ | on | off |

**Table 3.11: Switching Table for the forced switching module of the ideal boost converter**

In both cases, the selection of the correct topology is made without computing any condition whatsoever. The (very simple) switching table is represented on Table 3.11.

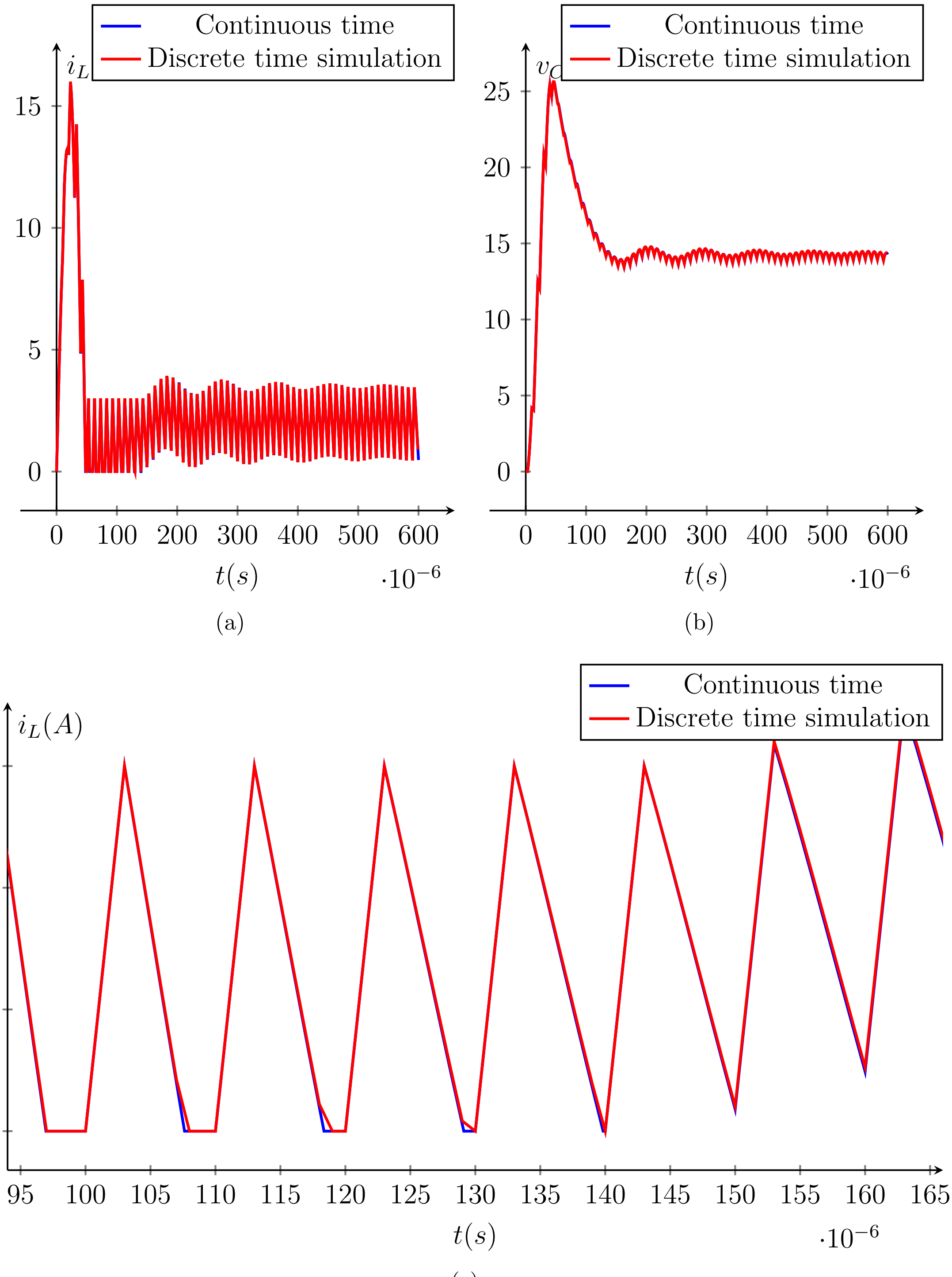
### Simulations

To assess the performance of the simulation, we select the following parameters for the elements of the circuit

$$\begin{aligned} L &= 10\mu H \\ C &= 10\mu F \\ R &= 10\Omega \\ E &= 10V \end{aligned} \tag{3.85}$$

These values correspond to a relatively fast circuit. For instance, the output RC circuit is characterized by a time-constant of $RC = 100\mu s$ while the resonant frequency of the LC is $f_c = \frac{1}{2\pi\sqrt{LC}} \approx 16kHz$. The transistor is controlled with a $100kHz$, 30% duty cycle PWM. This frequency allows us to verify how well the simulator performs when the switching period is only equal to ten time steps.

The results, shown on Figure 3.22, are promising. Looking at the global waveforms of the current $i_L$ (Figure 3.22a)and the voltage $v_C$ (Figure 3.22b), it is almost impossible to dissociate the simulation from the continuous-time plot. The only way to show any



(a)

(b)

(c)

**Figure 3.22: Simulation of the boost converter of Figure 3.21a with the parameters stated in** (3.85)**. The transistor is controlled with a** $100kHz$**,** 30% **duty cycle PWM.**

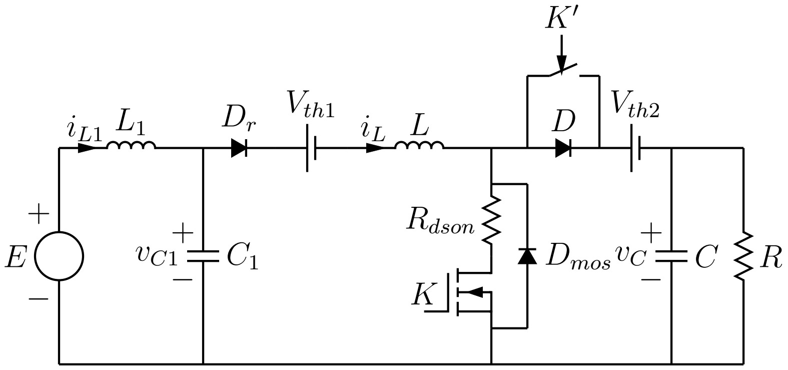


**Figure 3.23: More detailed implementation of a boost converter, with added elements at the input and a switch in parallel of $D$ to emulate a failure mode by shorting the diode**

difference is to zoom very closely on the current $i_L$ (Figure 3.22c): since we do not use any form of zero crossing, the we do not detect the exact moment at which the current reaches zero. However, thanks to the use of the output solver, the value of the current is forced to zero on the next time-step when the circuit switches to the topology where both switches are open and hence enters in discontinuous conduction mode.

### Interpretation of the results

The computing power needed to implement the natural switching module on a real-time platform is very small : an *adder* is enough to compute the condition, and the tables are easily implemented using a few conditional statements. The forced switching module is even simpler, and needs only a single *if* statement to control the diode. It makes sense that this circuit could be simulated with a very small time step, allowing a very precise representation of the signals. The simulations have proved the validity of our solving algorithm, which can now be used for more complex circuits.

## 3.5.3 More detailed boost converter implementation

### Circuit description

The circuit drawn on Figure 3.23 represents a more detailed implementation of a boost converter, with added elements to better emulate the real behavior of the switching elements. The diode $D_r$ is inserted at the input and acts as a reverse polarity protection. To better represent the imperfections of the switches, we add the forward voltage of the diode and the $R_{DS}$ resistance of the MOS transistor. The anti-parallel diode $D_{mos}$ of $K$ is also represented; a controlled switch $K'$ allows us to simulate the short-circuit failure of $D$. The failure detection can be made by supervising the current flowing through $D$.

### Topological study and automaton extraction

Since we did not add any reactive component, the amount of state variables in the modified circuit is still equal to two. Meanwhile, the threshold voltages of the diodes act as independent voltage sources, increasing the number of inputs to 3. Because of the

added switches, the circuit now contains $2^5 = 32$ topologies, each described by a $2 \times 2$ $A$ matrix and a $2 \times 3$ $B$ matrix .
To write the topologies in a compact form, we use the following notation : $T_{KK',DD_rD_{mos}}$. For instance, the topology with $D$ and $K'$ in the on-state and all other switches in the off-state is written as $T_{01,100}$. The systematic study of the circuits reveals that the major part of the topologies are unstable. For instance, seventeen topologies have a diode current equal to zero. In the end, only ten topologies are kept. Among them, the subgraph corresponding to $K = 1, K' = 0$ is the one containing the largest amount of stable nodes (i.e. four). Its subgraph is represented in Figure 3.24c: $T_{10,000}$, $T_{10,010}$ , $T_{10,100}$ and $T_{10,110}$ are stable, while $T_{10,001}$, $T_{10,011}$, $T_{10,101}$ and$T_{10,111}$ are unstable.

### Natural switching analysis

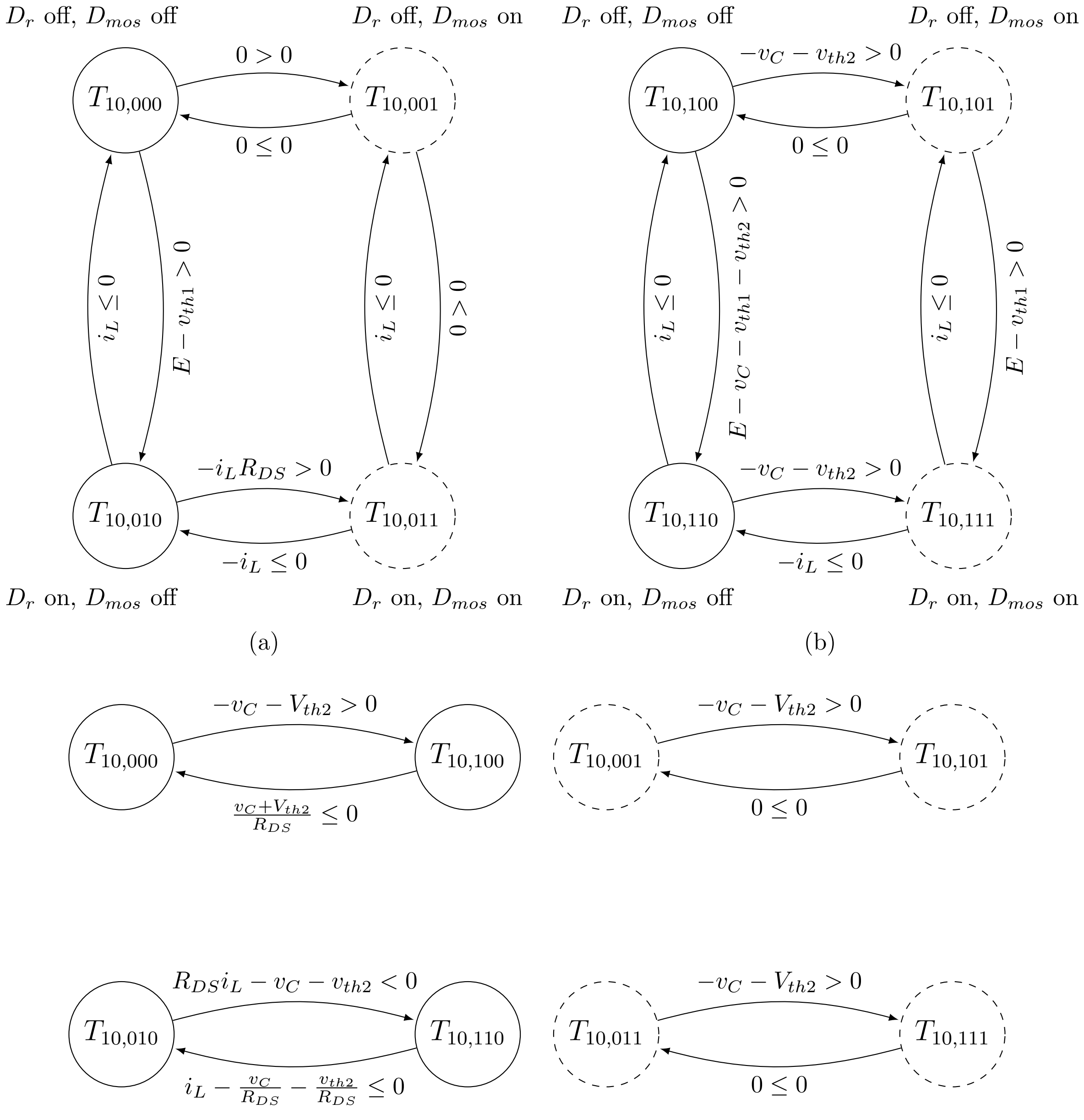
The study of the graphs reveals that at most four conditions are needed to find the correct topology. These conditions correspond to the reduced version of 3.24, when the circuit starts from the topology $T10,000$. The final graph is represented on Figure 3.25. Note that the transitions linking $T_{000}$ to $T_{100}$ and $T_{010}$ to $T_{110}$ should be the same since we start from a topology where $i_L = 0$. However, this kind of simplification is not performed in the natural switching algorithm for the moment. Integrating these additional constraints could lead to even fewer conditions to evaluate and an additional increase in performance.

### Forced switching analysis

Is most cases, there is no need to evaluate any conditions because only a single choice remains for the diodes after a change in the configuration of the active switches. Only three changes lead to the evaluation of a single condition:

- when the circuit starts from $T_{KK',DD_rD_{mos}} = T_{00,110}$ and $K$ is switched on
- when the circuit starts from $T_{01,010}$ and both switches are changed at the same time
- when the circuit starts from $T_{11,010}$ and $K'$ is switched off

In all three cases, the evaluation of the condition $R_{DS}i_L - v_C - v_{th2} \leq 0$ allows to select the state of the diode $D$.
Another important result is that if, starting from $T_{00,000}$ (ie. all switches off), we switch on $K$, the current $i_L$ will not start to flow immediately since we need an additional time-step to switch on $D_r$. If $K$ does not stay on for a long time (when compared to the time step), this could lead to distortions in the signal, as we will show in the next section.



(a)

(b)

(c)

**Figure 3.24: Switching graph of the practical boost from Figure 3.23 with $K = 1$ and $K' = 0$. The graph is divided in three parts for easier reading (a) $d$ off (b) $d$ on (c) links between the two.**

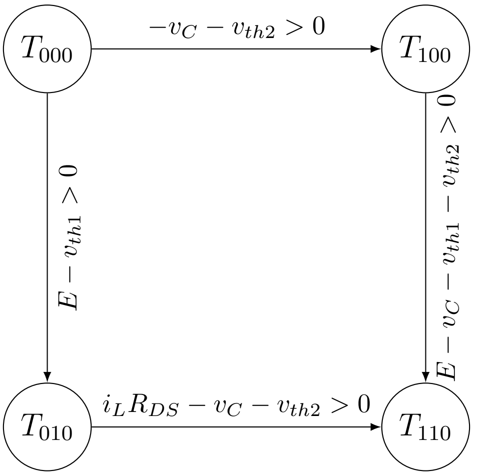


**Figure 3.25: Reduced version of the graph of Figure 3.24 after the natural switching analysis is performed**

### Simulations

To test the accuracy of the simulation, we select the following parameters

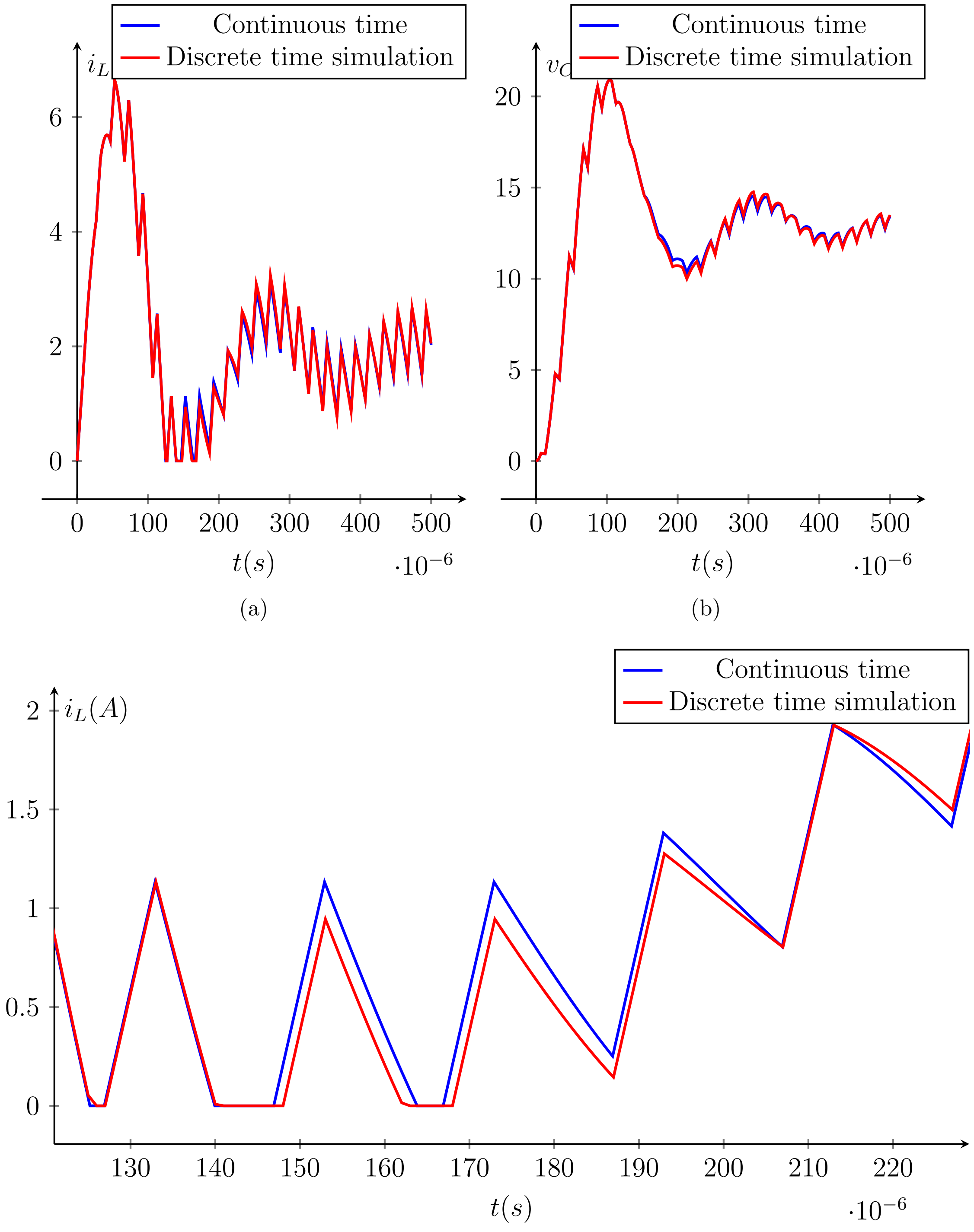
$$\begin{aligned} L &= 50\mu H \\ C &= 10\mu F \\ R &= 10\Omega \\ E &= 10V \\ v_{th1} = v_{th2} &= 0.5V \\ R_{DS} &= 100m\Omega \end{aligned} \tag{3.86}$$

These are mostly the same as those used for the ideal boost converter, except that we selected a bigger inductance for $L$, for reasons that will be explained shortly. The value of $R_{DS}$ is typical for low current (up to $5A$) MOS transistors. The simulation of the circuit with $K'$ left open and $K$ controlled with a $50kHZ, 30\%$ duty cycle PWM is represented in Figure 3.26.

While the long term ($500\mu s$) simulated waveforms seem to correspond to their continuous-time counterpart, the zoom of figure 3.26c shows a local distortion when $K$ is switched on. As explained previously, this is due to the fact that an additional time step is needed to switch $D_r$ on, and is unavoidable with our present forced switching analysis. Still, the signals quickly converge to their correct value once $i_L$ stays positive and $D_r$ stays in the on-state

A more complex analysis could follow the present version with a modified natural switching analysis to allow the switching of the diode. This additional step could make use of our knowledge on the previous topology to reduce the graphs.

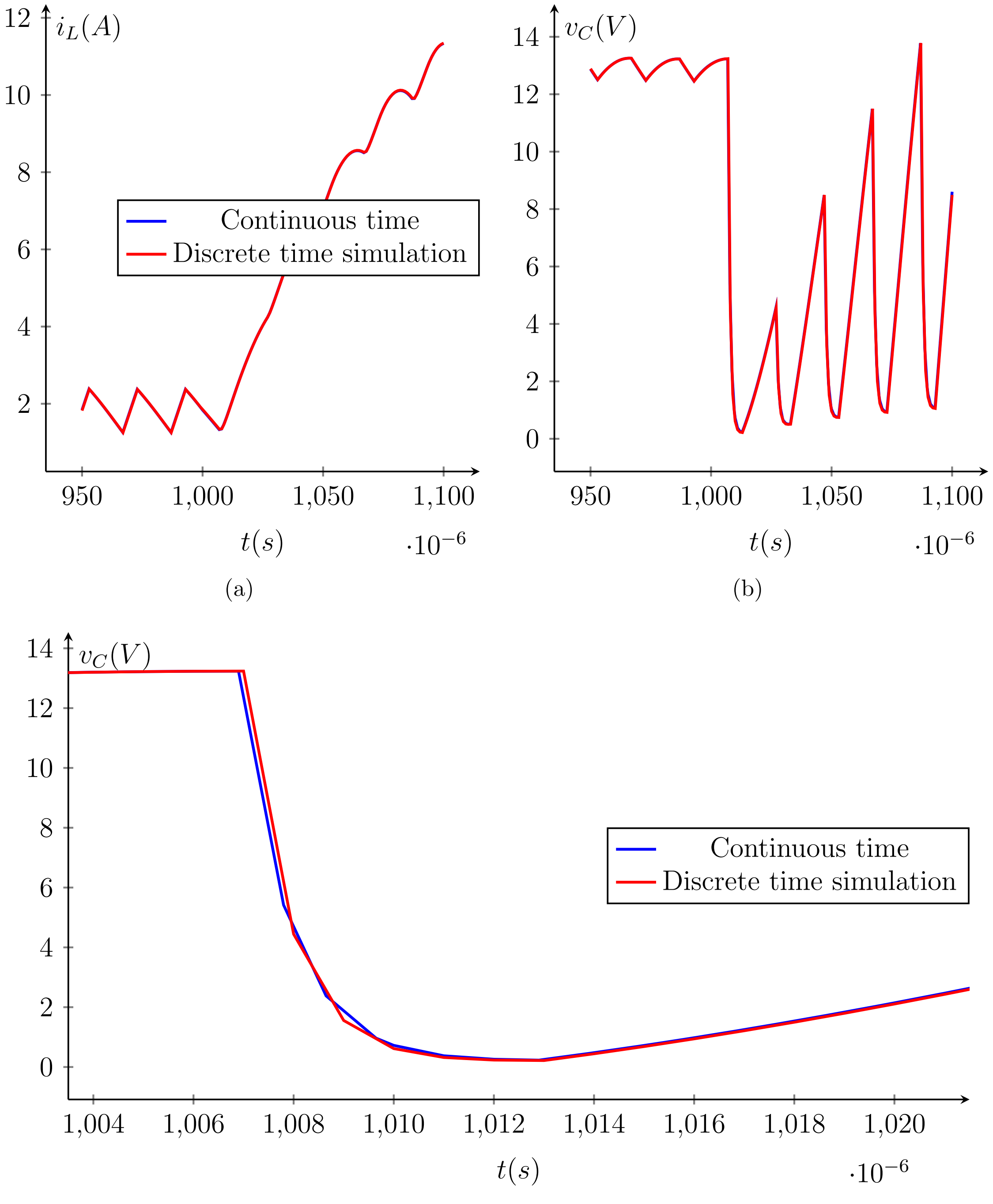
Another interesting test is to switch on $K'$ at some point during the simulation to emulate a short-circuit in the diode. When $K$ is switched on during the normal PWM cycle, the output capacitor should discharge quickly across $R_{DS}$ before being recharged by the current $i_L$ when $K$ is switched off. Since the time-constant $R_{DS}C = 1\mu s$ is equal to the time step during the discharge, this poses a nice challenge for our algorithm since we must


(a)

(b)

(c)

**Figure 3.26: Simulation of the boost converter of Figure 3.23 with the parameters stated in** (3.86)**. The transistor is controlled with a** $50kHz$**,** 30% **duty cycle PWM**



(a)

(b)

(c)

**Figure 3.27: Simulation of a short-circuit across $D$, showing the accuracy of the simulator even for signals with short time-constants : the capacitor $C$ is quickly discharged across $K$ during the on time of the PWM period, and is recharged by $i_L$ during the off time**

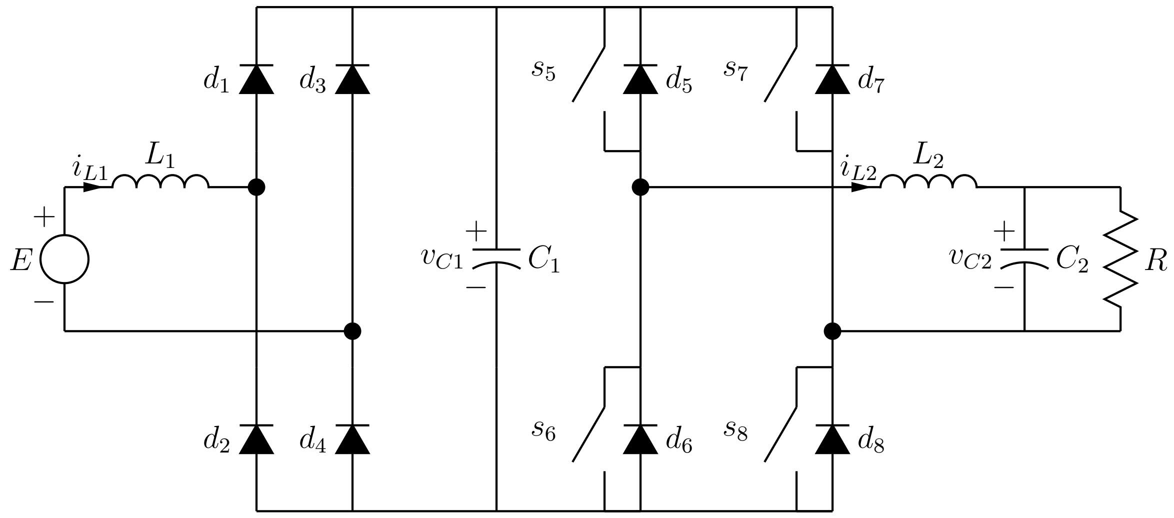


**Figure 3.28: Two stage AC/DC converter**

be able to track signals that change very quickly. As shown on Figure 3.27, these short time-constants do not pose any problem to our algorithm, which is very encouraging.

### Interpretation of the results

As shown in the previous section, adding these elements makes the analysis more complicated, because the base number of topologies is multiplied by eight (due to the addition of the three switches). However, we were able to keep the total amount of calculations low by removing many nodes from the graph.
The simulations have shown very accurate results (the error is inferior to 1%) as long as the circuit stays in continuous-conduction mode. When the circuit enters discontinuous conduction, the delay between the commutation of the active switches and the commutation of $D_r$ could pose problems if the control signals are not bigger than the sampling period by a factor of 10. In our case, the error between the two signals is equal to around 1 % of the nominal value of the current.
This problem could be alleviated by reducing the time step, which is certainly possible for such a small circuit, but could be more challenging for larger converters. Another way to eliminate this delay would be to develop a more complete forced switching algorithm, which integrates a simplified natural switching to change the state of the diodes.

## 3.5.4 The two-stage AC/DC converter

### Circuit description

The circuit shown in Figure 3.28 is an example of two-stage AC-to-DC power converter. The AC input is first connected to the main DC bus through a passive full-wave rectifier. The input inductance is placed to model the impedance of the connections. A four-quadrant step-down converter (also called full bridge or H-bridge) connects the DC bus to the load. The load is represented by a resistor, but it could be any load including a DC motor.

### Topological study and automaton extraction

The complete circuit contains 12 switches, leading to a base number of 4096 topologies. Applying the switching algorithms directly to the circuit would lead to a lot of useless computations and very large switching graphs (in the case of the natural switching module). Thankfully, only 102 of these topologies are stable, which greatly simplifies the process. During normal operation, only a smaller number of configurations are really used, but these simplifications are not detected by our current algorithm (for example, the fact that the capacitor voltages cannot be negative is not detected).

### Natural switching analysis

The four active switches give $2^4 = 16$ subgraphs and each of them contains $2^8 = 256$ topologies of 8 diodes; fortunately only a few of these topologies are stable. The maximum number of stable nodes found in a single subgraph is equal to 9.

The complete parsing of the graphs reveals that the maximum number of conditions is equal to seven, and is reached when the circuit starts from the following configurations:

- $d_6, d_7$ on, and $s_5, s_8$ on
- $d_5, d_8$ on, and $s_6, s_7$ on
- $d_1$ to $d_4$ on, and $s_5, s_8$ on
- $d_1$ to $d_4$ on, and $s_6, s_7$ on

In all of these case, the voltage on the DC bus is equal to zero while power is transmitted to the load, which should not happen in practice. However, these topologies cannot be considered as unstable, since they possess a non-null definition domain. If for example, we look at the first of these possibilities, we have the following diode voltages and currents:

$$\begin{aligned} v_{d1}, v_{d2}, v_{d3}, v_{d4} &= 0 \qquad \leq 0 \\ V_{d5}, v_{d8} &= 0 \qquad \leq 0 \\ i_{d6}, i_{d7} &= i_{L2} \quad > 0 \end{aligned} \tag{3.87}$$

These conditions can be respected if we select appropriate values of the inductor currents (as initial value, for example). It is also interesting to look at what happens when we start from the topology $S_0$ for which all active switches and diodes are off, as represented on figure 3.29. All nodes correspond to the topologies where the active switches are kept off and the legend indicates which diodes are on: we see that the transition to $S_2$ corresponds to an output voltage higher that the DC bus voltage. Again, this case should never happen in practice, unless the $C_2$ possesses an initial voltage and $C_1$ is initially discharged.

Nevertheless, the maximum number of conditions is lower than the number of signals that we would have to evaluate if we had used a non-iterative single-step natural switching module of an engine based on the modified nodal analysis (in which case, we would have to compute eight signals).

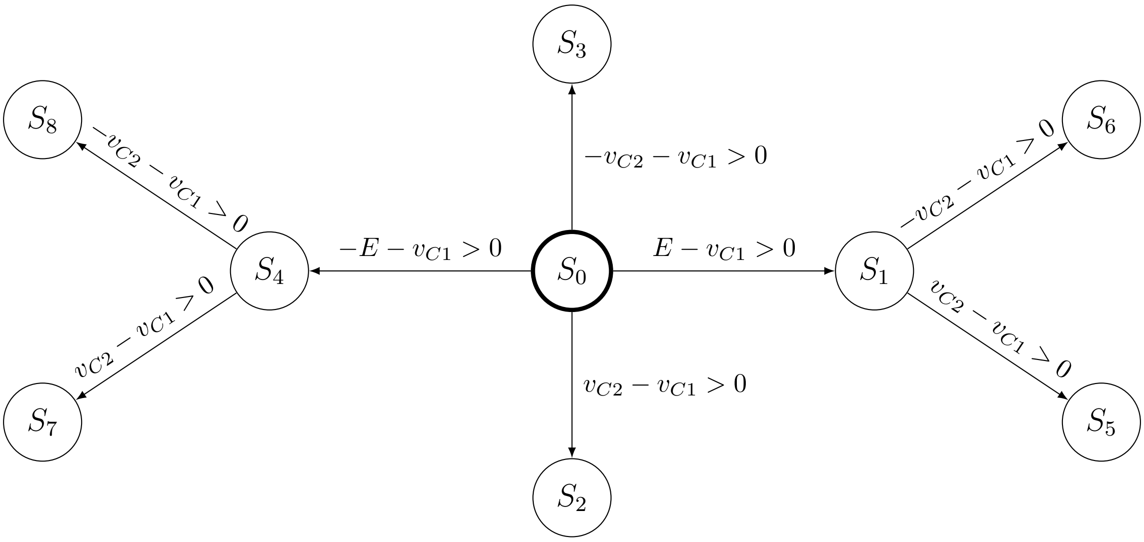


**Figure 3.29: Reduced switching graph for the circuit of figure 3.28, when starting from the topology with all switches in the off state ($S_0$), with the other nodes corresponding to the following diodes in the on-state $S_1 : (d_1, d_4)$, $S_2 : (d_5, d_8)$, $S_3 : (d_6, d_7)$, $S_4 : (d_2, d_3)$, $S_5 : (d_1, d_4, d_5, d_8)$, $S_6 : (d_1, d_4, d_6, d_7)$, $S_7 : (d_2, d_3, d_5, d_8)$, $S_8 : (d_2, d_3, d_6, d_7)$**

### Forced switching analysis

Despite the large number of elements in the circuit, the forced switching analysis reveals that only *one* needs to be evaluated at most to find the correct topology after any change in the configuration of the switches. This is expected since, when any of the switches is opened, one of the diodes in its leg will be switched on to allow $i_{L2}$ to flow. If, for instance, $s_5$ is opened, then $d_6$ will be switched on if $i_{L2} > 0$, and $d_5$ otherwise.
Another important result is that, thanks to the decoupling capacitor $C_1$, modifying the state of the switches has no impact on the diodes of the passive rectifier.

### Interpretation of the results

We have shown that our algorithm is able to greatly simplify the problem of simulating a circuit of increased size. Indeed, the natural switching module only need seven conditions to completely parse the switching graph, while a single condition is enough for the forced switching module. This study has shown that the topologies that lead to the largest number of conditions to evaluate correspond to modes that are not found in practice, unless we deliberately put inconsistent starting values for the state variables. We can partially solve this problem by explicitly limiting the range of some of the states by adding diodes in the circuit. For example, we can add a diode in parallel with $C_1$, as shown in Figure 3.30, to explicitly prevent a negative DC bus voltage. This reduces to five the number of conditions to evaluate during the natural switching analysis.
The adopted model for the converter is very useful to test the behavior of the controller, but is not able to properly simulate any failure. For example, we cannot assess the quality of the protection circuits when both transistors of a leg are erroneously switched on at the same time: this accident would result in the instantaneous discharge of $C_1$ and in an infinite current across the transistors. The system must be modified in order to be

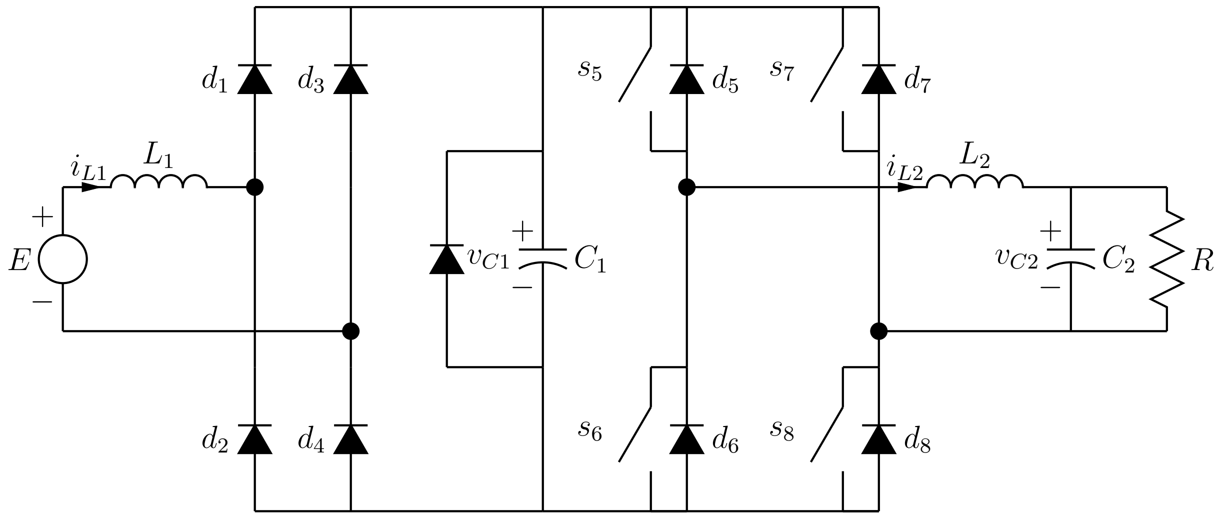


**Figure 3.30: Adding a diode in parallel with $C_1$ gives an explicit limit on the DC bus voltage**

compatible with failure mode analysis, which we shall present in the next section.

### 3.5.5 Practical implementation of the two-stage AC/DC converter

#### Circuit description

The modified circuit drawn in Figure 3.31 is able to emulate failure modes thanks to the addition of resistor $R_{mos}$. This resistance, which can be of very low value ($1m\Omega$ for low voltage, high current MOSFET transistors [79]), represents effective on-state resistance of the switches as well as the resistance of the connections. Another option to model these failure modes is to add an inductance instead, modeling the transients in the wires and connections. Thanks to this added impedance, we are now able to evaluate the very fast transients due to the discharge of the capacitor across the two transistors of a leg when both are on.

A switch connecting the output to an additional resistor is also added. This resistor can either model a sudden load change to test the response of the controller, or a low-impedance short-circuit at the output to verify if the protections are able to shutdown the system quickly enough. Note however that some drivers for the active switches are internally equipped with active protection systems which provide status bits indicating if the circuit is shorted or if the transistor could not switch. These bits can be easily modeled using external logic.

#### Topological study and automaton extraction

Thanks to $R_{mos}$, seven more topologies are available. These topologies correspond the various way we can short-circuit the active inverter. This number is doubled by the addition of the load switch: since the output resistors are decoupled from the power stage by the LC filter, changing the state of $s_L$ has absolutely no impact on the diodes.

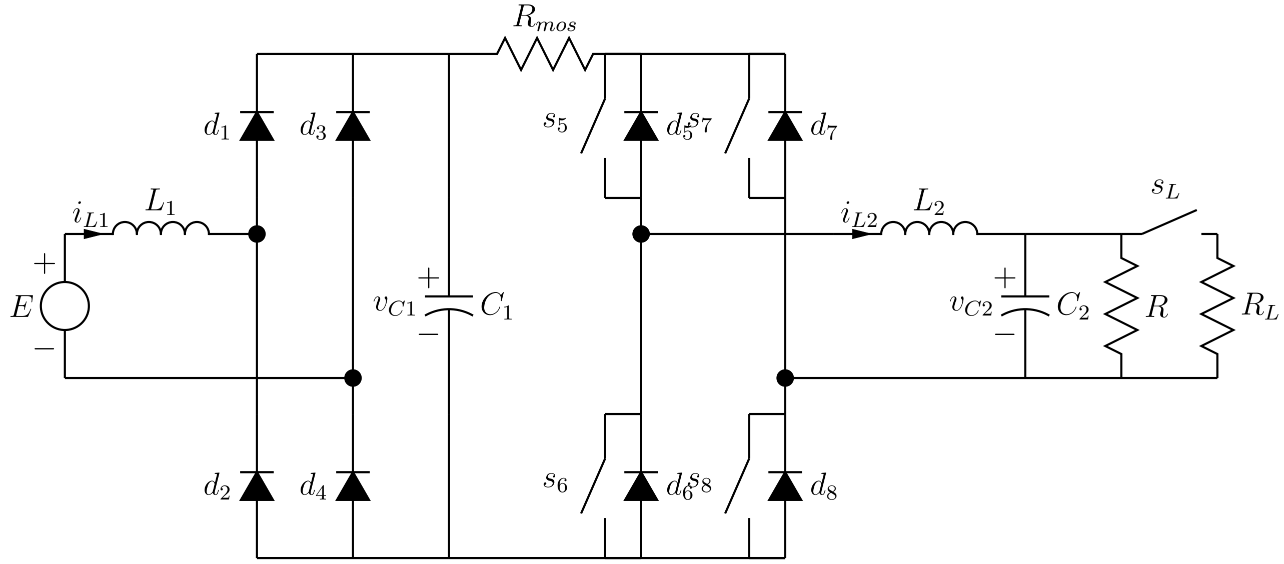


**Figure 3.31: The two-stage AC/DC circuit, with $R_{mos}$ added to allow failure modes, and an additional switch to simulate load changes**

### Natural switching analysis

The total number of conditions to evaluate is now equal to eight, one more than the result of the analysis for the converter without $R_{mos}$. This number is reached for all topologies where all the transistors of the inverter are off and both bridges are conducting (thanks to $d_1, d_4$ or $d_2, d_3$ for the rectifier, and to $d_5, d_8$ or $d_6, d_7$ for the inverter). The graph drawn in Figure 3.32 represents a part of the switching path starting from the topology with $d_1, d_4, d_6, d_7$ on. The transition linking $S_1$ to $S_2$ is associated with the condition $v_{C1} < 0$ which, as explained previously, should never be true during normal circuit operation, but could happen if $v_{C2}$ is initialized with a negative value.

### Forced switching analysis

Since the topologies for which a leg of the inverting stage is short-circuited are now allowed, additional choices are offered each time a transistor is commutated. This leads to the computation of two conditions at most (instead of one).

### Results interpretation

The overhead associated with the addition of a resistor allowing the emulation of failure mode has been shown to be relatively small: each of the commutation module needs to compute an additional condition, increasing the total amount of conditions to ten instead of eight.

Furthermore, it has been shown that, if sufficient decoupling is provided between the converter and the load, any change in the characteristics of the load (using a load switch for example) has absolutely no impact on the complexity of the switching modules. The largely used LC filter is one of the simplest ways to decouple two parts of a circuit.

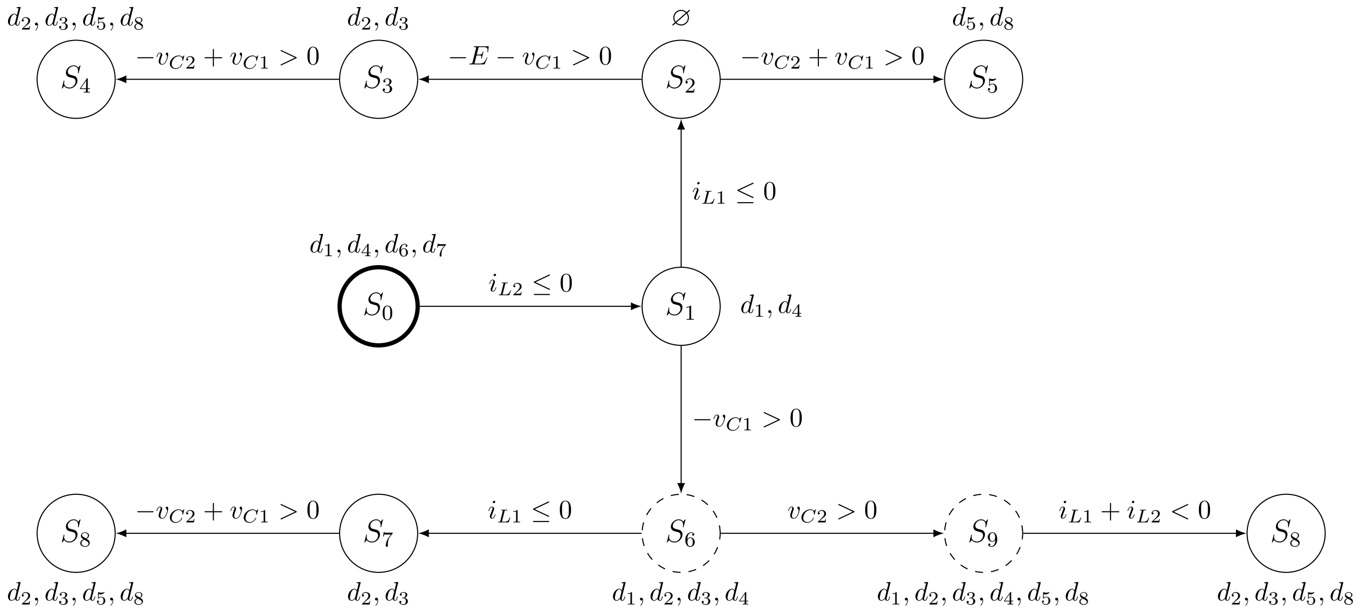


**Figure 3.32: Partial switching graph when the circuit starts in the topology with all transistors off and $d_1, d_4, d_6, d_7$ on. The $d_x$ next to the nodes indicates which diodes are conducting**

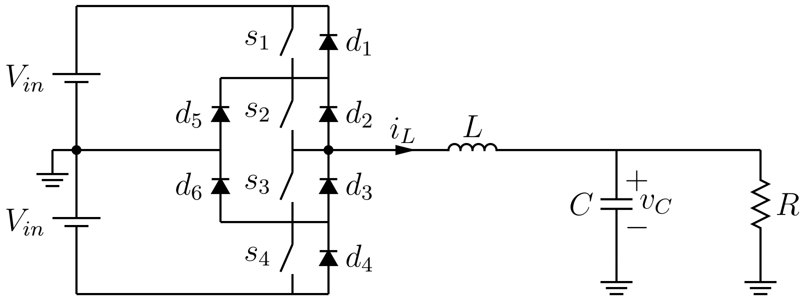


**Figure 3.33: Single Phase 3-level Neutral Point Clamped (NPC) Inverter**

### 3.5.6 The diode clamped single phase, three-level inverter

**Circuit description**

As a final test, we will illustrate the use of our algorithm on a more complex voltage-source inverter, known as the three-level neutral point clamped (NPC) inverter of Figure 3.33. The purpose of this leg is to drive the output to $0V$, $V_{in}$ or $-V_{in}$ depending on the configuration of the switches (hence the *three-level* denomination). With proper control, the use of this circuit reduces the amplitude of the harmonics in the output signals when compared to the classical two-level inverter. Furthermore, only half of the full scale voltage is perceived by the switching elements, which reduces the losses and allows the use of smaller, faster devices [80].

The main bridge configurations are :

- To drive the output to $V_{in}$
  - Switch on $s_1$ and $s_2$

  - Switch off $s_3$ and $s_4$
- To drive the output to $-V_{in}$
  - Switch on $s_3$ and $s_4$
  - Switch off $s_1$ and $s_2$
- To drive the output to $0V$
  - Switch on $s_2$ and $s_3$
  - Switch off $s_1$ and $s_4$
  - Either $d_5$ or $d_6$ will be turned on depending on the sign of the current $i_L$

These commands represent the final state of each switch. To properly switch the output voltage from one value to another, the controller must avoid any risk of cross-conduction by performing the on-to-off changes before the off-to-on changes.

The global PWM modulation can use very complex sequences of those 3 topologies. The actual sequence depends on the application and on the working point and can be aimed at reducing some harmonics or at maximizing the efficiency [81].

### Topological study and automaton extraction

This single-phase NPC contains four active switches and six diodes, leading to a base number of $2^{10} = 1024$ topologies. Of course, the amount of topologies used in practice will be much lower than that. For example, any configuration leading to a short-circuit between any two of the three input voltages ($V_{in}$, $-V_{in}$ and the ground) is eliminated. This single remark allows us to discard more than three-quarters of the topologies. Other eliminated topologies include those that lead to a conducting diode in series with an open circuit. In the end, only 56 topologies remain, which is about 5% of the initial number.

The automaton containing the largest amount of nodes corresponds to the graph obtained for $s_2$ and $s_3$ on, containing five stable topologies (with $(d_1, d_2)$, $(d_3, d_4)$, $d_5$ or $d_6$ on, or with all diodes off).

### Natural switching analysis

The analysis of the subgraphs leads to a maximum of four conditions to evaluate, which are obtained for multiple topologies.

### Forced switching analysis

Only a single condition is necessary to find the correct topology after any change in the state of the switches. For example, let us suppose that the circuit starts with $s_3$ and $s_4$ in the active state while all other switches are open, and that we suddenly open $s_3$. Only three of the reachable topologies do not lead to the loss of the state variable $i_L$ with either $(d_1, d_2)$, $(d_3, d4)$ or $d_6$ in the on-state. The first one requires $i_L < 0, V_{in} \leq 0$, the second $i_L > 0$ and the third $i_L < 0$. Since the conditions for $(d_1, d_2)$ are a super-set of the condition for $d_6$, we can safely eliminate the first choice. The conditions for the last remaining topologies are merged in a single if-else : $(d_3, d_4)$ is selected if $i_L < 0$ , and $d_6$ otherwise.

**Simulations**

To test the circuit, we must first choose a control scheme. One of the classical uses of this circuit is to generate a sinusoidal $v_C$ thanks to a PWM control signal. We choose a frequency of 500Hz for the sine wave, while the frequency of the PWM is equal to 10kHz. There are several more or less sophisticated ways to produce the 3-level control signals [81]. To demonstrate that the NPC can be emulated by our platform, a very simple 3-level PWM modulator can be built by comparing a sinusoidal reference signal to a sawtooth signal at the PWM frequency (Figure 3.34). Depending on the comparison, we select one of the three output voltages. The following values have been chosen for the components:

$$\begin{aligned} L &= 100\mu H \\ C &= 100\mu F \\ R &= 10\Omega \\ V_{in} &= 50V \end{aligned} \tag{3.88}$$

The cut-off frequency of the output LC filter is selected to lie between the sine frequency and the PWM frequency, a small value has been purposefully chosen for $L$ to accentuate the ripple. The plots of Figure 3.35 show the evolution of $i_L$ and $v_C$, and the signals obtained with our simulator are clearly almost indistinguishable from their continuous-time counterparts.

**Interpretation of the results**

The switching modules are relatively simple, since the natural switching module only requires four conditions while the forced switching module requires a single condition.

### 3.5.7 The Three-phase NPC converter

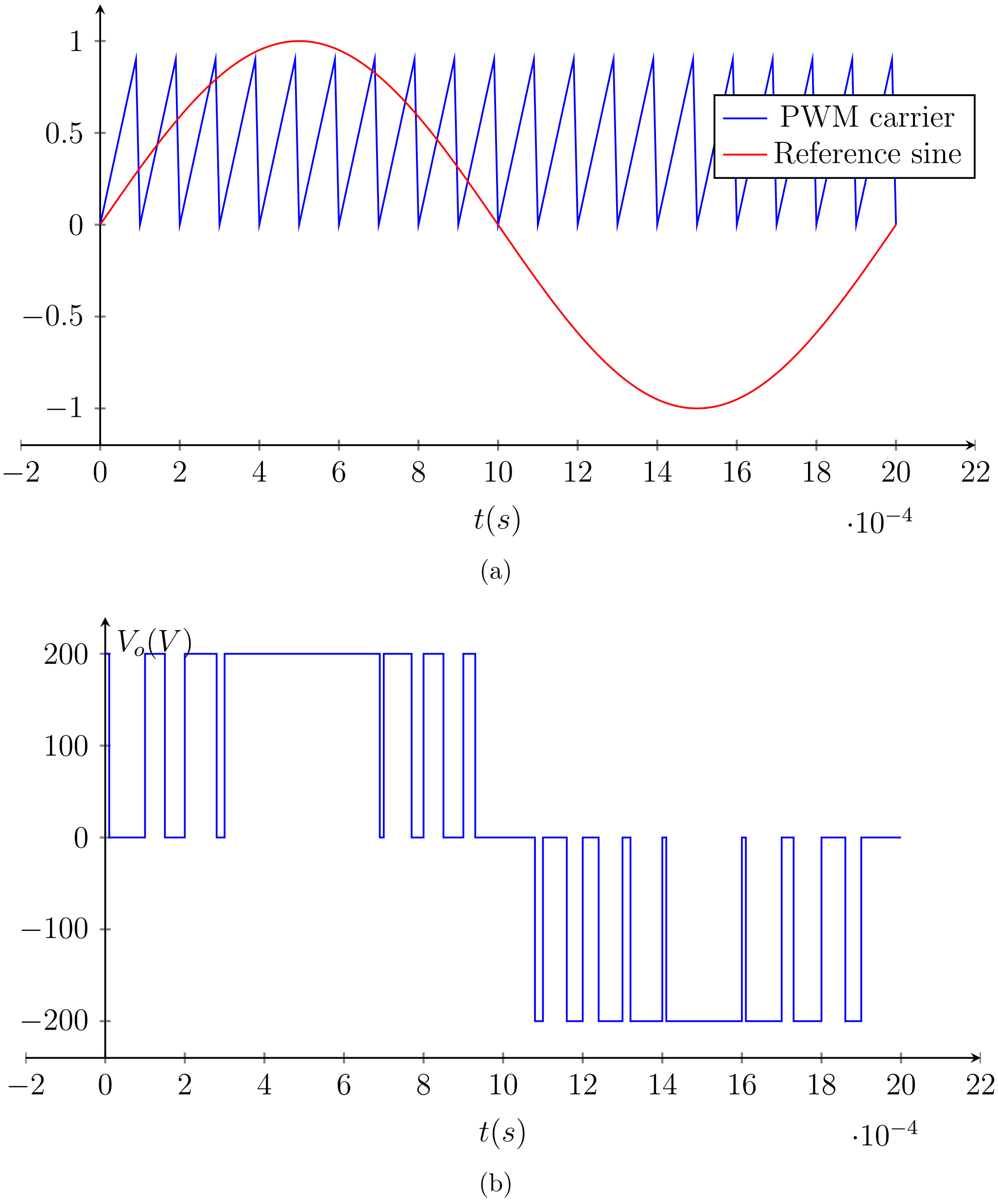
The NPC is most used in its three-phase version, represented in Figure 3.36. Unfortunately, this circuit cannot be simulated directly with our platform, since its 30 switches lead to more than a billion topologies and switching graphs containing a huge amount of nodes. This large number leads to vast overload in memory usage and a prohibitive time to analyze the transitions, and the tool itself crashed under the weight of the temporary data.

This problem, due to the exponential rise of the size of the data set, is analyzed in section 3.6.2 and a solution is given in the form of circuit partitioning in chapter 4

### 3.5.8 Comparison with solvers based on nodal analysis

At this point, it seems crucial to compare the outputs of our emulator with those obtained using a solver based on the very popular modified nodal analysis (MNA, see section 2.2.2). Indeed, this method is implemented in many circuit solvers, both offline and operating in real-time, and provides accurate results for a wide range of converters.

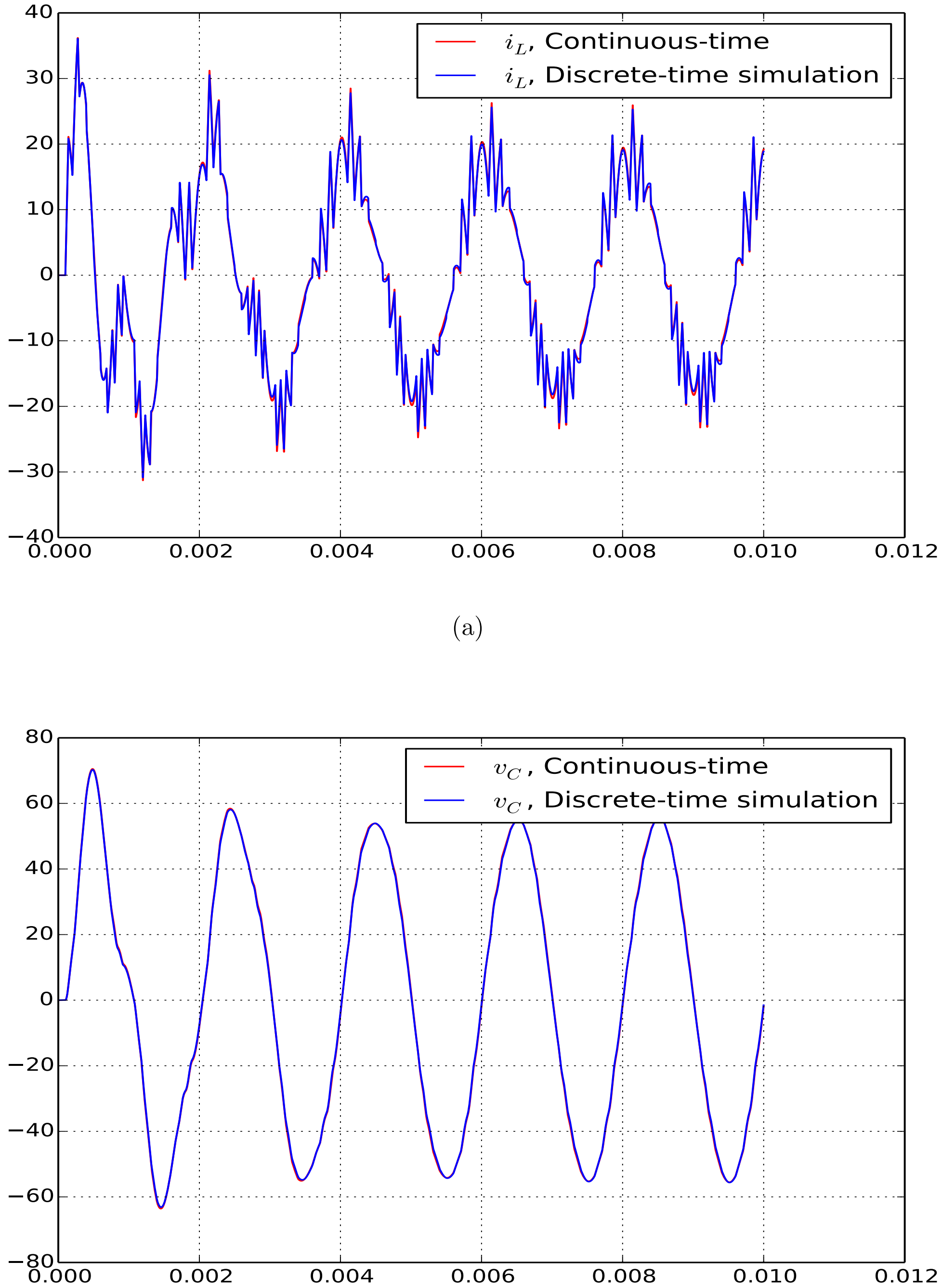
The major difference between the two methods is how the changes in the topologies are tracked. When using MNA, each switch is modeled by a current source in parallel with a resistor. By measuring the voltage and the current going through this dipole, it is for



(a)

(b)

**Figure 3.34: 3-level sinusoidal PWM modulation, based on a sawtooth PWM carrier**



(a)

(b)

**Figure 3.35: Simulation of the single-phase NPC, with the parameters given by** (3.88) **and using a 10kHz sawtooth PWM (a) inductor current (b) capacitor voltage**

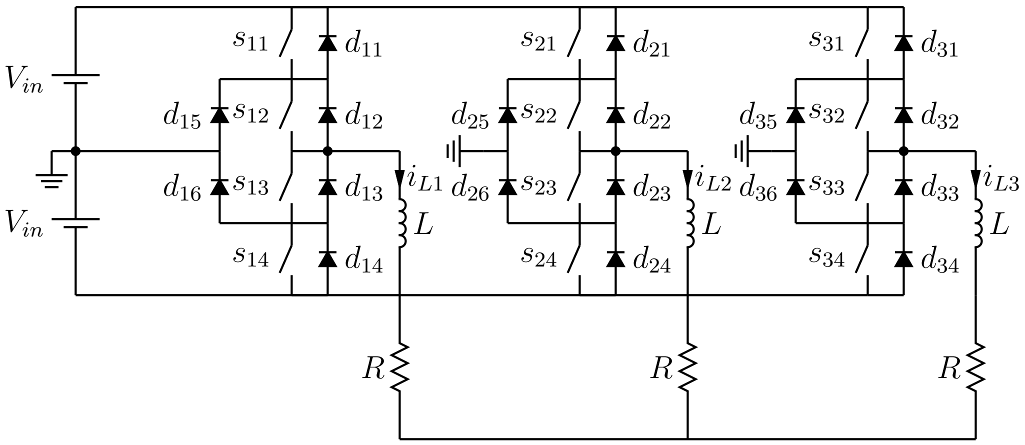


**Figure 3.36: 3-level 3-Phase Neutral Point Clamped (NPC) Inverter**

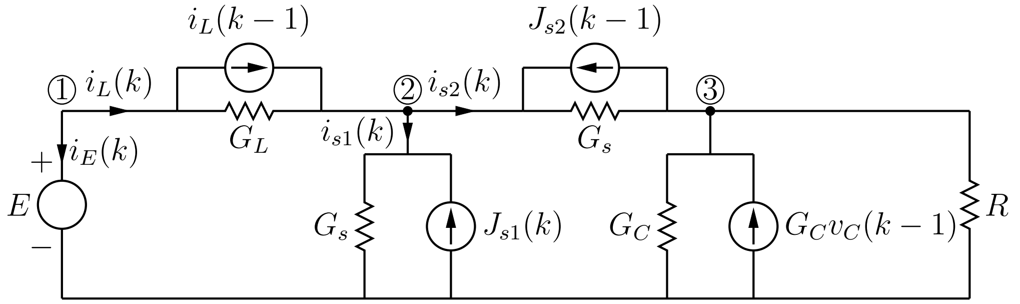


**Figure 3.37: Discrete-time model of the boost converter using MNA**

example possible to switch the diode if the current becomes negative, of if the voltage becomes positive. As such, there is no need to make a pre-analysis of the circuit since the structure of the circuit stays the same at all time, and only the current sources are modified depending on the state of the switch. Furthermore, all currents and voltages are computed as part of the solver step, making the whole algorithm easy to implement. While this could lead to think that the MNA solver is more convenient, we also have to compare the impact on the real-time platform (i.e. to find how many calculations have to be done). Indeed, the MNA solvers are also known for adding unneeded computations, as the representation of the circuit is not minimal. Furthermore, hidden in the solver is the fact that the sources have to be updated using the equivalent of a (simplified) dot product, increasing the overall complexity of the solver.

We shall compare the results of our algorithm applied to some of the circuits that were previously studied in this section with those obtained using the MNA solver. Note that comparing the waveforms is not really needed, since both methods have been shown to provide accurate results (and this error can be decreased by lowering the time-step). Instead, we will only look at the computing requirements, which could dictate the choice of the simulating platform, since a more compact design will typically use a lower time-step and allows to implement larger converters for a given platform.

### Ideal boost converter

This basic converter is actually studied in the original scientific article describing the use of MNA for switching converters [12], making the comparison immediate. The MNA solver uses five inputs (the main voltage source, the inductor current, the capacitor voltage and the current sources of the two switches) and five outputs (the currents across the inductor and the switches, and the voltage of the nodes connected to the switches and the capacitor), as shown in Figure 3.37. This leads to a 5-by-5 transition matrix to describe the whole system.

Meanwhile, all of the solvers of our algorithm only use three inputs (the voltage source, and the two states). For example, the state solver is represented by a 2-by-3 matrix computing the new value of the states; they take the form

$$\begin{bmatrix} i_L(k) \\ v_C(k) \end{bmatrix} = H \begin{bmatrix} i_L(k-1) \\ v_C(k-1) \\ E(k) \end{bmatrix} \tag{3.89}$$

This form is much more compact than the equation obtained using the MNA. Additionally, the natural switching module requires one condition, which is equivalent to dot product of the form

$$y_D = \begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} i_L(k-1) \\ v_C(k-1) \\ E(k) \end{bmatrix} \tag{3.90}$$

The forced state solver does not require any condition, since the new topology is only a function of the old topology and of the new configuration of the transistors. In total, this brings us to the equivalent of a 3-by-3 matrix, less than half of what is required by the MNA. Furthermore, is has been shown that the $k_j$ coefficients are either equal to ±1 or to 0, which means that it is possible to avoid the multiplications altogether. This is not possible using the MNA, for which all coefficients depend on the time-step.

While the boost converter is very simple and could be emulated in real-time by a wide range of computing platform including very small micro-controllers, this comparison shows the two algorithms behave for small converters, and, in this case, our analysis clearly leads to a much more compact design (only 9 products instead of 25, i.e. a 64% reduction in resource utilization).

### Two-stage AC/DC

The two stage converter was studied in section 3.5.4, and the analysis revealed that five conditions were needed for the natural switching module, and a single condition for the forced switching module. Each condition takes the source and all four states as input, which means that computing the conditions is equivalent to a 6-by-5 matrix. Updating the states requires a 4-by-5 matrix.

To use the MNA, we need to compute the current across each diode (eight independent variables), as well as the voltage of each node connected to a diode (five nodes). Updating the states adds two currents and one voltage (the output node) to the list, for a total of 16 variables. To compute these signals, the following inputs are needed:

- the input voltage

- the two currents corresponding to the previous value of the inductor currents
- the old value of the voltage of the four nodes connected to a capacitor
- the eight current sources modeling the switches

which means that the complete system requires a 16-by-15 matrix, corresponding to 240 elements. Again, the $10 * 5 = 50$ elements needed by our solver only require a fraction of the computing power needed by the MNA solver, without mentioning that many of the conditions can be written as a sum of states and inputs without requiring any multiplication, and do not need all five inputs to be computed.
Of course, these results do not take the switching tables into account. While these tables certainly require additional resources to select the correct conditions and modify the diodes, they are based on look-up tables and can be implemented using very basic elements such as multiplexers and RAM. As will be shows in section 5.3 devoted to the development of the platform, these tables can be optimized to avoid consuming too many clock cycles.

## 3.6 Known limitations

### 3.6.1 Limitations on the size of the data set

The most important limitation of our algorithm lies in the actual size of the data set, and more precisely in the risk of memory overflow in the computer running the offline analysis. For each reachable topology, we need to save

- the state matrices $A, B, C, D, C_x, D_x$, whose size depend on the number of states and independent sources
- the list of conditions and the switching table of the natural switching module
- the list of conditions and the switching table *for each combination of transistors* of the forced switching module

Each list of conditions is equivalent to a set of $C, D$ matrices whose amount of rows depends on the output of the automated analysis. Since the number of topologies is equal to $2^{\text{amount of switches}}$, all the conditions are met for a combinatorial explosion to occur should the number of switches grow. Assuming a circuit with six diodes and six transistors (eg. a three-phase inverter, which is a medium-sized circuit), we are facing 4096 topologies. For each topology, we have to store up to 256 lists of conditions in memory and 256 switching tables for the hard switching module alone. This can still be managed by a normal PC but, should this number rise to higher levels, the storage needed to keep all data in memory will rise exponentially and will eventually use all of the space available in RAM. This overflow will lead to the resort to virtual memory on disk, with lots of swaps, slowing the process to a crawl.
Of course, thanks to the reduction of the amount of reachable topologies by our algorithm, the effective size will probably be much lower than that. For many circuits, the number of reachable topologies can be reduced by a factor of ten or more, which greatly simplifies

the problem and the amount of lines in the tables. Additional procedures may lead to a reduction of the memory footprint:

1. the $A, B$ matrices are never used during the course of our algorithm. They can be stored separately on the hard drive until the final port on the real-time platform.

2. at any time during our algorithm, we only need to load all topologies whose transistors are in the correct configuration. By doing a partial loading of the data set, the effect of the RAM is limited

3. for the unstable topologies, only the $C_D$ and $D_D$ matrices are needed to draw the graph during the soft switching analysis. All other information of the topology can be discarded

Nevertheless, these tricks might not be enough if the amount of switches rises. The test cases used in section 3.5 were small enough to be studied easily, but circuits with more than 15 to 20 switches have been impossible to study without dedicating days to the analysis. A more permanent solution is to split the circuit into smaller parts which are studied independently. This concept, known as *circuit partitioning*, is the subject of chapter 4.

### 3.6.2 Limitations on the size of the output

The amount of data to store also has an impact on the size of the output and on the performance of the real-time platform. Indeed we have to remember that each state variable, output variable and condition must be evaluated using a dot product. Depending on the implementation, each partial product requires a multiplication and an addition. Multipliers and multipliers-adders are limited resources on FPGAs (fortunately the amount of such units is quickly rising at each new generation), and too many simultaneous dot products will saturate the chip completely. The tables seem relatively straightforward in their implementation, as they are basically large multiplexers. However, due to their size, they may consume a large amount of logic slices in the FPGA, even if these tables have been reduced to a minimum.

As an added problem, using more resources reduces the maximum speed, since it becomes more difficult for the HDL synthesizer of find the optimal placement and routing inside the chip, which lowers maximum clock frequency.

To reduce the impact, we will need to carefully select the implementation of the functional modules to avoid using too many logical units. But, no matter which tricks are used, we will always reach a limit for a given number of switches. Let us also recall that the traditional implementation of real-time solvers based upon the modified nodal analysis would require more resources: while the platform does not need to store the switching tables, the amount of dot products is typically much larger than the total amount of dot products required by the linear solver and by the two switching modules. Moreover, the dot products linked to conditions obtained using our method generally contain a large amount of $(-1, 0, 1)$ coefficients which may be removed or simplified during the translation to the HDL code, so that no hardware multiplier will be recruited in the FPGA for this product. Since the amount of mathematical units inside a FPGA is typically much lower than the number of logical slices, we expect to be able to implement larger converters on our platform. These considerations are studied with more precision in sections 5.3.2, 5.3.3 and 5.3.4.

## 3.7 Possible improvements and future works

The parsing algorithm works under the assumption that the user may want to test the circuit in absolutely all of its functioning mode. This includes considering cases that should never happen in practice if the circuit is correctly controlled. More often than not, these corner cases lead to complex interaction between the topologies and force us to compute many conditions to correctly track the state of the circuit. Many of those cases are normally part of the Failure Mode Effects and Critical Analysis, which are generally handled by off-line simulation, while our real-time platform should only emulate the failure modes that have to be handled by the controller under design.

Hence, we could want to explicitly remove some of the cases by entering user's constraints. For instance, we can limit an input voltage to positive values to avoid analyzing a large part of the graph. Another reason to explicitly limit the range of some variables is that this range cannot be defined automatically when the order of the system is higher than two. This concept, known as reachability, was introduced in section 3.3.2. Because of this, some transitions that might never be true could be found in the graphs or in the conditions of the forced switching module even after all the simplifications described in the previous sections.

However, as expert users, we are generally able to tell the range of all variables through intuition or after extensive testing. Passing these restriction to the offline tool could greatly accelerate the analysis and reduce the size of the graphs, or mark more topologies as unreachable. As it stands, the most pertinent way to introduce these constraints is in the form of additional inequalities using the same formula as the constraints, i.e.

$$\begin{aligned} y_1(t) &= Cx(t) + Du(t) \quad \leq 0 \\ y_2(t) &= Cx(t) + Du(t) \quad > 0 \end{aligned} \tag{3.91}$$

These constraints can then be integrated in the automated tools. For example, let us write user's constraints of the form

$$\begin{aligned} y_1 &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \geq 0 \\ y_2 &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \geq 0 \end{aligned} \tag{3.92}$$

and let us suppose that we are studying a topology whose definition domain contains the following condition:

$$v_D = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq 0 \tag{3.93}$$

By virtue of the additional constraints, this signal will never become negative. Hence, the transition corresponding to this diode can be removed from the transition graph. Proper integration requires the use of the simplex algorithm to test if the condition can be written as a linear combination of the user's constraints.

## 3.8 Conclusions

In this chapter, we have developed a set of tools able to automatically study a power converter and extract all pertinent information before porting the design to the selected real-time platform. First, we have described in section 3.2 an iterative method based on Dijkstra algorithm to draw the hybrid automata associated with the converter.

Then, in section 3.3, we have introduced a new tool able to parse the automata in order to extract the minimal set of signals that must be computed to properly control the diodes in reaction to changes in the value of the state variables. On the real-time platform, these signals are evaluated by the natural switching module (see section 2.6.3). The reduction of the graphs is based on the study of the definition domain of the topologies connected by the graph, including a systematic research of the links between the transitions. We have developed multiple methods to find transitions that automatically validate or invalidate each other.

This analysis allows us to find the correct state of the diodes in a single pass, even if multiple diodes must be switched in sequence. Hence, the overhead introduced by the switching module and the associated computing power are kept to a minimum, and the minimization of the signals reduces the required processing power.

A similar analysis has been made in section 3.4 for the signals needed to control the diodes after any change in the configuration of the active switches. Again, we introduced methods to systematically predict how the power converter will react to the changes, and extract the minimum number of signals which have to be computed in order to pinpoint the correct transition. For this purpose, we make an intensive use of the supposed continuity of the states during any instantaneous change in the configuration of the switches.

The various test cases of section 3.5 allowed us to illustrate how these algorithms work. We carried out simulations to better highlight the performance as well as the limitations of our current method. We have shown that the results were very accurate is most cases, with a relative error, defined as the difference between the simulated and expected signals divided by the average value of the signal, of less that 1%. Only the simulation of Figure 3.26 presents a larger error, due to a time step which is too large when compared to the internal dynamics of the circuit.

The two automated exploration methods provide a set of tables linking the current topology to the signals that must be evaluated, as well as another set of tables connecting the result of the evaluation to the correct configuration of the switches. This information will be used by the real-time platform, as will be described in chapter 5.

The crucial problem associated with this methodology lies in the combinatorial explosion linked to the size of the data set. We have shown that, as the circuit grows, the tables become exponentially larger, which leads to very long compilation time and to memory issues if the circuit contains more than fifteen to twenty switches. Since these circuits are widely used, propose a solution based upon circuit partitioning in the next chapter.

# Chapter 4

# Modeling larger circuits by circuit partitioning

## 4.1 Introduction

The suite of algorithms introduced in chapter 3 performs well for small- and medium-sized circuits (up to twenty switches), but tends to be inefficient or even unusable for larger converters. Indeed, the exponential growth of the size of the graphs and tables leads to a high memory usage, while the amount of combinations results in a very long off line processing. As an example, it takes more that 10 hours on a modern computer based on an Intel i7 processor and 8GB of RAM to completely parse a circuit composed of twenty switches and four state variables. While this limitation is acceptable for a large range of circuits, it prevents us from studying complex circuit such as the three-level, three phase, neutral-point clamped (NPC) inverter of Figure 4.1 composed of 30 switches. Such circuits are widely used, hence it is necessary to provide a solution to this problem.

Since the size of the tables is typically proportional to $2^{n_s}$, where $n_s$ is the total amount of switches in the circuit, a solution is to split the circuit into multiple sub-circuits, each containing a reduced amount of switching elements. For example, this separation could

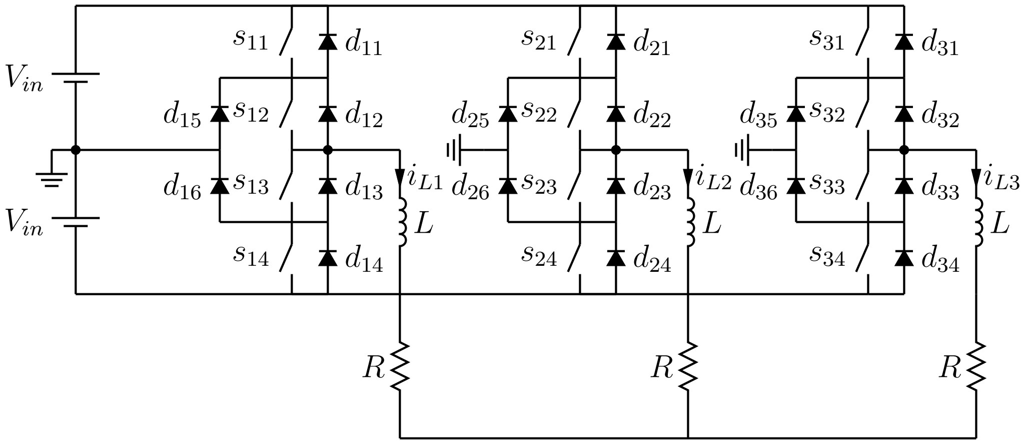


**Figure 4.1: 3-Level 3-Phase Neutral Point Clamped (NPC) Converter**

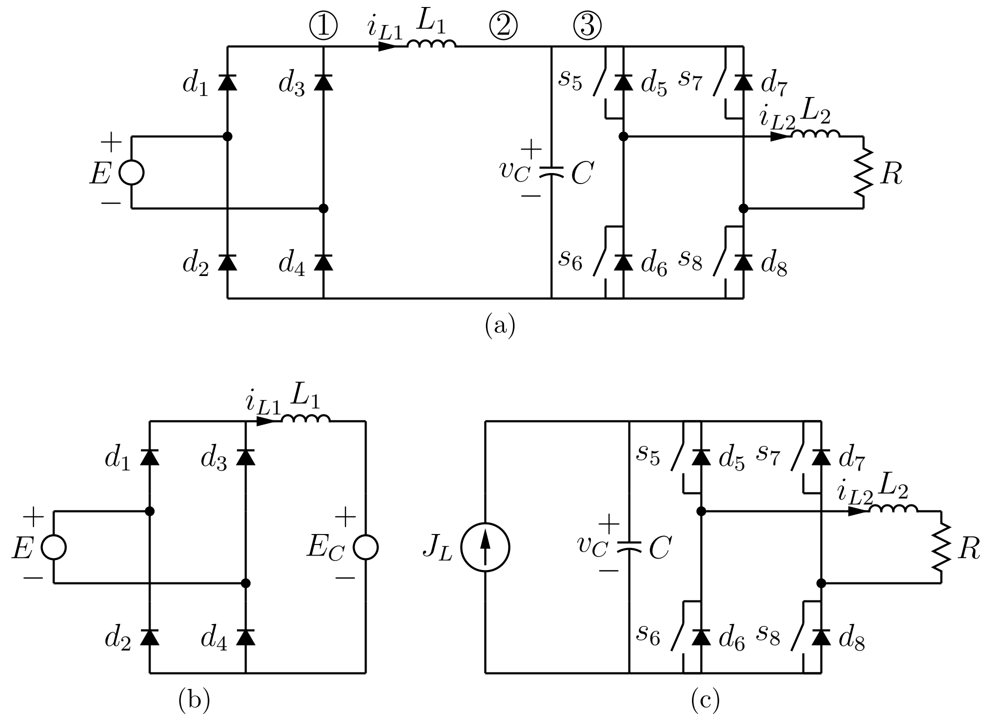


**Figure 4.2: Two-stage converter composed of a rectifier and an inverter (a) Complete circuit (b)(c) Partitioned circuit**

transform the 3-phase NPC inverter into three manageable circuits of 10 switches. Each sub-circuit is then studied separately, and additional steps are taken to link the sub-circuits.
This process, known as *circuit partitioning*, enables the study of a much larger range of circuits. Unfortunately we can foresee a degradation in the performances of the real-time implementation. Indeed, since each sub-circuit is studied independently, the analysis tools no longer have a global view of the system which reduces the range of optimizations, such as the removal of equivalent conditions across two or more parts of the converter.
To properly integrate circuit partitioning in our analysis system, we have to answer the following questions:

- Where should we split the circuit? Are there any limitations?
- How can we recombine the different parts?
- What is the impact on the accuracy of the simulation?

These questions are intrinsically linked, and the answer could vary depending on the structure of the studied circuit. We shall start with a simple example to illustrate the basic principles.

**Example 15.** Let us take a look at the two-stage circuit of Figure 4.2, and let us try to split it into two sub-circuits. It seems natural to perform the partitioning between the rectifier and the inverter, since these two sub-circuits are controlled by different signals. Furthermore, this would allow us to easily mix-and-match the two modules: we could for example easily add a second inverter in parallel with the first one by slightly modifying the connections.

Still, this leaves us with multiple choices: we could split the circuit in any of three points (marked 1 to 3 on the schematics), or even use more than two sub-circuits. For the sake of simplicity, and because this is again the most natural choice, we will first split the circuit at point ② between the inductor and the capacitor (appropriately named *decoupling capacitor*), resulting in the two sub-circuits of Figures 4.2b and 4.2c. Each of these modules can be analyzed separately, and will get its own set of tables for the natural and forced switching modules, as well as its own state equations. However, one question remains: how can the behavior of each sub-circuit have an effect on the other one? To answer this, we first write the continuous-time state equations for the complete circuit:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_{L1}(t) \\ v_C(t) \\ i_{L2}(t) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} i_{L1}(t) \\ v_C(t) \\ i_{L2}(t) \end{bmatrix} + \begin{bmatrix} b_1 \\ 0 \\ 0 \end{bmatrix} E(t) \tag{4.1}$$

where the $a, b$ coefficients depend on the topology. From these equations, it appears that $v_c$ is the only state variable influenced by both converters. After introducing two auxilary controlled sources $J_L(t){=}i_{L1}(t)$ and $E_C(t){=}v_C(t)$, we may equivalently write these equations as

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_{L1}(t) \\ v_C(t) \\ i_{L2}(t) \end{bmatrix} &= \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} i_{L1}(t) \\ v_C(t) \\ i_{L2}(t) \end{bmatrix} + \begin{bmatrix} b_1 & 0 & a_{12} \\ 0 & a_{21} & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} E(t) \\ J_L(t) \\ E_C(t) \end{bmatrix} \\ \begin{bmatrix} J_L(t) \\ E_C(t) \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i_{L1}(t) \\ v_C(t) \\ i_{L2}(t) \end{bmatrix} \end{aligned} \tag{4.2}$$

These sources allow us to virtually connect the two parts of the circuits, as shown in Figures 4.2b and 4.2c: each sub-circuit evolves according to its own state equations

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_{L1} \end{bmatrix} &= \begin{bmatrix} a_{11} \end{bmatrix} \begin{bmatrix} i_{L1}(t) \end{bmatrix} + \begin{bmatrix} b_1 & a_{12} \end{bmatrix} \begin{bmatrix} E(t) \\ E_C(t) \end{bmatrix} \\ \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} v_C(t) \\ i_{L2}(t) \end{bmatrix} &= \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} v_C(t) \\ i_{L2}(t) \end{bmatrix} + \begin{bmatrix} a_{21} \\ 0 \end{bmatrix} \begin{bmatrix} J_L(t) \end{bmatrix} \end{aligned} \tag{4.3}$$

while the communication between the sub-circuits is made through the update of the new controlled sources:

$$\begin{aligned} \begin{bmatrix} J_L(t) \end{bmatrix} &= \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} i_{L1}(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} E(t) \\ E_C(t) \end{bmatrix} \\ \begin{bmatrix} E_C(t) \end{bmatrix} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} v_C(t) \\ i_{L2}(t) \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} J_L(t) \end{bmatrix} \end{aligned} \tag{4.4}$$

The partitioning has absolutely no impact on the waveforms as long as we perform a continuous-time study. Of course, we now have to adapt the discrete-time simulation to take this into account. As shown in figure 4.3, this is done by inserting an additional module named ”sources update” before the state solver. Because the two partitions are no longer linked directly, any change on a state variables will only be perceived by the

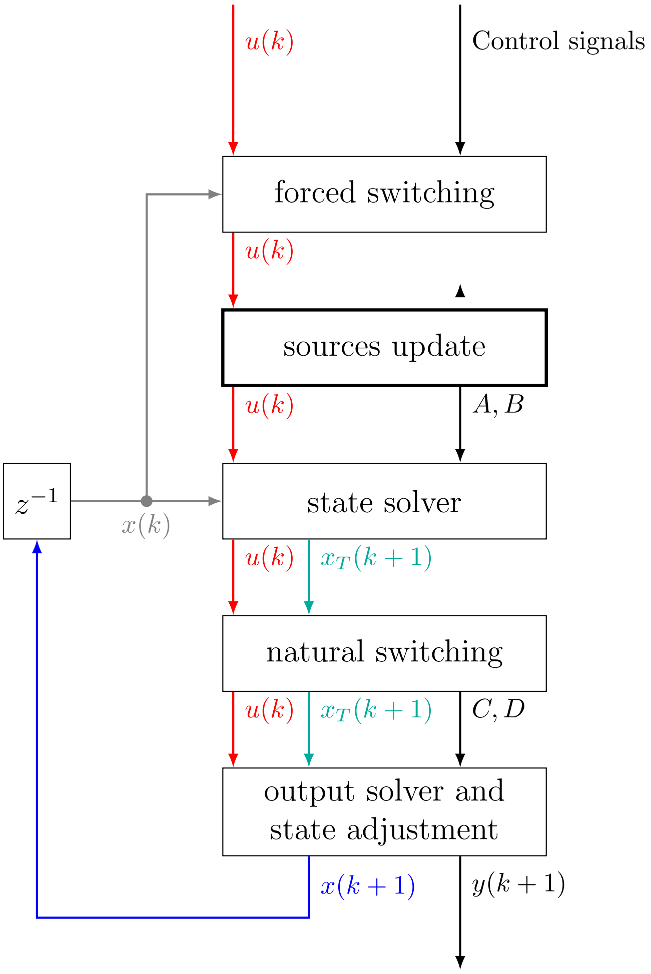


**Figure 4.3: Adding the update of the controlled sources to the interleaved discrete solver**

other sub-circuit a single time-step later, which leads to an additional error whose value will depend on the chosen solver.
To illustrate this, we will apply the partitioning to the following linear system:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}x_1(t)\\x_2(t)\end{bmatrix} = \begin{bmatrix}a_{11} & a_{12}\\a_{21} & a_{22}\end{bmatrix}\begin{bmatrix}x_1(t)\\x_2(t)\end{bmatrix} + \begin{bmatrix}b_1\\0\end{bmatrix}u(t) = A_1\begin{bmatrix}x_1(t)\\x_2(t)\end{bmatrix} + B_1u(t) \tag{4.5}$$

We split the system between the two state variables by introducing the auxiliary variables $y_1(t) = x_1(t)$ and $y_2(t) = x_2(t)$ :

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}x_1(t)\\x_2(t)\end{bmatrix} = \begin{bmatrix}a_{11} & 0\\0 & a_{22}\end{bmatrix}\begin{bmatrix}x_1(t)\\x_2(t)\end{bmatrix} + \begin{bmatrix}b_1 & 0 & a_{12}\\0 & a_{21} & 0\end{bmatrix}\begin{bmatrix}u(t)\\y_1(t)\\y_2(t)\end{bmatrix} = A_2\begin{bmatrix}x_1(t)\\x_2(t)\end{bmatrix} + B_2u(t) \tag{4.6}$$

Some elements that were in the $A$ matrix in the first system are now in the $B$ matrix. We will now apply the backward Euler approximation (BEA, see section 2.2.3) to the two systems and compare the results. As a reminder, this method corresponds to the following approximation :

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = Ax(t) + Bu(t) \Rightarrow x(k) \approx (I - TA)^{-1}x(k-1) + (I - TA)^{-1}TBu(k) \tag{4.7}$$

Applying the BEA to the unique system (4.5) yields

$$\begin{bmatrix}x_1(k)\\x_2(k)\end{bmatrix} = \frac{1}{\Delta}\begin{bmatrix}1 - Ta_{22} & Ta_{12}\\Ta_{21} & 1 - Ta_{11}\end{bmatrix}\begin{bmatrix}x_1(k-1)\\x_2(k-1)\end{bmatrix} + \frac{1}{\Delta}\begin{bmatrix}-T(Ta_{22} - 1)b_1\\T^2a_{21}b_1\end{bmatrix}u(k) \tag{4.8}$$

where $\Delta = -T * a_{11} - T * a_{22} + T^2 * a_{11} * a_{22} - T^2 * a_{12} * a_{21} + 1$ is the determinant of $I + TA_1$ Applying the BEA to the partitioned system yields:

$$\begin{bmatrix}x'_1(k)\\x'_2(k)\end{bmatrix} = \begin{bmatrix}\frac{-1}{Ta_{11}-1} & 0\\0 & \frac{-1}{Ta_{22}-1}\end{bmatrix}\begin{bmatrix}x_1(k-1)\\x_2(k-1)\end{bmatrix} + \frac{1}{\Delta}\begin{bmatrix}\frac{-Tb_1}{Ta_{11}-1} & 0 & \frac{-Ta_{12}}{Ta_{11}-1}\\0 & \frac{-Ta_{21}}{Ta_{22}-1} & 0\end{bmatrix}\begin{bmatrix}u(k)\\y_1(k)\\y_2(k)\end{bmatrix} \tag{4.9}$$

Replacing the auxiliary variables by their value:

$$\begin{bmatrix}x'_1(k)\\x'_2(k)\end{bmatrix} = \begin{bmatrix}\frac{-1}{Ta_{11}-1} & \frac{-Ta_{12}}{Ta_{11}-1}\\\frac{-Ta_{21}}{Ta_{22}-1} & \frac{-1}{Ta_{22}-1}\end{bmatrix}\begin{bmatrix}x_1(k-1)\\x_2(k-1)\end{bmatrix} + \frac{1}{\Delta}\begin{bmatrix}\frac{-Tb_1}{Ta_{11}-1}\\0\end{bmatrix}\begin{bmatrix}u(k)\end{bmatrix} \tag{4.10}$$

The local error (ie. the error due to a single approximation step) is given by

$$\begin{bmatrix}x_1(k)\\x_2(k)\end{bmatrix} - \begin{bmatrix}x'_1(k)\\x'_2(k)\end{bmatrix} \tag{4.11}$$

which converges to zero as the time step $T$ gets smaller (i.e., all elements of the matrices are $\mathcal{O}(T^n)$ , with $n \geq 1$).
To assess if the simulation still provides accurate results after the partitioning, we will now make a series of comparative tests on the circuit of Figure 4.2a. We assume the following parameters:

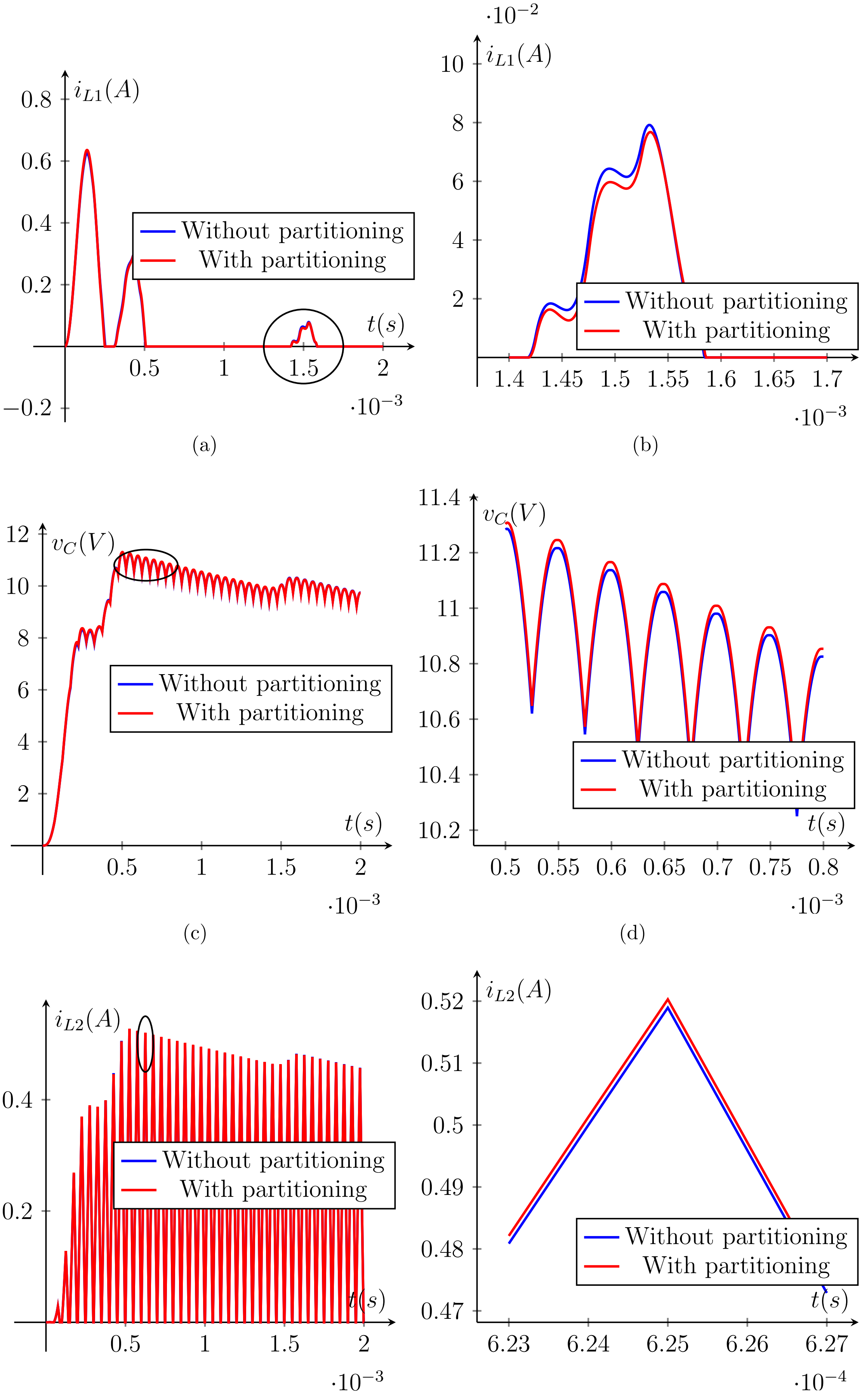
- $E$ is a 10 volts, 500Hz sinusoidal source

- $L_1 = 200\mu H$
- $C = 10\mu F$
- $L_2 = 500\mu F$
- $R = 2\Omega$

A 500Hz source is used instead of the more traditional 50Hz to reduce the length of the simulation and allowing us to better zoom on the combined effects of the input source and of the switching process. The converter is simulated with the interleaved solver presented in Figure 4.3, and the circuits are discretized using the trapezoidal approximation.
We first simulate the circuit with a time-step $T$ of $1\mu s$. Transistors $T_1, T_4$ are controlled by a $20kHz$, 50% duty-cycle PWM while $T_2, T_3$ are always off. Figures 4.4a to 4.4f show the results of the simulation of the full circuit, as well as of the partitioned version. These results show that the signals obtained are very close to each other. From Figure 4.4b, we see that $i_{L1}$ is the only signal showing a significant difference, which is however limited to about 1% of its maximum value. Changing the PWM frequency to 100kHz results in the plots of Figure 4.5. The relative error (again defined as the ratio of the absolute error divided by the maximum value of the emulated signal) on the DC bus voltage is greater than before, but is still kept below 1%. Again, the output current is virtually the same in both cases. The error increases again when the frequency of the sinusoidal voltage source is changed to 5kHz, as shown in Figure 4.6. From this example, we can infer the first partitioning rule: *we should always split the circuit on a point where the signal evolve slowly when compared to the sampling period.*

## 4.2 Converters operating in discontinuous conduction mode

Discontinuous-conduction mode (DCM) is a behavior that appears in some circuits containing an inductor placed in series with one or more diode. When the current flowing through the inductor and the diode becomes negative, the diode turns off, forcing the current to zero (or, more generally, to a value controlled by an algebraic equation instead of a differential equation). When the circuit is not partitioned, the value of the current is forced by the output solver of the simulation algorithm (see section 2.6).
Difficulties arise when the circuit is partitioned, as we could have to force a current that does not belong to the current partition to zero. In some cases, this does not lead to any major change. For example, the inductor $L_1$ of the circuit of Figure 4.2a can enter DCM, but it did not prevent us from partitioning at its right-hand node. This is explained by noting that $L_1$ is included within the rectifier stage, which also contains the diodes that could force the inductor current to zero.
Let us now study what happens when the inductor is separated from the diodes that could lead to DCM: let us split the inverter of Figure 4.7a. We could take the left switching branch combined with the central branch as the first partition (in black on the circuit), while the second partition (drawn in gray) only contains the other branch.
To virtually connect the partitions, we will first try to use the same auxiliary current



(e)

(f)

**Figure 4.4: Simulation of the circuit of Figure 4.2a, for the partitioned and non-partitioned versions, $T = 1\mu s, T_{PWM} = 50\mu s$. The plots on the right are zooms of the left plots on the circled region**

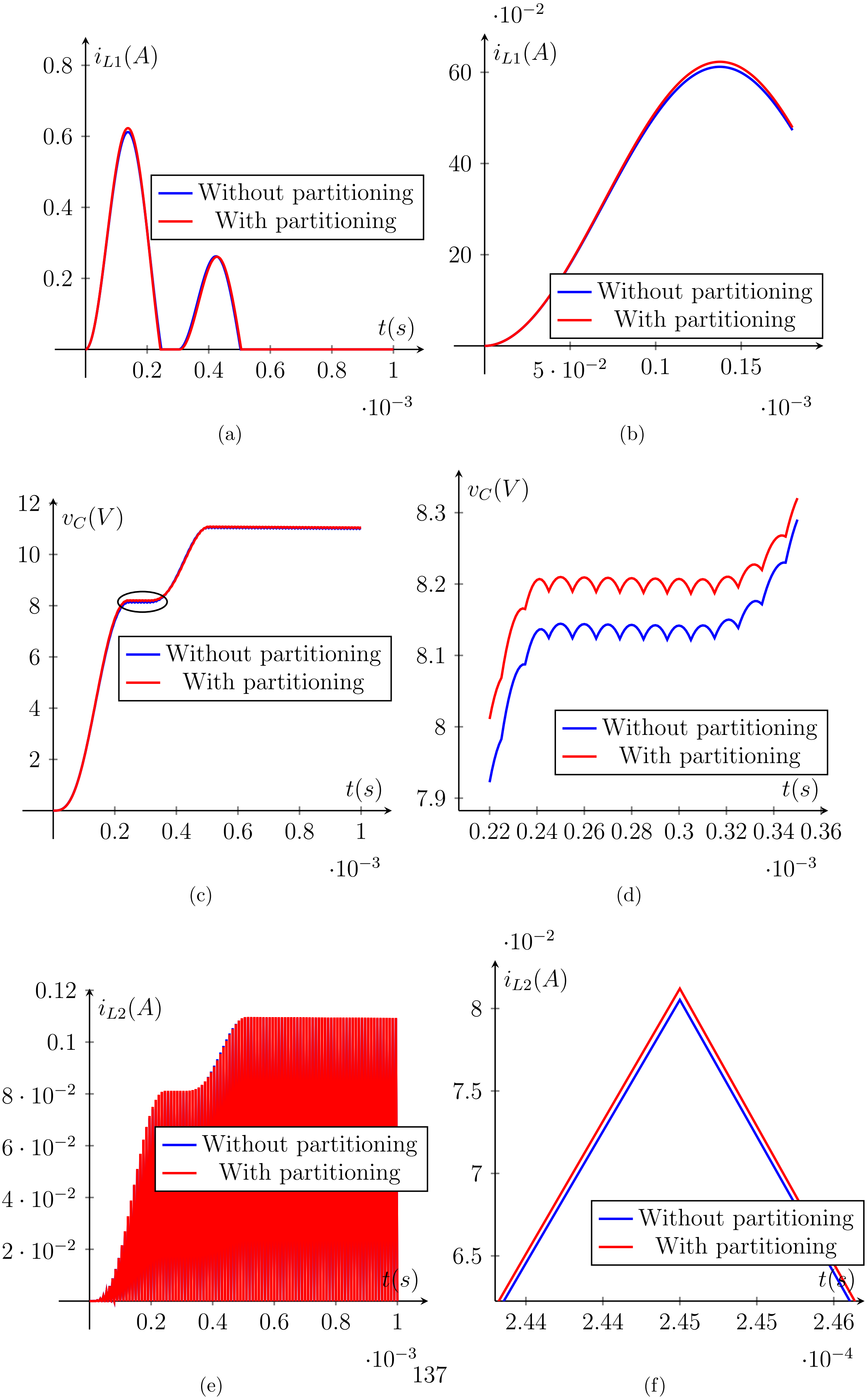
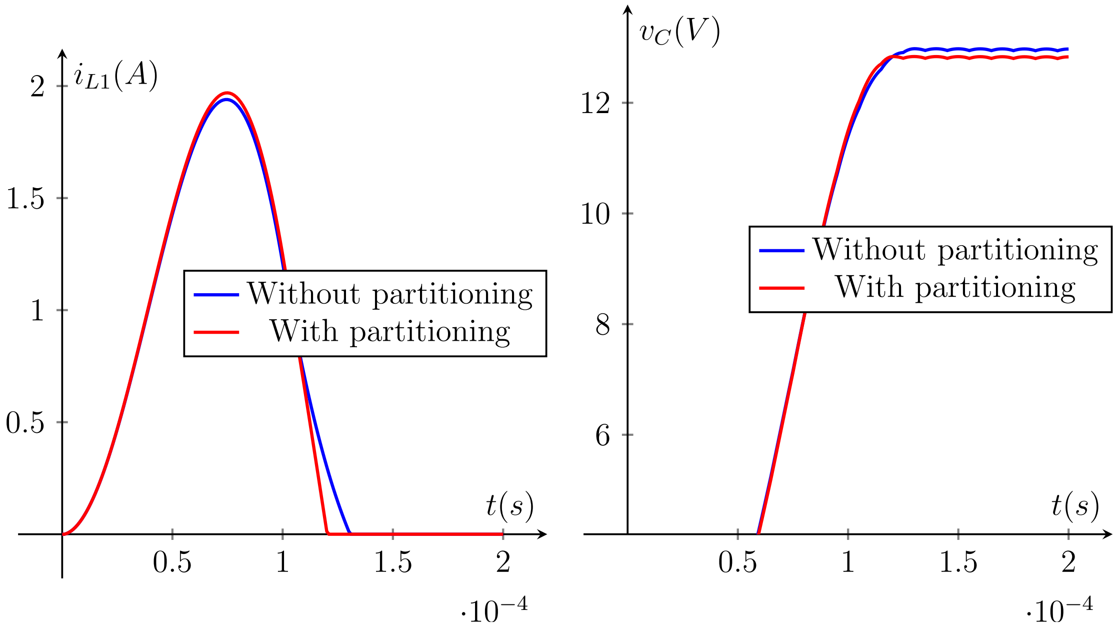


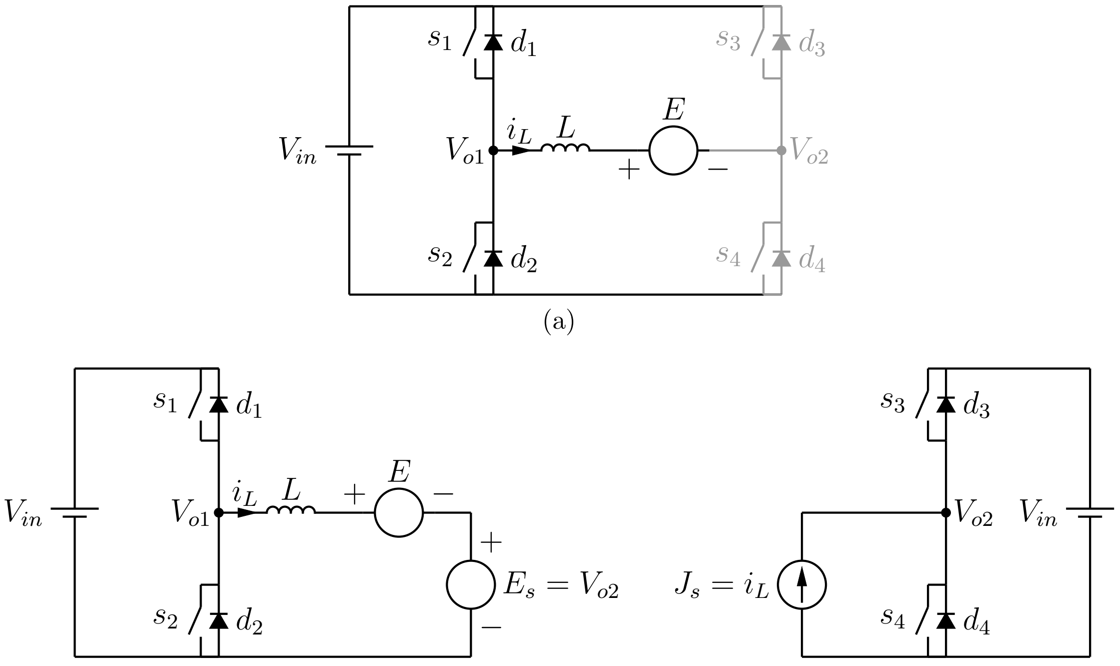
**Figure 4.5: Simulation of the circuit of Figure 4.2a, for the partitioned and non-partitioned versions, $T = 1\mu s, T_{PWM} = 10\mu s$. The plots on the right side are zoomed versions of the left plots**



(a)

(b)

**Figure 4.6: Simulation of the circuit of Figure 4.2a, for the partitioned and non-partitioned versions, $T = 1\mu s, T_{PWM} = 10\mu s$ and input frequency increased to 5kHz**



(a)

(b)

(c)

**Figure 4.7: Partitioning of a full bridge converter (a) Complete circuit, with the two partitions in black and gray (b)(c) Partitioned circuit, with the controlled sources used to virtually reconnect the two sub-circuits. The law controlling the sources are $E_s(k) = V_{o2}(k-1)$ and $J_s(k) = i_L(k-1)$**

.

and voltage sources as in the previous example, as shown in Figures 4.7b and 4.7c. Unfortunately, this method only works here if both branches have a switch in the on-state, which allows the currents $i_l$ and thus $J_s = i_L$ to flow freely. When DCM occurs and all switches are off, the update of the auxiliary sources is impossible and the link between the two sub-circuits is broken, as it will be explained in the next section.

To illustrate this problem, let us now assume that the current flows through $d_1$ and $d_4$ (ie. $i_L$ is negative) and that $V_{in} > E > 0$. The inductor is discharging, its current will cross zero and reach a small positive value at some time step; in the left sub-circuit $d_1$ will be switched off and the inductor current forced to 0. At the next time step, $J_s$ will be updated to 0, which will switch off $d_4$. If we look at the switching graph of the complete inverter in Figure 4.8, we see that the state has evolved from T1 to T0; we should stay in T0 until the transistors are switched on or until $E > |V_{in}|$. Let us suppose that E evolves so that this last condition becomes true. In the full circuit, $d_1$ and $d_4$ will become forward-biased and be switched on again; the state changes back to T1. In the partitioned circuit, since $J_s$=0, $d_3$ and $d_4$ are blocked in the off state and $V_{o2}$ is undefined because the node is floating. In the left sub-circuit, the condition to reach T1 $(E + E_s) > V_{in} \equiv (E + V_{o2}) > V_{in}$ is impossible to evaluate and we cannot get out of this topology.

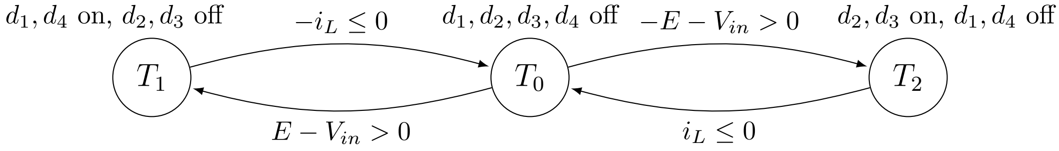


**Figure 4.8: Topology graph for the full bridge circuit of Figure 4.7a when all active switches are off**

To avoid this problem, a first solution is to add polarizing resistors $R$, as shown in Figures 4.9a and 4.9b, which always provide a path for the current and ensures that all voltages are always defined. To avoid modifying the general behavior of the circuit, we select a value of $100k\Omega$ for these resistances so that their current adopts a value close to the leakage current of the diodes.

We will now analyze the effect of these resistances on the switching behavior and on

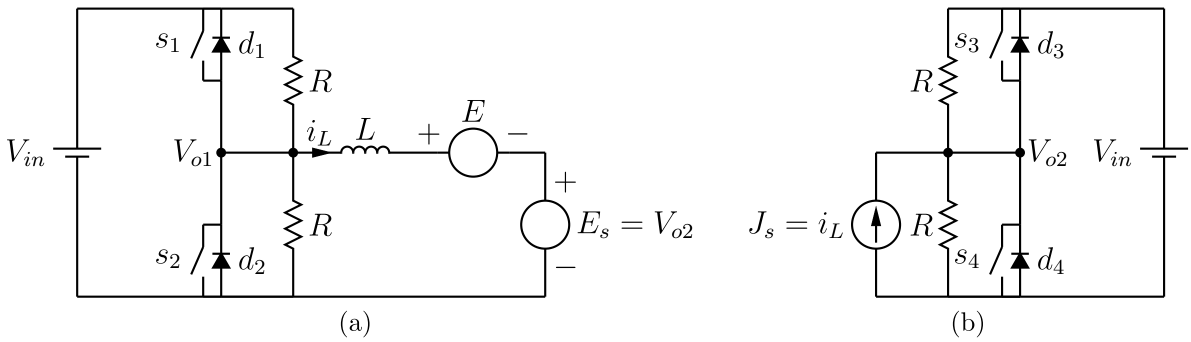


**Figure 4.9: Adding polarizing resistors allows us to compute $V_{o1}$ and $V_{o2}$, which are needed to control the diodes**

the waveforms. Let us assume that $i_L \leq 0$ flows through $d_1$ and $d_4$; the current evolves following the law

$$i_L(k) = i_L(k-1) + \frac{T}{L}(V_{in}(k) - E(k) - E_s(k)) \tag{4.12}$$

where $E_s(k) = V_{o2}(k) = 0V$ since $d_4$ was in the on-state during the previous step. This law is the same as the one used to simulate the whole circuit at once. The current in the diodes is given by

$$\begin{aligned} i_{d1}(k) &= -\frac{V_{in}}{R} - i_L(k) \\ i_{d4}(k) &= -\frac{V_{in}}{R} - J_s(k) = -\frac{V_{in}}{R} - i_L(k-1) \end{aligned} \tag{4.13}$$

These equations already show the effects of the partitioning: first, the current across the diodes (and hence the input current of the converter) is offset by $-V_{in}/R$. This effect can degrade the accuracy of the simulation if this offset is not significantly lower than the real value of the current. Secondly, since the new value of the current is only seen on the *next* step by the second branch, $d_4$ will not be switched off simultaneously with $d_1$. As explained in the previous section, this delay introduces errors if the dynamics are not much slower than the time-step.

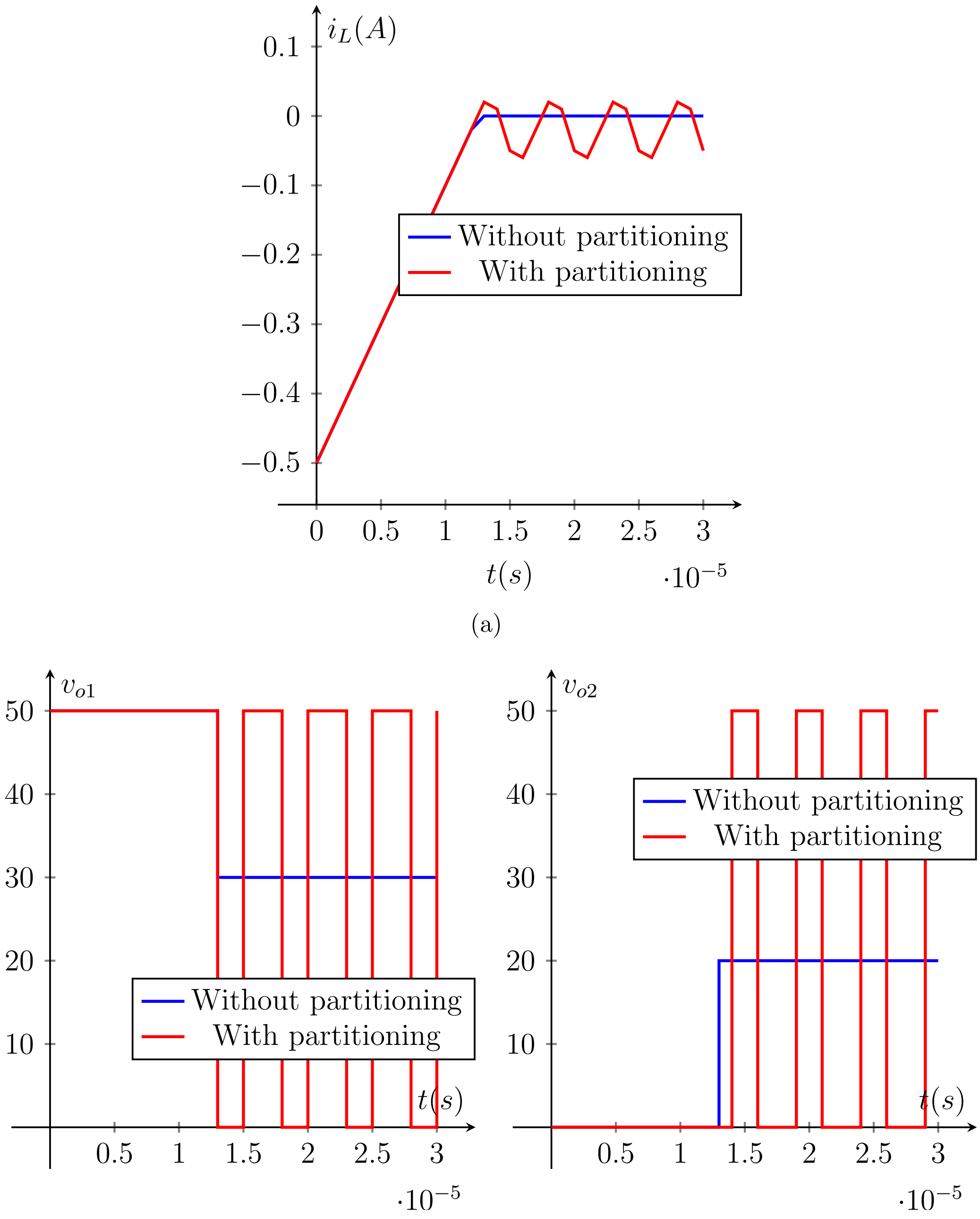
The graphs presented in Figure 4.10 show the simulation of the partitioned circuit, along with the results of the simulation of the whole circuit at once. To better understand these results, we also need the switching graphs of the two partitions : these are drawn in Figure 4.11. With this in mind, the oscillations are easily explained. When the current $i_L$ goes from negative to positive, the following events happen:

1. the natural switching module first switches $d_1$ off
2. since $-\frac{R}{2}i_L$ is positive, $d_2$ is switched on
3. on the next clock cycle, $J_s$ is updated to the positive value, which blocks $d_4$ and turns $d_3$ on
4. since a negative voltage is forced across the inductor, the current decreases
5. the current becomes negative and the process is repeated

This oscillation between the different topologies has an effect on the voltage at the output of the branches: instead of having a constant value of $\frac{V_{in}}{2} \pm \frac{E}{2}$, the voltage goes back and forth between $0V$ and $V_{in}$. The ripples present in the inductor current degrade the accuracy of the simulation, and the effect is more present when the circuit is driven by a high-speed PWM signal. Assuming $E = 0V$, the amplitude of this noise is easily obtained by discretizing the inductor on a single sampling period :

$$L\frac{\mathrm{d}i_L}{\mathrm{d}t} = V_{in} - 0V \Rightarrow \Delta i_{L,noise} = V_{in}\frac{T}{L} \tag{4.14}$$

We now assume that $s_1$ and $s_4$ are driven by a PWM signal characterized by a period $T_{PWM}$ and a duty-cycle $\delta$ while $s_2$ and $s_4$ are left open. The circuit should go through the following modes:



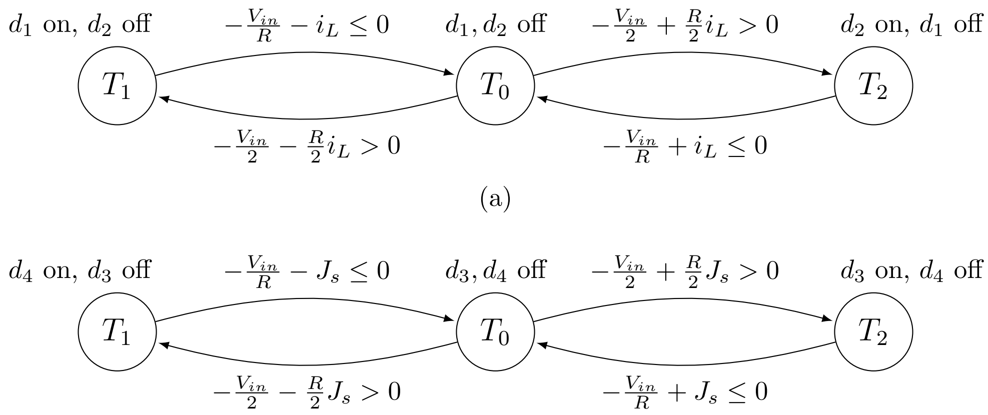
(a)

(b)

(c)

**Figure 4.10: Simulation of the circuit of Figure 4.7, when all active switches are kept in the off-state and the inductor current $i_L$ starts with a negative value. The two voltages go back and forth instead of converging towards $V_{o1} = \frac{V_{in}+E}{2}$ and $V_{o2} = \frac{V_{in}-E}{2}$**

.


$d_1$ on, $d_2$ off
$-\frac{V_{in}}{R} - i_L \leq 0$
$d_1, d_2$ off
$-\frac{V_{in}}{2} + \frac{R}{2} i_L > 0$
$d_2$ on, $d_1$ off
$T_1$
$T_0$
$T_2$
$-\frac{V_{in}}{2} - \frac{R}{2} i_L > 0$
$-\frac{V_{in}}{R} + i_L \leq 0$
$d_4$ on, $d_3$ off
$-\frac{V_{in}}{R} - J_s \leq 0$
$d_3, d_4$ off
$-\frac{V_{in}}{2} + \frac{R}{2} J_s > 0$
$d_3$ on, $d_4$ off
$T_1$
$T_0$
$T_2$
$-\frac{V_{in}}{2} - \frac{R}{2} J_s > 0$
$-\frac{V_{in}}{R} + J_s \leq 0$

(a)

(b)

**Figure 4.11: Topology graphs for the two partitions (a) Left branch (b) Right branch.**

1. when the controlled switches are closed, all diodes are open and the inductor current increases.

2. when the controlled switches are turned off, $d_2$ and $d_3$ turn on to keep the current flowing. The current decreases, since the voltage on the inductor is now negative.

3. when the current becomes negative, all diodes should turn off until $s_1$ and $s_4$ are closed again.

As explained previously, the circuit will not stay in the third topology, but instead oscillate between the topology where $(d_2, d_3)$ are closed and the topology where $(d_1, d_4)$ are closed. The ripple on the current due to the PWM control is equal to

$$\Delta i_{L,PWM} = V_{in} \frac{\delta T_{PWM}}{L} \tag{4.15}$$

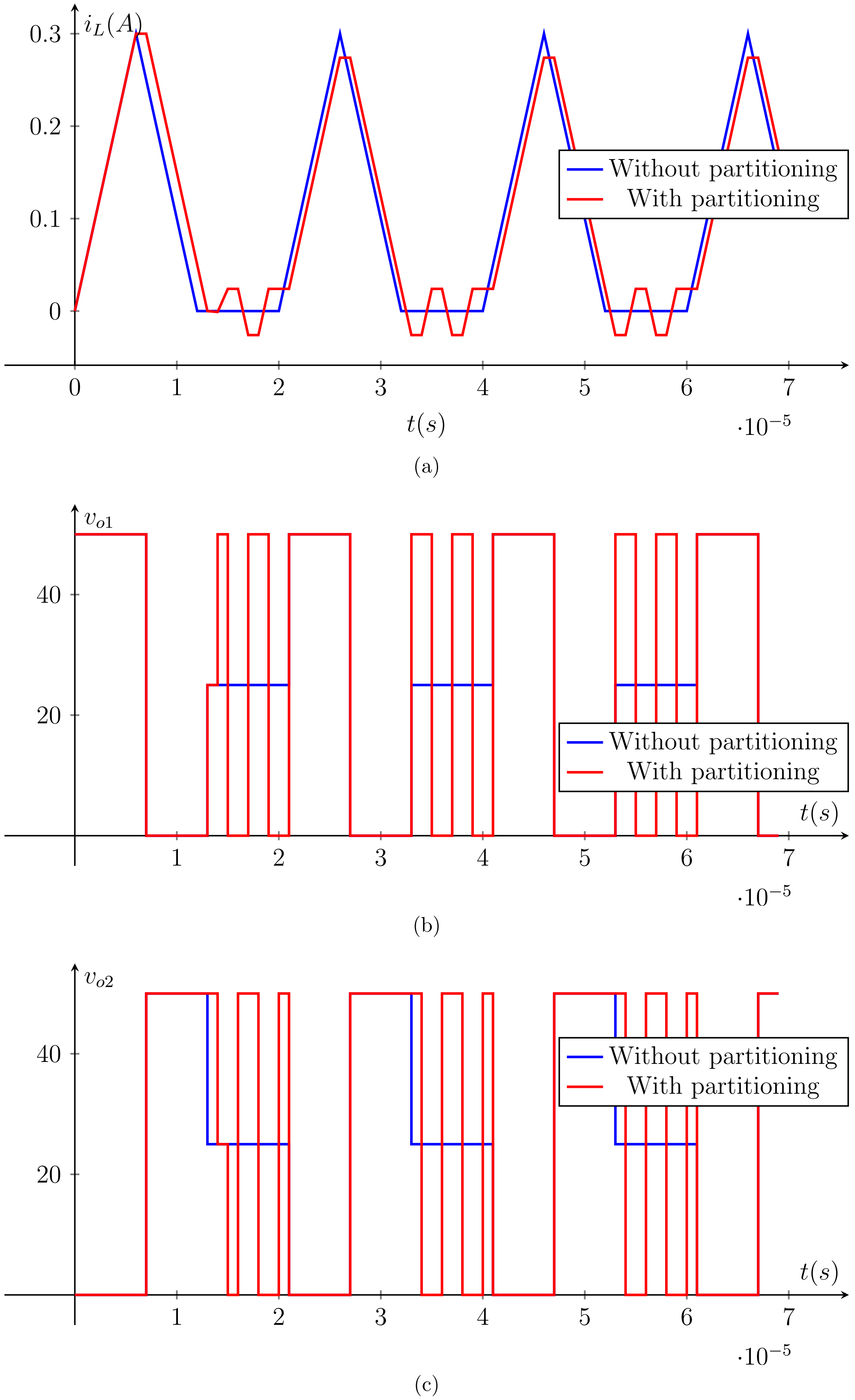
And we can define the signal-to-noise ratio (SNR) of the simulation using the formula

$$SNR = \frac{\Delta i_{L,PWM}}{\Delta i_{L,noise}} = \frac{\delta T_{PWM}}{T} \tag{4.16}$$

Assuming $T = 1\mu s$, $T_{PWM} = 20\mu s$ and $\delta = 0.3$, we only have a $6 : 1$ ratio between the ripples due to the PWM and those due to the additional commutations in the system, as shown on the graphs of Figure 4.12. Furthermore, the current does not rise as high as it should.
This result is not accurate enough for our needs, and we have to develop another method to keep the errors as low as possible

### 4.2.1 Using topology-dependent sources

Since keeping the same controlled sources for interconnecting all topologies does not yield acceptable results, we need to change this form along with the topology of each partition. To avoid falling back into the combinatorial explosion problem, we impose



(a)

(b)

(c)

**Figure 4.12: Simulation of the circuit of Figure 4.7, for the partitioned and non-partitioned versions, $T = 1\mu s, T_{PWM} = 20\mu s$, 30% duty cycle**

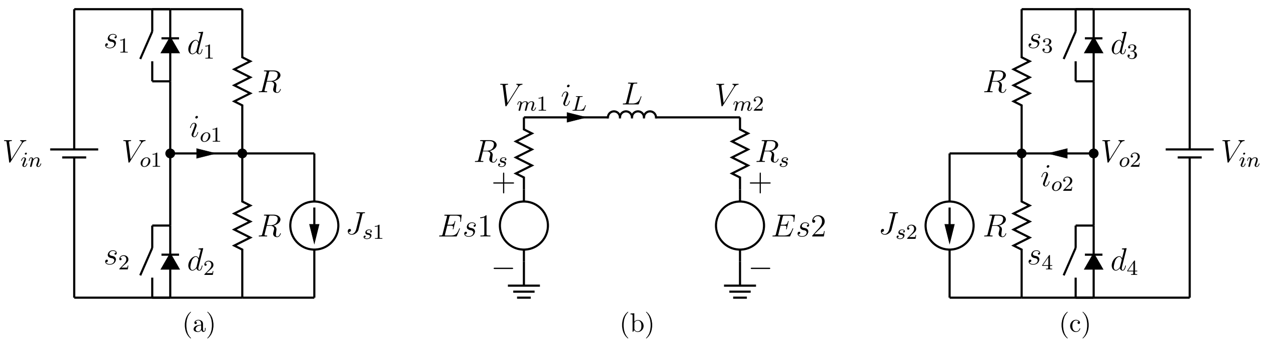


**Figure 4.13: Partitioning the circuit in three with full Thevenin equivalent for the interconnection circuit**

that the equation controlling each of the sources can only depend on the topology taken by a single partition. Furthermore, we want to be able to implement basic modules, such as a single branch of a converter, independently of the rest of the circuit. These modules should then be connected using an *interconnection circuit* composed exclusively of linear elements that are easily simulated using well-known techniques.

A method based on the study of linear reactive interconnection elements is proposed in [82] and refined in [42,83]. This method detects and discretizes the connection elements using the forward Euler method. Those elements are simulated alongside the circuits, and their effects are transmitted from one partition to another using controlled sources.

However, this method is used in association with the nodal analysis, which does not require to change the equations when the topology is modified. Similar methods are presented in [84] and [85, 86], again for solvers based on the modified nodal analysis. Another method [24] uses a state-space solver for each partition while modeling the connections using the modified nodal analysis. However, this method is used for circuits without forced states, avoiding the problem that arises only when the converter can no longer be considered as a voltage source (or as a current source for current-based converters). In light of these results, it seems evident that we have to develop our own method, compatible with the rest of the work presented in this thesis. Using the previous simulation of the full bridge as a baseline, one of the criteria to avoid unwanted oscillations is to somehow force the inductor current to drop to zero as quickly as possible when one of the branches switches off. For this purpose, we propose to split the circuit into three parts, as drawn in Figure 4.13: each of the branches constitutes its own partition. The interconnection (i.e. the inductor and the voltage source $E$ in our case) constitutes the third partition. Instead of using ideal sources to control the inductor, we now use a full Thevenin equivalent with a controlled source $E_{s1,s2}$ and a resistance $R_s$. This resistance ensures us that the current will drop to zero with a time constant $\tau = \frac{L}{2R_s}$ if the voltage sources are kept constant. We now have to write the control laws for the source to ensure that the circuit converges to the correct behavior, i.e.

- For the switching branches
  - when any of the switches of the left branch is closed (i.e. the branch is active), the current $i_{o1}$ flowing through by this branch must be equal to $i_L$. Similarly,

the current $i_{o2}$ must be equal to $-i_L$ when a switch of the right branch is closed.

- when all switches of a branch are off (i.e. the branch is inactive), the voltages $V_{o1,2}$ can be computed using standard state-space equations.

- For the interconnection network
  - when a branch is active, its output voltage must be present and measurable at the corresponding inductor terminal.
  - when at least one branch is inactive, the current must decay to zero as quickly as possible.
  - when a branch is inactive, the voltage measured at the corresponding inductor terminal $V_{m1,2}$ must be equal to the voltage measured at the same point in the real circuit.

Let us start with the case where both branches are active: the real inductor current is controlled by the law

$$L\frac{\mathrm{d}i_L(t)}{\mathrm{d}t} = V_{o1}(t) - E(t) - V_{o2}(t) \tag{4.17}$$

where $V_{o1}$ and $V_{o2}$ are equal to $0V$ or $V_{in}$ depending on which switch is active. Using the forward Euler approximation (FEA) on this equation leads to

$$i_L(k) = i_L(k-1) + \frac{T}{L}(V_{o1} - E - V_{o2}) \tag{4.18}$$

Meanwhile, the differential equation controlling the interconnection branch is

$$L\frac{\mathrm{d}i_L(t)}{\mathrm{d}t} = E_{s1}(t) - 2R_s i_L(t) - E_{s2}(t) \tag{4.19}$$

which is discretized using the FEA:

$$i_L(k) = i_L(k-1) + \frac{T}{L}(E_{s1}(k) - 2R_s i_L(k-1) - E_{s2}(k)) \tag{4.20}$$

The law defined by (4.18) will be equal to (4.20) if

$$E_{s1}(k) - 2R_s i_L(k-1) - E_{s2}(k) = V_{o1} - E - V_{o2} \tag{4.21}$$

Furthermore, imposing that the voltage $V_{m1,2}$ measured at the inductor terminals is equal to $V_{o1,2}$ leads to

$$\begin{aligned} E_{s1} &= V_{o1} + R_s i_L(k-1) \\ E_{s2} &= V_{o2} - R_s i_L(k-1) \end{aligned} \tag{4.22}$$

Let us now take a look at the controlled current source present in the left converter branch (Figure 4.13a). On the equivalent ideal (without the added resistors) converter, the output current $i_{o1}$ is equal to the inductor current $i_L$, while $i_{o2} = -i_L$. Meanwhile, the same currents in the partitioned branches are equal to

$$\begin{aligned} i_{o1} &= J_{s1} + \frac{2V_{o1} - V_{in}}{R} \\ i_{o2} &= J_{s2} + \frac{2V_{o1} - V_{in}}{R} \end{aligned} \tag{4.23}$$

which allows us to find the law controlling the current source while the branches are active

$$J_{s1}(k) = i_L(k-1) + \frac{V_{in}(k) - 2V_{o1}(k)}{R} J_{s2}(k) \quad = -i_L(k-1) + \frac{V_{in}(k) - 2V_{o1}(k)}{R} \tag{4.24}$$

These results can be used as-is while the branches remain in an active state. When a branch is inactive, the current $i_{o1}$ or $i_{o2}$ must be equal to zero, i.e. the law controlling the current source is changed to

$$J_{s1,2}(k) = \frac{V_{in} - 2V_{o1}}{R} \tag{4.25}$$

Furthermore, $V_{o1,2}$ is not given by the current partition, but must be measured on the interconnection circuit. In this case, we have

$$\begin{aligned} V_{o1}(k) = V_{m1}(k-1) = E_{s1}(k-1) - R_s i_L(k-1) \\ V_{o2}(k) = V_{m2}(k-1) = E_{s2}(k-1) + R_s i_L(k-1) \end{aligned} \tag{4.26}$$

The law controlling the voltage sources $E_{s1}$ and $E_{s2}$ must also be changed. Assuming $i_k$ converges to zero, we have

$$\begin{aligned} V_{m1}(k) = E_{s1}(k) \\ V_{m2}(k) = E_{s2}(k) \end{aligned} \tag{4.27}$$

And, since this voltage must be equal to the voltage measured at the output of the branches, we have

$$\begin{aligned} E_{s1}(k) = V_{o1}(k-1) = E_{s1}(k-2) - Ri_L(k-2) \\ E_{s2}(k) = V_{o2}(k-1) = E_{s2}(k-2) + Ri_L(k-2) \end{aligned} \tag{4.28}$$

We shall now study the interconnection circuit more carefully in order to select the appropriate value for $R_s$. We convert (4.20) and (4.28) to a first-order recurrence system by introducing the following variables

$$\begin{aligned} y_1(k) &= E_{s1}(k-1) \\ y_2(k) &= E_{s2}(k-1) \\ y_I(k) &= i_L(k-1) \end{aligned} \tag{4.29}$$

The system is now written as

$$\begin{bmatrix} i(k) \\ E_{s1}(k) \\ E_{s2}(k) \\ y_1(k) \\ y_2(k) \\ y_I(k) \end{bmatrix} = A \begin{bmatrix} i(k-1) \\ E_{s1}(k-1) \\ E_{s2}(k-1) \\ y_1(k-1) \\ y_2(k-1) \\ y_I(k-1) \end{bmatrix} = \begin{bmatrix} 1 - \frac{2RT}{L} & 0 & 0 & \frac{T}{L} & -\frac{T}{L} & -\frac{2RT}{L} \\ 0 & 0 & 0 & 1 & 0 & -R \\ 0 & 0 & 0 & 0 & 1 & R \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i(k-1) \\ E_{s1}(k-1) \\ E_{s2}(k-1) \\ y_1(k-1) \\ y_2(k-1) \\ y_I(k-1) \end{bmatrix} \tag{4.30}$$

The stability and the convergence speed of this system depend on the eigenvalues of $A$, obtained by solving $det(A - \lambda I) = 0$ for $\lambda$. As a reminder, an eigenvalue $|\lambda| > 1$ will lead to an unstable system, and the system will converge faster when the eigenvalues are close

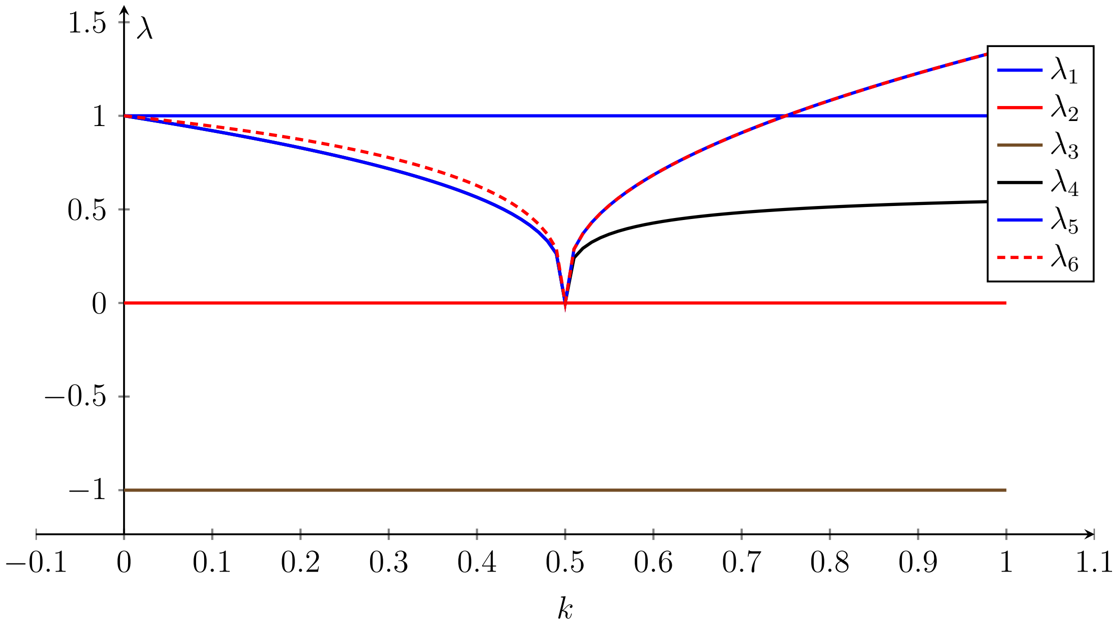


**Figure 4.14: Evolution of the eigenvalues of** (4.30) **as a function of** $k = T\frac{R_s}{L}$

to the origin [87]. To give a better perspective of the impact of the resistance, we first write

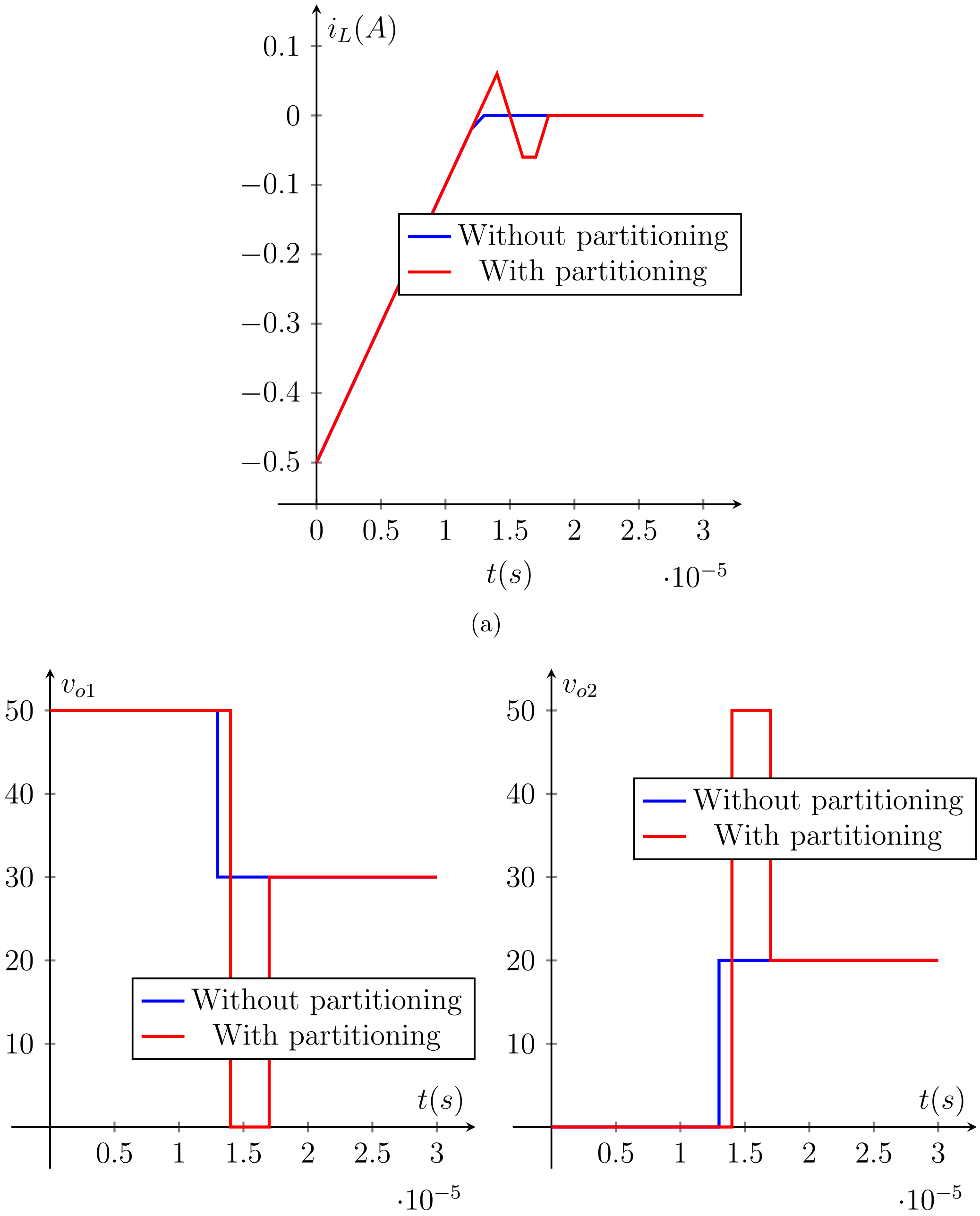
$$R_s = k\frac{L}{T} \tag{4.31}$$

The evolution of the modulus of the eigenvalues as a function of $k$ is plotted in Figure 4.14. A first observation is that selecting $k > 0.75$ will lead to an unstable system, which must absolutely be avoided. On the other hand, choosing $k = 0.5$ will result in eigenvalues placed in $\lambda = 0$, which means that the system will converge in a single time step. This choice corresponds to equating the time constant $\tau = \frac{L}{2R_s}$ to the time step $T$. The graphs of Figure 4.15 show the results of a simulation similar to the one performed in Figure 4.10: the inductor current is initialized with a negative value, and $d_1, d_4$ are conducting, allowing the current to evolve following the law $L\frac{\mathrm{d}i_L}{\mathrm{d}t} = V_{in} - E$. For this test, we have chosen $R = 100k\Omega$ for the polarizing resistors, and a value of $10V$ is chosen for $E$. The results are much better this time, with the current quickly converging to zero after the initial overshoot while the output voltages stabilize at the correct value in two clock cycles. The results are less than ideal when the value of $E$ is decreased to $2V$ or even to negative values: in this case, the current oscillates for many periods until it finally converges to zero (Figure 4.16). The explanation for this is simple: when the sign of the current is changed, the diode is first switched off. Then, we test if

$$J_s\frac{R}{2} + \frac{V_{in}}{2} > \pm V_{in} \tag{4.32}$$

The sign of the right hand sign depends on the previous topology. If this test is true, then the other diode of the bridge is switched on in turn, leading to an oscillating behavior similar to the one obtained with the previous method.

The performance is greatly enhanced by reducing the value of the polarizing resistances

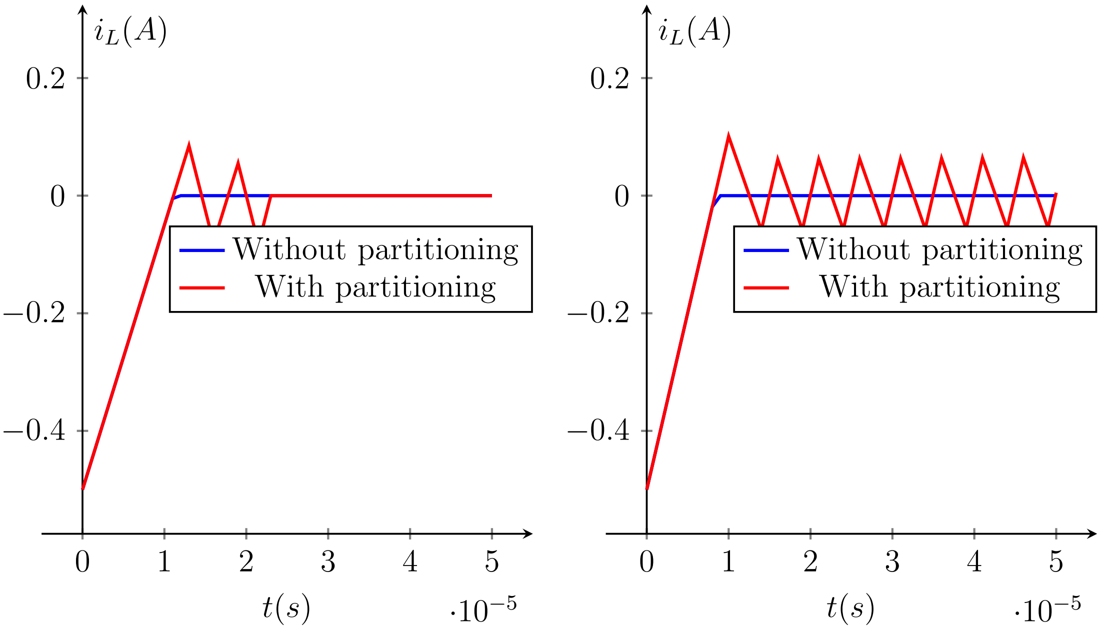


(a)

(b)

(c)

**Figure 4.15: Simulation of the circuit of Figure 4.7, when all active switches kept off and the inductor current $i_L$ starts with a negative value.**



(a)

(b)

**Figure 4.16: Simulation of the circuit of Figure 4.7, with $R = 100k\Omega$ and (a) $E = 5V$ (b) $E = -10V$**

so that (4.32) will never be true after a sign change. The highest value of the current after a sign change is

$$i_{max} = V_{in} - E\frac{T}{L} \tag{4.33}$$

The minimal value of $E$ that allows the circuit to stay in discontinuous-conduction mode is $E = -V_{in}$, leading to

$$i_{max} = 2V_{in}\frac{T}{L} \tag{4.34}$$

Putting this value into the equation for $J_s$ (defined by (4.24)) allows us to write the condition on $R$ to ensure that the diodes will stay off (written here for the case where the current goes from negative to positive values):

$$J_s\frac{R}{2} + \frac{V_{in}}{2} \leq V_{in} \Rightarrow R \leq \frac{L}{T} \tag{4.35}$$

If the resistance is any higher, the circuit will oscillate until the new value of the current is low enough for (4.32) to be false. This is what happens on Figure 4.16b The graph presented in Figure 4.17b shows that the current converges even for negative values of $E$ when $R = 1k\Omega$. This method improves the convergence of the circuit, but draws an additional current from the input source. It is thus limited to the cases where this additional current is negligible when compared to the typical currents of the application. The plots of Figure 4.18 show the behavior of the circuit during PWM control. Again, a slight overshoot is observed when the current converges to zero, but the results show a real improvement compared to those obtained using the previous method (shown on Figure 4.12).

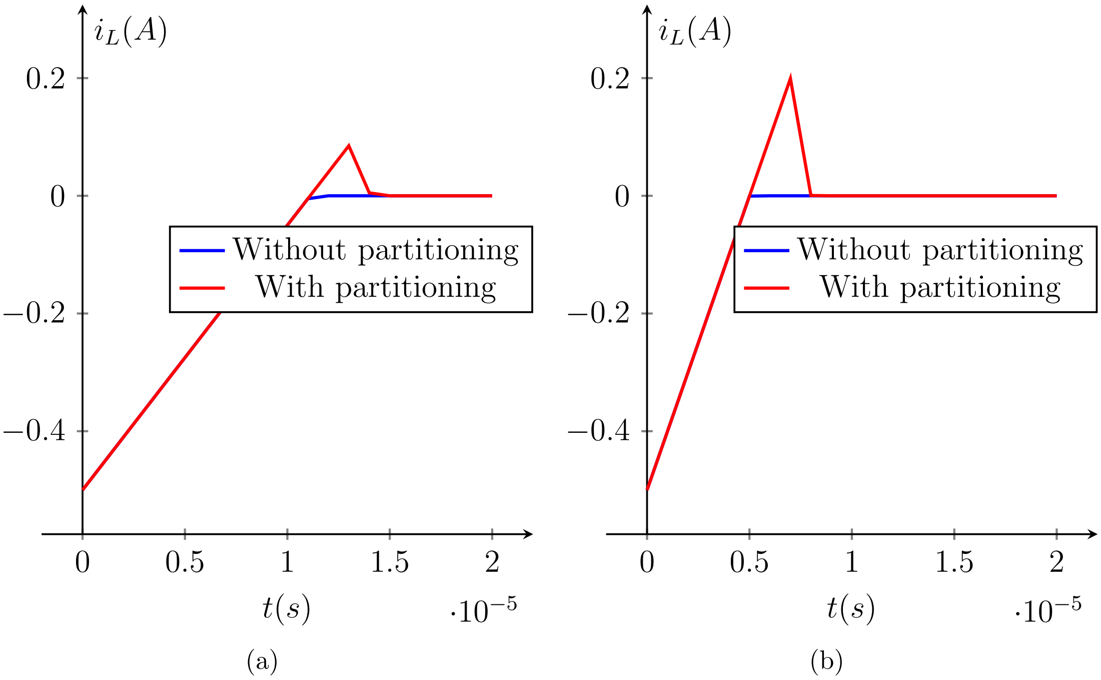
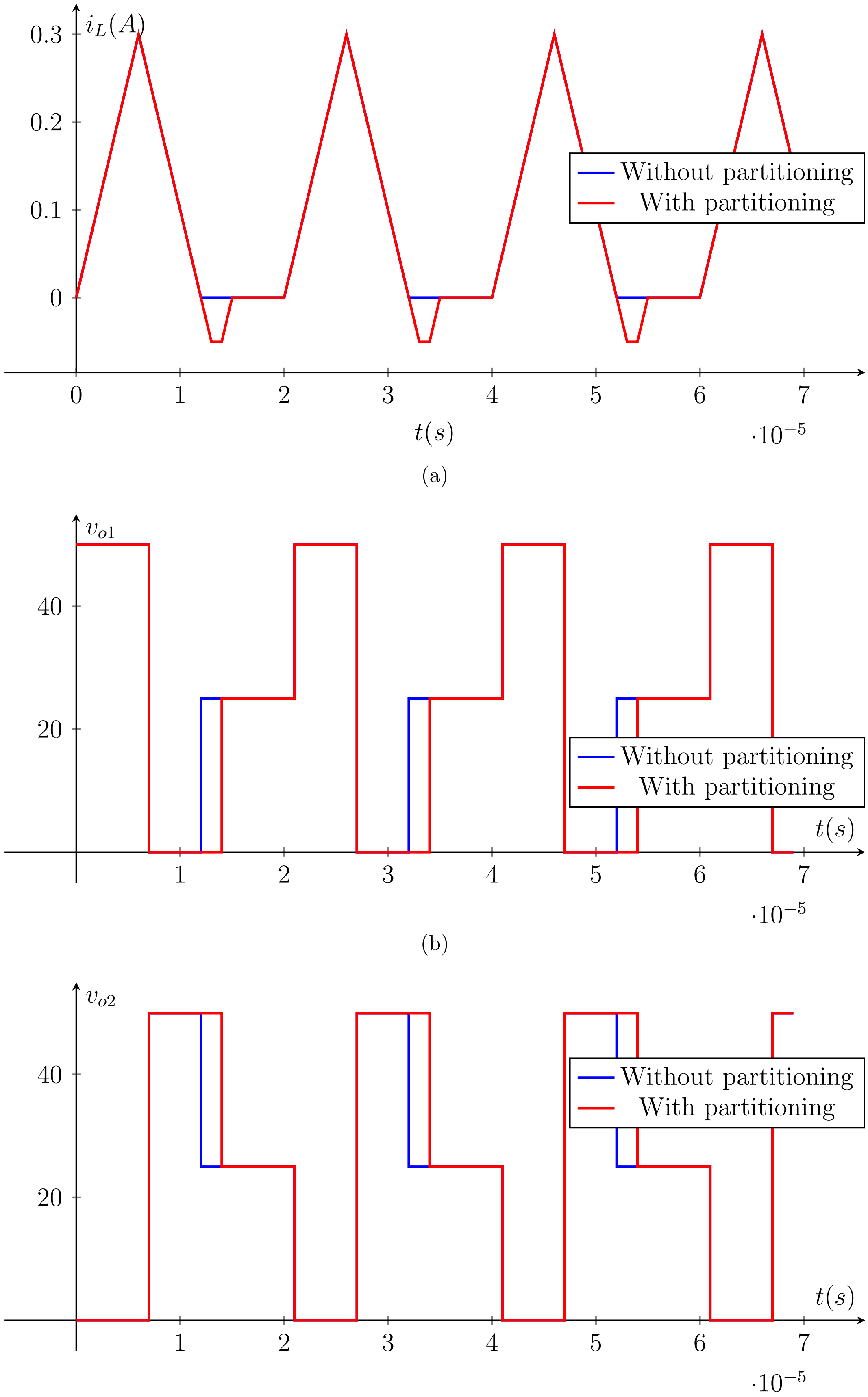


**Figure 4.17: Simulation of the circuit of Figure 4.7, with $R = 1k\Omega$ and (a) $E = 5V$ (b) $E = -50V$**

Finally, we will show the impact of the resistance placed in series with the inductance. Let us make the same test as the one performed on Figure 4.17, but this time we keep $s_1$ in the on state. The current originally flows through $d_4$, and converges to zero, at which point all the diodes of the right branch go into blocking state. The results, drawn on Figures 4.19a and 4.19b for $E = 20V$, show that the current and the output voltage oscillate while converging to a constant value. Since the left bridge stays in conducting mode, the corresponding $R_s$ is compensated by the source $E_{s1}$. As a result, the inductor only sees $R_s$ instead of $2R_s$ and the eigenvalues are not placed at the origin, which in this case introduces an oscillatory behavior.
The same test is repeated for $E = 0V$ on Figures 4.19c and 4.19d. This time, the diode $d_3$ is turned on when the sign of the current is changed and cannot be turned off since the current across the inductor does not decrease (the voltage is equal to $V_{in}$ on both sides of the inductor).
This section has proven that it is possible to partition a circuit operating in DCM in a way that keeps the inductor outside of the partition which causes the discontinuity, as long as proper adjustments are made to the circuit.

## 4.3 Generalizing the results to any converter

In this section, we shall integrate all the concepts previously studied to deduce a general method for partitioning any circuit. We will start by introducing some definitions before describing the steps and applying the methodology to a few examples. A typical power converter (or partition thereof) may be represented as shown in Figure 4.20. The power



(a)

(b)

(c)

**Figure 4.18: Simulation of the circuit of Figure 4.7, for the partitioned and non-partitioned versions, $T = 1\mu s, T_{PWM} = 20\mu s$, 30% duty cycle**

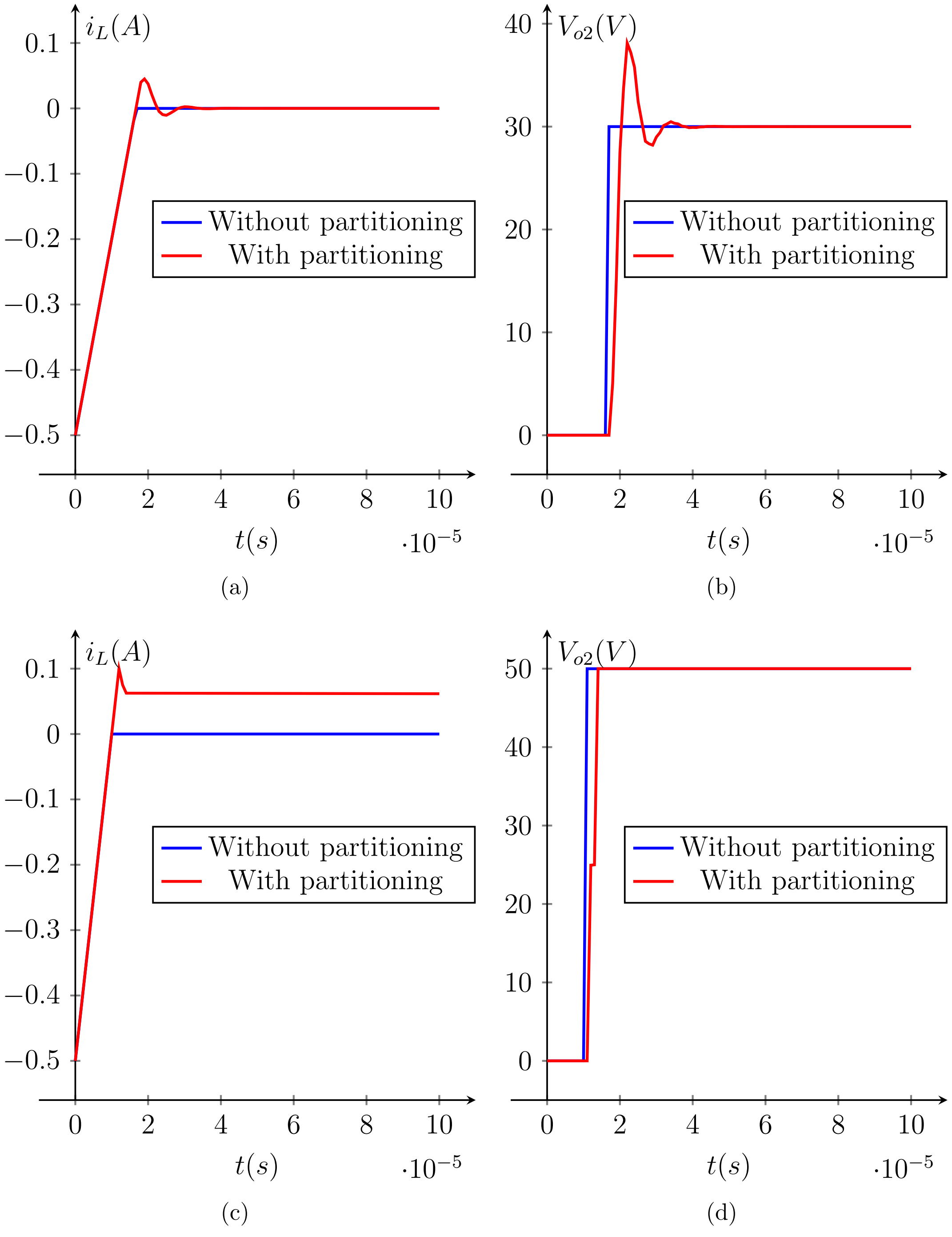


**Figure 4.19: Simulation of the circuit of Figure 4.7, with $R = 1k\Omega$, $s_1$ is kept on and (a)(b) $E = 20V$ (c)(d) $E = 0V$**

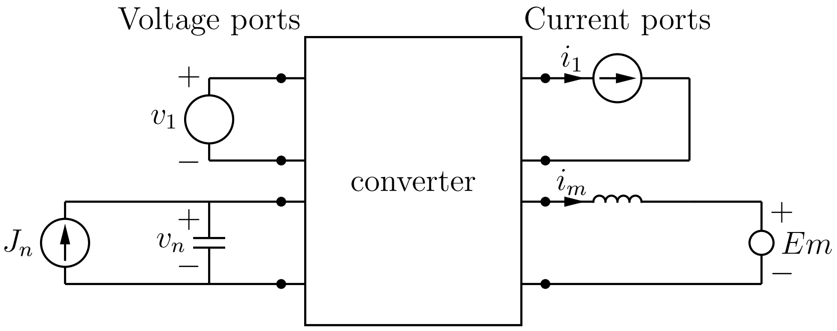


**Figure 4.20: Generic converter surrounded by its voltage and current input ports**

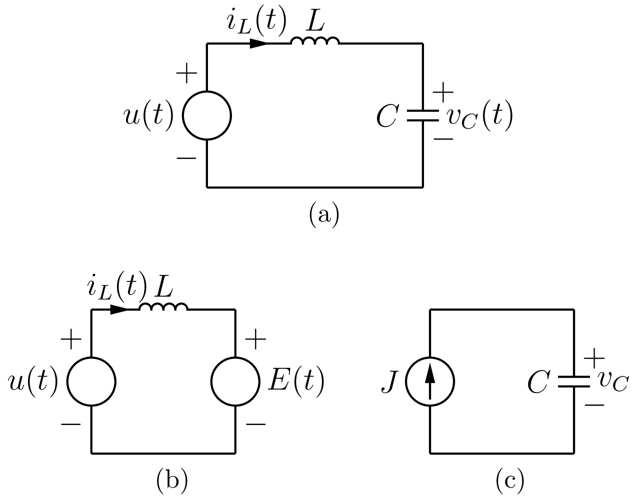


**Figure 4.21: Partitioning the circuit in three with full Thevenin equivalent for the interconnection circuit**

converter is connected to the outside world through voltage sources (called voltage ports) and current sources (current ports). These (controlled) sources may represent actual sources, but more generally correspond to any device that can be represented by its Thevenin or Norton equivalent. When a circuit is split into multiple partitions, these sources are the means by which the sub-circuits see each other.

### 4.3.1 Choosing the partitions

A given circuit may be partitioned in many different ways, some of which leading to better results than the others. As such, it would be useful to have some guidelines about the selection process. We already mentioned the first rule, which is to split the circuit in a point controlled by time constants much larger than the sampling period. The reason is that replacing a partition by a controlled source is equivalent to using the forward Euler approximation (FEA), which is prone to instabilities and less accurate than the backward Euler approximation (BEA) or the trapezoidal approximation [33] when the time constants of the system are in the same order of magnitude as the sampling period. Let us illustrate this by observing the LC circuit of figure 4.21a. The continuous-time differential equation governing the state variables are

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix}\begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u(t) \tag{4.36}$$

The FEA corresponding to this equation, obtained for a sampling period $T$, is

$$\begin{bmatrix} i_L(k) \\ v_C(k) \end{bmatrix} = \begin{bmatrix} i_L(k-1) \\ v_C(k-1) \end{bmatrix} + T\begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix}\begin{bmatrix} i_L(k-1) \\ v_C(k-1) \end{bmatrix} + T\begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u(k) \tag{4.37}$$

Let us now split the circuit between the inductor and the capacitor and replace each of the missing part by a controlled source, as shown -n Figure 4.21b and 4.21c. This circuit is now described by

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} i_L(t) \\ v_C(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} & -\frac{1}{L} & 0 \\ 0 & 0 & \frac{1}{C} \end{bmatrix}\begin{bmatrix} u(t) \\ E(t) \\ J(t) \end{bmatrix} \tag{4.38}$$

Since the coupled effects are now replaced by sources, using any single-step integration method on this system will always lead to the same recurrence equation:

$$\begin{bmatrix} i_L(k) \\ v_C(k) \end{bmatrix} = \begin{bmatrix} i_L(k-1) \\ v_C(k-1) \end{bmatrix} + T\begin{bmatrix} \frac{1}{L} & -\frac{1}{L} & 0 \\ 0 & 0 & \frac{1}{C} \end{bmatrix}\begin{bmatrix} u(k) \\ E(k) \\ J(k) \end{bmatrix} \tag{4.39}$$

The equations of the sources are

$$\begin{aligned} J(k) &= i_L(k-1) \\ E(k) &= v_C(k-1) \end{aligned} \tag{4.40}$$

Replacing into (4.39):

$$\begin{bmatrix} i_L(k) \\ v_C(k) \end{bmatrix} = \begin{bmatrix} i_L(k-1) \\ v_C(k-1) \end{bmatrix} + T\begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix}\begin{bmatrix} i_L(k-1) \\ v_C(k-1) \end{bmatrix} + T\begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u(k) \tag{4.41}$$

Which is exactly the same as (4.37), obtained from the complete circuit with the FEA.

As explained in section 4.2, circuits operating in discontinuous-conduction mode (DCM) must be handled with care. As a rule, we should always try to keep the inductor in the same partition as the diodes that may force it to enter DCM, the reason being that the current is then properly controlled and forced by the output equations of the partition. If separating the inductor from the diodes is unavoidable, then we must use a more complex control scheme for the sources, similar to the process that we used in section 4.2. As explained previously, the reason for this is that using sources with a constant form do not always allow the current to properly converge to zero, which can introduce unwanted oscillations and commutations.
If the inductor can be forced into DCM because of diodes present in two (or more) partitions, then it should be kept outside of all of these partitions. Indeed, the modified procedure studied in section 4.2 advocates the use of additional resistors to allow the

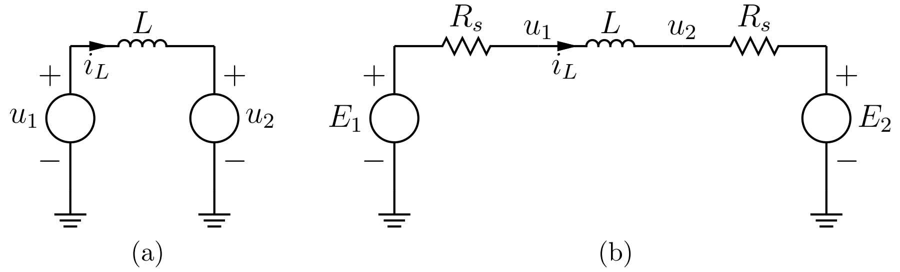


**Figure 4.22: (a) Definition of the inductor signals (b) Inductor with added convergence resistors**

measurement of the voltage on both sides of the inductors. These resistors are compensated by adding a $\pm Ri$ term to the sources $E_{1,2}$ while the circuit operates in continuous conduction mode. Omitting this additional term allows the current to converge to zero when the inductor should operate in DCM. However, this compensation is *only realized when using the FEA as the discretization method.* Let us take a look at the inductor on Figure 4.22a: the differential equation is simply

$$\frac{\mathrm{d}i_L(t)}{\mathrm{d}t} = \frac{1}{L}(u_1(t) - u_2(t)) \tag{4.42}$$

The circuit with the added resistors (Figure 4.22b) in described by

$$\begin{aligned} E_1(t) &= u_1(t) + R_s i_L(t) \\ E_2(t) &= u_2(t) - R_s i_L(t) \\ \frac{\mathrm{d}i_L(t)}{\mathrm{d}t} &= -\frac{2R_s}{L} i_L(t) + \frac{1}{L}(E_1(t) - E_2(t)) \end{aligned} \tag{4.43}$$

These equations are of course equivalent as long as the circuit is studied in continuous time. Discretizing (4.43) with the FEA leads to

$$\begin{aligned} E_1(k) &= u_1(k) + R_s i_L(k-1) \\ E_2(k) &= u_2(k) - R_s i_L(k-1) \\ i_L(k) &= i_L(k-1) - T\frac{2R_s}{L} i_L(k-1) + \frac{T}{L}(E_1(k) - E_2(k)) \\ &= i_L(k-1) + \frac{T}{L}(u_1(k) - u_2(k)) \end{aligned} \tag{4.44}$$

which is exactly the same as (4.42) discretized with the FEA, showing that the effect of the resistors is properly compensated. Assuming we apply the backward Euler approximation to (4.43) instead, the discrete system is now

$$\begin{aligned} E_1(k) &= u_1(k) + R_s i_L(k-1) \\ E_2(k) &= u_2(k) - R_s i_L(k-1) \\ i_L(k) &= \frac{1}{1 + T\frac{2R_s}{L}} i_L(k-1) + \frac{1}{1 + T\frac{2R_s}{L}} \frac{T}{L}(E_1(k) - E_2(k)) \end{aligned} \tag{4.45}$$

The resistors are not compensated anymore and appear in the equations, resulting in a system that converges to $i_L = 0$ with a time constant $\tau = \frac{L}{2R_s}$
As a consequence, *the partition containing the inductor must be discretized using the FEA*. Again, this method is far less accurate and stable than some of the other solvers, and we should reduce its usage to a minimum. Therefore, we should keep the inductor in its own partition.
When partitioning this way, it is crucial to remember that the current is *indirectly* forced to zero, which leads to longer convergence time and additional transients during a few clock cycles. Combined, these effects can significantly modify the behavior of the circuit when in presence of DCM.

### 4.3.2 Analyzing the partitioned circuit

Once the partitions are chosen, the next step is to perform the analysis of the complete circuit according to the methods introduced in chapter 3. This is done by following these steps

1. Split the circuit into partitions

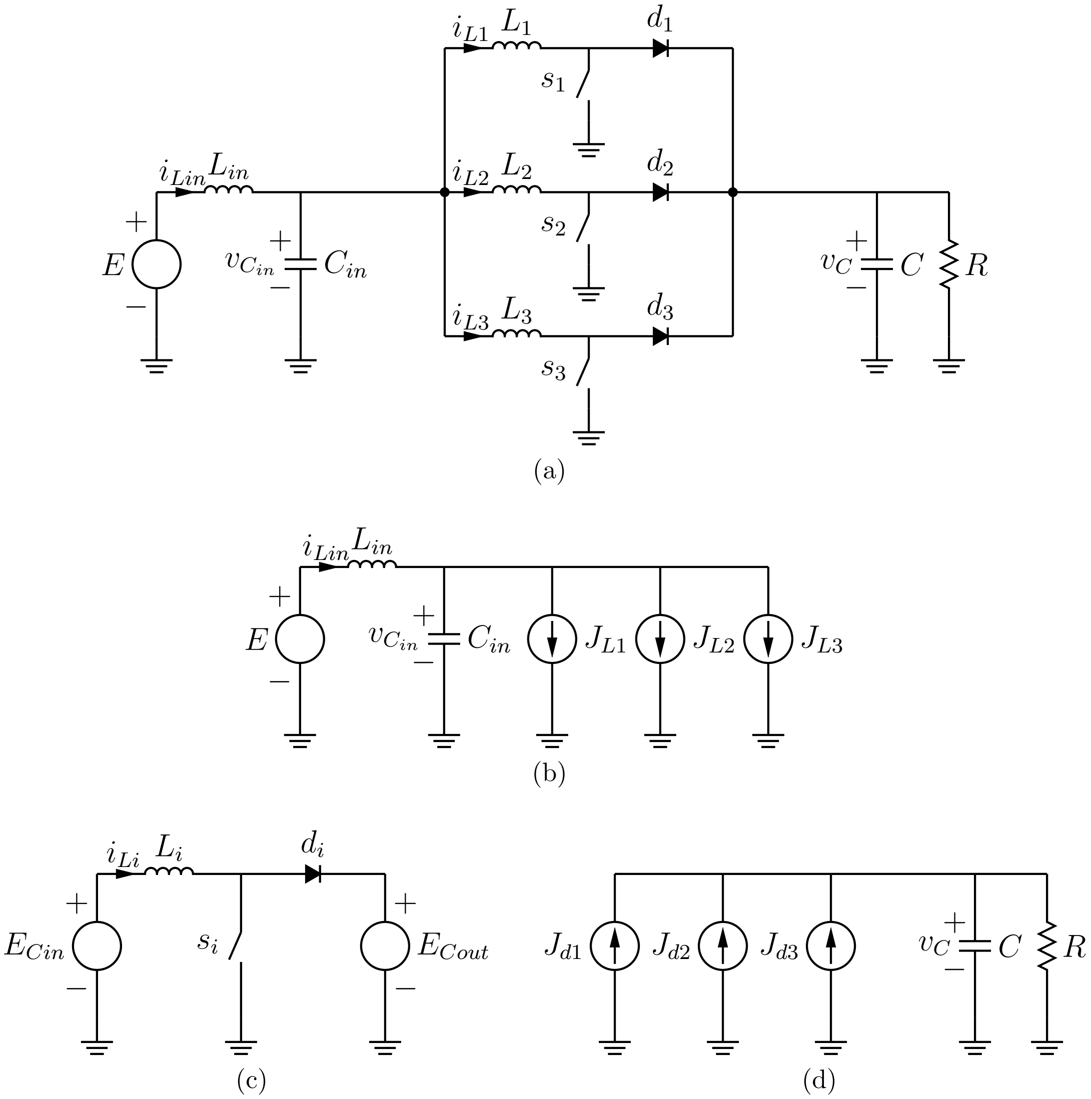
2. For each partition

   - Add voltage sources at each voltage port, and current sources at each current port
   - Add a measurement of the input current at each voltage port, and a measurement of the voltage at each current port. These are used to update the sources of the connected partitions
   - Carry the automated natural and forced switching analysis

3. Establish the laws controlling the sources

   - If the circuit cannot enter DCM, simply use the measurements taken at the ports of the partitions
   - If the circuit can enter DCM, write laws based on the example of (4.24) and (4.25). The goal of this controlling scheme is to ensure that the correct current is measured at the output of the converter while keeping the correct output voltage

The main difficulty of this procedure is to find if the circuit can operate in DCM and how exactly it modifies the equations. And unfortunately, we have not found any completely generic method at this time.
However, we are able to define whole classes of converters that obey the same laws. For example, any switching branch based on an ideal voltage source (also called *voltage source converter*) and connected to an inductor will have a controlled current source whose laws are governed by (4.24) and (4.25), and the partition containing the inductor will have equations similar to (4.22). Since inductors are often present at the output of converters to filter the current (or they represent the equivalent inductance of a transformer or of a motor), these equations will be found in applications.



(a)

(b)

(c)

(d)

**Figure 4.23: Multiple parallel boosts, and the three selected partitions**

In the next section, we will apply the partitioning methodology to a range of circuits to show how universal it really is and what are the practical limitations

## 4.4 Test cases

### 4.4.1 Introduction

In this section, we will perform an analysis similar to the one we did in section 3.5. The goal is to study the partitioned circuits not only in terms of accuracy, but also in terms of complexity of the tables and quantity of conditions to evaluate.

### 4.4.2 Parallel boost converters

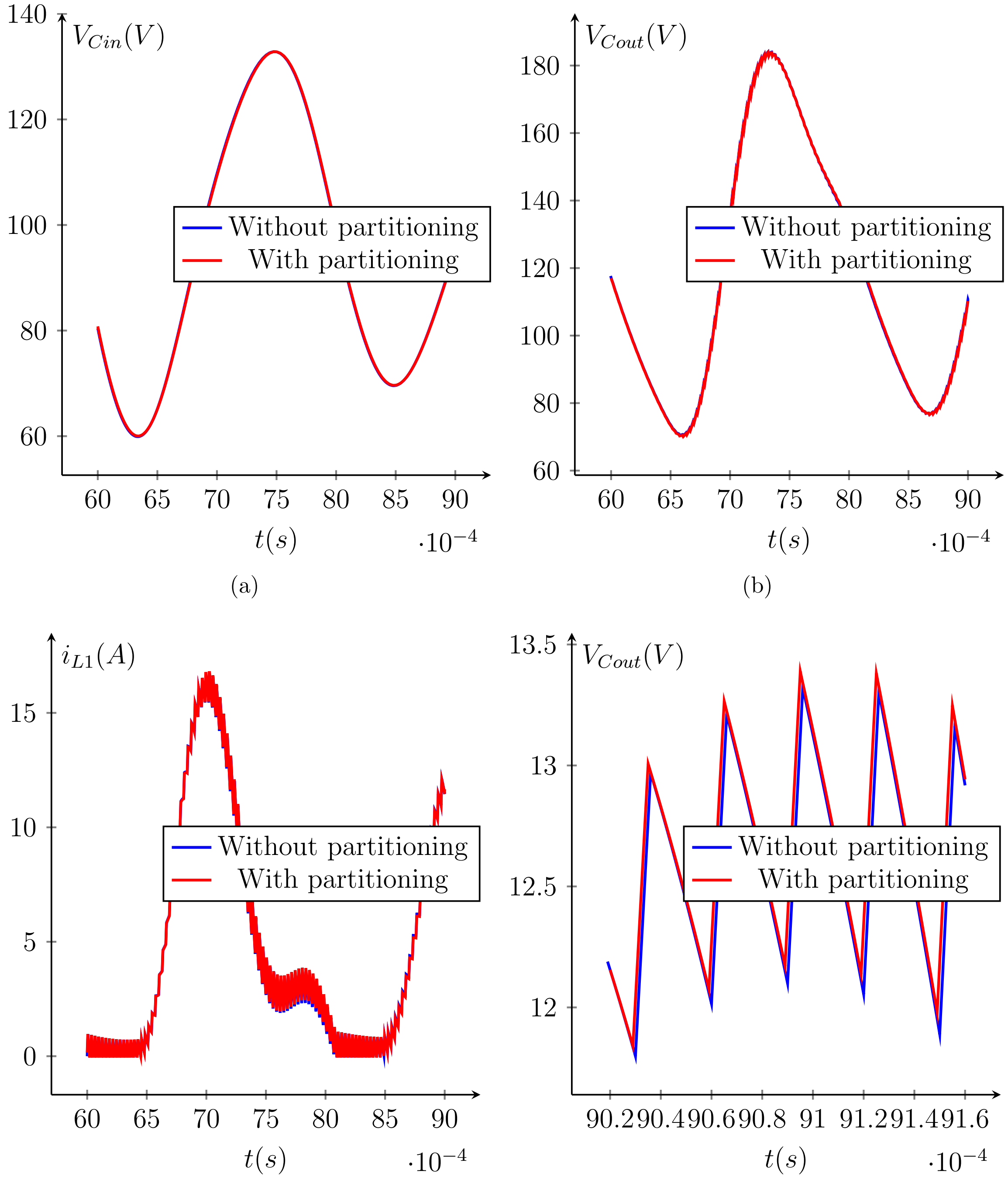
Our first example will be the circuit of Figure 4.23. This circuit is composed of three boost converters placed in parallel. Compared to the standard boost architecture, this variant provides redundancy and allows us to use components which are rated for lower currents [1]. By shifting the PWM signals by a fraction of the period, the switching frequency is virtually multiplied by three, which allows us to select smaller reactive components.
To partition this circuit, we choose to dissociate the boost converters from the input and output capacitors. Since the inductors $L_1, L_2, L_3$ can enter DCM, they are best kept inside the boosts. This leads to the circuit of Figures 4.23b to 4.23d. Note that the boost circuit itself is used three times (but represented only once at 4.23b with an index $i = 1, 2, 3$), showing how this method enables us to easily reuse pre-existing modules. The simulation of Figure 4.24 show the behavior of the partitioned circuit compared to the behavior of the unpartioned one for the following values:

$$\begin{aligned} C_{in} &= 1mF \\ C_{out} &= 100\mu F \\ L_i &= 500\mu H \\ R &= 10\Omega \\ T_{PWM} &= 30\mu s \\ \text{duty cycle} &= 20\% \end{aligned} \tag{4.46}$$

The input and output voltages are almost exactly the same in both versions, while a small error (around 0.3% of the maximum value) is present on the waveform of $i_{L1}$. This error is very small and should not hamper any real-world application of the emulator.
Unfortunately, the performance is significantly degraded when the input capacitor $C_{in}$ is reduced to $100\mu F$, Figure 4.25 clearly shows that the oscillations are not correctly damped. This result confirms the limitations of the partitioning method, since faster circuits will lead to less accurate waveforms. As mentioned previously, this effect is partially due to the fact the splitting the circuit is similar to using the forward Euler approximation, which is far less accurate than the trapezoidal approximation used inside the partitions.
However, in this particular case, we would expect the circuit to have a large input capacitor so the effect should be limited in practice. Moreover, simulating the circuit with a smaller time step also reduces the error due to the update delay of the sources.
Let us now look at the performance of the automated parsing algorithms, starting with the study of the topologies. Each of the boost converters has three stables topologies (see section 3.5.2 for the complete analysis), and the total number of topologies for the complete circuit is equal to $3 * 3 * 3 = 27$, the same as we would have by studying the complete circuit at once. This result makes sense since the internal configuration of the boost converters does not depend on the other partitions. For the same reasons, the total amount of conditions to evaluate for the natural and forced switching module is also the same:

- Each modules needs the test of $i_{Li} \leq 0$ or $E_{Cin} - E_{Cout} > 0$ to switch its diode

[1] Especially if a MOS transistor is used to implement the switch, since the power dissipation is given by $P = R_{DSON} \times i^2$



(a)

(b)

(c)

(d)

**Figure 4.24: Simulation of the parallel boost converter, with the parameters given in** (4.46)

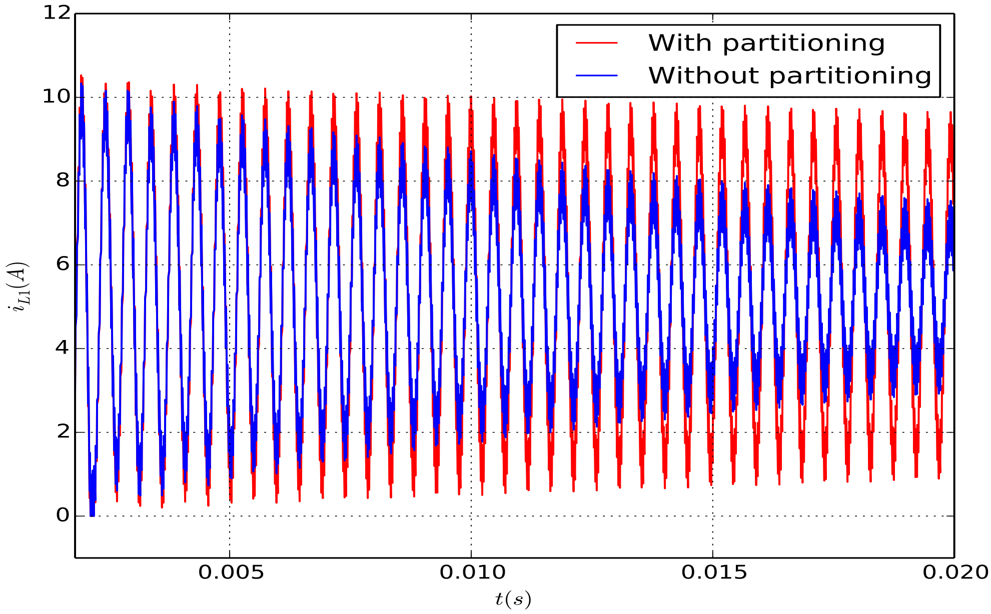


**Figure 4.25: Simulation of the parallel boost converter, with $C_{in} = 100\mu F$. All other parameters are defined in** (4.46)

- The forced switching module does not require any condition, as switching $s_i$ forces $d_i$ to change its state

These results show that splitting the circuit does not always lead to an increase in resource usage, since the tables of the whole converter would have been the concatenation of the local tables. The only real impact of the separation lies in the accuracy of the simulation, which is degraded because of the delays between the modification of a state variable and its effect on the rest of the circuit.

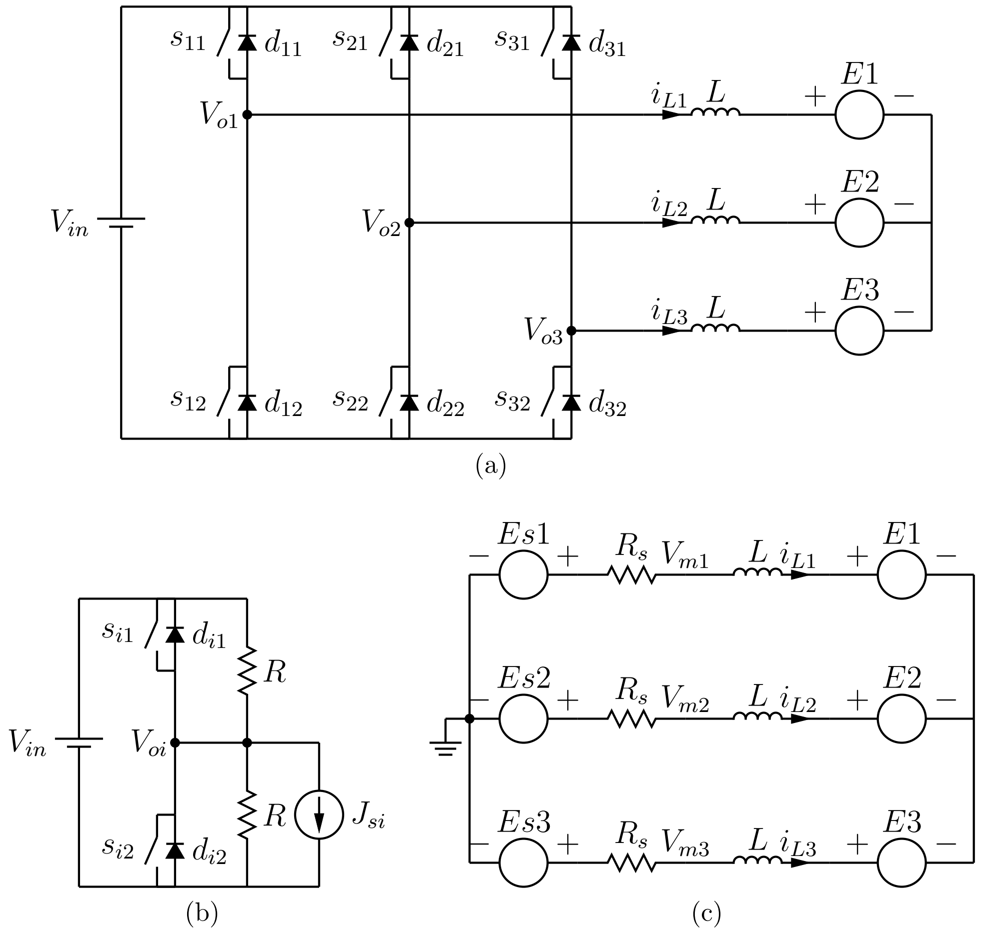
### 4.4.3 Three-phase inverter

As an example, we will try to generalize the results obtained with the full bridge to the three-phase circuit of Figure 4.26a. This circuit contains three switching branches which are modeled using the circuit of Figure 4.26b, and an interconnection circuit containing three inductors and three voltage sources, representing the load. The convergence resistors are added to the interconnection circuit, as shown on Figure 4.26c.

Since the switching branches are exactly the same as the ones we used in the full bridge, their current source is controlled by (4.24) and (4.25), and we can use our pre-studied modules without any modifications. The inductor current $i_L$ in the equations is replaced by $i_{L1}$, $i_{L2}$ or $i_{L3}$ depending on the studied branch.

To select the value of the convergence resistors $R_s$, we first assume that the voltage sources $E_{s1}$ to $E_{s3}$ are constants. The total inductance and resistance seen by each source are equal to $L_{tot} = L + L\|L = \frac{3}{2}L$ and $R_{tot} = \frac{3}{2}R$. The value of $R$ is chosen such that

$$\tau = \frac{L_{tot}}{R_{tot}} = \frac{L}{R} = T \Rightarrow R = \frac{L}{T} \tag{4.47}$$



(a)

(b)

(c)

**Figure 4.26: Partitioning the circuit in three with full Thevenin equivalent for the interconnection circuit. On (b), $i \in (1, 2, 3)$ corresponds to the $i^{th}$ branch**

| | Partitioned circuit | | Non partitioned circuit |
|---|---|---|---|
| | Per branch | Global | |
| Topologies | 7 | $7^3 = 343$ | 232 |
| Conditions (forced) | 4 | $4 * 3 = 12$ | 9 |
| Conditions (natural) | 3 | $3 * 3 = 9$ | 22 |

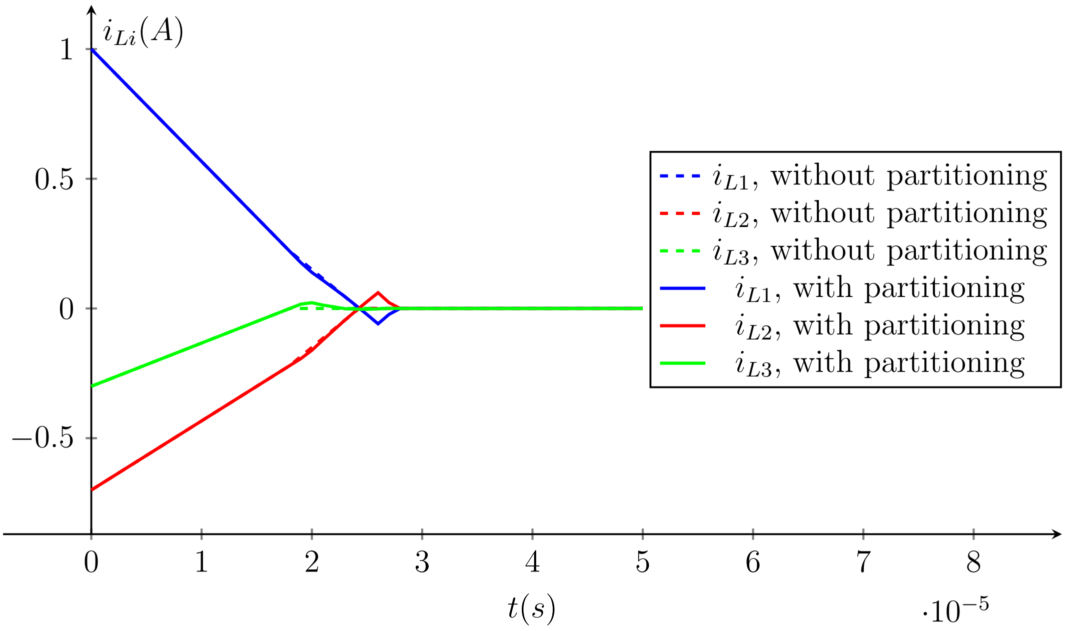
**Table 4.1: Ressources needed to simulate the three-phase circuit, for the partitioned and non-partitioned versions**

The plots of Figure 4.27 allow us to compare the results of the simulation of the partitioned circuit with the signals measured on the original circuit when the circuit operates in discontinuous conduction mode, and using the following parameters

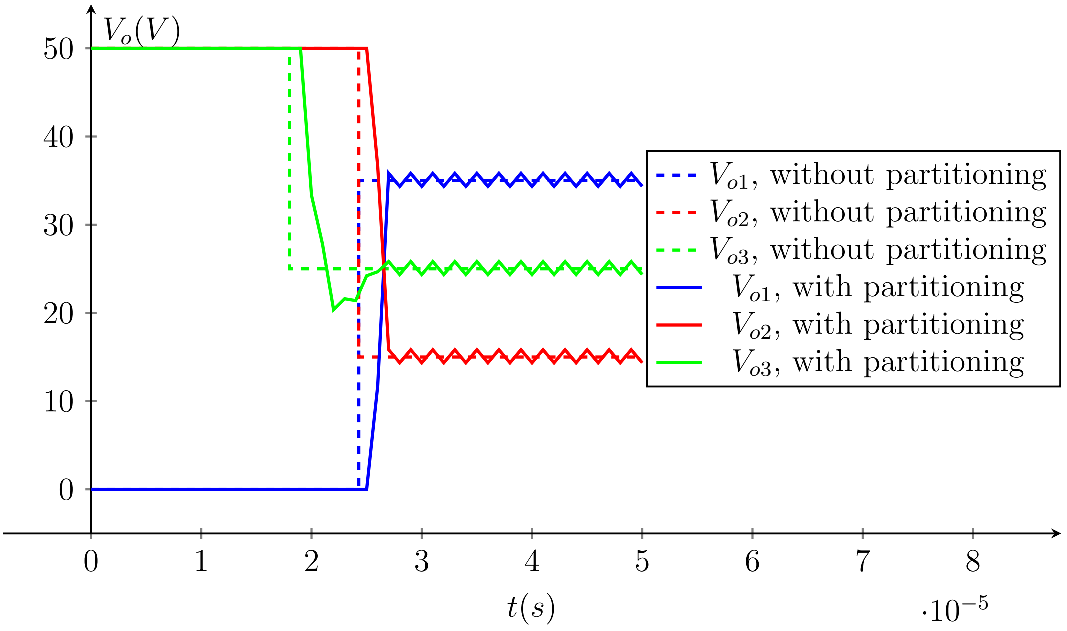
$$\begin{aligned}
V_{in} &= 50V \\
L_1 = L_2 &= L_3 = 1mH \\
i_{L1}(0) = 1A, i_{L2}(0) &= -0.7A, i_{L3}(0) = -0.3A \\
E_1 = 10V, E_2 &= -10V, E_3 = 0V
\end{aligned} \tag{4.48}$$

The currents of the partitioned circuit follow accurately the model until they hit zero. At this point, small oscillations are present until they all converge after a few clock cycles. The output voltages of the branches also present oscillations and never converge to the correct value, but instead oscillate around it. As it was the case previously, these results are perfectly fine as long as the very short term (between one to ten time steps) behavior of the circuit is not crucial. To test if the few time-steps required for the convergence of the currents have an impact on the global waveforms, we feed the switches $s_{11}$, $s_{21}$ and $s_{32}$ with a $20kHZ$, 60% duty cycle PWM signal. As shown of Figure 4.28b, the current $i_{L1}$ enters DCM, leading to a transient until it finally converges to zero. While the transient disturbs the current locally, the global waveforms are almost not disturbed. The effect is much more pronounced if we do not let the current converge to zero by increasing the PWM frequency to 40kHz. As shown in Figure 4.29, the waveform of the emulated current is certainly not accurate enough. This imposes a maximum on the PWM frequency, as we need to let the current reach zero before switching on the transistors again.

Since the partitions are not intrinsically decoupled, forcing the separation leads to a larger amount of conditions to evaluate. One of the reasons is that the optimization tool has no longer a global view on the circuit, and is unable to eliminate some topologies that would have deemed unstable. For example, the topology where $d_{11}$, $d_{21}$ and $d_{31}$ are on should be unstable, since the sum of the diode currents is equal to zero (see section 3.3.3). Since each branch is studied separately, this kind of simplification is no longer done. The net result is that, while the study of each part leads to a reduced local complexity for the forced and natural switching modules, simulating the global circuit will probably require more resources. Interestingly, this is not really the case in this example. As shown on Table 4.1, while the total amount of conditions required by the forced switching module is larger when the circuit is partitioned, the natural switching module requires less than half the conditions needed when the circuit is kept in one piece. The reason for this is that the branches cannot directly influence each other, reducing the size of the graphs.

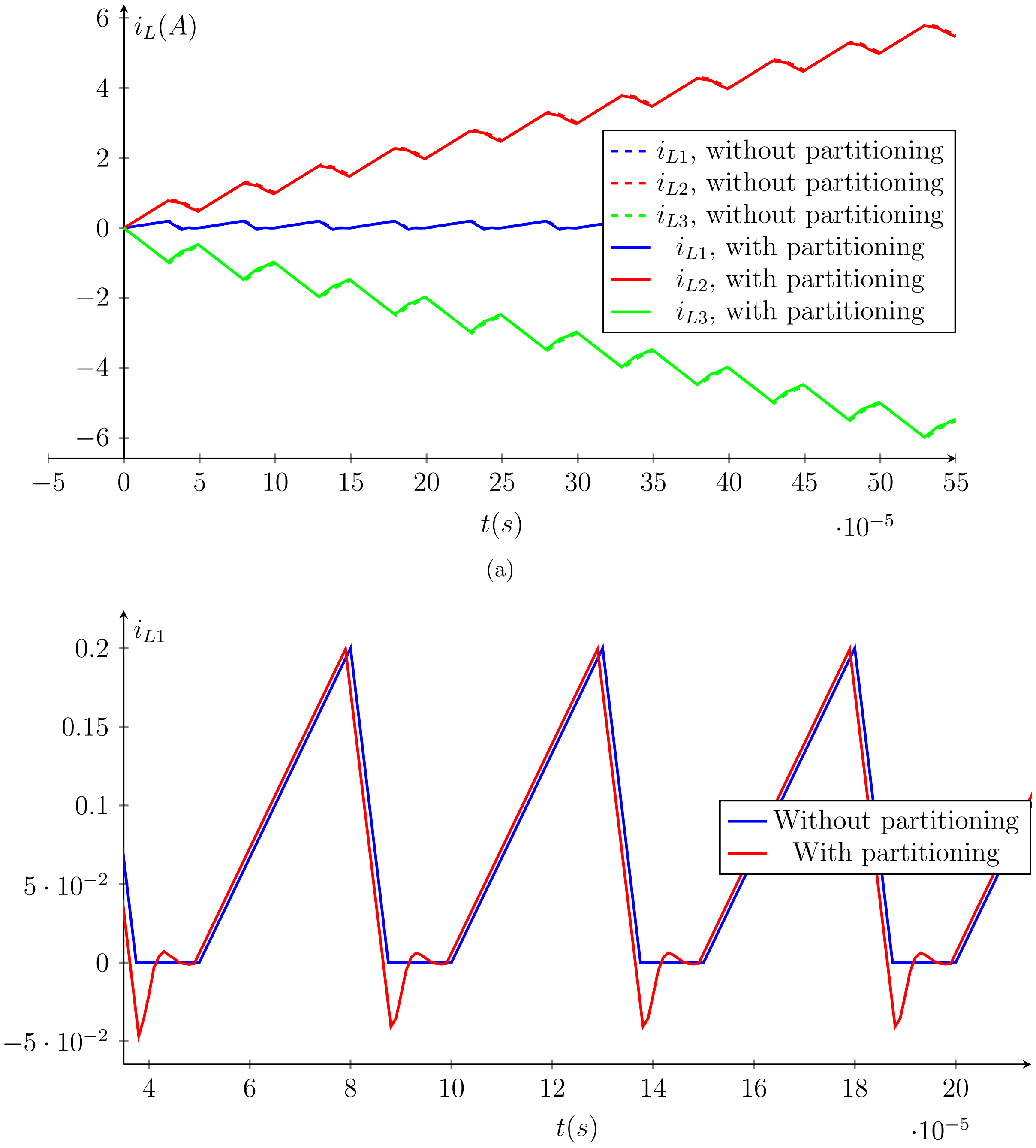


(a)



(b)

**Figure 4.27: Simulation of the three-phase circuit of Figure 4.26a when the circuit operates in natural conduction mode with all active switches left open, with the parameters defined by** (4.48)



(a)

(b)

**Figure 4.28: Simulation of the three-phase circuit of Figure 4.26a in PWM mode with $V_{in} = 50V, L = 1\mu H, E_1 = 10V, E_2 = -10V, E_3 = 0V$ and 60% duty cycle for the switches $s_{11}$, $s_{21}$ and $s_{32}$ (b) Zoom on $i_{L1}$, showing the transients as the first branch enters DCM**

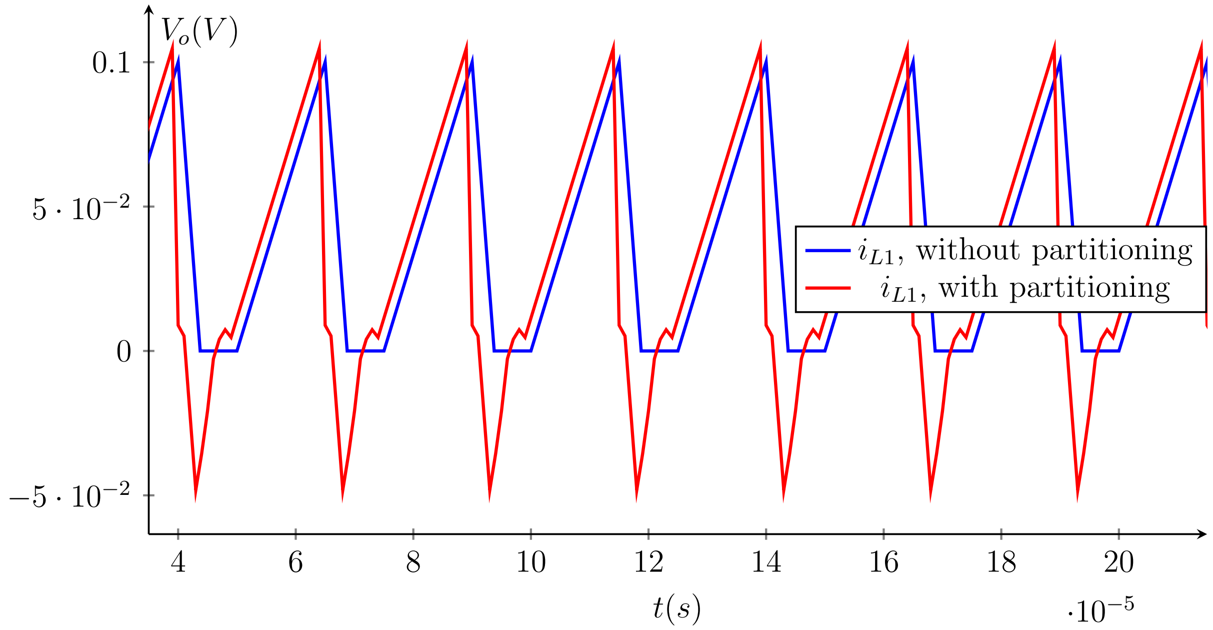
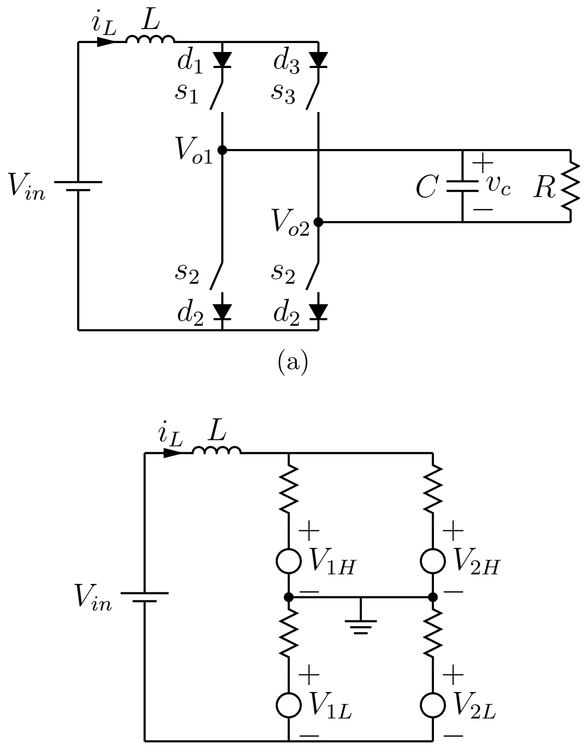


**Figure 4.29: Simulation of the three-phase circuit of Figure 4.26a, with the PWM frequencyincreased to** $40kHz$

Of course, this is also the reason why the simulation is less accurate to begin with.
This reduction in the amount of conditions is far from being a general result. For example, a two-phase bridge would require *one* condition for the forced switching module, and *three* conditions for the natural switching module, while a partitioned circuit would respectively require $4 * 2 = 8$ and $3 * 2 = 6$ conditions. But then again, a simple full bridge does normally not need to be partitioned.
This example shows once more the limitations of splitting a circuit operating in DCM. Because of the delay between the detection of the negative currents (made by measuring the diode currents) and the convergence of the inductor currents, the circuit is not able to react correctly to fast changing signals.

### 4.4.4 Current source inverter

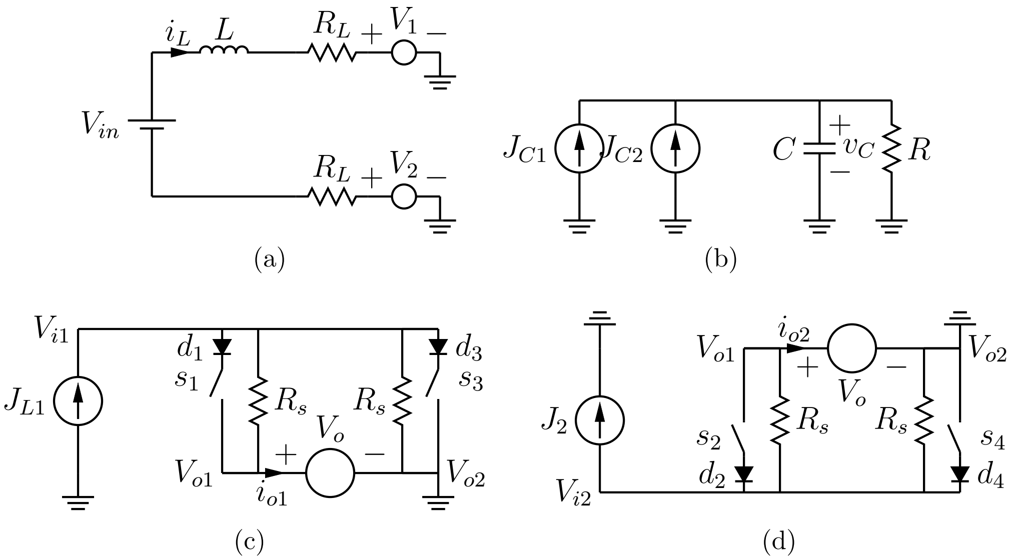
The three-phase circuit studied in the previous section acts as a voltage-to-current converter, converting a single voltage source to three different currents. The next example acts in the opposite way, and converts a current source into a controlled voltage. The circuit is represented in Figure 4.30a: the inductor current is selectively sent to one (and only one) of the output branches by switching the correct transistors, or to none of the phases by closing the two switches of a single branch.
The first question is how to split the circuit. Since the inductor can enter DCM, and since we will partition the bridge itself, it makes sense to keep it apart from the rest of the circuit, just like we did during the study of the full bridge. The next problem is how to split the converter itself. While separating the branches vertically seems to most logical choice, the connections between the partitions are relatively complex. For instance, let us study the inductor partition. Let us try to find the correct configuration for the sources which represent the branches. There must be a parallel connection between both, as the



(a)

(b)

**Figure 4.30: Current source inverter (a) Full circuit (b)Inductor partition corresponding to a vertical split of the converter**



(a)

(b)

(c)

(d)

**Figure 4.31: Horizontal partitioning of the current source inverter (a) Inductor partition (b) Capacitor partition (c) High side of the bridge (d) Low side of the bridge**

circuit will only enter DCM if both branches are switched off. Furthermore, each branch must be represented by *two* sources, as they may be connected to the high side or the low side of the inductor. This leads to the circuit of Figure 4.30b, which is rather complex. While adding all these elements seems to provide an acceptable partitioning of the circuit, there are a lot of additional components for a circuit that should not be more complex than a simple full bridge. Let us partition the bridge *horizontally* instead of vertically. The two partitions are now represented by the circuits of Figures 4.31c and 4.31d. These circuits, which look like maximum/minimum selectors, also contain the two polarization resistors $R_s$. The other circuits, drawn in Figures 4.31a and 4.31b, are much simpler and provide a good comparison point with the full bridge. Indeed the circuits in Figures 4.31a, 4.31c and 4.31d are almost exactly the same partitions as those drawn in Figure 4.13, which helps us to write the equations controlling the sources. This way, current source converters (CSCs) show a duality with the voltage source converters (VSCs): whereas the VSC is split vertically to separate the branches, it is more efficient to split a CSC horizontally to isolate the parts fed by the same current source.

The connections between the two half-bridge partitions and the output partition are straightforward:

$$\begin{aligned} V_o(k) &= v_C(k-1) \\ J_{C1}(k) &= i_{o1}(k-1) \\ J_{C2}(k) &= i_{o1}(k-1) \end{aligned} \tag{4.49}$$

Since the inductor can go into DCM, the equations depend on the topology. If at least one diode of the upper half is conducting, the connections between this partition and the inductor are

$$\begin{aligned} J_{L1}(k) &= i_L(k-1) + \frac{2V_{i1} - V_o(k-1)}{R_s} \\ V_1(k) &= -R_L i_L(k-1) + V_{i1}(k-1) \end{aligned} \tag{4.50}$$

If both diodes are off, then the terms proportional to $i_L$ are dropped from the equations. By symmetry, we also have

$$\begin{aligned} J_{L2}(k) &= i_L(k-1) - \frac{2V_{i2}(k-1) - V_o(k-1)}{R_s} \\ V_2(k) &= R_L i_L(k-1) + V_{i2}(k-1) \end{aligned} \tag{4.51}$$

To test the accuracy of the simulation, we first assume the following parameters

$$\begin{aligned} V_{in} &= 10V \\ L &= 100\mu H \\ C &= 1mF \\ R &= 1\Omega \end{aligned} \tag{4.52}$$

The control signals are: $s_1$ and $s_4$ are switched on and $s_2$ and $s_3$ are switched off during 80% of the PWM period, fixed to $50\mu s$. Then, for the remainder of the PWM period, the commands are inverted. With these parameters, the circuit stays in continuous conduction mode, as shown on the plot of Figure 4.32. Under these constraints, the circuit converges rather quickly with an error of about 1% on the voltage and 4% on the current. This error is essentially due to the lag of two time steps between the modification of $v_C$ and its effect on the current (one time step to update $V_o$ and another one to update $V_1$ and $V_2$). While this error is a bit higher than usual, the simulation remains perfectly usable for most applications. Again, the performances are reduced when the circuit enters discontinuous conduction mode. To illustrate this, we change $R$ to $10\Omega$, and change the control signals so that $s_1$ remains in the on-state all the time, $s_2$ stays in the on-state for 30% of $T_{PWM}$ and $s_4$ is switched on for the remained of the PWM period. Under these conditions, the circuit can be assimilated to a boost converter with additional diodes placed in series with the inductor, which is a case similar to the circuit studied in section 3.5.3. And, just like in this circuit, a full time step is required to switch the diodes, reducing the effective duty cycle and disturbing the signals when the circuit starts with its diodes off. Because of the partitioning, an additional transient of a time-step appears when the current starts to rise. These effects can be seen in Figure 4.33c.
Together, these time-steps reduce the effective duty cycle by more than 10%, which has a visible effect on the plots of Figure 4.33. This limits the application of the partitioning when the circuit is used in DCM, and solving this problem should be one of the major points of future works.

### 4.4.5 Three-Phases Neutral Point Clamped Inverter

Finally, we shall study the three-phase, three-level, neutral-point-clamped inverter (NPC) of Figure 4.34. This circuit contains 30 switches (18 diodes and 12 transistors), and is able to drive each of its three outputs to $0V$, $V_{in}$ or $-V_{in}$, depending on the configuration of the transistors and the diodes. In this particular case, each phase of the converter is loaded by an inductor $L$ placed in series with a resistor $R_L$, and the three load branches are star-connected.
While the single phase NPC was successfully emulated in section 3.5.6, let us recall that the three-phase variant cannot be studied as a single circuit using our algorithm. Indeed,

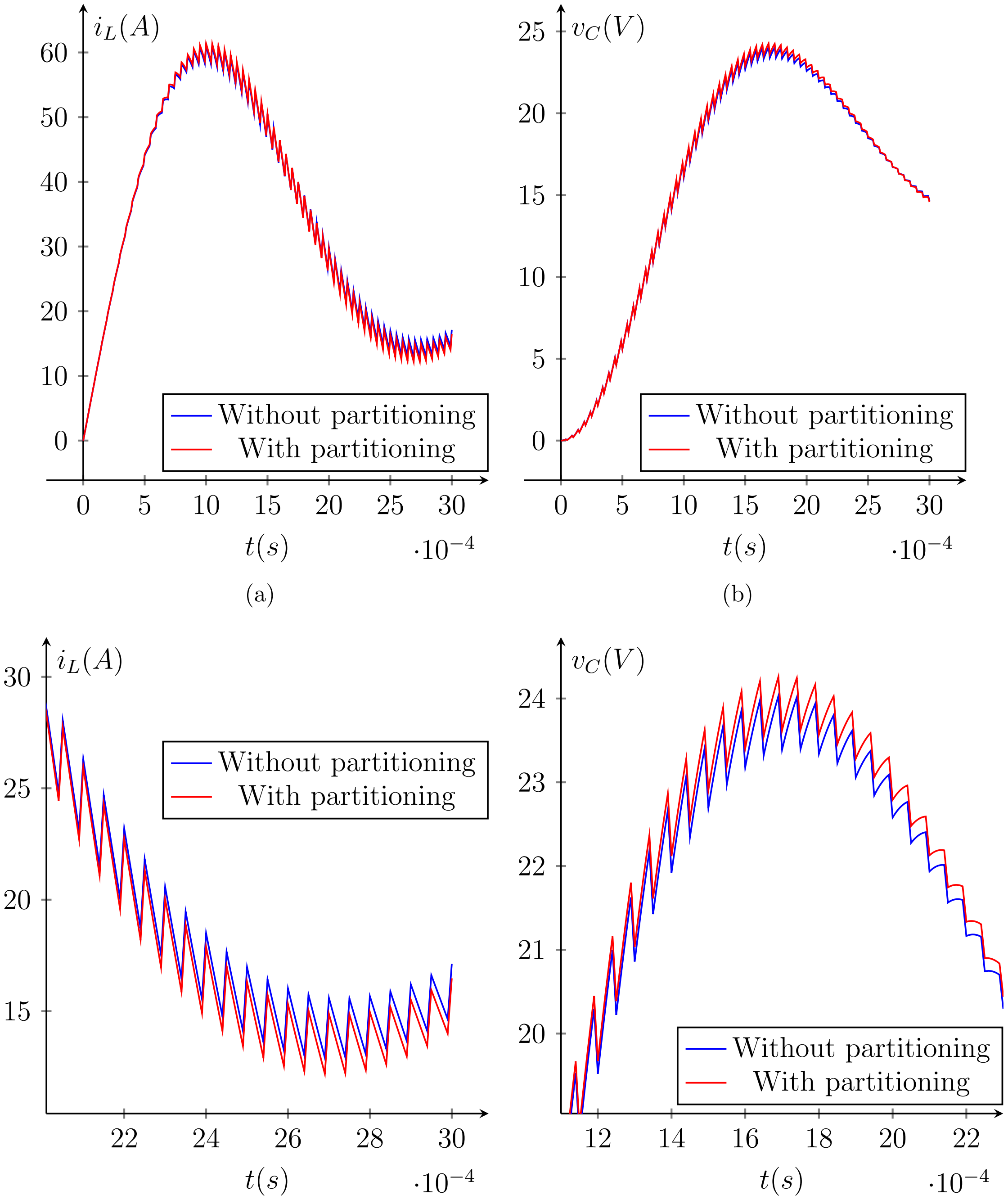
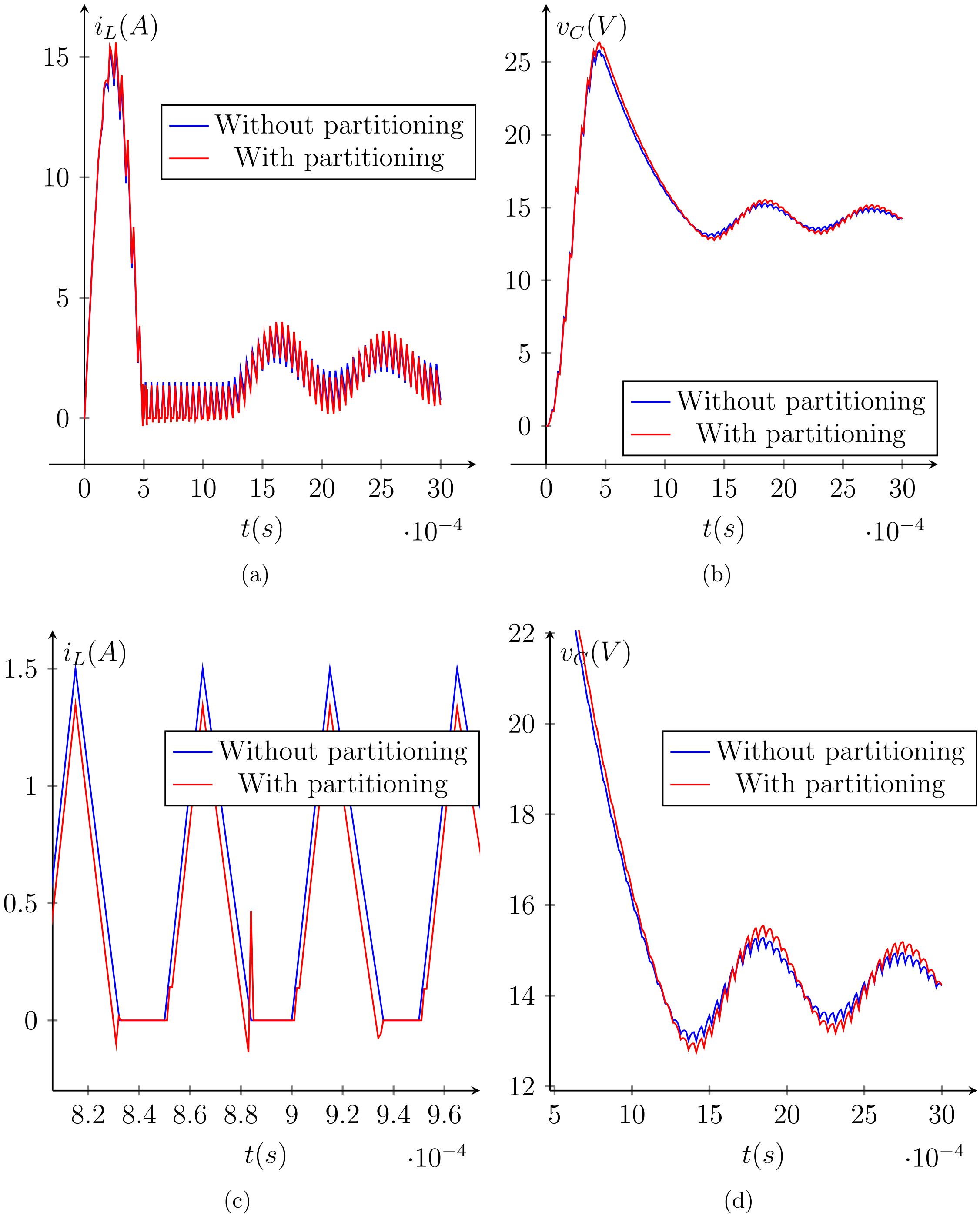


**Figure 4.32: Simulation of the current source inverter, with the parameters given by** (4.52)



(a)

(b)

(c)

(d)

**Figure 4.33: Effect of DCM on the signals**

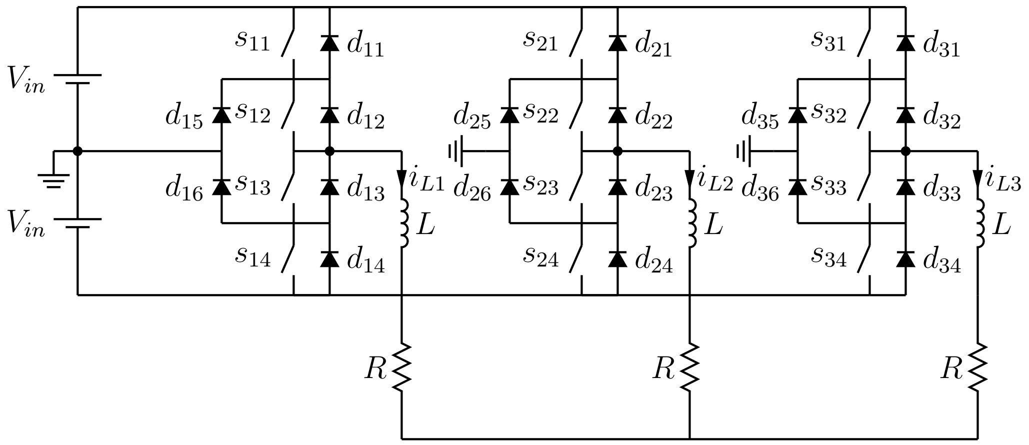


**Figure 4.34: 3-Phase Neutral Point Clamped (NPC) Inverter**

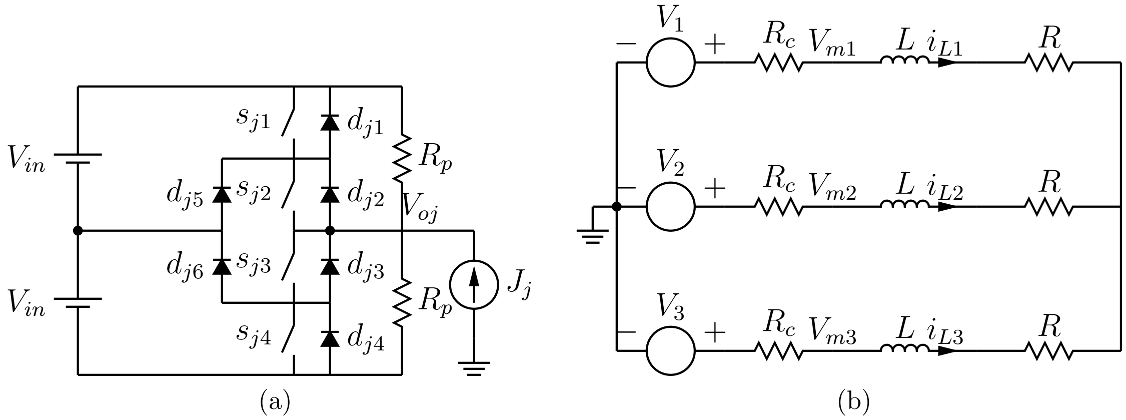


**Figure 4.35: Partitions of the three-phases NPC converter**

the large amount of switches leads to more than a billion topologies ($2^{30} \approx 10^9$) and extremely large graphs. Even though our algorithm could in theory study this circuit, this task would require an unaffordable amount of time (and computer memory). Thus, being able to properly partition and emulate the circuit is of capital importance. Fortunately, partitioning this circuit is very similar to some of the previous cases. We will study each of the three branches separately, and leave the inductor branches it its own partition, as shown in Figure 4.35. As usual, the polarizing resistors $R_p$ are added to the branches and the convergence resistors $R_c$ are added in series with the inductors to ensure the correct behavior when the output of a branch is disconnected from all the voltage sources.

This structure is very similar to the partitions of the three-phase inverter studied previously, and confirms that many converters may be split in similar ways. The law governing the voltage sources $V_1$ to $V_3$ is actually independent of the structure of the branches and is exactly the same as the one used for the full-bridge and the three-phase converter: if the output of the branch $j$ is connected to one of its input voltages, this law is

$$V_j(k) = V_{oj}(k-1) + R_c i_{Lj}(k-1) \tag{4.53}$$

which correctly cancels the voltage drop on $R_c$. If the branch is disconnected, we must force the current $i_{Lj}$ to converge towards zero, i.e.

$$V_j(k) = V_{oj}(k-1) \tag{4.54}$$

The equations governing the current sources $J_j$ are obtained using the same reasoning that lead us to write (4.24): we first write the Kirchhoff's current law at the output node of the branch:

$$J_j = i_{oj} + \frac{V_{in} - V_{oj}}{R_p} + \frac{-V_{in} - V_{oj}}{R_p} = i_{oj} - \frac{2V_{oj}}{R_p} \tag{4.55}$$

If the branch is active, we have $i_{oj} = i_{Lj}$, leading to

$$J_j(k) = i_{Lj}(k-1) - \frac{2V_{oj}(k-1)}{R_p} \tag{4.56}$$

in practice, $V_{oj}$ will be equal to $0V$, $V_{in}$ or $-V_{in}$ depending on the topology. If the branch is inactive, we have $i_{oj} = 0$. Furthermore, $V_{oj}$ is now measured on the interconnection partition, and is equal to $V_{mj}$. The current source is thus equal to

$$J_j(k) = -\frac{2V_{mj}(k-1)}{R_p} \tag{4.57}$$

Since each of the branch partitions only contains ten switches, they are easily studied using the traditional algorithm. To simulate the circuit, we first select the following parameters:

$$\begin{aligned} V_{in} &= 200V \\ L &= 1mH \\ R_L &= 1\Omega \end{aligned} \tag{4.58}$$

The output circuit has a time constant equal to $L/R = 1ms$. We control the switches of each bridges so that the currents are sinusoidal and phase shifted by $2\pi/3$ from each other. The frequency of the sines is 500Hz, while the frequency of the PWM is equal to 10kHz. The control signals are obtained by comparing a sinusoidal reference signal to a sawtooth signal at the PWM frequency (Figure 4.36). Depending on the comparison, we either switch

- The two upper transistors to force $V_{in}$ at the output
- The two lower transistors to force $-V_{in}$ at the output
- The two central transistors, which forces $0V$ at the output as soon as one of the two clamping diodes is switched on

With the chosen parameters, the circuit stays in CCM. The plots of Figure 4.37 represent the simulation of the circuit with the three inductor currents, along with the plots obtained from a continuous-time simulation. The simulation is very close to the theoretical value, which is an encouraging result since, as a reminder, this circuit could not be simulated in one piece.

With these results in mind, let us look at the performance of the automated parsing procedures. In theory, each of the branches requires 13 conditions for the natural switching

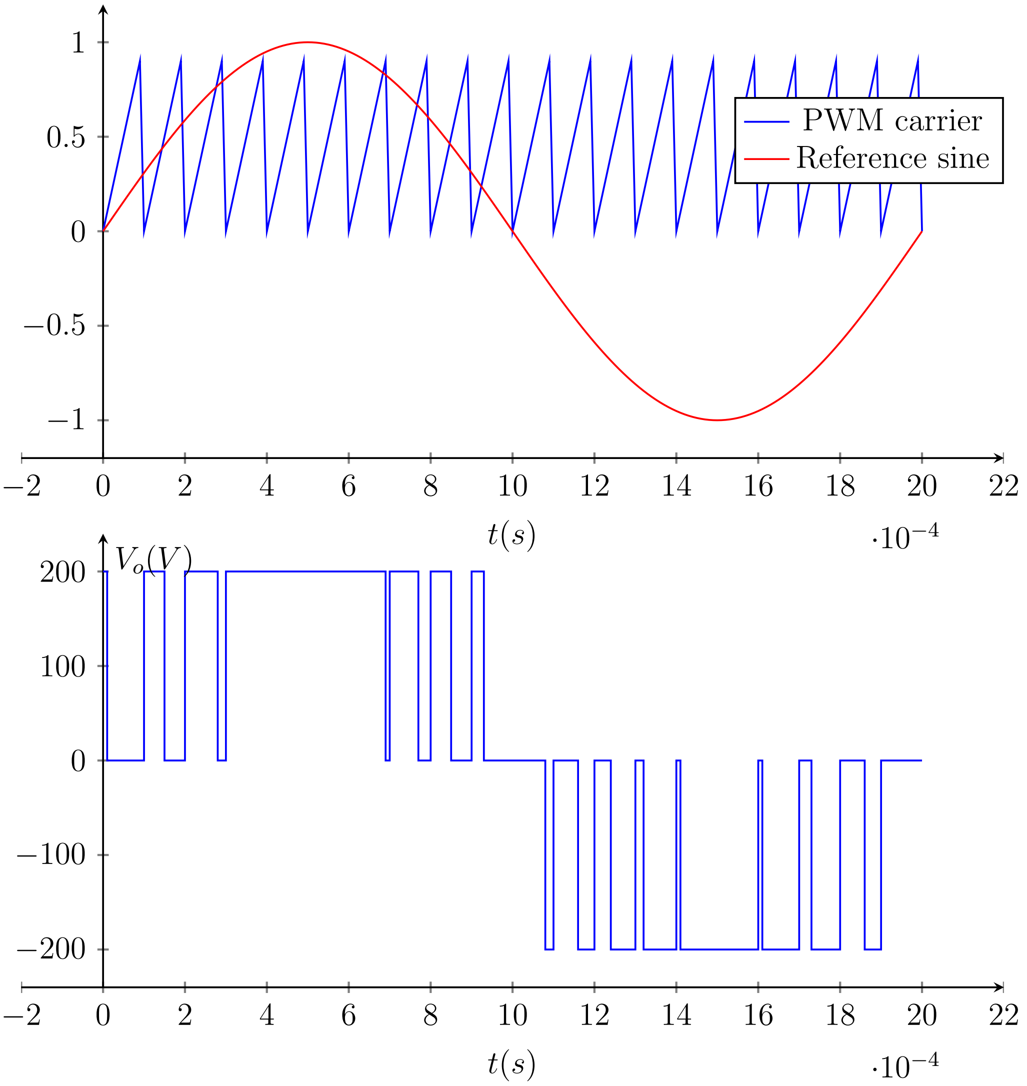
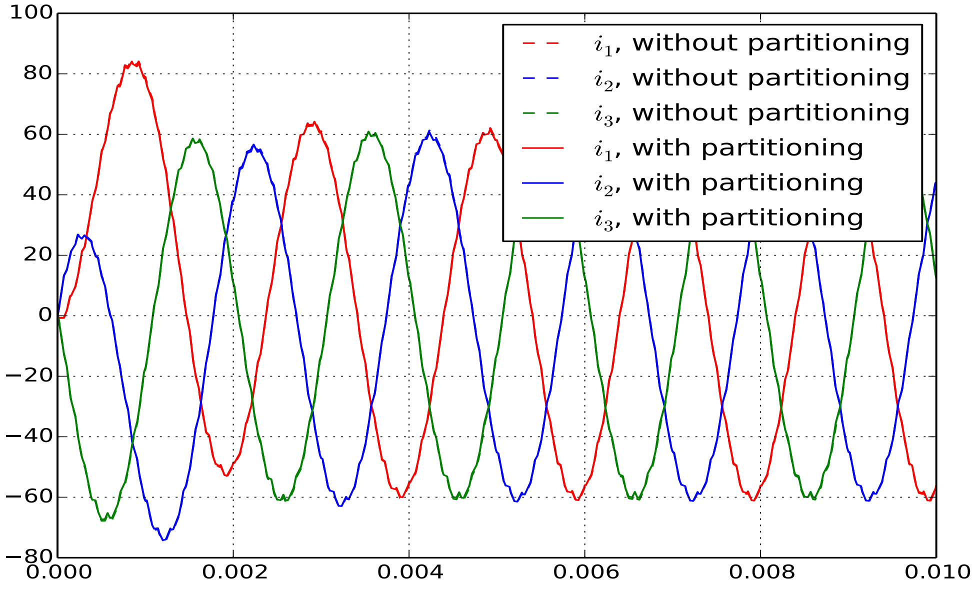
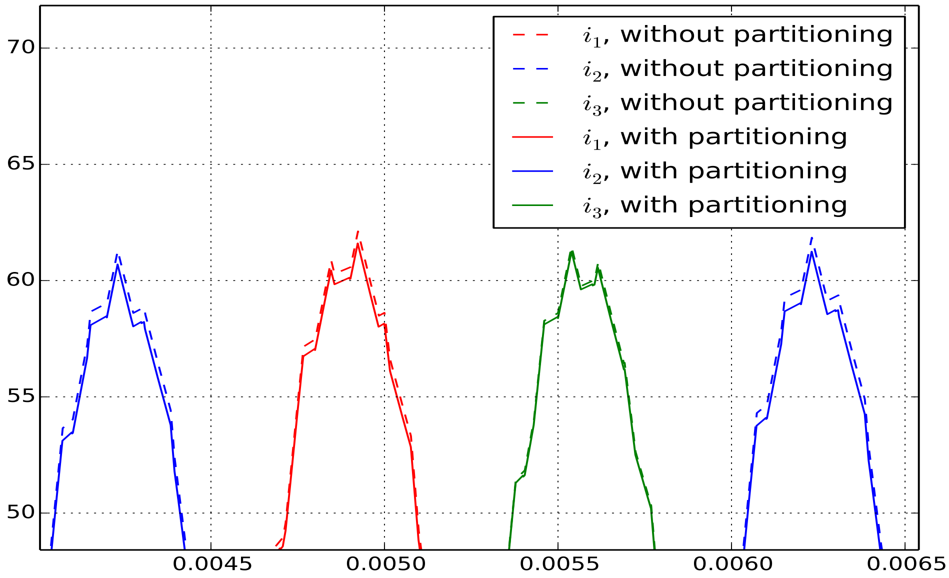


**Figure 4.36: Sinusoidal PWM modulation, based on a sawtooth PWM carrier**



(a)



(b)

**Figure 4.37: Simulation of the three-phase NPC, with the parameters given by** (4.58) **and using a 10kHz sawtooth PWM (b) zoom to show the differences between the continuous-time plot and the simulation**

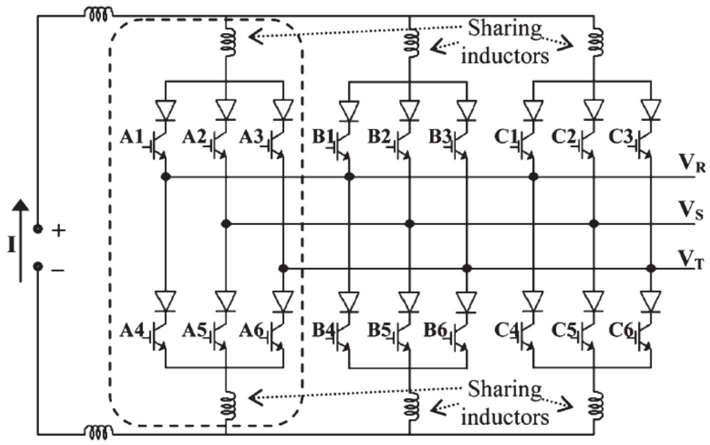


**Figure 4.38: Typical multilevel current source inverter (taken from [88])**

module and 5 conditions for the forced switching module. This high amount is mainly due to topologies that are reachable in theory, but are completely uninteresting in practice since they correspond to negative values for the input voltages. If we further specify that these voltages must be positive, the amount of conditions drops to 5 and 3 respectively, for a grand total of 15 conditions for the natural switching module of the complete circuit, and 9 conditions for the forced switching module.

### 4.4.6 Other circuits

The studies carried in the last few sections should allow us to properly simulate a large range of power circuits based on the same principles. One of the key conclusions is that the splitting algorithm tends to work better if the partitions of the converter follow a *one current-many voltages* paradigm, meaning that the converter is fed by a single current source, while multiple voltage sources can be connected. This is especially true if the current corresponds to an inductor current that can enter DCM, since these requires particular attention to ensure their convergence, and connecting the current to many partitions lead to additional resistors in the circuit.

For voltage source inverters, such as the full bridge, the three-phase bridge or the NPC, this paradigm corresponds to split the circuit by isolating each of its phases (branches). This partition is also the most natural one, because the converter is effectively composed of multiple identical converters placed in parallel. Since the complexity of the branches typically depends on the amount of different output voltages (as illustrated by the NPC converter), a circuit with more levels will lead to larger partitions. On the other hand, the number of phases in the circuit has no impact on the complexity of the branches. The opposite is true for current source inverters, as it is better to connect the current to each of its output phases (represented by a voltage source) to easily detect discontinuous conduction. This means that the number of partitions will grow if we add more phases to the circuit. Meanwhile, a multiple level current source inverter is generally made of multiple identical bridges connected to the outputs voltages [88,89]. A typical multilevel current source is represented on Figure 4.38. The current injections to the same node are effectively decoupled thanks to the sharing inductors. Note that each of the branches (contained inside the dotted line on the figure) can again be split in two partitions like we did during the study of the single phase current source inverter.

## 4.5 Perspectives and conclusions

The simulations carried out in this section have shown that the introduced partitioning method is a viable technique to separate a large circuit into multiple sub-modules which are studied separately before being recombined in a later phase. If the circuit does not operate in discontinuous conduction mode (DCM), or if the inductors that operate in DCM are completely contained in a single partition, the partitioning procedure is extremely simple and provides sufficient accuracy for the control application for which it is designed. Some circuits containing many switches are very difficult to analyze due to their intrinsic exponential complexity, and breaking them in multiple partitions greatly simplifies the analysis.

We have also developed a more advanced partitioning method in section 4.2, which is used when DCM involves several partitions. This method is based on controlled current and voltage sources whose control laws depend on the topology of the partitions. Additional resistors must be placed in the circuit to ensure that the currents will converge to zero as quickly as possible, and the value of these resistors must be carefully selected to avoid long transients or unwanted oscillations. We have shown through multiples examples that this method leads to acceptable results as long as the time constants and switching periods are noticeably larger than the time-step of the solver. It should be noted however that each source depends only on the topology of a single partition, avoiding the combinatorial explosion that lead us to develop this partitioning methodology. At this point, no other alternative has been found in the literature, and the usefulness of this method will still improve in the future.

Another advantage of the partitioning methodology is that it allows to define a library of circuits in the form of pre-compiled partitions. Since many converters are based on the same core components, the future designs are simplified because we only need to redefine a part of the circuit[2] instead of the whole converter.

Through the examples, we have tried to show the limitations of this system and to infer its validity domain. One of the most critical point is that signals that are coupled through fast dynamics (compared to the time-step) should not be separated, the reason being that replacing a partition by a source is roughly equivalent to using the less accurate forward Euler approximation, and splitting the circuit can lead to a divergence between the emulated signals and their real value. Of course, this effect is highly dependent on the time-step, and is reduced by using modern computing platforms able to refresh the signals at a higher rate. It also means that a circuit that cannot be properly partitioned today could well be used in a few years when new, faster components are available.

The fact that this algorithm is not completely automated can also be seen as a limitation of the system in its current state. While we have defined guidelines helping the designer to split the circuit it the best possible way, the algorithm is not complete. Indeed we have not developed automated methods able to generate and recombine all the tables needed by the switching modules, and all simulations were made by entering manually the laws

[2] Typically, the only parts that must be redefined are the output circuits representing the application, and the input circuits representing the sources

corresponding to the interconnections between the partitions. This is particularly crucial for the sources whose control law varies, because it means that we have to generate more tables that modify the transfer laws as a function of the partial topologies. In order to propose a complete turn-key solution, these points will have to be solved as part of a future project.

# Chapter 5

# Real-Time platform development

## 5.1 Introduction

### 5.1.1 Goals and requirements

Before going into the development of the real-time platform, it is a good idea to restate our goals and to reintroduce the workflow of our toolchain. The steps required to go from an offline simulation performed on a computer using a tool like *Plecs*® or *Spice* to a fully real-time digital equivalent are represented on Figure 5.1. Some of these steps were the subject of the previous chapter.

More precisely, in chapter 2, we have compared the multiple ways a power circuit can be emulated. After reviewing how to simulate a linear circuit in section 2.2 and how to model ideal and non-ideal switches in section 2.3, we have selected a simulation procedure based on the state-space representation of the linear elements. The equations governing the state-space system are dependent on the configuration of the switches, which are represented either as an open-circuit or as a short circuit. In practice, the extraction of the state matrices necessary to perform any kind of advanced analysis is performed using *Matlab*®, taking circuits drawn in *Plecs*® as a starting point.

We have also introduced linear automata as a mathematical tool allowing us to conveniently represent the transition between the topologies of the power circuit. This allowed us to define a few high-level simulation procedures in section 2.6.

The procedures were refined in chapter 3 where we introduced multiple steps that significantly reduce the complexity of the system by studying all possible transitions between the topologies. Once this automated offline study is done, we obtain four sets of tables:

- two sets (one for the forced switching module, and one for the natural switching module) providing the list of conditions (diode currents or voltages) required to select the next topology, based on the current configuration of the switches
- two sets that connect the result of the evaluation to the correct topology

In its current state, the analysis algorithms are implemented in *Python*, and produce *.dat* binary files that will be used downstream.

Since the selection of the topologies happens in real-time, the tables must be implemented on the emulation platform in a way that is both memory- and time-efficient. Indeed, a

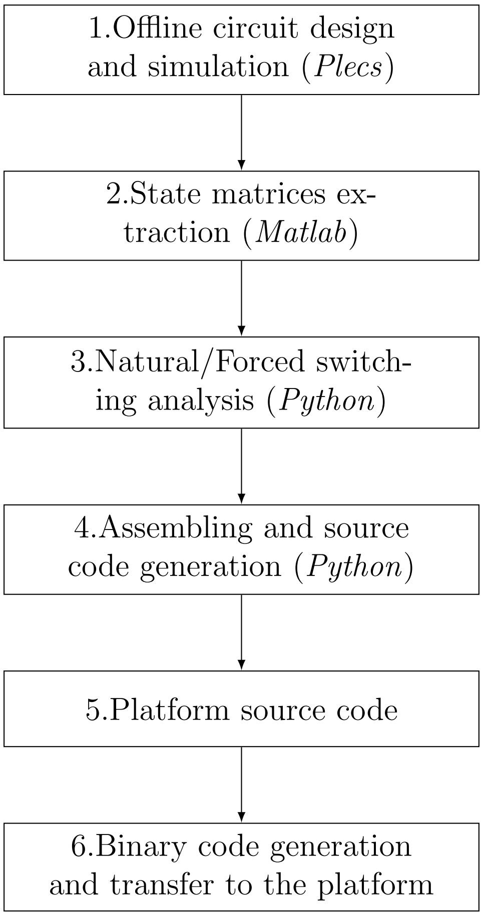

1.Offline circuit design and simulation (*Plecs*)
2.State matrices extraction (*Matlab*)
3.Natural/Forced switching analysis (*Python*)
4.Assembling and source code generation (*Python*)
5.Platform source code
6.Binary code generation and transfer to the platform

**Figure 5.1: The complete design flow of the rapid prototyping procedure. Steps 1 and 2 are described in chapter 2 while step 3 is described in chapters 3 and 4. The remaining steps are the subject of this chapter**

wrong representation of these wide selectors could lead to a large overhead in memory, which in turn would limit the speed of the platform.

The exponential growth of the tables lead us in section 4 to introduce a modified representation of the circuit in order to split the system in multiple sub-circuits, each one being simulated thanks to the previously defined procedure; the connections between the sub-circuits are realized in the nodal approach. While the platform must be adapted to take this change into account, the main steps are still the same: each of the sub-circuits defines its own tables leading to a set on conditions to evaluate, and the results from all parts are aggregated to obtain the global result. The most important change is that we have to introduce a feedback through the modification of the sources instead of the states: any topology change inside a sub-circuit has an incidence on the controlled sources of the other parts. The assembling of the partitions, i.e. the generation of the state-space equations and switching conditions corresponding to the complete circuit, is again done in *Python*.

This discussion allows us to define the main requirements for the real-time platform. First, the architecture must be scalable, in order to adapt itself to many circuits of varying size. This implies that the design of the different parts must be properly separated

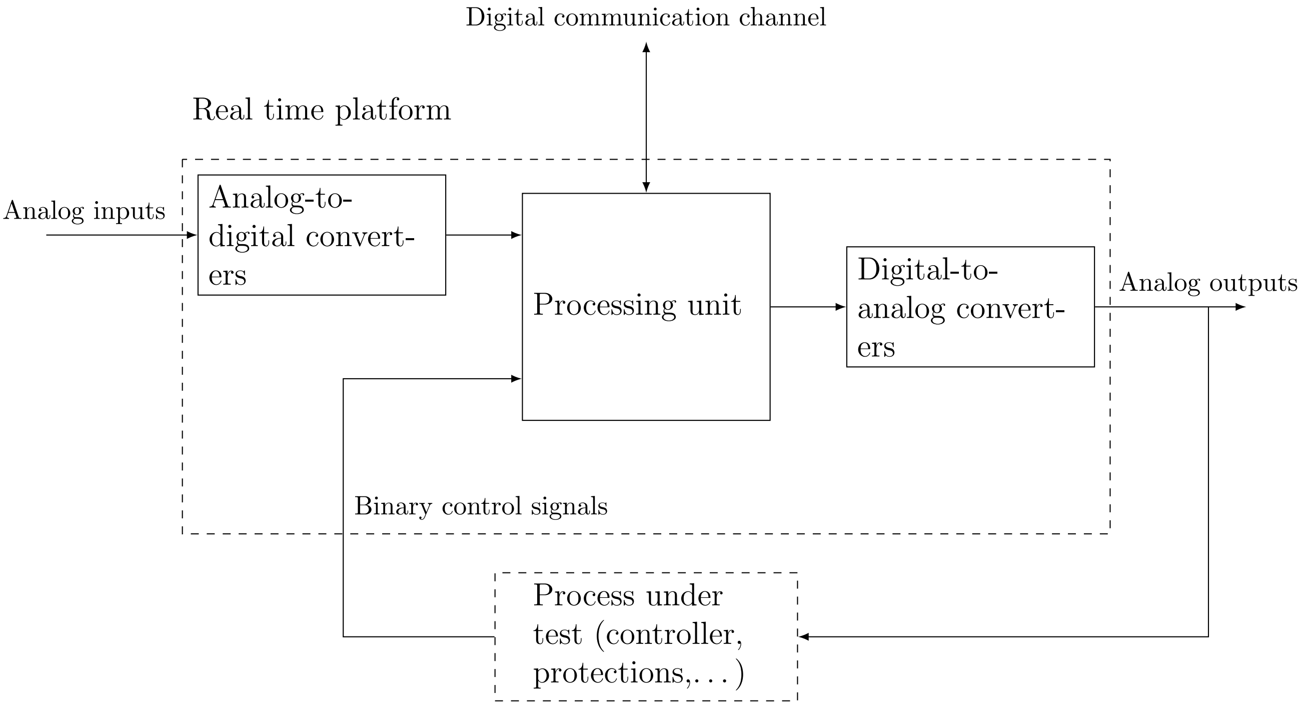

Digital communication channel
Real time platform
Analog inputs
Analog-to-digital converters
Processing unit
Digital-to-analog converters
Analog outputs
Binary control signals
Process under test (controller, protections,...)

**Figure 5.2: Physical structure of the real-time platform, composed of a main processing unit and data converters. The process under test typically uses the analog output signals as its own inputs and provides the binary control signals used to select the state of the active switches**

in sub-modules to enhance the parallel processing capabilities of the procedure. Thanks to the proper application of pipe-lining techniques, we expect to obtain a design whose speed is largely independent from the size of the emulated circuit.

Since many parts of the simulation algorithm hang on matrix products (for example, the evolution of the state variables and the conditions are based on a series of dot products), it makes sense to optimize this part of the algorithm to avoid overusing the resources of the platform, while keeping an high computation speed. Finally, in order for our platform to be as user friendly as possible, we must provide multiple modules to simplify the implementation of the emulator. This includes the automatic creation of the sources, interfacing the platform to the outside world and provide additional capabilities such as the representation of sensor defects and measurement noise.

This chapter is divided in multiple sections in which we shall describe the parts contained in our platform. In section 5.2, we compare the different emulation procedures in order to select the best choice for our application. The implementations of the low level modules (notably, the dot products and the selection tables) are studied and compared in section 5.3 before being used in the higher level modules such as the switching modules in section 5.4.

### 5.1.2 Structure of the platform

The physical structure of the real-time platform (RTP) is represented in Figure 5.2. The processing unit is the central element of the platform, and implements the emulator as well as all communication protocols allowing the emulator to interact with the outside world. Multiple inputs and outputs are provided to use the platform in its intended way (i.e. in closed-loop control). The signals controlling the active switches are the main inputs. Since these signals are intrinsically binary (the switches are either on or off), they are easily connected to the digital inputs of the processing unit using a single wire (assuming compatible logic levels). In closed loop applications, the control signals are provided by the process under test. For instance, the tested controller might use the feedback from the analog output signals coming from the emulated converter to generate the PWM control signals

To represent the measurements that would be taken on the real circuit, we have to translate the digital signals computed by the platform into a measurable voltage (or current) using a digital-to-analog converter (DAC). These devices are typically placed outside of the processing unit and are connected either through a parallel bus, or through a serial protocol such as SPI or I$^2$C if the sampling rate is less than a few mega samples per second. Analog-to-digital converters can also be added to provide analog inputs allowing to externally modify the value of the sources.

Additional communication channels may provide advanced capabilities not directly considered in this thesis. An example would be a high-speed channel allowing to modify the state matrices on-the-fly or to send the computed values of the signals to a computer without using an analog channel.

### 5.1.3 Selection of the processing unit

While computers and micro-controllers are often used in embedded control systems, they are not the best choice for our application. Indeed, these systems are intrinsically sequential, which means that the time needed to execute all the instructions included in a time step is greatly impacted by the size of the emulated circuit. For instance, multiplying a $m-by-n$ matrix by a $n-by-1$ vector requires the sequential computation of $m \times n$ partial products, leading to a long computational delay when the circuit grows in complexity. Hence, larger circuits have to be simulated with a larger time-step. Some parallelism can be obtained in multi-core processors, but computing tens of partial products in parallel remains difficult to achieve.

Field Programmable Gate Arrays (FPGAs) are another type of digital devices, which put an emphasis on massive parallel processing. Internally, an FPGA is composed of a high number of small cells (or slices) providing basic operations [44]:

- Logic slices are composed of a small (4bit or 6bit) RAM acting as a look-up table and allowing the implementation of basic logic functions, multiplexers and additional general purpose logic such as latching elements. These slices constitute what is known as the FPGA *fabric.*

- Multiplying slices implement high speed multipliers in high speed dedicated logic, providing better performance that a multiplier implemented in general purpose logic. Modern FPGAs provide DSP units instead able to perform a multiplication and/or an addition in the same slice

- Block RAMs are memories of a few kilobits able to contain small tables.

- High speed IO slices allow the use of modern communication protocols such as PCI .

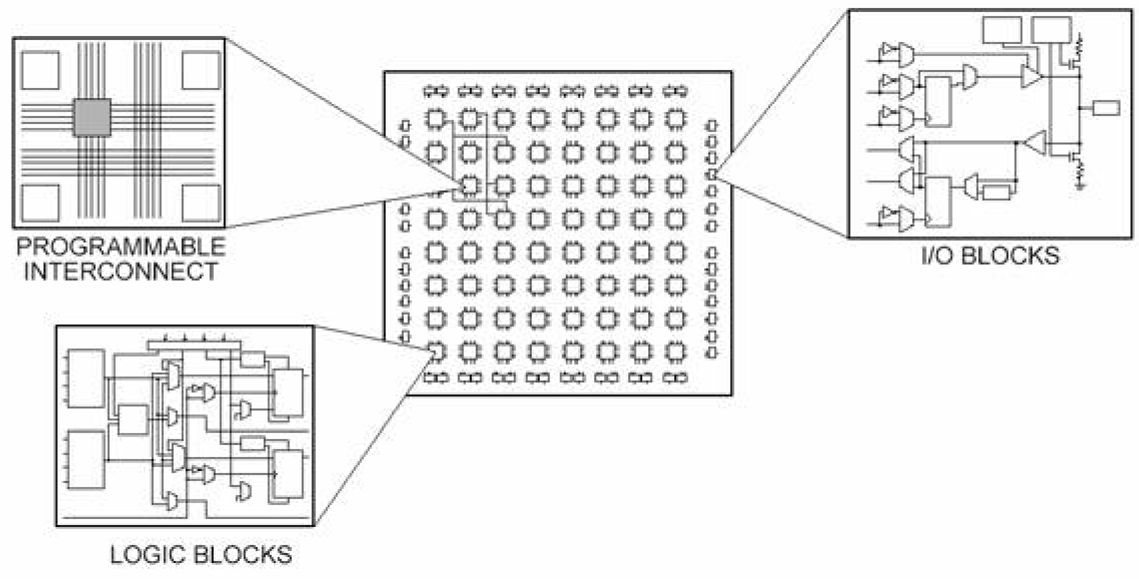


**Figure 5.3: Architecture of an FPGA (taken from [90])**

All these elements are interconnected thanks to a large configurable network of wires across the device, as represented on figure 5.3. Since the slices are mostly independent, we are allowed to use the functions of many slices at the same time, providing a very high degree of parallel processing. This means that we could, for example, compute all partial products of a matrix product simultaneously, while updating the DACs with the new value of the signals and reading the new value of the controlling signals at the same time. It also means that the emulation of a larger circuit will require a larger amount of resources, but will not necessarily need more time to perform the operations. Hence, we are provided with a more precise knowledge of the minimum time step, and this time step is mostly independent of the emulated converter. On large FPGAs containing many slices, we can compute hundreds of operations simultaneously, leading to high data rates that cannot be reached by most processors.

For the current version of the platform, we have chosen to use FPGAs manufactured by *Xilinx* ® because of our familiarity with their products. However, the basic concepts are the same for the devices built by other vendors, and their lineups are mostly equivalent.

One of the biggest challenges when designing an application for FPGAs is to ensure what is known as the *timing closure*. To explain this concept, we first define a *path* as a chain of logic operations that must be done during a single clock cycle. The path is bounded on both ends by memory elements that are updated at each clock cycle and acting as buffers allowing to spread complex operations over multiple clock cycles. The path will be within the timing parameters of the design if the data is guaranteed to reach the end of the path before the next clock cycle. This imposes a limit on the number of operations that can be performed in a single clock cycle, as each slice adds a propagation delay. Furthermore, the time needed to propagate the signals across the interconnection cannot be neglected, and may constitute the major part of the total delay when connecting slices

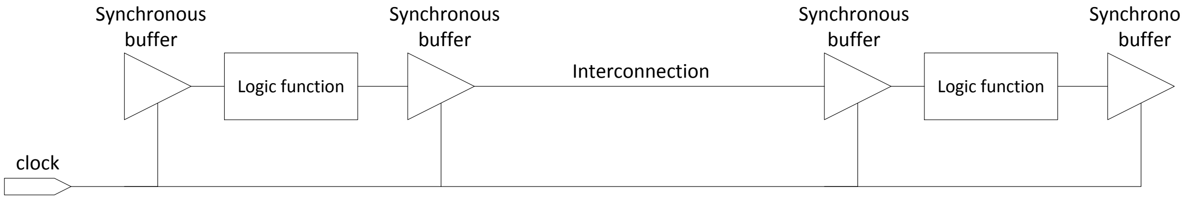


**Figure 5.4: The double buffering isolates the interconnections from the modules**

placed far apart on the chip.

The maximum allowed clock frequency is limited by the *critical path*, which is the longest sequence of operations that must be performed during a single clock cycle. The critical path is reduced by adding intermediary buffers in the design, effectively spreading (or pipe-lining) the operations on several consecutive clock cycles.

### The FPGA workflow

A FPGA is not *programmed* like a processor, we do not load a memory with a list of instructions that are executed by a central unit. Instead, we configure the function of each slice as well as the interconnections between the slices during the compilation. However, we can still draw parallels between the two workflows transforming a high-level description of the application into an executable file.

In both cases, a first step converts a file written in a high-level language (C/JAVA/Python... for a computer, VHDL/Verilog for a FPGA) into a low-level description that corresponds to the chosen platform. For a computer, this step is performed by the *compiler*, while the *synthesizer* converts an application written in VHDL or Verilog into low-level logic functions (multiplexers, registers, multipliers,...). These functions are then *mapped* to the basic functions provided by the slices of the specific FPGA target.

The different parts of the application are then connected to provide the complete design. This is known as the *link* step when compiling a program for a computer. For a FPGA design, this step is called *place and route*, and consists in physically selecting which slices will be used for the different modules of the design, and in configuring the interconnection network to provide the correct electrical path.

## 5.1.4 Challenges

From a technological standpoint, the difficulties in designing the platform arise from the need to ensure timing closure. Hence, each module must be tested and validated separately against the timing requirements. However, this does not guarantee that the module will still meet the closure when used as part of the complete system. Indeed, the synthetiser may decide to place two successive modules far from each other during the *place and route* step, and the resulting additional propagation delay may cause the design to fail. This effect is reduced by placing buffers at the beginning and at the end of each major module. With this double buffering, the communication between the modules is separated from the logic operations, and is allowed to take up to a full clock cycle to transmit the signals (Figure 5.4).

Reducing the global utilization of logic cells is another goal of this design, because a reduced logic circuit will more easily meet the timing requirements and can be implemented on a smaller FPGA. The reduction of the automaton representing the circuit was developed in chapter 3: by limiting the number of conditions we need to evaluate, we kept the resource usage to a minimum. Of course, additional optimizations can be made at the implementation level to avoid using too many unnecessary slices. These reductions will be introduced in this chapter.

Since we do not expect the user of the platform to be an expert in FPGA design, we have to design the platform to be as user-friendly as possible. This means that the platform must be internally universal, and should be configured thanks to configuration files that depend on the FPGA and on the emulated circuit. An optimal design flow should require as few interventions as possible from the user between the initial design of the circuit in *Simulink* and the implementation on the FPGA, which means that me must automatically convert the tables established during the offline analysis to a format understood by the design.

Finally, we have to remember that the main goal of this thesis is to emulate circuits with fast transients and high switching frequencies. As an immediate implication, all modules must be optimized to take as few clock cycles as possible while allowing a high clock rate. This means that all computations must be kept to a minimum, and that we must avoid developing pipelines that are unnecessary long. We must also ensure that the behavior of the design in closed loop (i.e., when the output signals are used by an external apparatus such as a controller to modify the value of the input signals) does not lead to unrealistic signals [1].

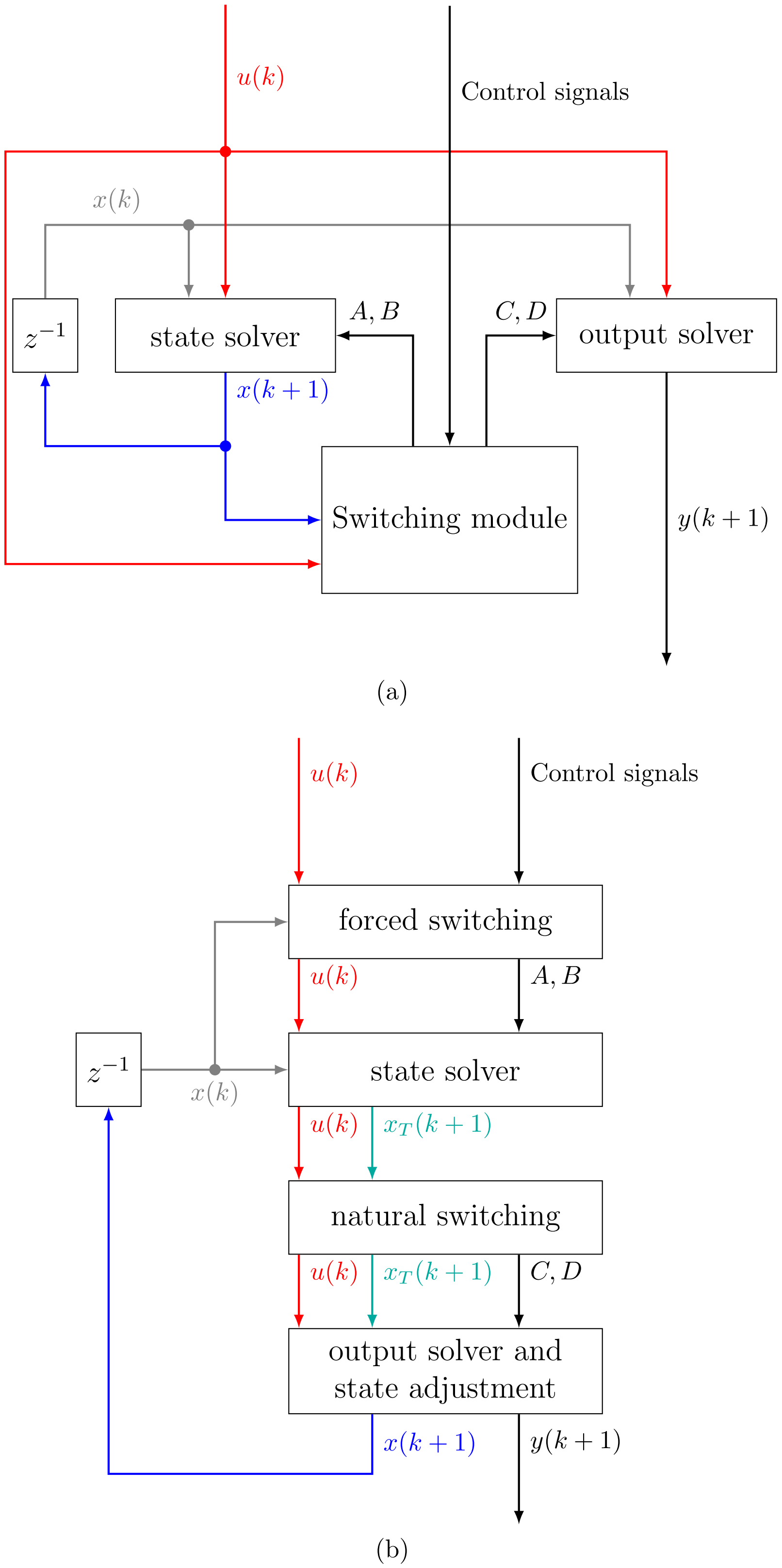
## 5.2 Emulation procedure

### 5.2.1 Structure comparison

We already introduced multiple high-level procedures for the real-time platform in section 2.6, and only the main results will be repeated here. Each of the possibilities have the same base properties, which are the implementation of the state solver, the output solver and the switching modules. The two main possibilities are drawn in Figure 5.5. The first structure, represented in Figure 5.5a, shows a split solver with the parallel computation of the states $x(k+1)$ and of the outputs $y(k+1)$. This method allows for a high degree of parallelism, since more signals may be computed at the same time. In turn, this reduces the total latency of the solver. The main disadvantage of this structure is that the outputs are computed using the states and inputs of the previous cycle (i.e. $y(k) = f(x(k-1), u(k-1))$, which introduces a delay equal to a single time-step and may degrade the performance of the controls if the time constants are not significantly larger that the fundamental time step of the solver due to the implied reduction of the phase margin.

In contrast, the interleaved structure of Figure 5.5b allows to update the outputs as soon as the new value of the states is known ($y(k) = f(x(k), u(k))$). Furthermore, inserting the natural switching module between the state solver and the output solver adds the possibility of adjusting the value of a state if it becomes forced after a topology change.

[1] Since we are using a discrete solver, some distortion in unavoidable due to the lags in the signals.



(a)

(b)

**Figure 5.5: The two main structures (a) Split structure with parallel computation of the states and outputs (b) Interleaved structure**

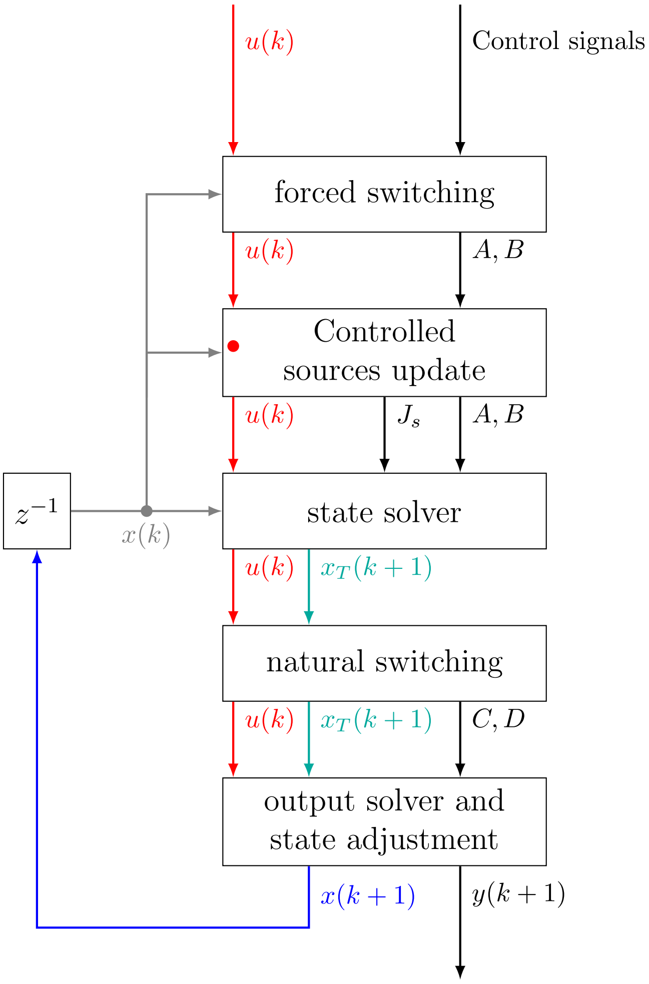


**Figure 5.6: Solver with interleaved structure, modified to allow the use of the partitioning thanks to the inclusion of the *controlled sources update* module**

Since the modules are computed in a serial manner, the total latency of the system is larger. Hence, if we succeed to keep the same time step for both structures, the interleaved variant will provide the outputs sooner than the parallel structure. As the circuits we hope to simulate can include state variables with very short time constants (notably, the discharge of a decoupling capacitor across a leg of a converter could be described by a time-constant of around 1 microsecond or less, as was shown in section 3.5.3), we selected the interleaved structure for our platform. If we use circuit partitioning, we have to take the adjustment of the controlled sources into account. As shown in chapter 4, the sources are computed before the states, leading to the structure of Figure 5.6. This method lengthens the simulation procedure, as more computations must be performed in series. Unfortunately, this structure is also required to avoid a lag between a change in a state variable and its effect on the other partitions, and to keep a high fidelity in the emulated signals. This is even more true if the partitioning leads to the separation of an inductor from a partition that can force it to enter discontinuous-conduction mode, as these circuits are more sensitive to the lag (this effect was shown in section 4.2). Again, this effect is less pronounced when the sampling rate increases.

# 5.3 Fundamental modules

## 5.3.1 Data representation

In the previous chapters, all computations have been performed in Matlab® and Python, which use by default double precision floating-point numbers. These floating-point numbers are composed of:

- a sign bit
- a mantissa, containing the most significant digits of the number (23 bits for the IEEE-754 in 32-bit format, 52 bits for the 64-bit representation)
- an exponent, or scaling factor, which multiplies the mantissa by $2^{\text{exponent}}$ (8 bits for the 32-bit representation, 11 bits for the 64-bit representation)

The main advantage of floating-point numbers is their very wide dynamic range, indicating that we can represent very small numbers and very large numbers with the same precision. If we take the IEEE-754 representation as an example, the full 23 bits of the mantissa (equivalent to about seven decimal digits) are available for the complete range of exponents (-127 to 127).

Computations using floating-point numbers tend to take a lot of clock cycles. For example, computing the product of two 32-bit floating-point numbers require the multiplication of the two mantissas, the normalization of the exponent and the insertion of the sign bit in sequence and may take up to 14 clock cycles for a signed product [91]. Floating-point adders require even more steps, and can require more time that the floating-point product [92]. Integrated floating-point units are typically not present in FPGAs, and these operations must be partially performed in the slower general purpose slices of the device. FPGAs with integrated high speed floating-point DSP slices appeared on the market recently [93], but were not available when our platform was developed. When these devices become more common, switching the platform to floating-point calculations could provide a more customizable design able to emulate a wider range of signals.

During this thesis, we had to choose a *fixed-point representation* for our numbers. With this paradigm, all numbers are integers scaled by an implicit factor. As an example, let us take the $Q15$ fixed-point representation, meaning that all numbers are multiplied by $2^{15}$ before being encoded. We can write the following;

- 1 is encoded as $1 \times 2^{15} = 32768$
- 123 is encoded as 4030464
- 0.05 is encoded as 1638 (rounded down from $0.05 \times 2^{15} = 1638.4$)

Hence, numbers with a fractional part may be represented using an integer with a precision of $2^{-15}$. This comes at the expense of the maximum value that can be encoded with this representation. For example, the maximum value of a signed 32-bit Q15 number is $(2^{31} - 1)/2^{15} \approx 65536$ and corresponds to the binary number $0111\ldots1$ (the left-most bit is reserved for the sign). In conclusion, the number is composed of:

- a sign bit

- 16 bits for the integer part
- 15 bits for the fractional part

The biggest advantage of this representation lies in its computational efficiency. Since all numbers are integers, we can use the dedicated integer DSP units available in the FPGA to their full power. The only additional step required by this method is to properly rescale the values after applying any arithmetical operation. Let us assume two real numbers $a$ and $b$, encoded in the fixed-point numbers $A$ and $B$ with their respective representation $Qm$ and $Qn$. We have

$$\begin{aligned} A &= a \times 2^m \\ B &= b \times 2^n \end{aligned} \tag{5.1}$$

If we want the compute the product of $a$ and $b$, and encode the result in $X$, defined as a $Qk$ fixed-point number, we first perform the integer multiplication

$$AB = ab \times 2^{m+n} \tag{5.2}$$

The final representation of $X$ is obtained by scaling the output

$$X = AB \times 2^{k-(m+n)} \tag{5.3}$$

Multiplying (or dividing) an integer in two's-complement representation by a power of two is equivalent to performing a binary shift. This operation is very fast when the amount of bits by which we must shift the signals is constant, and takes at most one clock cycle [2].

We now have to select the exact representation that will be used in our platform. A first immediate idea is to use numbers of a traditional length such as 16, 32 or 64 bits; these numbers are indeed heavily used on computers and micro-controllers. However, using these representation do not allow us to use the full precision of the integrated hardware multipliers of the FPGA, which are generally 18 × 18 bits [3]. Hence, we studied the use of the 18-bit and 36-bit representation.

To select the final representation, we must first find the needed range and precision. Let us take a look at the following first-order dynamical system:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= ax(t) + bu(t) \\ y(t) &= cx(t) \end{aligned} \tag{5.4}$$

For the sake of simplicity, we assume that this system is sampled and solved using the Forward Euler Method and a time step $T$, leading to

$$\begin{aligned} x(k+1) &= (1 + aT)x(k) + bTu(k) \\ y(k) &= cx(k) \end{aligned} \tag{5.5}$$

[2] Shifting a number by a variable amount of bits requires a barrel shifter, which is a more complex device that is also much slower (i.e. it either requires more clock cycles, or a single much longer clock cycle). This is one of the reasons why floating point calculations take so much time

[3] The origin of the 18 bits representation lies in the fact FPGA memories are typically 9-bit wide in order to provide a parity bit along with the 8 bit data. If we do not use the parity bit, we can freely use the ninth bit as a basis for our numbers, leading to 9- , 18- and 36-bit representations

the values $(1+aT)$, $bT$ and $c$ must be encoded in the system as 18- or 36-bit integers. The $aT$ term can be rewritten as $\frac{T}{\tau}$; $\tau = 1/a$ is the time constant of the system. As explained in section 2.2, it makes sense to use a time step smaller than the time-constant. For slow systems, the time-constant can be as high as one second or even more. For a short time-step equal to $1\mu s$, $aT$ is equal to $10^{-6}$. As a result, the encoding of $1 + aT$ requires at least 21 bits [4], which excludes the use of the 18 bits representation.
Hence, we have selected 36-bit integers. Meanwhile, we chose to allot 18 bits to the fractional part, two fewer bits than the 20 bits computed previously. This format allows us to represent numbers ranging from $-2^{18}$ to $2^{18}$ (i.e. from $-262144$ to $262144$) with a resolution of $2^{-18}$. This symmetrical choice allows us to represent both $x$ and $1/x$ at the same time for $x \leq 2^{18}$, which is useful for circuits that include resistances (i.e. the resistance $R$ will be at the numerator for current-to-voltage relations and at the denominator for voltage-to-current relations). Note however that changing the format only requires to recompute the matrices and the scaling factor.
In practice, 35-bit numbers are used. The reason for this is that the product of two 18-bit signed numbers (containing 17 significant bits and the sign bit) is a number composed of a sign bit and $2 \times 17 = 34$ number bits, hence a 35-bits number.

### 5.3.2 Dot products

The dot products are present in multiple places in our algorithm. Indeed, we have the state equations that use any of those forms depending on chosen solver (as described in section 2.2.3)

$$\begin{aligned}
x_T(k) &= (I - TA)^{-1}(x(k-1) + TBu(k)) && \text{Backward Euler approximation}\\
x_T(k) &= (I - \frac{T}{2}A)^{-1}((I + \frac{T}{2}A)x(k-1) + \frac{T}{2}Bu(k)) && \text{Trapezoidal approximation}\\
x^*(k) &= x(k-1) + T(Ax(k-1) + Bu(k)) && \\
x_T(k) &= x(k-1) + TA\frac{x(k-1) + x^*(k)}{2} + TBu(k) && \text{Improved Euler approximation}
\end{aligned} \tag{5.6}$$

The output equation, augmented by the modified value of the states (see section 2.6), is defined by

$$\begin{bmatrix} y(k) \\ x(k) \end{bmatrix} = \begin{bmatrix} C \\ C_x \end{bmatrix} x_T(k) + \begin{bmatrix} D \\ D_x \end{bmatrix} u(k) \tag{5.7}$$

The switching modules are based on the computation of a set of conditions that depend on the current topology. These conditions are of the form

$$y_D = Ex(k) + Fu(k) \tag{5.8}$$

where $y_D$ either represents a diode current or a diode voltage. Finally, the value of the controlled current sources that appear when using circuit decoupling is updated using

$$J_s = C_J x(k) + D_J u(k) \tag{5.9}$$

[4] To encode any number ranging from 0 to 1, with a precision of $10^{-6}$, we must have at least $10^6$ different values at our disposal. The minimal $n$ such that $2^n \geq 10^6$ is 20. An additional bit is required to encode the sign

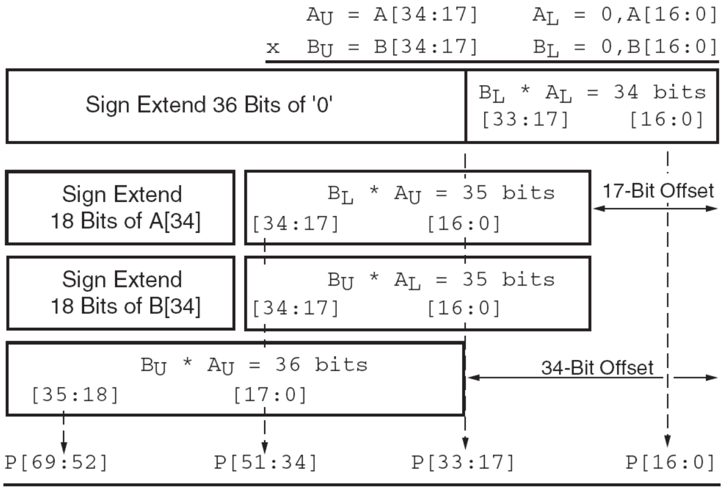


**Figure 5.7: Steps of a 35-by-35 product using a 18-by-18 multiplier (taken from [94])**

Each of these matrix products is trivially split into a series of dot products of the form

$$z_i = \sum_j a_{ij} x_j + \sum_j b_{ij} u_j \tag{5.10}$$

The calculation of the dot products involves computing the partial products $a_{ij}x_j$ and $b_{ij}u_j$ before summing the results. The implementation depends on the exact goal pursued (size vs latency vs throughput), as we shall show now.

### Computing the partial products

Multiplying two signals seems straightforward at first glance, but the implementation still requires a bit of thought. Indeed, since we have chosen to represent the signals as 35-bit variables, we cannot simply use an internal 18-by-18 multiplier. We can, however, find a succession of operations that allow us to perform the 35-by-35 product. To better describe our possibilities, we shall assume that we want to compute the product of the variables $A$ and $B$, each 35-bit wide, and write the result in $P$, which is a 70-bit variable. We will furthermore assume that the input variables are decomposed into $A = \begin{bmatrix} A_U & A_L \end{bmatrix}$ and $B = \begin{bmatrix} B_U & B_L \end{bmatrix}$ where $A_L, B_L$ are the 18 least significant bits while $A_U, B_U$ are the 17 most significant bits. The product is then decomposed into four partial products, which are then shifted and summed as shown in Figure 5.7. We will now review different implementations of this product, and select the most appropriate for our application. The canonical implementation is to use four DSP units connected to perform the products, the shifting and the summation in dedicated high-speed logic, as shown in Figure 5.8 (where the $z^{-1}$ block is a single clock delay). This version has the advantage of being fully pipe-lined, which means that a new product is computed at each clock cycle (albeit with a latency of four clock cycles). The biggest drawback of the pipe-lined implementation is that it requires four DSP slices per 70-bit product. In older FPGAs like the *Spartan 6* series, the shift had to be done in fabric (generic logic), which leads to a longer data path and hence a lower maximal clock rate [96] (this effect can be counteracted by using a longer pipe-line, which allows a higher frequency in exchange of a longer latency). A first single-slice variant is shown in Figure 5.9. Here, the same slice is used four

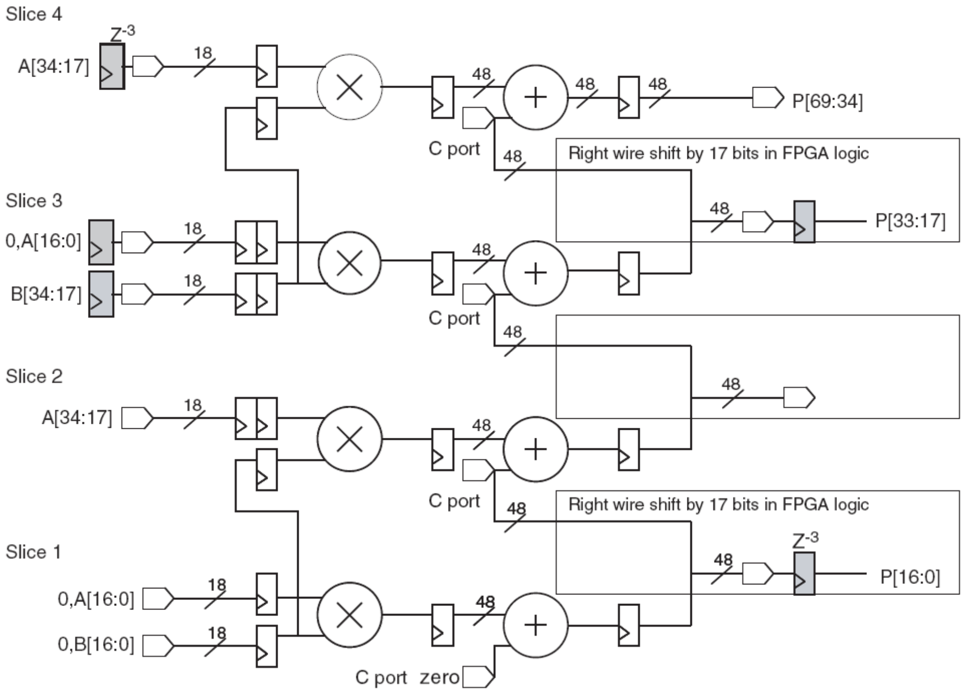


**Figure 5.8: Pipe-lined implementation of a 35-by-35 product using four 18-by-18 multipliers (taken from [94]). At each stage, a DSP slice includes the logic needed to perform the multiplication, the addition and the internal buffering. Additional buffers are added in front of the slices and at the output in order to synchronize the data paths (i.e. the inputs of the second stage must appear one clock later than the inputs of the first state**

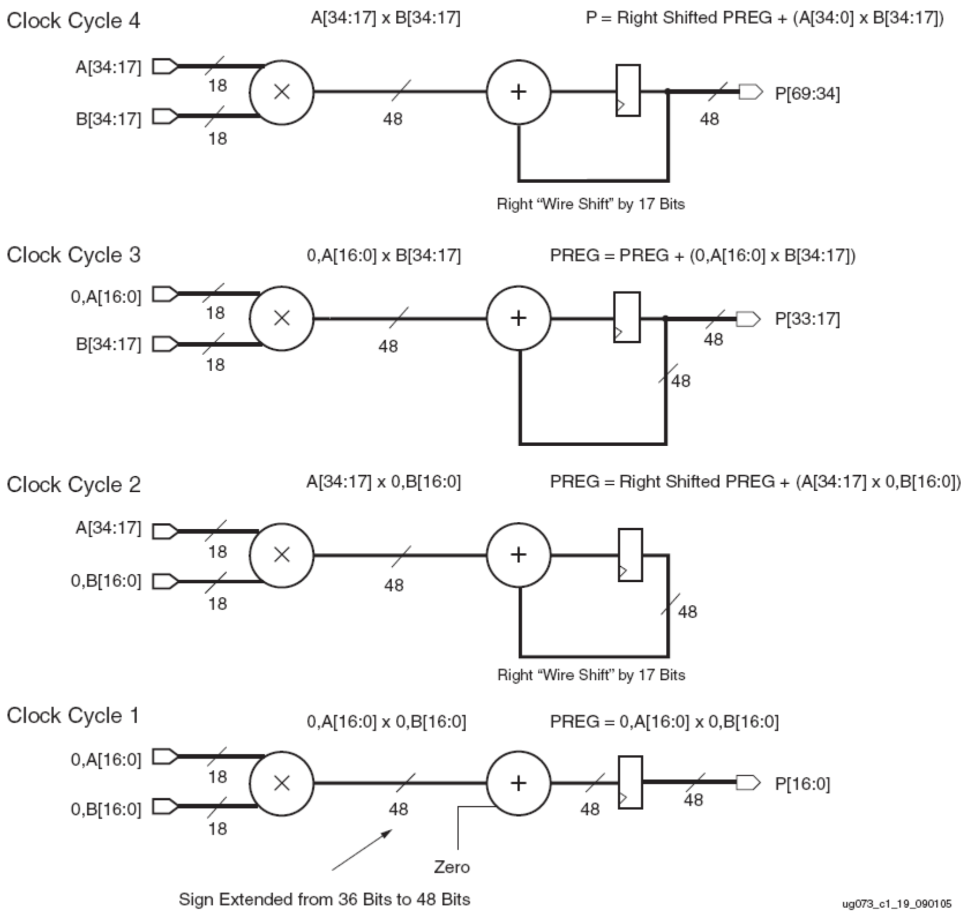


**Figure 5.9: Implementation of a 35-by-35 product using the 18-by-18 multiplier and the adder of a single DSP slice (taken from [95]): the four partial products are computed in succession, and are accumulated using the adder**

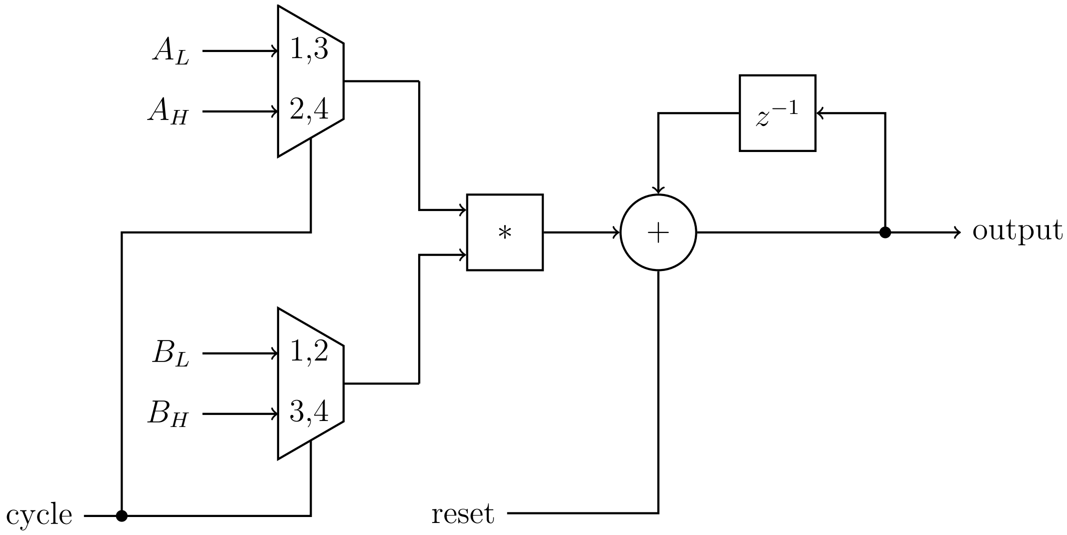


**Figure 5.10: Implementation of a 35-by-35 product using the 18-by-18 multiplier and an accumulator**

times with different inputs, successively implementing the four slices of the pipe-lined version [95]. While this could work for lower clock frequencies, this design is limited by the feedback from the output to the adder input. Indeed, there are two ways to implement the selective shifting required by the product. First, the shift may be realized in fabric, limiting the speed for the reasons described previously. On newer platforms, the slices can be configured to shift the adder input before performing the addition, which provides a more compact solution [5]. While this solution uses less logic, the slices are not built to support high frequency mode changes, limiting the clock frequency to about 200 MHz [44]. To eliminate this problem, we suggest a modified implementation, represented in Figure 5.10. Here, the DSP slice is used as a basic 18-by-18 multiplier, outputting in sequence each of the partial products, which are then summed using an accumulator placed downstream. The accumulator is controlled thanks to a multiplexer, allowing to selectively shift the signal or reset the accumulator before performing the product. Since the shifter is placed in series with the multiplier, it adds a single clock cycle to the operation. The steps required by this design are described on table 5.1. This pipe-lining allows the design to perform at higher clock frequencies.

To further compare the methods, we implemented the algorithms on two devices (including a second variant of the fully pipe-lined with added buffer stages, leading to a deeper pipe-lining) and wrote their performances based on the maximal clock rate and the amount of LUT (look-up tables, which are the base logical units of the FPGA). The first device is a Spartan 6 FPGA, which is a low-cost family typically used is smaller boards and is part of the 2008 lineup of devices. The second device tested is part of the Artix 7 family of FPGAs (sold since 2011) which are much more powerful while being the smallest and the cheapest of the new Xilinx chips. The results are written in Table 5.2, with the following remarks:

- To ensure proper decoupling with the rest of the design, buffers (D flip-flops) were

[5] These slices implement many more modes, including the possibility to perform a subtraction instead of an addition, carrying a logical operation (OR,AND), building a three-way adder,...

| Cycle | Multiplier | | Adder | |
|---|---|---|---|---|
| | Input | Output | Input $u(k)$ | Output $y(k)$ |
| 1 | $A_L$,$B_L$ | $A_L \times B_L$ | 0 | $y(1) = 0$ |
| 2 | $A_H$,$B_L$ | $A_H \times B_L$ | 0 | $y(2) = y(1) + u(2) = 0$ |
| 3 | $A_L$,$B_H$ | $A_L \times B_H$ | $A_L \times B_L$ | $y(3) = y(2) + u(3)$<br>$= A_L \times B_L$ |
| 4 | $A_H$,$B_H$ | $A_H \times B_H$ | $(A_H \times B_L)$<<17 | $y(4) = y(3) + u(4)$<br>$= A_L \times B_L + (A_H \times B_L)$<<17 |
| 5 | 0,0 | 0 | $(A_L \times B_H)$<<17 | $y(5) = y(4) + u(5)$<br>$= A_L \times B_L + (A_H \times B_L + A_L \times B_H)$<<17 |
| 6 | 0,0 | 0 | $(A_H \times B_H)$<<34 | $y(6) = y(5) + u(6)$<br>$= A_L \times B_L + (A_H \times B_L + A_L \times B_H)$<<17<br>$+ (A_H \times B_H)$<<34<br>$= A \times B$ |

**Table 5.1: Steps of the single slice 35-by-35 multiplier of Figure 5.10. The <<$n$ operation represent a $n$ bits arithmetical left shift. The delay between the output of the multiplier and the input of the accumulator is due to the shifter, which requires a single clock cycle**

| Method | DSP—Latency | Spartan 6 | | Artix 7 | | |
|---|---|---|---|---|---|---|
| | slices | (cycles) | LUT | Frequency | LUT | Frequency |
| Fully pipelined | 4 | 4 | 212 | 160 MHz | 212 | 330 MHz |
| Fully pipelined | 4 | 8 | 300 | 280 MHz | 212 | 435 MHz |
| Single slice, first version | 1 | 5 | 220 | 210 MHz | 230 | 250 MHz |
| Single slice, second version | 1 | 5 | 225 | 230 MHz | 215 | 280 MHz |

**Table 5.2: Speed and resource usage of the multipliers**

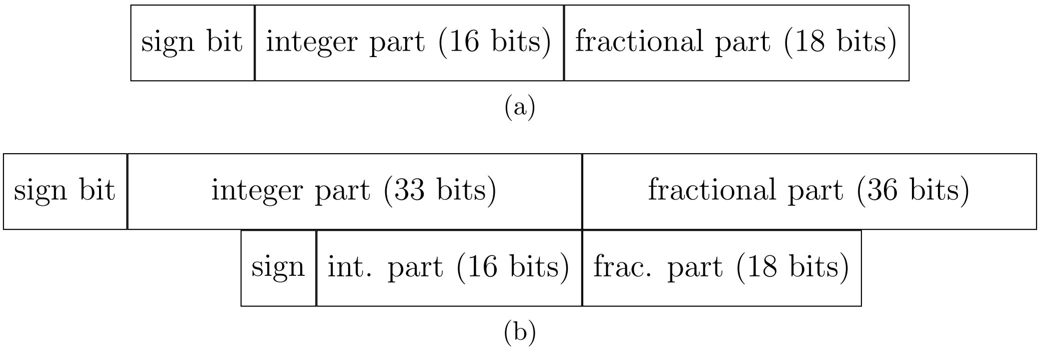


**Figure 5.11: Slicing the output of the 70-bit multiplier (a) operands in Q18 format (b) 70-bit output in Q36 format and extraction of a new Q18 number**

added before and after the module, and the LUT numbers takes them into account. Since the multipliers would be buffered in the platform anyway, the numbers reflect the correct real-world logic utilization.

- The maximum frequencies were assessed by compiling the Hardware Description Language (HDL) files, and running the *Trace* tool integrated with the *Xilinx* integrated software environment, which automatically generates the timings of all logical paths including the propagation delays (which could be larger than the logic delays if the design uses modules far apart from each other).

Multiple observations may be done from these tests. First, it seems clear that implementing pipe-lined multipliers on the less powerful Spartan 6 will either require a slower clock rate or a longer internal pipe-line to separate the product into more basic operations. In general, keeping a high clock frequency is more important since it allows a global acceleration of the whole design, while a shorter latency only accelerates the current module. In general, the pipe-lined design can handle higher clock rates, which makes sense since we use fast dedicated blocks while the single slice designs use more fabric logic. Secondly, it seems clear that a big jump in performance has been made between the previous low-cost family and the newer one. In particular, the maximum allowed frequency of the four-stages pipe-lined implementation has been doubled.
In light of these tests, it seems clear that computing the products quickly while keeping a high clock rate is perfectly possible using both platforms, and that the pipe-lined and the single-slice implementations are both viable. Among the two single-slice implementation, the second version allows a higher clock frequency, and will be preferred. Note that the speed of the single-slice design is essentially limited by the accumulator, which is implemented in generic logic. The output of the multiplier is coded on a 70-bit signed number. Since all signals are coded on 35 bits, we have to extract the correct range of bits corresponding to the chosen fixed-point representation. As an example, let us take the product of a state variable by a coefficient of the $A$ matrix. Since the values are coded in the Q18 format (see section 5.3.1), the output will be coded in the Q36 format (just as the product of two decimal numbers with one fractional digit leads to a number with two decimal digits at most). Since the output also corresponds to a state, we drop the 18 least significant bits and the 18 most significant bits (Figure 5.11). Of course, this technique assumes that the signals are properly scaled so that their value will never

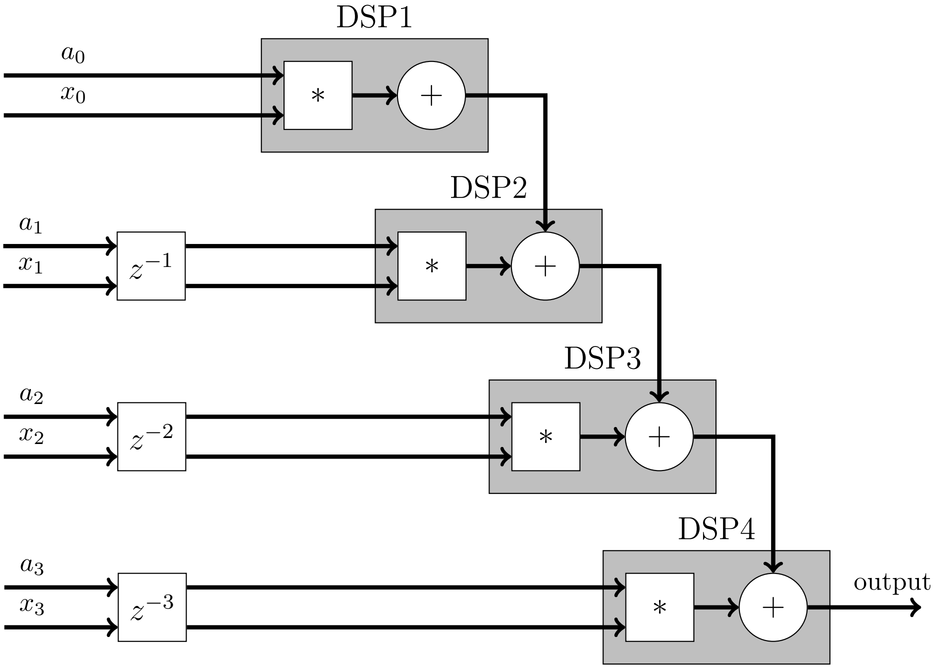


**Figure 5.12: Cascaded implementation of a dot product. The computations of the partial products are synchronized by the $z^{-n}$ blocks, and added together to produce the output**

exceed the 17-bit integer part. If this rule is respected, the value of all bits at the left of the 17-bit integer is equal to the sign bit (this is due to the *sign extension* principle).

### Dot product structure

Multiple choices are offered to implement fixed-point dot product, all of which having to do with the order and timing of the addition of the partial products. To provide a comparison, we first define $n$ as the length of the vectors and $T_{mul}$ and $T_{add}$ as the number of clock cycles needed to respectively perform a multiplication and an addition.
A first implementation, called the cascade adder or systolic adder, is shown in Figure 5.12. Here, the partial products are arranged as follows:

$$z = a_0x_0 + (a_1x_1 + (a_2x_2 + \dots)) \tag{5.11}$$

The result of a multiplier is added with the partial sum of all previous products. This method is very often used in DSP applications due to its high level of pipe-lining and the use of the integrated adder of the DSP slices. In our case however, this adder is already used as part of the computation of the products, resulting in the utilization of generic logic (or of an additional DSP slice) to perform the addition instead. The total latency of this implementation is equal to $T_{mul} + (n-1)T_{add}$. The number of stages may be reduced by implementing an adder tree instead, as shown in Figure 5.13 with 2-way adders. The partial products are now arranged as:

$$z = ((a_0x_0 + a_1x_1) + (a_2x_2 + a_3x_3)) + (\dots) \tag{5.12}$$

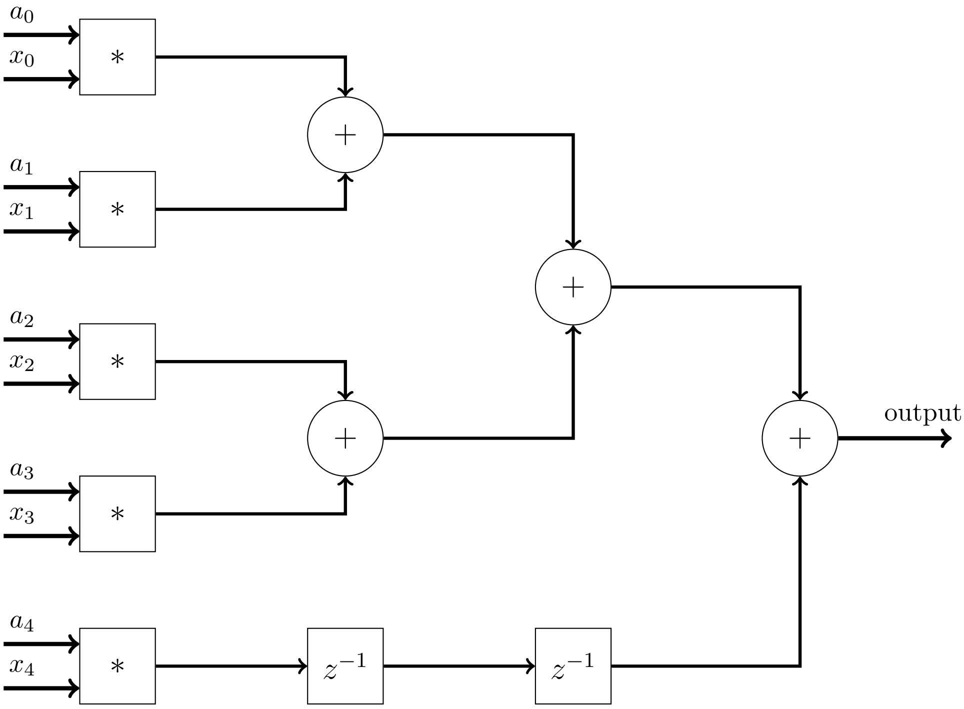


**Figure 5.13: Adder tree implementation of a dot product**

Thanks to this arrangement, the total latency is reduced to $T_{mul} + T_{add}\lceil\log_2(n)\rceil$. If the FPGA is fast enough, we can use adders with more inputs to further reduce the latency to

$$\text{latency}_{dot} = T_{mul} + T_{add}\lceil\log_k(n)\rceil \tag{5.13}$$

where $k$ is the amount of inputs of the adders. Again, this latency does not include the buffers that are typically added to reduce the critical path.

| Method | Latency | DSP | Spartan 6 | | Artix 7 | |
|---|---|---|---|---|---|---|
| | (cycles) | slices | LUT | Frequency | LUT | Frequency |
| Pipelined multipliers | 9 | 20 | 1200 | 225 MHz | 1100 | 345 MHz |
| Single slice multipliers | 12 | 4 | 2300 | 266 MHz | 1900 | 310 MHz |

**Table 5.3: Speed and resource usage of a dot-product of vectors with five elements, using 4-way adders**

In practice, the use of 4-way adders seems to be a good compromise. The result of the implementation of a dot product of two vectors of five elements is shown in Table 5.3 for the two FPGAs and the two styles of multipliers. The tradeoff between the usage of DSP slices and generic LUTs is apparent, as the single-slice implementation uses around twice as much LUTs as its pipe-lined equivalent.

A curious result is that the speed of the single-slice implementation is actually *faster* in the case of the Spartan 6. This is easily explained by the fact the limit comes from the connections between the multipliers, and not from the adder tree. By implementing the multiplier in a single multiplier followed by a buffer, we reduce the critical path and keep a higher clock rate. Adding two additional pipe-lining stages in the implementation raises

the clock rate to 280MHz.
The results are clearly in favor of the Artix 7, since the pipe-lined version is faster, needs fewer clock cycles and consumes fewer generic resources.

### 5.3.3 Multipliers optimization

The resource utilization of the multipliers can, under certain conditions, be drastically reduced. For example, we can completely remove a multiplier if we know that one of the operands is always equal to zero, and replace its output by a constant zero. The number of elements in the dot product is reduced, limiting the amount of stages in the adder tree. Likewise,

- if one of the operands is always equal to 1, the multiplier is replaced by a buffer
- if one of the operands is always equal to $-1$, the multiplier is replaced by an inverting buffer
- if one of the operands is always equal to a power of two, the multiplier is replaced by a shifter

These cases can be combined into the simplified multiplier of Figure 5.14 when $a_i \in \left[-1, 0, 1\right]$: a multiplexer allows to output $x_i, -x_i$ or 0 depending on the sign of $a_i$, where $sign(a_i)$ is defined as

$$sign(a) = \begin{cases} 1 \text{ if } a > 0 \\ -1 \text{ if } a < 0 \\ 0 \text{ if } a = 0 \end{cases} \tag{5.14}$$

For some circuits, this optimization leads to a very large reduction in the number of used DSP slices.

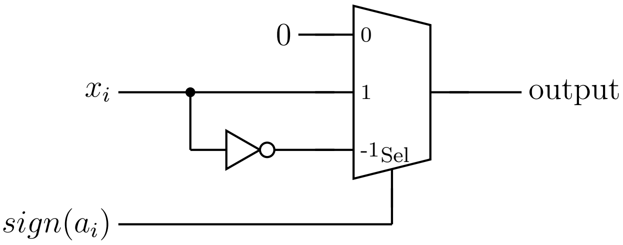


**Figure 5.14: Simplified multiplier used when $a_i \in [-1, 0, 1]$, where $sign(a_i)$ is defined by** (5.14)

**Example 16.** Let us study the equations the circuit of Figure 5.15. Since it contains four state variables (2 inductors and 2 capacitors) and one source $E$, we have a base number of $4 * (4 + 1) = 20$ elements in the $A, B$ matrices. However, it can be shown easily that these matrices are of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} i_{L1} \\ v_{C1} \\ i_{L2} \\ v_{C2} \end{bmatrix} = \begin{bmatrix} 0 & x & 0 & 0 \\ x & x & x & 0 \\ 0 & x & x & x \\ 0 & 0 & x & x \end{bmatrix} \begin{bmatrix} i_{L1} \\ v_{C1} \\ i_{L2} \\ v_{C2} \end{bmatrix} + \begin{bmatrix} x \\ 0 \\ 0 \\ 0 \end{bmatrix} E \tag{5.15}$$

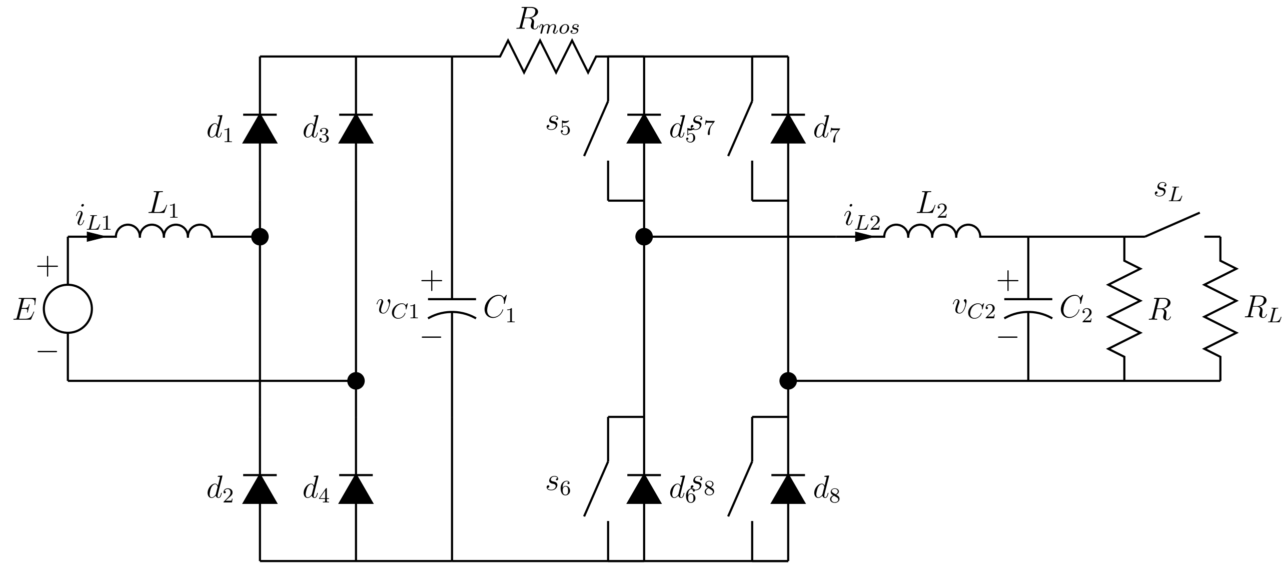


**Figure 5.15: In this circuit, most elements are equal to zero**

where $x$ is a non-zero value fixed by the topology. Only half of the coefficients have to be computed, which divides the number of used slices by two and reduces the size of the vectors in the dot products from five to three. Note however that this is only true if we use a discrete solver that keeps the form factor of the matrices intact, such as the Improved Euler Method described in section 2.2.3: all other methods disrupt the structure by inverting the state matrices. The reduction on the state-output matrices $C_X, D_X$ is even more impressive. The matrices are written as

$$\begin{bmatrix} i_{L1} \\ v_{C1} \\ i_{L2} \\ v_{C2} \end{bmatrix} = \begin{bmatrix} 0|1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0|1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_{L1,T} \\ v_{C1,T} \\ i_{L2,T} \\ v_{C2,T} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} E \tag{5.16}$$

Only four coefficients are not equal to zero, and all the remaining products can be computed using a simplified multiplier.

### 5.3.4 Selection tables

The selection tables are used to select

- the conditions to evaluate using the current topology (and the new state of the transistors in the case of the active switching module)
- the new state of the diodes using the result of the evaluation of the conditions

In both cases, the problem is stated as finding the best implementation of a $n-$ to $-1$ selector, translating the input vector into the correct answer. The ideal implementation should be fast and memory-wise efficient. It should also have a low latency and allow a fast clock rate.

### RAM-based implementation

| $i_L + \dfrac{V2 - v_C}{R_2} > 0$ | $\dfrac{V2 - v_C}{R_2} > 0$ | $i_L > 0$ | $V_2 - v_C \leq 0$ | $D_1$ | $D_2$ |
|---|---|---|---|---|---|
| 1 | 1 | - | - | on | on |
| 0 | - | 1 | 1 | on | off |
| - | 0 | 1 | 1 | on | off |
| All other combinations | | | | off | on |

**Table 5.4: Example of switching table**

One of the most straightforward implementations is to use a RAM-based table. With this method, the input is used as an index allowing to easily find the correct value using

$$value = Table[input] \tag{5.17}$$

The most important drawback of this implementation is the memory required to hold all the values, even if most of them are not useful. For example, let us take a look at the switching Table 5.4 (obtained in Section 3.4.3) containing four address bits corresponding to each of the four conditions. To hold all the values, we first need to expand the *don't cares* to obtain all sixteen values. Then, each line is mapped to an element of the table using the result of the evaluations as the binary address. Thus, instead of only having the four lines of the original table, we use four times as much memory. Since the same process must be repeated for all available combinations of switches, we could end with a large amount of data to store in memory. For this reason, we encourage a multiplexer-based solution, as presented in the next section

### Muxtiplexers-based implementation

Looking back at the same table, a multiplexer (mux) is another natural solution to our problem. A multiplexer is a logic construction which allows one (and only one) of its inputs to be connected to the output, based on the value of a set of control signals. It corresponds to the *switch* or *case* statement present in many programming languages. For example, Table 5.4 is written in C code as:

```
switch(address)
{
        case(11--):
                out = 11;
                break;
        case(0-11):
                out = 10;
                break;
        case(-011):
                out = 10;
                break;
        default:
                out = 01;
}
```

The implementation of a mux on a FPGA is similarly straightforward. The logic blocks contain basic multiplexers with two inputs, which are connected to form larger modules. Unfortunately, the tables cannot be implemented as-is because they may contain equivalent lines. For example, both the second and third line of Table 5.4 will react if the input is equal to 0011. While this does not seem to be a problem since the outputs are the same, it is difficult to know how the internal logic of the FPGA will behave when dealing with this case. This problem is avoided by expanding the *don't cares* (written as −− in the table), which leads to the same problem as the RAM-based implementation. Another limitation is due to the VHDL language used to program the FPGA. While this language supports the creation of multiplexers using the *case* statement, this construct do not accept any − in the list of inputs, which completely voids its application to our case since it again means that we would have to expand the *don't cares* [6].

### Priority Encoder

The multiple selection problem is avoided by introducing a priority mechanism that tests the entries one by one until a correspondence is found. This mechanism, known as a *priority encoder*, corresponds to the following C code:

```
if (address == 11−−)
        out = 11;
        else if (address == 0−11)
                out = 10;
                else if (address == −011)
                        out = 10;
                        else
                                out = 01;
```

The most important disadvantage of this method is that all the *if* statements have to be evaluated in sequence until a match is found, leading to a non-constant evaluation time. The critical path is a function of the number of comparison stages, which can limit its usage for larger tables. Nevertheless, this method seems to be the best match for our application due tot he fact that we do not need to expand the don't care statements, and that the priority order is respected. The equivalent logic circuit for FPGA is presented in Figure 5.16: the first stage makes all comparisons in parallel, and the second selects the output by activating the correct buffer.

The design is accelerated by first combining all inputs leading to the same output, which corresponds to the following code:

[6] While the possibility of using *don't cares* was added in the 2007 version of the VHDL language, it is not yet understood by the systhesis tools.

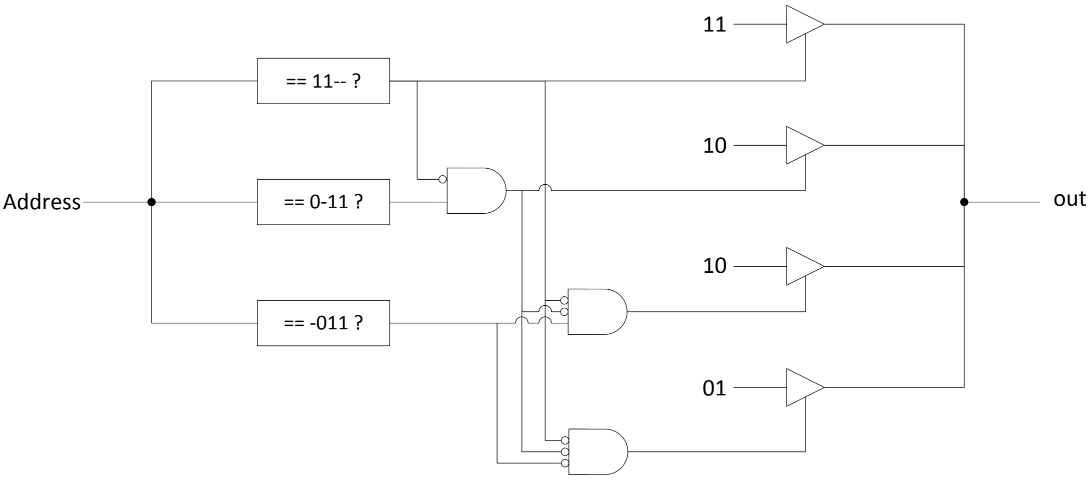


**Figure 5.16: Logic circuit of a priority encoder**

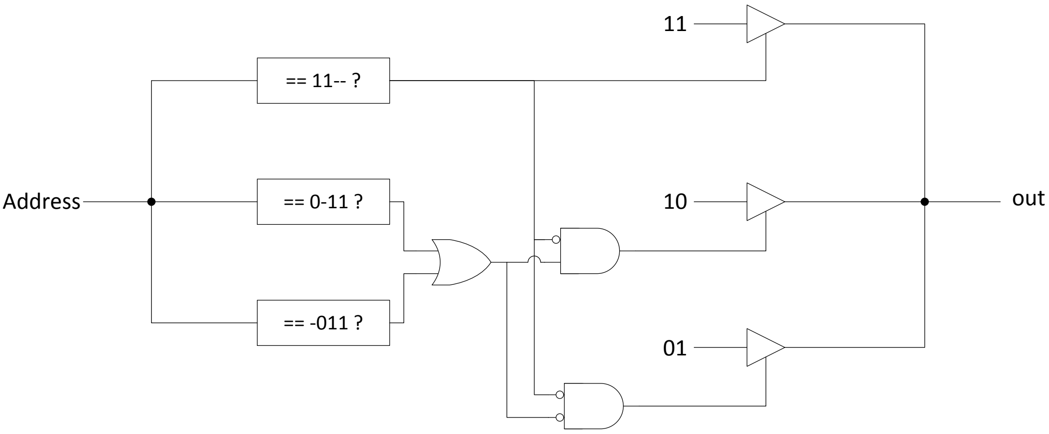


**Figure 5.17: Logic circuit of a priority encoder with combined inputs**

```
if (address == 11--)
        out = 11;
        else if (address == 0-11 || address == -011)
                out = 10;
                else
                        out = 01;
```

and leads to the circuit of Figure 5.17, where the equivalent inputs are combined using an OR gate. The final circuit has now three sequential stages:

1. all comparisons are made simultaneously during the first stage
2. the results of the the comparisons leading to the same output are combined (i.e. OR-ed together)
3. the priority encoder properly enables the correct output

Since the tables can include hundreds of lines, we have decided to add buffering gates between each stage. This procedure reduces the critical path, allowing higher clock frequencies at the cost of two additional clock cycles.

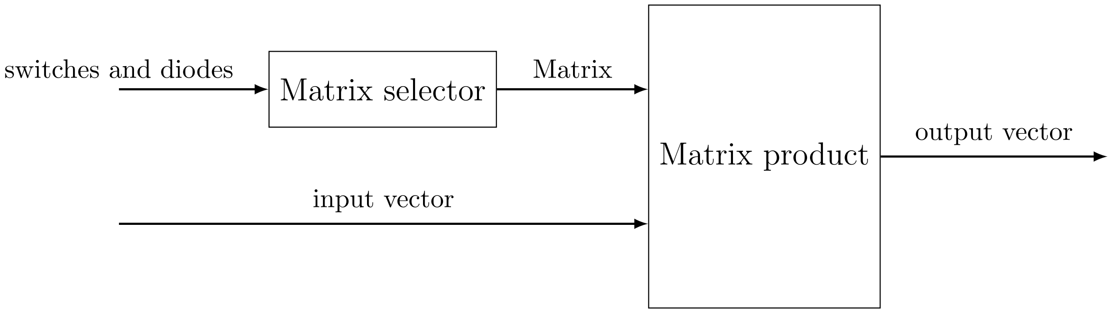


**Figure 5.18: Structure of the solvers**

### Table reduction

In some cases, multiple lines corresponding to the same output can be combined into a single one using logic reduction. For example, if the inputs 1000 and 0000 lead to the same output, these two lines can be combined into −000. Normally, this reduction is made automatically by the HDL compiler during the synthesis of the logic circuit. However, making such a reduction ourselves allow us to select the best evaluation order for the lines and, most importantly, to select which value we should assign to the *others*[7] entry of the table (see Table5.4).
To reduce the logic, a practical choice for the last line is to select the output with the largest amount of corresponding inputs. The reasoning behind this choice is that the last line is not explicitly evaluated (i.e. is selected only when all other comparisons have failed), and the comparison and combination logic is not generated.
Many automated reduction algorithms exist, each with their strengths and weaknesses. For example, the Quine-Mc Cluskey algorithm guarantees that the result is the minimal representation of the system, but its execution time is $\mathcal{O}2^n$, where $n$ is the number of inputs to the function [97]. In comparison, heuristic reducers find an *almost* optimal result, but do so in much less time. The *Espresso Logic Minimizer* algorithm, originally developed by IBM, is the de-facto standard and is implemented in one form or another in almost all HDL synthesizers [98].

# 5.4 High level modules

## 5.4.1 Introduction

The modules presented in section 5.3 are the basic bricks that will allow us to build the more complex parts of the simulator. In this section, we will study each of the parts individually and show the modules can be sued to optimize the design.

### 5.4.2 Linear Solver and output solver

#### Internal structure

The internal configuration of the two linear solvers is quite simple, as shown in Figure 5.18. Both modules are based on the same sub-functions. First, the value of the state matrices ($A, B$ for the state solver, $C, D$ for the output solver) is obtained using a selection table that takes the current configuration of the switches as its input. Then, each of the outputs are computed in parallel using its own dedicated dot product.

#### Selection tables

Instead of writing a single selection table for the complete matrix, each of the elements of the matrices is obtained using a dedicated multiplexer. This way, we avoid generating extra logic modules if only some of the elements need a wide selection table while some others have fewer values (or are constant).

**Example 17.** Let us take a circuit with two switches, leading to four topologies. We have four $A$ matrices:

$$\begin{aligned} A00 &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, A01 &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \\ A10 &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, A11 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{5.18}$$

Furthermore, we assume that the selection table, written in C, is the following

```
if (switches == 00)
        A = A00;
        else if (switches == 01)
                A = A01;
                else if (switches == 10)
                        A = A10;
                        else
                                A = A11;
```

Four lines are needed while each coefficient has at most two different values. If we separate the elements of the matrix, we end up with four independent selectors:

[7] The *others* entry of the *select* VHDL instruction corresponds to the *default* entry of the *switch* instruction present in many programming languages.

```
//Selector for element (0,0)
if (switches == -1)
        A[0][0] = 1;
        else
                A[0][0] = 0;

//Selector for element (0,1)
        A[0][1] = 0

//Selector for element (1,0)
        A[1][0] = 0;

//Selector for element (1,1)
if (switches == 0-)
        A[1][1] = 1;
        else
                A[1][1] = 0;
```

Only two elements require proper selection modules: the two other ones are assigned a constant value. Furthermore, the remaining individual selectors are much simpler than before (they only require the value of a single bit).

**Matrix product**

The individual dot products are computed in parallel using the modules presented in section 5.3.2. Since the solvers are only used once for every time-step, there is no need to use a pipe-lined design, and the non-pipe-lined version of the multipliers (shown in Figure 5.10) is generated to reduce usage of the DSP slices. The optimizations presented in section 5.3.3 are used to eliminate partial products with a constant zero coefficient or equal to ±1. With this modification, the different dot products do not contain the same amount of partial products and adder stages, and additional buffers are added to equalize the length of the paths, as shown in Figure 5.19.

### 5.4.3 Natural Switching module

The structure on the natural switching module is similar to the linear solvers:

1. A matrix multiplexer selects the conditions to evaluate according to the current configuration of the switches. These conditions are the outputs of the natural switching analysis carried out in section 3.3

2. The conditions are evaluated by performing a matrix product

3. We test if the conditions (of the form $i_D > 0$ or $v_D \leq 0$) are verified or not

4. A second multiplexer selects the new configuration of the diodes based on the previous configuration and on the result of the evaluation

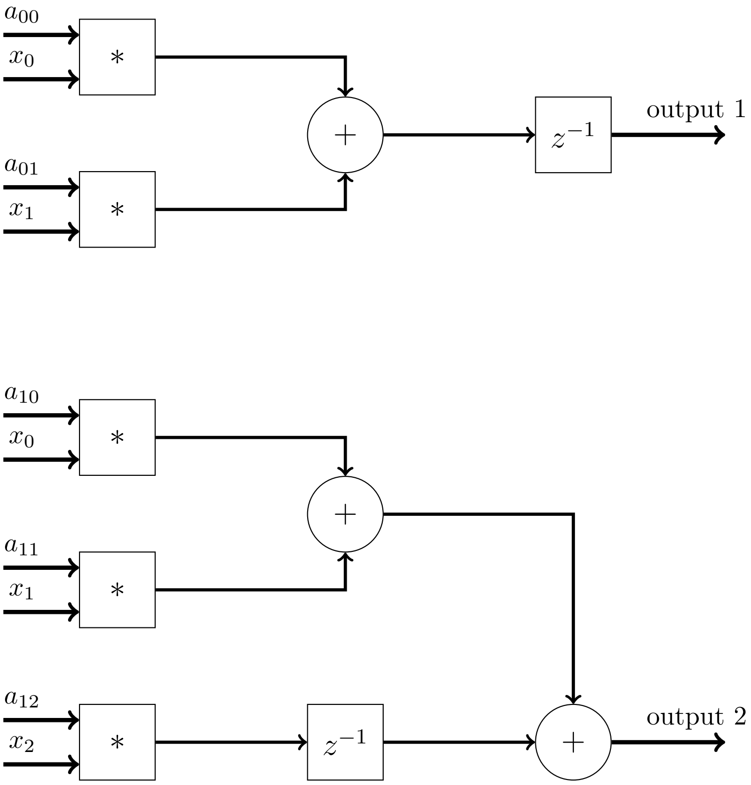


**Figure 5.19: The length of the paths corresponding to the individual dot products are equalized by inserting extra buffers**

The first two modules are the same as those present in the linear solvers, and do not need additional information. Since the natural switching analysis is only carried out once for each time-step [8], the non-pipelined variant of the multipliers is again used.

Testing the validity of the condition can be done by evaluating a single bit if we rewrite the tests in a slightly different way. Let us assume that we rewrite the conditions to be of the form $y \geq 0$. The test can be performed by looking at the sign bit, which is always equal to zero for a positive number, including zero. Hence

- Instead of testing $v_D \leq 0$, we test $-v_D \geq 0$
- Instead of testing $i_D > 0$, we test $-i_D \geq 0$ and invert the result of the evaluation

These changes do not add any complexity to the design, as long as the tables are accordingly modified during the offline analysis. First, we change the sign of all the coefficients written in the matrix selector, which means we will compute $-i_D$ or $-v_D$. Then, if the condition corresponds to a current, we invert the *True/False* values in the selection tables. The two tables of Table 5.5 provide an example of modification

Finally, the second table which selects the state of the diodes is implemented using a single priority encoder.

[8] as a reminder, a natural switching event is a change in a diode in reaction to a modification of the value of a state or an input, which can only happen during the computation of the new time-step.

| $E - v_C \leq 0$ | $i_L > 0$ | $D_1$ | $D_2$ |
|---|---|---|---|
| 1 | - | on | off |
| 0 | 1 | off | on |

(a)

| $-E + v_C \geq 0$ | $-i_L \geq 0$ | $D_1$ | $D_2$ |
|---|---|---|---|
| 1 | - | on | off |
| 0 | 0 | off | on |

(b)

**Table 5.5: Modifying the conditions to allow for a faster evaluation of their validity (a) Before modification (b) After modification**

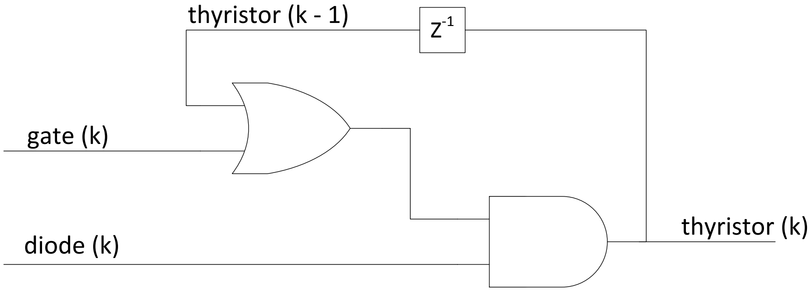


**Figure 5.20: The thyristor control circuit, where the conduction state of the thyristor depends on its state at the previous iteration, the state of the diode given by the selection tables, and by the gate control signal**

### Thyristor control

Until now, we have not explicitly talked about implementing hybrid switching devices, such as the thyristors, in the platform. This can actually be performed with a small addition to the natural switching module, as we shall show now.
As a reminder, the thyristor is a device that acts as a diode with an added enabling pin called the gate. The device can only switch to the on-state if the cathode to anode voltage becomes positive (like a diode) *and* a direct current is injected in the gate-cathode junction. Once turned-on, the device will remain in this state until the current drops to zero, at which point it turns off. Hence, an ideal thyristor can be modeled as a simple diode, which allows us to use all of our algorithms seamlessly.

An additional module is placed after the second selection table, and allows us to force the mode of the diode to a particular value. This module is a simple combination of an OR-gate and an AND-gate, as shown in Figure 5.20: the feedback allows us to lock the device in the on-state until the current becomes negative, even if the gate voltage disappears.

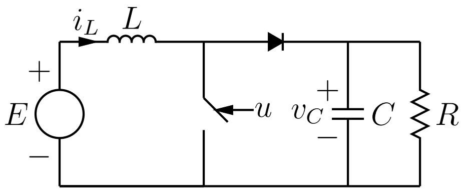


**Figure 5.21: Boost Converter used to show the effect of inter sample-time switching events**

### 5.4.4 Forced Switching Module

#### Structure

At first sight, the structure of the forced switching module is exactly the same as the natural switching module. Indeed, this selector is also composed of a matrix selector followed by a matrix product between the provided and the states and inputs. Again, the outputs of this products represent the diode currents and voltages, which are then compared to zero to select the new configuration of the diodes.

The only difference between the forced switching module and the natural switching module is that the former reacts to a change in the switch control signals. This is the reason why the selector takes both the old configuration of the switches and the new value of the control signals as its input.

The control signals are not synchronous with the time-step and may change at any time during this period. This leads to the apparition of *Inter Sample-time Switching Events* (ISSE), whose effects will be analyzed now

#### Managing Inter Sample-time switching events

The control signals of the active switches can be modified (by the controller or by the protection systems among others) at any point during the time-step. How we manage these events can have a profound effect on the accuracy of the waveforms, as will be shown shortly.

The simplest way to deal with these events is to simply sample the value of the control signals at the beginning of the time step. With this method, the forced switching engine is called once at every time-step, and is placed directly in series with the rest of the modules. The problem with this approach is that the resolution on the timing of the switching events is equal to the time-step, which degrades the quality of the emulation when the control signals change rapidly.

As an example, let us take a 100kHz PWM control signal and a time step of $1\mu s$. With this configuration, the resolution on the duty cycle is equal to $\frac{1\mu s}{10\mu s} = 10\%$. To analyze the effect of this loss of precision, let us take the boost converter of Figure 5.21 with the following parameters

$$\begin{aligned} E &= 100V \\ L &= 2mH \\ C &= 100\mu F \\ R &= 10\Omega \end{aligned} \tag{5.19}$$

Let us further assume that the switch is driven by a 100kHz PWM characterized by a

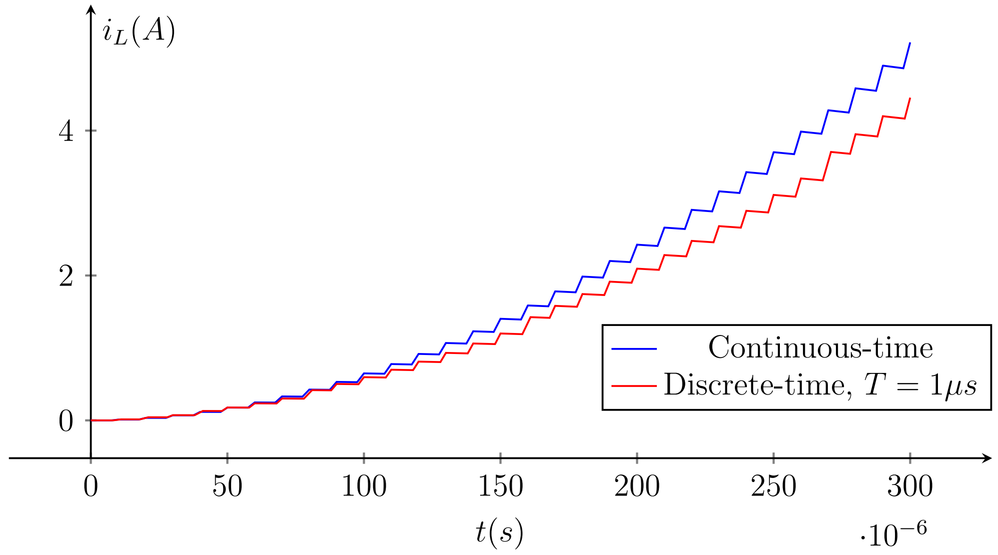


**Figure 5.22: Simulation of the boost converter when the control signal of a switch occurs during the time step and is sampled only once per simulation time step : the simulation diverges quickly from the expected result**

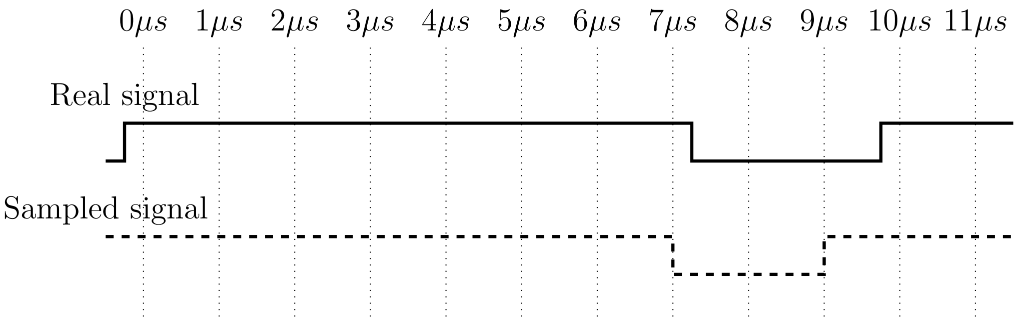


**Figure 5.23: Effect of the sampling on the control signal, the effective duty-cycle is modified from 75% to 80% and the commutations are delayed**

duty-cycle of 0.75 and a simulation time-step equal to $1\mu s$. The simulation of Figure 5.22 compares the result of the emulated waveforms with the ideal result. Obviously, the effect on the outputs are significant. This is to be expected, since the emulated duty-cycle $\delta$ will either be equal to 0.7 or to 0.8 depending on the time-shift between the PWM and the start of the time steps (Figure 5.23). More precisely, the current system assumes that the sampled value of the control signal stayed the same for the whole duration of the time-step. Once the inductor and the capacitor have reached their nominal charge, the steady-state average value at the output of the boost is given by

$$v_C = \frac{1}{1-\delta}E \tag{5.20}$$

Because of the sampling, the average output voltage will become $\frac{1}{1-0.8}E = 5E$ instead of $\frac{1}{1-0.75}E = 4E$, i.e. an error of 25%.
This behavior is unacceptable, and a new way of dealing with ISSE must be developed. Multiple methods have been introduced to modify the outputs when a switching event occurs between two time-steps. Some of them are presented in [99]. Let us assume that the switching event happens at some point during the time-step from $kT$ to $kT + T$, and that the exact switching moment is $kT + \tau$. We also assume that $\tau$ is known exactly by sampling the input signals with a sampling period much shorter than the time-step. This is not a problem, as reading a binary signal from an input pin can be done at the full clock frequency of the FPGA (250MHz or more). Most correction methods are based on those steps:

1. compute $x(k + 1)$ normally
2. if a switching event has occurred at time $kT+\tau$, interpolate linearly to find $x(kT+\tau)$

$$x(kT + \tau) \approx x(kT) + \frac{\tau}{T}(x((k + 1)T) - x(kT)) \tag{5.21}$$

3. apply a correction to $x((k + 1)T)$ using $x(kT + \tau)$

Another method is to iteratively solve the circuit for each time event. For our example, it means we would first use the linear solver to find $x(kT + \tau)$

$$x(kT + \tau) = A_{d1}x(kT) + B_{d1}u(kT) \tag{5.22}$$

where $A_{d1}, B_{d2}$ depend on the chosen integration method. Then, we use a second linear step to find $x((k + 1)T)$

$$x((k + 1)T) = A_{d2}x(kT + \tau) + B_{d2}u(kT) \tag{5.23}$$

The second method is more accurate, and is effectively equivalent to a variable time-step solver. However, both solutions have the same problem: since the number of iterations is not known at synthesis time (i.e. multiple switching event may occur during a single time-step), we cannot ensure that the computations will take less than a single time-step. Depending on the chosen discrete-time approximation, the second method may also require the online inversion of the state matrices (for example, the Backward Euler

Approximation forces us to compute $(I - \tau A)^{-1})$ which a long procedure that requires a lot of resources.

Instead, we suggest a procedure based on the *averaged state-space approximation*, which has been used for decades in offline simulators to quickly obtain the long-term behavior of power circuits. As implied by its name, this approximation avoids the computation of fast transients by averaging the state-matrices along the time-step [9]. For our example, it means we could directly compute $x((k+1)T)$ using

$$x((k+1)T) = (\frac{\tau}{T}A_{d1} + \frac{T-\tau}{T}A_{d2})x(kT) + (\frac{\tau}{T}B_{d1} + \frac{T-\tau}{T}B_{d2})u(kT) \tag{5.24}$$

With this approximation, the solver remains a single-step method, and the latency remains the same even if multiple switching events occur. If we know the switch configuration at all time, we can infer $A(t), B(t)$ and the averaged state-space is written as

$$x((k+1)T) = \left(\frac{1}{T}\int_{kT}^{(k+1)T} A(t)dt\right) x(kT) + \left(\frac{1}{T}\int_{kT}^{(k+1)T} B(t)dt\right) u(kT) \tag{5.25}$$

To compute the integral, multiple changes must be made to our algorithm. First, we have to sample the control signal at a higher sampling rate. This is not a problem since, as already mentionned, reading a binary signal from an input pin can be done at the full clock frequency of the FPGA (250MHz or more).

Since we must know the switch configuration at each clock cycle, the forced switching module is modified to use the pipe-lined version of the multipliers. This way, their output is refreshed at each clock cycle, allowing us to know the current value of the $A$ and $B$ matrices.

Finally, the integration is performed by accumulating the value of the matrices during the time-step. We first define $n$ as the ratio between the sample period and the FPGA clock cycle $T_{FPGA}$

$$n = \frac{T}{T_{FPGA}} \tag{5.26}$$

We then compute the integral using

$$\frac{1}{T}\int_{kT}^{(k+1)T} A(t)dt \approx \sum_{j=0}^{n} \frac{1}{n} A(jT_{FPGA}) \tag{5.27}$$

When a new time-step starts, the accumulated values are sent to the state-solver while the accumulators are reset to zero. With these modifications, the selection of the state matrices is moved from the state solver to the forced switching engine, which now operates at the FPGA clock rate instead of the sampling rate.

The final problem lies in the division present in (5.27). To avoid the use of a classical divider, which requires at least one clock cycle per bit, we select a factor $n$ equal to a power of two ($n = 2^h$). This way, the division can be replaced by an arithmetical right shift of $h$ bits. This operation is easily implemented in the generic slices of the FPGA

[9] Originally, this method was used to avoid the simulation of the switching effects, representing only the average effects of the PWM control

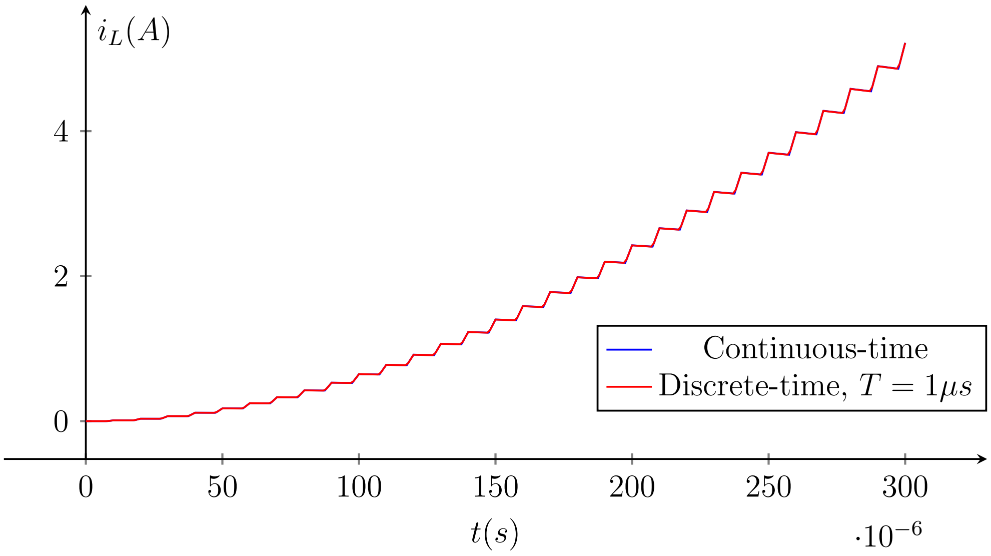


**Figure 5.24: Simulation of the boost converter with switching events during the time step. The control signals are sampled at 100MHz**

and is performed in a single clock cycle. Since $n$ is constant, we could also pre-compute $\frac{1}{n}$ and multiply by this factor instead. However, this alternative requires an additional pipe-lined multiplier per coefficient. The results for the simulation of the boost converter with a time-step of $1\mu s$ and a sampling period of $10ns$ for the control signals is presented in Figure 5.24. This time, the two plots are almost perfectly aligned thanks to the oversampling of the PWM signal.

### 5.4.5 Outputs interfacing

The results obtained with our platform are meaningless if we are not able to somehow output their value to the outside world. Since the main goal of our platform is to emulate the signals obtained by the voltage and current sensors, it make sense to recreate the analog voltages corresponding to the measurements. Hence, digital-to-analog converters (DACs) must be added to the platform, and the communication protocol must be implemented.

To avoid designing complex circuit boards based on fast (10 megasamples per second or more) DACs with a parallel interface, we have selected the much smaller AD5453 chip manufactured by *Analog Designs*. This integrated circuit is characterized by

- a serial interface using the SPI protocol
- a maximum sample rate of 2.7 mega samples per second
- a resolution of 14 bits

Physically connecting these circuits to our FPGA only requires three signals (the clock signal, the data signal and a chip select signal), which simplifies the design.

The main disadvantage of using serial DACs is the increased lag in the control loop. Indeed, sending an output to the DAC requires sixteen SPI clock cycles (which is slower than the FPGA clock), leading to an added lag of about $350ns$. Taking the time to develop a front-end based on faster converters would be a way to reduce the loop lag.

### Output signal scaling

To make full use of the output range of the DACs, we have to properly scale the signals beforehand. To better understand the concepts introduced in this section, we first need to remember how a DAC works. An unipolar, voltage-based DAC transforms a binary number $N$ composed on $n$ bits ($n$ being equal to the resolution of the converter) into a voltage $v_o$ using a simple rule of thirds :

$$v_o = \frac{N}{2^n - 1} V_{fs} \tag{5.28}$$

where $V_{fs}$ is the full-scale voltage of the converter. To use the converter to the best of its abilities, we must ensure that we use the whole range of values for $N$. Let us assume that the signals are scaled in a way that the maximum value sent to a 14-bit DAC is equal to 1000, which can be coded on 10 bits only. Not only do we have a lower resolution, meaning that fewer different values can be sent to the converter, but we barely use 1/16 of the voltage range. This reduced voltage is more sensitive to electromagnetic interferences, hence the signal-to-noise ration will be worse, which may degrade the performance of the control loop. We have thus to carefully select the representation of the output signals.
The output equation, augmented by the modified value of the states (see section 2.6), is defined by

$$\begin{bmatrix} y(k) \\ x(k) \end{bmatrix} = \begin{bmatrix} C \\ C_x \end{bmatrix} x_T(k) + \begin{bmatrix} D \\ D_x \end{bmatrix} u(k) \tag{5.29}$$

The $y(k)$ vector corresponds to the measurements that will be sent to the converters, while $x(k)$ is an internal vector containing the state variables. The format of the state vector is fixed to 18 fractional bits to remain consistent with the rest of the signals. Meanwhile, we are free to chose the best scaling for $y(k)$.
Let us first look at what would have happened if we had kept the Q18 format for $y(k)$. The maximum value for this signal is equal to $2^{14} = 16384$. If the actual value of $x(k)$ the does not go higher than 100 during the simulation, we use less than ten percent of the scale and the four most significant bits are lost. Thus, sending the 14 most significant bits to the converter is very inefficient in this case.
A better choice is to select a full scale value for each of the outputs (for example, 100 volts or 10 amperes), and normalize the signals by dividing by this value. Let us first assume that $y(k)$ is scalar (i.e. there is only one output), and that we select a maximum value of $y_{max}$ for this signal. The equation for the normalized signal $y_n(k)$ is

$$y_n(k) = \frac{1}{y_{max}} C x_T(k) + \frac{1}{y_{max}} C u(k) \tag{5.30}$$

Since the maximum value of normalized signal is equal to 1, all 34 data bits are now part of the fractional part: $y(k)$ is now expressed in the Q34 format (it codes the *per unit* value of the signal), with $100\ldots0$ corresponding to $-1 \times y_{max}$ and $011\ldots1$ to $(1-2^{-34}) \times y_{max}$. This choice brings multiple advantages. First, it allows us to make a direct comparison with the sensors that would have been used on the real converter: all we have to do is

select $y_{max}$ equal to the full range value of the sensor.
For our goals, the biggest advantage is that we are now sure that all the most significant bits are used, and we can simply take the fourteen most significant bits and send them to the converter, whereas a badly scaled signal would have many unused bits, which reduces the effective resolution of the converter.

### 5.4.6 Input Interfacing

The main inputs of our platform are split between the binary control signals of the switches and the independent sources of the design. Due to the differences in their nature, these two types of inputs are addressed separately in the next sections.

#### Switch controls

The control signals of the switches are, by nature, binary values. These signals can be generated externally by a controller or a PWM modulator, or internally by an additional module. In the current version of the platform, only externally generated signals are supported. Interfacing external binary signals to the platform is easily done by connecting the binary control signal to the correct IO pin, after ensuring that the input is electrically compatible with the board.

#### Electrical sources modeling

The electrical sources can be of two types. Independent sources correspond to external signals whose values do no depend on the evolution of the system, while controlled sources are used to interconnect the partitions of a single circuit.
The two main kinds of sources found in electrical devices are constant value sources and sinusoidal sources. While the first kind is trivial to generate, we must handle the sinusoidal sources carefully, as a bad implementation could lead to offsets or drifts in the signal. Let us study the simple integrating system

$$\frac{\mathrm{d}x}{\mathrm{d}t} = bu(t) \tag{5.31}$$

where $u(t) = \sin(\omega t)$ is an AC source. The discrete version of this equation is

$$x(k+1) = x(k) + T_s bu(k) = x(k) + T_s b \sin(\omega k T_s) \tag{5.32}$$

Assuming $\omega T_s = n \in \mathbb{N}$ (i.e., the period of the signal is a multiple of the time-step), we have

$$x(k+n) = x(k) + \sum_{i=0}^{n-1} \sin(ni) \tag{5.33}$$

To ensure that $x(k)$ is periodic and does not present any drift, we must verify that the following is true

$$\sum_{i=0}^{n-1} \sin(ni) = 0 \tag{5.34}$$

In other words, we have to ensure that the sum of all the values taken by the source over one period is equal to zero. This is easily done using a table based implementation of the sine over a half-period. Assuming a period of $n$ points, we first generate a sawtooth signal $z(k)$ with increases from 0 to $n-1$ at each new time step before rolling back to zero. Then, assuming $n$ even, we simply encode the following

$$u(k) = \begin{cases} \sin(\frac{z(k)}{n}) & \text{if } 0 \leq z(k) \leq n/2-1 \\ -\sin(\frac{z(k)}{n}) & \text{if } n/2 \leq z(k) \leq n \end{cases} \tag{5.35}$$

This implementation ensures that the opposite of each positive value will appear in the sequence, guaranteeing that the average of $u(k)$ over a period is equal to zero.
Note than this table-based implementation of the sine/cosine is not the only one. Another popular variant is the *Cordic* algorithm [100], which provide similar results. Controlled sources are handled very differently. Indeed, as explained in section 5.2, the value of these sources is recalculated at the beginning of each step as a linear combination of the previous value of the sources and measurements.

### 5.4.7 Source code generation

The analysis performed in chapters 3 and 4 results in the creation of data files containing

- the correspondence between the topology and the state matrices
- the conditions that must be evaluated at each time step
- the correspondence between the result of this evaluation and the new topology

The results lead to the implementation of the selection tables as seen in section 5.3.4. In practice, the VHDL code corresponding to the selectors is completely generated by a *Python* script. This is due to the fact that these selectors are completely specific to a particular circuit. These are the only modules of the real-time platform that are directly generated during the rapid prototyping process.
In contrast, the dot products used in the solvers only need a few pieces of information to be configured. More precisely, each dot product only needs to know which elements of its input vectors must be multiplied together (i.e. which partial products will not always result in a zero value), and which partial partial products can be replaced by a simplified multiplier as described in section 5.3.3. Hence, we decided to code a static generic dot-product VHDL module which is configured during the synthesis step using the data produced during the compilation of the results of the analysis step (see the design workflow on Figure 5.1).

## 5.5 Tests and validations

### 5.5.1 Validation procedure

In this section, we will perform various tests in order to assess the actual performance or our platform. These tests will be slightly different from those carried in sections

3 and 4, and their purpose is not the same. The previous tests were made to prove that our algorithm was a correct representation of the dynamics of the emulated circuit. Our goal in this section is to prove that the signals obtained with the real-time platform (RTP) are the same as those that were obtained in simulation, even with the quantization introduced by the fixed-point representation of the signals, and the limited precision of the multipliers. Furthermore, we need to quantify performances that have no impact on the offline simulation, such as the loop latency or the resources usage.

The assessment can be made at different points during the compilation of the HDL files. Since the ultimate purpose of this section is to look at the quality of the signals present at the output of the DACs, this test requires us to go through the (rather long) mapping and place/route steps. Furthermore, this test does not allow us to easily access the internal signals of the FPGA, and the results are impacted by the quality of the DACs and the measuring devices, which are not crucial parts of this thesis.

To circumvent this limitation, we will mostly perform tests at the functional level of the platform. This is an offline simulation performed using the program *ModelSim* which allows us to verify if the logical equations correspond exactly to what we wanted. Since this simulation is accurate at the clock-cycle level, we shall also use it to exactly measure the latency in terms of clock cycles. By adding stimulus files (emulating the outside world, such as a PWM or a controller) to the simulation, we have a good idea of the performance of the platform without actually having to synthesize the HDL files.

The two main missing pieces of information, the timing-closure and the resources usage, are obtained by performing the full synthesis of the logical circuit. If the design respects the timing parameters and performs the correct logical function, it is almost guaranteed that the real-time platform will behave as expected [10].

### 5.5.2 Platform description

Before performing the tests, let us first quickly introduce the hardware used for the validation. The target processor is a *Spartan 6 SLX 150* FPGA. This is the most powerful chip of the SPARTAN family, i.e. the low-end range of the 2009 Xilinx catalog. This choice allows us to see what kind of circuit we can hope to emulate on a low-cost platform, as developer boards based on this chip can be bought for less that 100 euros. In this case, we used a *TE0630* FPGA board from *Trenz Electronics*, fitted on a *TE0303* carrier board that includes all power management integrated circuits as well as clock generators, and provides easy access to most FPGA pins. An external board containing eight AD5453 digital-to-analog converters was designed in-house, and is connected to the *TE0303* using a ribbon cable, which carries the individual SPI channels. Both boards are represented on the picture of Figure 5.25. The complete schematics of the external board are presented in Appendix A.

The FPGA is controlled by a 125MHz external clock, which is converted to a 200MHz signal inside the chip by a dedicated clock manager. As explained in section 5.4.4, we need to select a sampling period equal to the clock period multiplied by a power of two. In this case, we have chosen a time step of 128 clock cycles, i.e. 640ns.

[10] A second simulation can be performed at the electrical level after the place and route step. However this simulation is extremely slow due the representation of all the electrical properties of the FPGA, down to the transmission-line models of the interconnections, and is only used to track precise bugs.

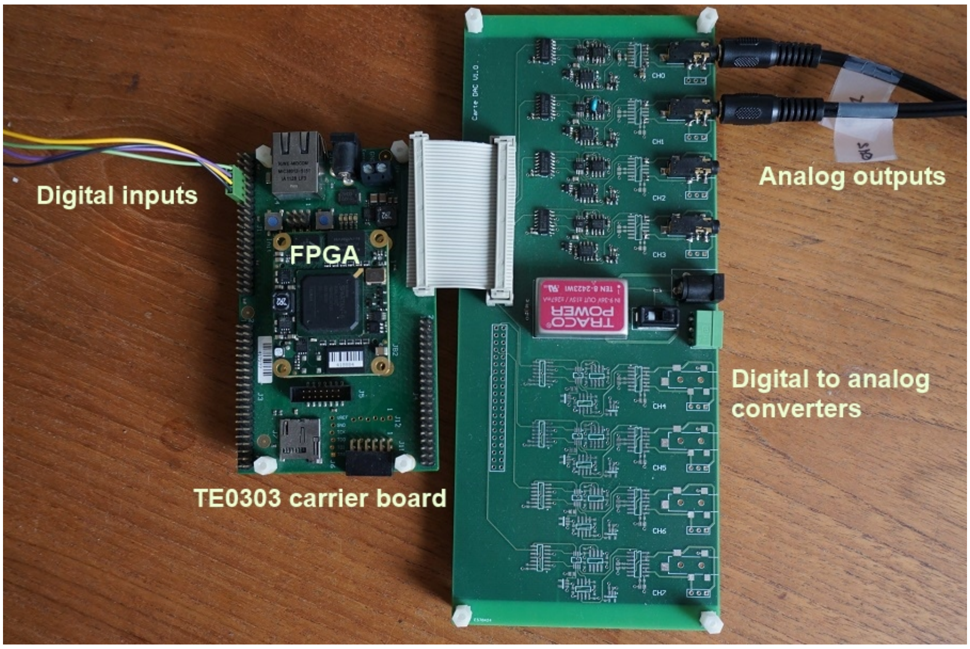


**Figure 5.25: Real-time platform used for the tests, with the TE0630 FPGA board plugged into its TE0303 carrier board on the left, and the custom-made DAC board on the right. A ribbon cable carrying the eight SPI channels connects the two board.**

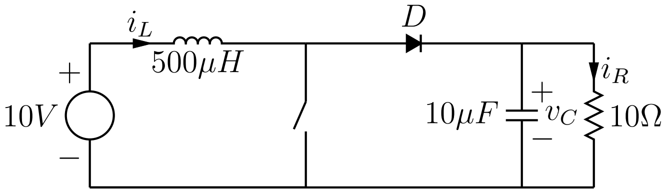


**Figure 5.26: Boost converter used to test the real-time algorithm**

### 5.5.3 Algorithm validation : the boost converter used in open loop

#### Introduction

Before testing larger converters containing many switches and components, we will first try to simulate in real-time the simple boost circuit of Figure 5.26. The reason for this is twofold. First, the simple waveforms allow us to easily verify the correct behavior of each sub-module individually. Secondly, it provides us with as estimation of the highest speed reachable by our design, since the boost is one of the simplest and smallest converter (along with the buck converter, which is composed of the same electrical components).
While we have designed the solver to be as independent of the size of the circuit as possible, a larger converter will probably need more adder stages in the dot products or limit the clock rate if the logic functions become too complex.

### Offline Tests

The goal of the tests in this section is to verify if the platform performs the correct function. These tests are made offline using *ModelSim*, which gives us access to all internal signals such as the state variables and the signals evaluated by the switching engines.

The first test consists in controlling the circuit with a $100kHz$ PWM with 50% duty cycle, which corresponds to the higher limit of our requirements. As mentioned, the sampling period is fixed to $\frac{128}{200}\mu s = 640ns$, resulting in about 15 computations for each PWM period. The *ModelSim* simulation is performed by adding a stimulus file which controls the inputs of the platform (here, these inputs consists in the clock and the PWM signal). The results are presented on the graphs of Figure 5.27. The plots of Figures 5.27a and 5.27b show the starting transient. The emulated signal matches the continuous time simulation obtained using Matlab with a good accuracy. The plots of Figure 5.27c and 5.27d, showing a zoom on the ripple of the inductor current and capacitor voltage, are even more interesting since they allow us to show how close the emulated signals are to the continuous-time results. We are able to closely track a high-frequency signal with a peak-to-peak amplitude equal to 2% of its average value.

The error on the amplitude of the ripple itself is equal to 3% in the worst case, as illustrated in the black box of Figure 5.27c. However, this error is mainly due to the fact that the highest point of the ripple is reached between two time-steps. The next value is very close to what it should be thanks to the averaging of the PWM signal along the period ( as described in section 5.4.6 ).

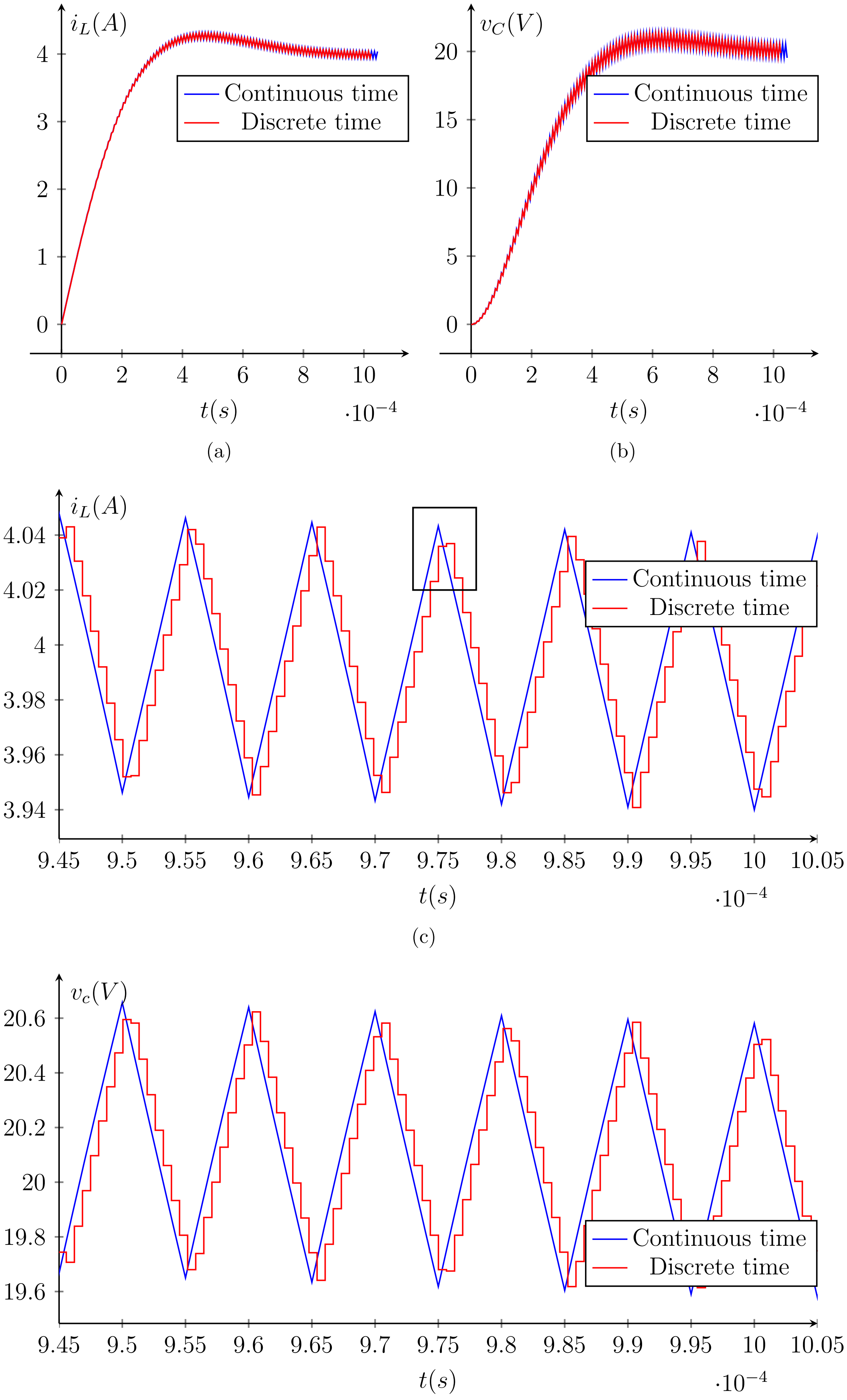
The delay between the continuous-time plot and the emulated plot correspond to the latency due to the computations performed at each time-step, and is effectively the smallest reachable time-step for this circuit. This delay is equal to *200ns*, which means that the design could run faster to produce more accurate results [11].

| Module | Time (ns) |
|---|---|
| State solver | 60 |
| Natural switching module : solver | 25 |
| Natural switching module : selectors | 40 |
| Selection of the (C,D) matrices | 30 |
| Output solver | 25 |

**Table 5.6: Latencies of the modules for the emulation of the converter of Figure 5.26**

Table 5.6 provides the breakdown of the latencies of the individual blocks. Almost one third of the delay is due to the state solver, while other dot products take fewer time. The reason for this is that, for this particular design, the state solver is the only module which requires full multipliers while the other product only need the simplified multipliers described in section 5.3.3. Indeed, as we previously said during the offline analysis performed in section 3.5.2, all the coefficients in the dot products are equal to 1, $-1$ or 0.

[11] Again, we are limited by the sampling rate of the chosen DACs, and accelerating the design would require faster converters with more complex interfaces



(a)

(b)

(c)

(d)

**Figure 5.27: Open-loop simulation of the circuit of Figure 5.26, with a 100kHz, 50% pwm input**

To obtain accurate numbers in terms of timing closure and resources occupation, we have run the complete synthesis process on the design (this step also generates the binary files that are used to program the platform).

| Module | Slices | DSP48 |
|---|---|---|
| State solver | 572 | 6 |
| Natural switching module Solver | 89 | 0 |
| Output solver | 77 | 0 |
| Forced switching module solver | 16 | 0 |
| Averaging of the state matrices | 120 | 0 |
| All modules | 1070 | 6 |
| Available on FPGA | 23000 | 180 |

**Table 5.7: Resources occupation of the largest modules for the emulation of the converter of Figure 5.26**

Let us first look at the resources usage, with the individual contribution of each submodule presented on Table 5.7. As expected, a circuit as small as a boost converter only requires a small part of the resources of the FPGA: only 1000 of the 23000 available slices are used, while the state solver only requires 6 of the 180 DSP multiplying slices to perform its function. Again, the fact that all other modules use simplified multipliers largely reduces the usage of this particular resources.

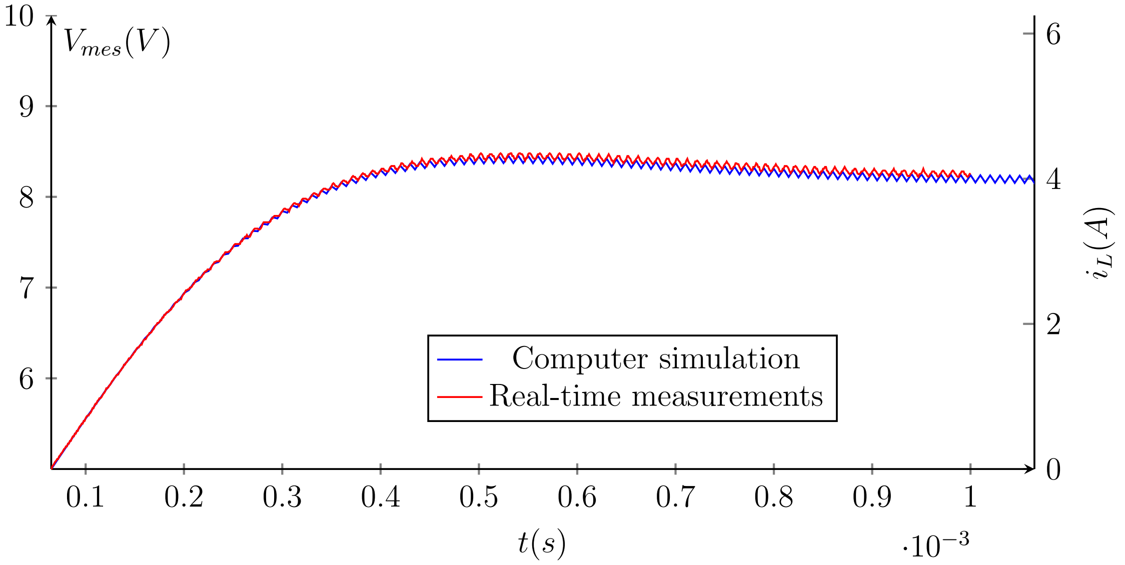
This is also why the state solver also requires the most generic slices. Most of these slices (502 out of 572) are used as buffers for the signal during the multiplication itself.

Let us now look at the timing closure results. In contrast with the resources usage, the compiler does not provide us with a precise rundown for each module, but only outputs the most critical paths (i.e. the paths that require the most time between two clock edges). For this design, the critical path only needs 4.3ns (out of a target clock cycle of 5ns). It should be noted however that this does not mean that this is the absolute best performance for the design, since the synthesizer does not try to find a better result once a design that respects the timing closure has been found. Still, we are left with a design that matches the requirements with a relatively large margin. Since the design has been developed with scalability in mind, we expect that these results should hold true for a large number of converters. We should also remind the reader that these results were obtained on a low-end (by 2014 standards) FPGA, and a more modern device should handle faster clock rates with ease.

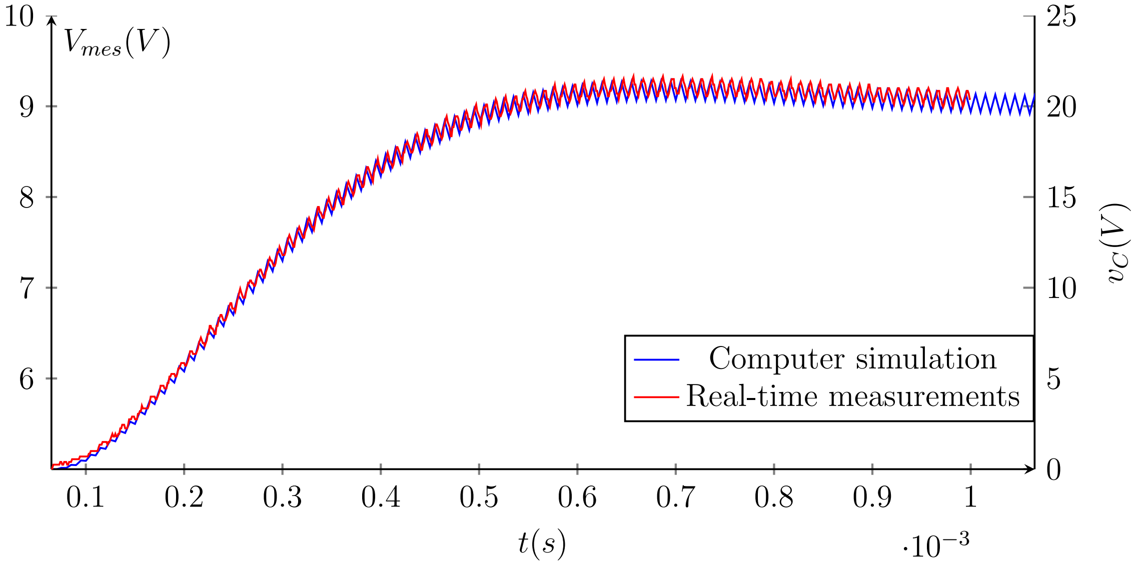
### Implementation on the physical platform

Since the design passes the timing requirements, it can be implemented on the actual Spartan 6 platform described in section 5.5.2. This is done by generating the bytecode corresponding to the complete design obtained after the place and route step.

A 100kHz PWM is connected to a digital input, and the outputs ($i_L(t)$ and $v_C(t)$) are measured at the outputs of the digital-to-analog converters (DACs) using a four-channels *Teledyne LeCroy* WJ314A oscilloscope. These converters translate the 14 most significant bits of the outputs signals into a voltage ranging from 0V to 10V, with 5V corresponding to a digital value of 0. Due to the multiple conversion factors in the platform, we have

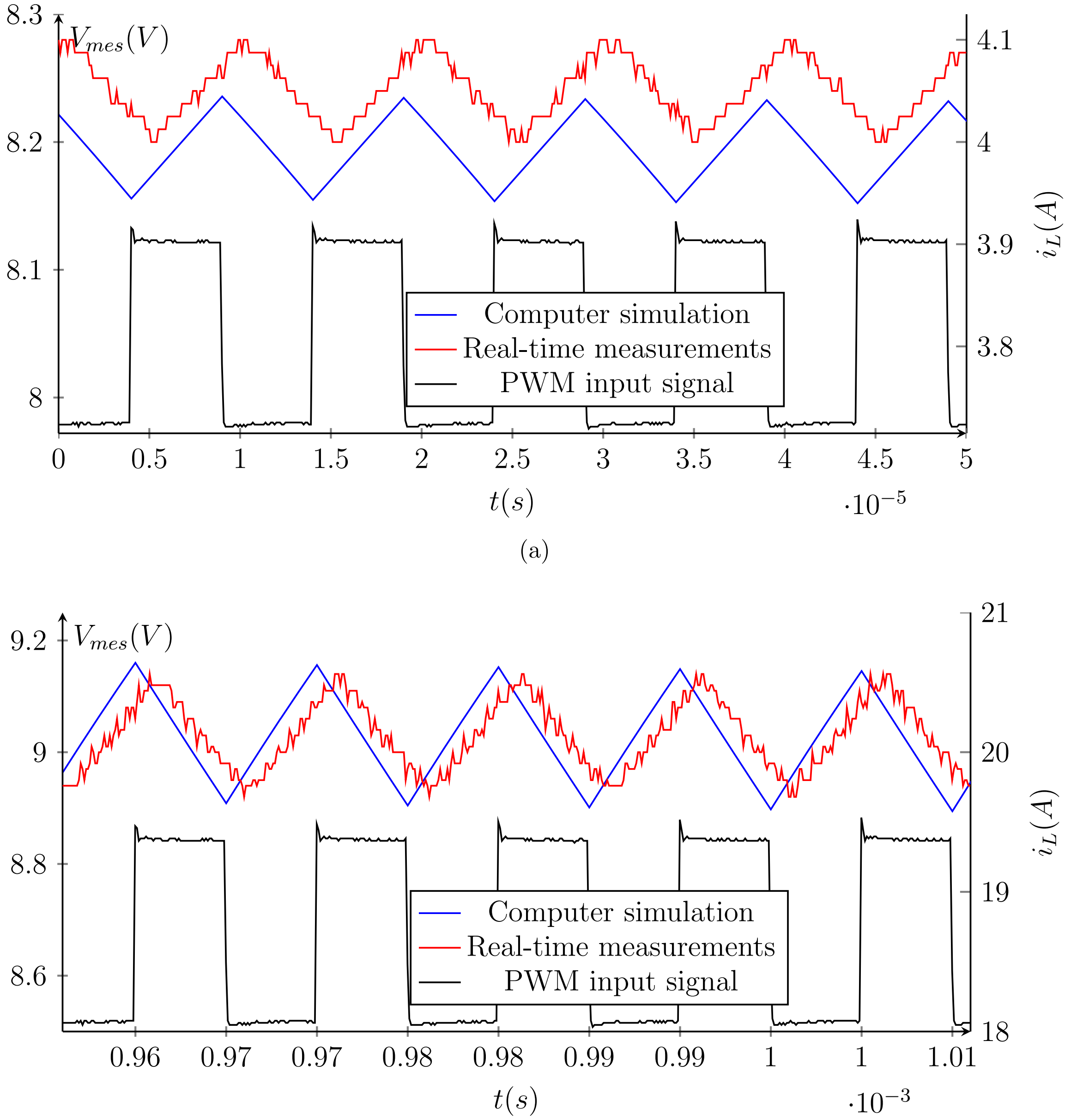


(a)



(b)

**Figure 5.28: Analog measurements at the outputs of the digital-to-analog converters of the real-time platform for the circuit of Figure 5.26, with a 100kHz, 50% pwm input : initial transient of the circuit when starting from $i_L = 0A$ and $v_C = 0V$. The measured voltage is translated into the actual value of the outputs by using (5.36).**



(a)

(b)

**Figure 5.29: Analog measurements at the outputs of the digital-to-analog converters of the real-time platform for the circuit of Figure 5.26, with a 100kHz, 50% pwm input : zoom on the commutations due to the PWM input signal. The measured voltage is translated into the actual value of the outputs by using** (5.36)**.**

the following relations between the measured voltage $V_{mes}$ and the actual values of the output signals :

$$\begin{aligned} i_L(t) &= (V_{mes}(t) - 5V) \times 1.25\frac{A}{V} \\ v_C(t) &= (V_{mes}(t) - 5V) \times 5\frac{V}{V} \end{aligned} \tag{5.36}$$

The start-up of the circuit is shown in Figure 5.28. As expected, the obtained waveforms are close to their theoretical values obtained using a computer simulation, even if a small offset appears at the tail end of the waveforms. Since the digital waveforms obtained in Figure 5.27 were more accurate and closer to the theoretical values than their analog counterparts, we can attribute this offset to

- a gain error in the analog chain
- a phase shift between the PWM signal used in the digital simulation and in the real-time process
- a loss of accuracy due to the fact that the 21 least significant bits are discarded

A zoom on the commutations events is plotted in Figure 5.29 [12]. On these plots, the offset between the analog signals and their expected values is clearly visible. The peak-to-peak amplitude of the signals show a much better accuracy, and are usable for most applications. The time-delay between the PWM signal and its effect on the analog outputs correspond to the total latency of the real-platform. This latency is composed of

- The latency of the simulator itself : 200 ns (see Table 5.6)
- The delay between a change in the PWM signal and the start of a new time-step : anywhere between 0ns and 640ns
- The time needed to transmit the data to the DACs : 350ns
- The settling time of the converters : 20ns

These results show that the communication delay between the platform and the DACs cannot be neglected. Indeed, this lag is equal to almost twice the latency of the simulator itself. To reduce this effect, we can use a DAC with a higher sampling rate (which can be expected to have a high-speed communication channel) or a DAC with a parallel interface, which avoids the need of sending the bits in series at the expense of a more complex interface.

### 5.5.4 Boost converter in closed loop

#### The effects of latency and time-discretization

When a digital platform is used to model the behavior of a continuous-time process, it invariably adds delays between a change in any of the inputs and the measured effects at the output when compared to the original plant. Examples of delays are

[12] Note : the plots corresponding to the computed simulations have been slightly time-shifted (when compared to 5.27) in order to synchronize them with the PWM input signals

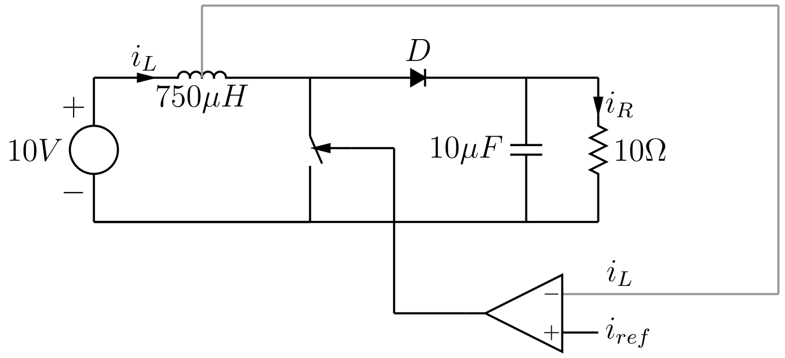


**Figure 5.30: Boost converter with a current mode control loop : the comparator controls the switch thanks to its internal hysteresis**

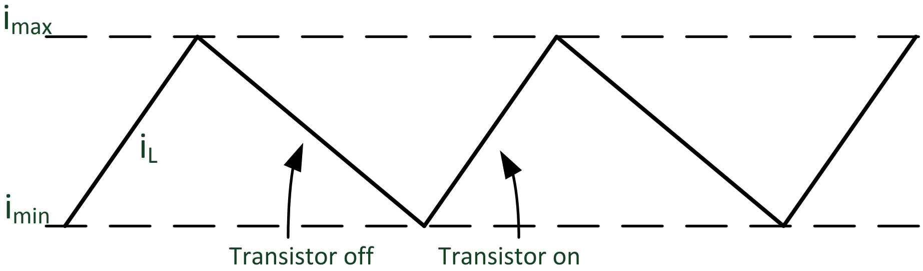


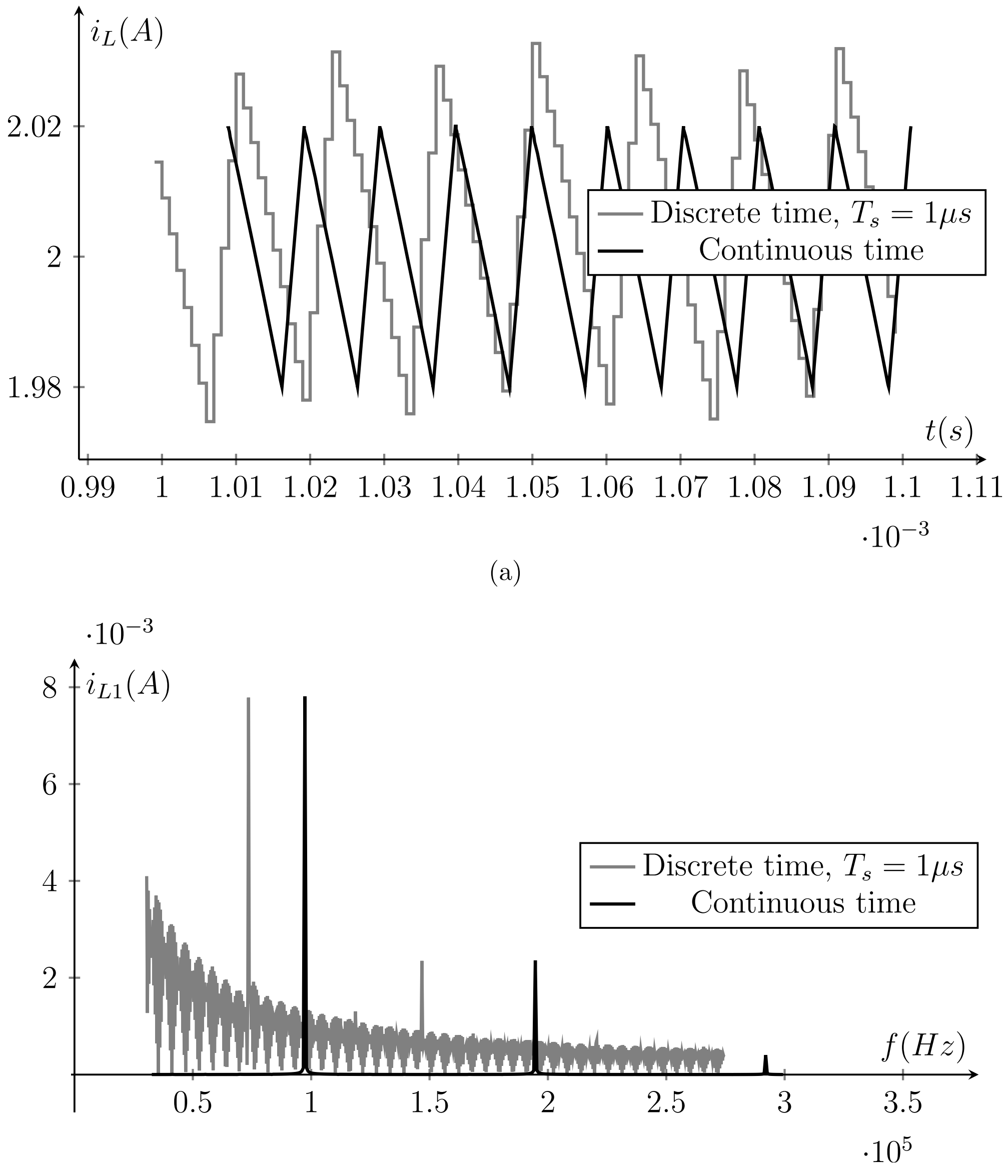
**Figure 5.31: Current mode control with hysteresis : the transistor is switched off when $i_L$ reaches $i_{max}$ and is switched on again when $i_L$ goes below $i_{min}$**

- the computing time of the loop itself
- the transmission time needed to send the signals to the digital-to-analog converters
- the DAC's conversion time

These delays are unavoidable, and can have major effects on the system when the platform is used in a closed-loop operation where a controller uses the measured signals to modify the inputs of the emulator. In practice, this delay introduces a phase shift in the signal, which degrades the phase margin of the closed loop system.

It should be noted that transmission delays also appear on the real converter, mainly due to the opto-coupler used to isolate the power-stage from the controlling system and the filtering of the switching noise. Even if the system did not present any computation delay, it would still be influenced by the discrete sampling time. We illustrate this point by using the circuit depicted in Figure 5.30. Here, a sliding-mode current controller ensures that the inductor current $i_L$ stays inside a defined hysteresis band by switching the transistor whenever the current goes above the upper limit or below the lower limit of the hysteresis band (Figure 5.31). This simple control loop is often used in power electronics because it is very simple to implement and insures that the current will not increase above safety levels.

If the power converter is emulated using a platform with a fixed time-step, we cannot guarantee that the current will remain inside the defined zone, as illustrated by Figure 5.32a where the current goes beyond both limits. The difference between the actual peak



(a)

(b)

**Figure 5.32: Effects of the sampling on the closed-loop accuracy of the sliding-mode control of $i_L$ in the circuit of Figure 5.30, with a hysteresis band going from $1.98A$ to $2.02A$ (a) Measure of the current (b) Fourier transform of $i_L$**

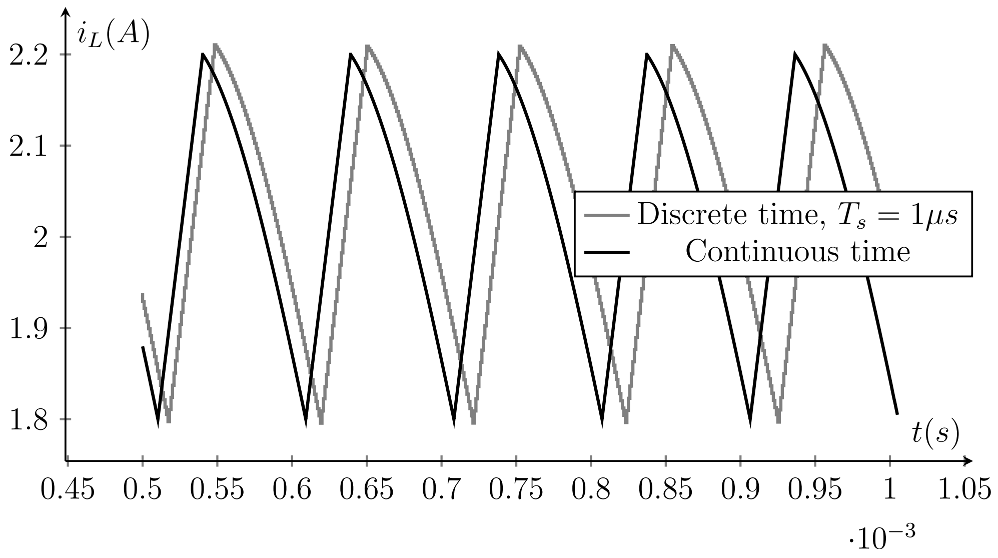


**Figure 5.33: Effects of the sampling on the accuracy of the sliding-mode control of $i_L$ in the circuit of Figure 5.30, with a hysteresis defined from $1.8A$ to $2.2A$**

value of the current and the expected upper limit $i_M$, in this case, equal to

$$\Delta i(k+1) = i_L(k+1) - i_M = \frac{\delta T_s}{L} E \tag{5.37}$$

where $\delta T_s$ is the fraction of the sampling period corresponding the overshoot. On average, this fraction will be equal to $\frac{T_s}{2}$.
Since the time spent in the same topology is increased, the equivalent switching frequency of the system is lower than on the real converter. Let us assume that, during each period, the converter spends $\alpha T_s$ with the transistor in the on-mode, and $\beta T_s$ with the transistor in the off-mode. The equivalent switching frequency of this device is

$$f_{sw} = \frac{1}{(\alpha + \beta) T_s} \tag{5.38}$$

Because of the discrete time, the time spent in the two modes is now equal to $(\alpha + \delta_\alpha)T_s$ and $(\beta + \delta_\beta)T_s$, and the switching frequency is lowered to

$$f'_{sw} = \frac{1}{(\alpha + \beta + \delta_\alpha + \delta_\beta) T_s} \tag{5.39}$$

Since $\delta_\alpha$ and $\delta_\beta$ can be assumed as random, the switching frequency will vary, leading to additional frequencies in the current spectrum, as shown in Figure 5.32b. Notice that the main frequency is $75kHz$ in the sampled system instead of $100kHz$. Of course, this effect is reduced when the time-step becomes much smaller than the switching period. Using a hysteresis band which is 10 times larger reduces the relative effect of the discrete-time as shown in Figure 5.33. Naturally, a faster design helps reaching higher switching speeds.
As a final remark, it should be noted that this effect is much less pronounced when using a controller that does not directly control the digital inputs. A widely-used example is a controller that modifies the duty-cycle of a constant-frequency PWM modulator using a filtered (using a low-pass filter) form of the converter outputs. Since these controllers have a much lower bandwidth than sliding-mode controllers, the error between the simulated signal and its real-world equivalent is similarly lower.

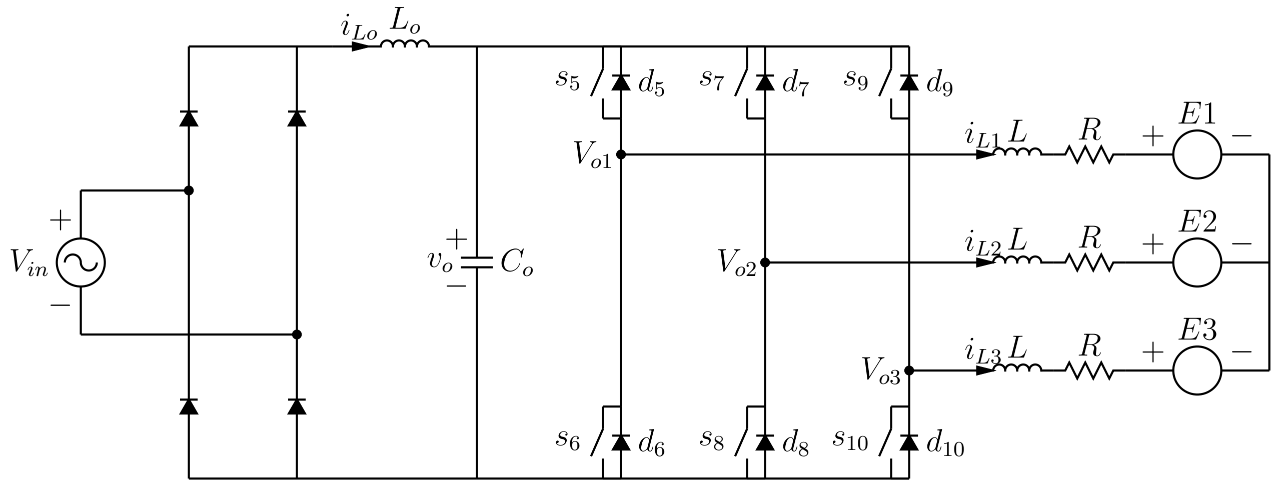


**Figure 5.34: A circuit composed of a passive rectifier followed by a three-phase inverter**

### 5.5.5 Implementation of a two-stage, three-phase AC-DC-AC converter

#### Introduction

While the boost converter used in section 5.5.3 was a good verification circuit, it is by no means a demonstration of the power of the emulation platform. Therefore, we will now show the results of the much larger converter circuit of Figure 5.34. This converter contains two stages : a passive AC to DC rectifier followed by a three-phase two-level inverter.

A total of ten diodes and six transistors are used, and we decided to use the partitioning algorithm to avoid the use of large switching tables. A first separation is performed by splitting the circuit between the decoupling capacitor $C_o$ and the inverter.

Our small FPGA cannot handle a three-phase inverter in a single piece, so we also split the second stage into its three branches and its load circuit, for a grand total of five partitions (Figure 5.35). The offline study of the inverter, along with the reasoning behind the choice of the partitions, is carried in section 4.4.3. Since the output inductors can go into discontinuous conduction mode, the law governing the controlled sources is topology-dependent, as explained in section 4.2.

The circuit is rather complex, and we will first study the implementation of the inverter alone without the rectifier. This will provide us with a validation of our simulation algorithm in presence of multiple partitions interconnected by variable sources. Once the inverter is validated, we will proceed with the emulation and the validation of the complete circuit.

#### Implementation of the inverter

The inverter has been split into the three switching legs and the interconnection circuit using the partitioning method described in section 4.4.3. Let us first study the legs. They are easily connected to the first power stage by representing the DC voltage $V_{in}$ using

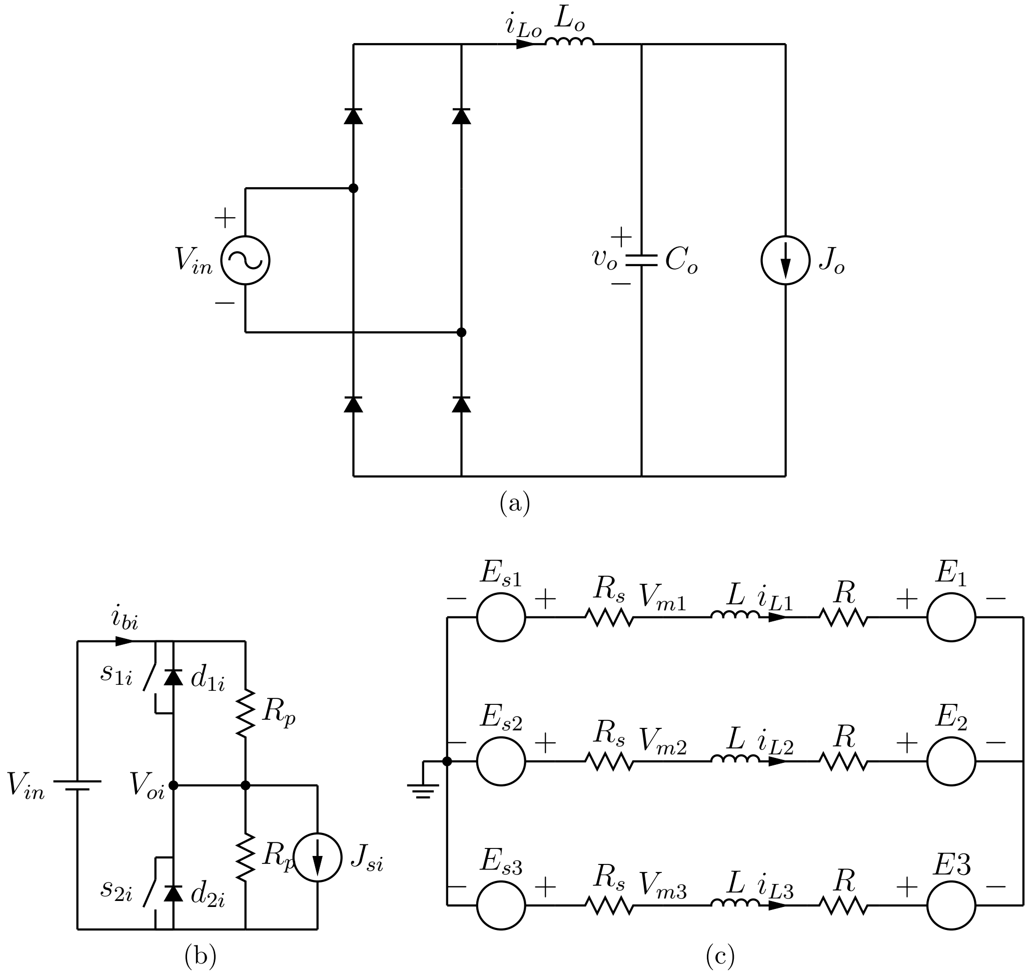


**Figure 5.35: The partitions corresponding to the circuit of Figure 5.34: (a) Input rectifier (b) Each of the three individual branches, where $i = 1, 2, 3$ (c) Load and its interconnection circuit**

a controlled voltage source equal to $v_C$. A measure of the input currents is needed to represent their effect on the capacitor. The connection with the interconnection circuit is more complex, since the output current can go in discontinuous current mode (DCM). Hence, we use sources with a variable control law: if at least one switch is in the on-state, this law is

$$J_{si}(k) = i_{Li}(k-1) + \frac{V_{in}(k-1) - 2V_{oi}(k-1)}{R_p} \tag{5.40}$$

If the leg is completely switched off, the law is changed to

$$J_s(k) = \frac{V_{in}(k-1) - 2V_{Mi}(k-1)}{R_p} \tag{5.41}$$

Meanwhile, the voltage sources $E_{s1}, E_{s2}, E_{s3}$ are equal to

$$E_{si} = V_{oi}(k-1) + R_s i_{Li}(k-1) \tag{5.42}$$

as long as the corresponding branch is active. This law is reduced to

$$E_{si} = V_{oi}(k-1) \tag{5.43}$$

when the branch is inactive. As explained previously, these relations have to be created manually for the time being.

To analyze the performance of the design, we will first emulate the inverter by forcing the input voltage source to a constant value of $1V$[13].The other parameters are

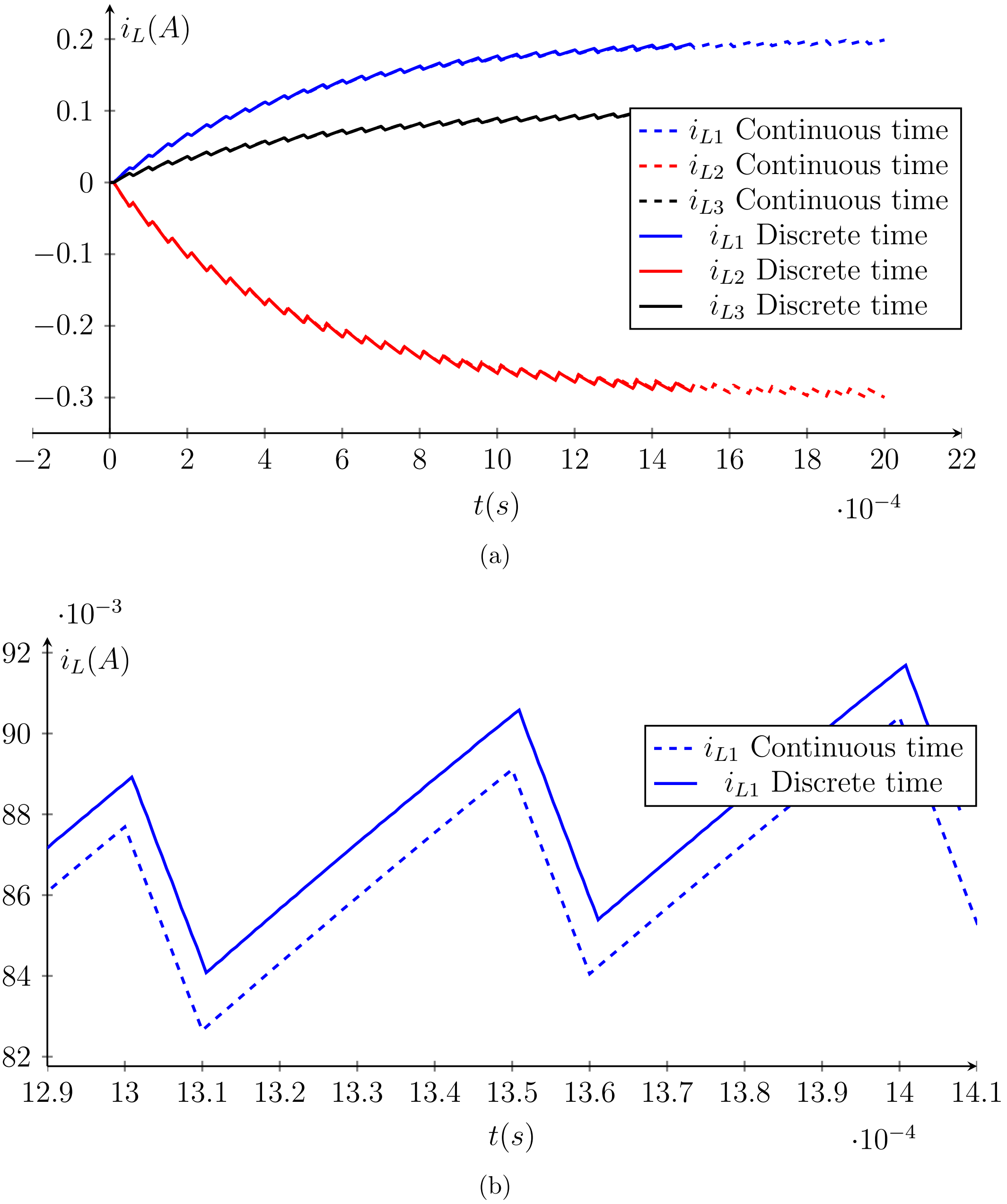
$$\begin{aligned} T_s &= 640ns \\ V_{in} &= 1V \\ R_p &= 1k\Omega \\ R &= 2\Omega \\ L &= 1mH \\ R_s &= 1560\Omega \\ E_1 &= -0.2V \\ E_2 &= 0.2V \\ E_3 &= 0V \end{aligned} \tag{5.44}$$

The value of $R_s$ is chosen so that the time constant of each branch is equal to the time-step, i.e.

$$T_s = \frac{L}{R + R_s} \tag{5.45}$$

As usual, the resulting outputs will be compared to a high resolution offline simulation performed in *Plecs*. First, we drive $s_5, s_8$ and $s_9$ with a $20kHz$ PWM characterized by a 80% duty cycle, while the other switches are kept in the off-mode. The evolution of the output currents is represented on Figure 5.36a, showing that the long term behavior

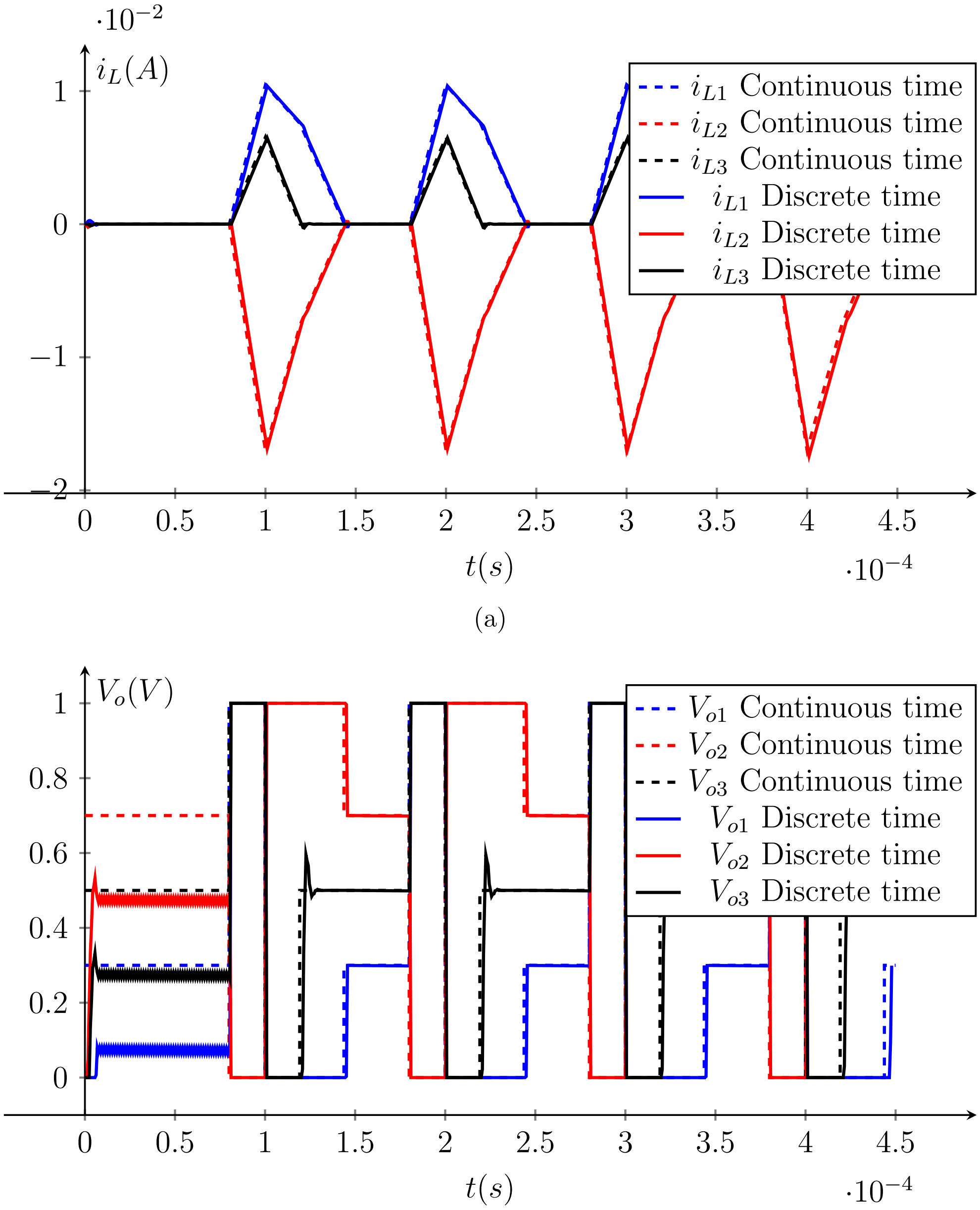
[13] A value of 1V is normally much too low for power electronics applications. In this case, it can be seen as a normalized value of 1 per-unit. Since the equations are linear with regard to the inputs, this value has no impact of the waveform other than adding a scaling



(a)

(b)

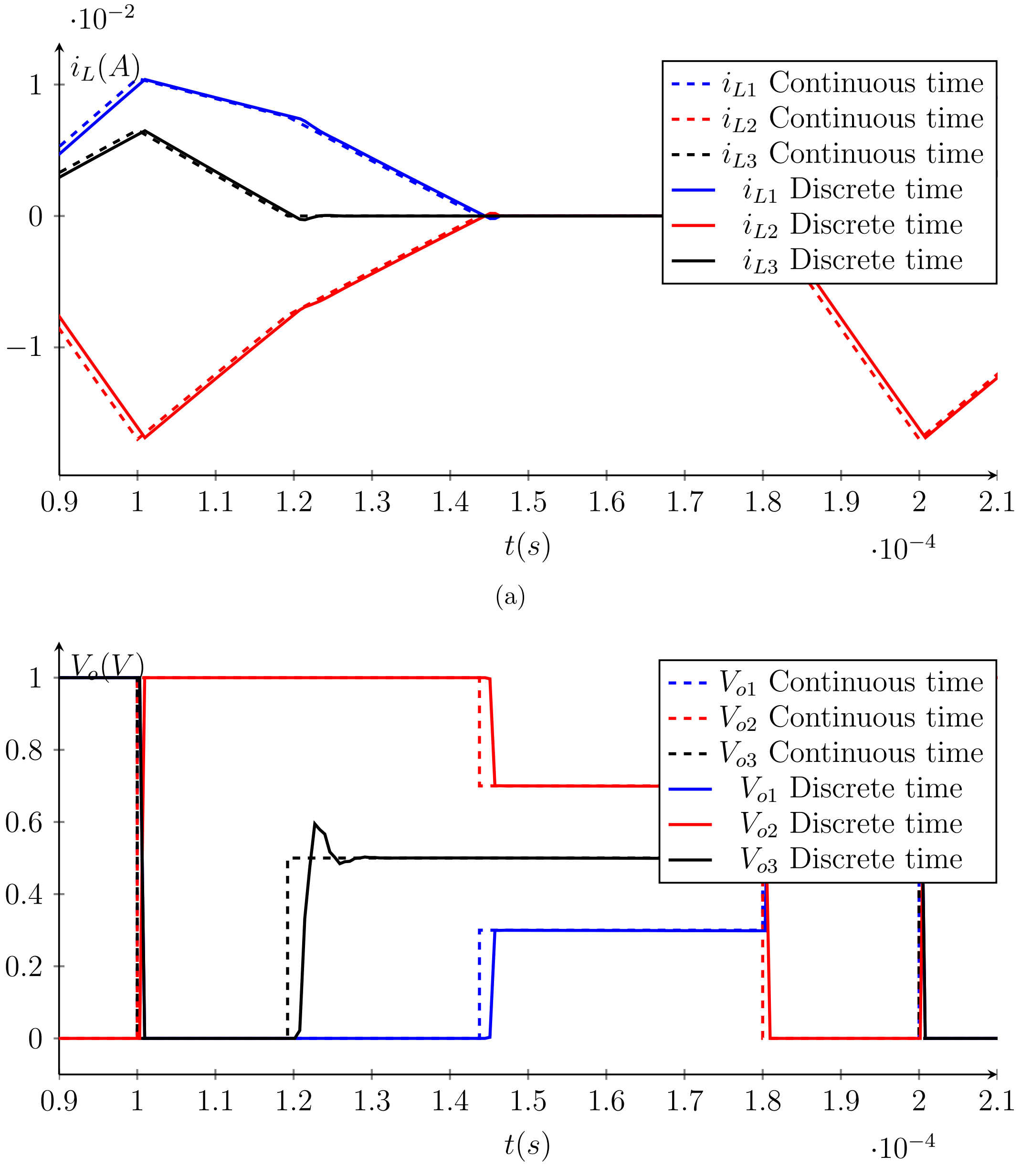
**Figure 5.36: Simulation of the three-phase inverter operating in continuous conduction mode**



(a)

(b)

**Figure 5.37: Simulation of the three-phases inverter operating in DCM. (a) inductor currents (b) leg voltages**



(a)

(b)

**Figure 5.38: Zoom on the simulation of Figure 5.37. (a) inductor currents (b) branch voltages**

of the emulated signals is a close approximation of the real behavior. This is confirmed by the plots of Figure 5.36b, showing the output current $i_{L1}$. The plot shows a constant offset between the emulated signal and its expected value, but this offset is equal to 0.5%, which is negligible for most applications. It should also be noted that the amplitude of the peak-to-peak is also very close to the expected value, which means that the DC and the AC parts of the signal are represented accurately.

To assess the performance of the emulation when the circuit operates in discontinuous conduction mode, we decrease the switching frequency to $10kHz$ and the duty cycle to 20%. The results are shown in Figure 5.37a and 5.37b, showing the output currents and the voltages at the output of the branches. The error at the start of the simulation is due to a bad initialization of the sources, which could be corrected in a future evolution of the platform. Nevertheless, these results are promising, as confirmed by the zooms of Figures 5.38a and 5.38b. Just like the offline simulations made in section 4.4, errors occur when a branch switches off: the output voltage takes a few clock cycles to converge, which induces a distortion in the current waveforms.

Since the emulation works as expected, we can now analyze the timing performance and the required resources. As shown in Table 5.8, the total time needed to solve a single step is equal to $250ns$. This result highlights one of the key advantages of our FPGA implementation : the discrete step-time is almost independent of the size of the circuit. Indeed, the computation time for the much smaller boost converter was previously measured equal to 200ns. Meanwhile, the resource occupation is shown in Table 5.9.

| Module | Time (ns) |
|---|---|
| State solver | 65 |
| Natural switching module : solver | 60 |
| Natural switching module : selectors | 35 |
| Selection of the CD matrices | 30 |
| Output solver | 60 |
| Selection of the sources matrices | 30 |
| Computing the sources | 60 |
| Complete Loop | 340 |

**Table 5.8: Latencies of the modules for the emulation of the three-phase inverter**

About a third of the LUTs and DSP slices are used by our design, confirming that the platform can theoretically handle larger designs. This design passes the place-and route step with a small margin ($4.9ns$ out of the maximum of $5ns$). Note that our tests have shown that this particular FPGA could not handle the non-partitioned version of the inverter due to large transmission delays in the design.

### Implementation of the complete circuit

Now that the inverter is validated, we are able to simulate the complete circuit [14]. Thanks to the partitioning algorithm, we only have to adapt the laws governing the sources. The

[14] Since the rectifying stage is only composed of a few elements, it will not be studied here.

| Module | Slices | DSP48 |
|---|---|---|
| State solver | 1200 | 12 |
| Natural switching module Solver | 1000 | 6 |
| Output solver | 1400 | 11 |
| Forced switching module solver | 500 | 24 |
| Sources computation | 1600 | 12 |
| Averaging of the state matrices | 250 | 0 |
| All modules | 7000 | 60 |
| Available on FPGA | 23000 | 180 |

**Table 5.9: Resources occupation of the biggest modules for the emulation of the three-phase inverter**

input voltage $V_{in}$ of the branches is equal to the capacitor voltage:

$$V_{in}(k) = v_o(k-1) \tag{5.46}$$

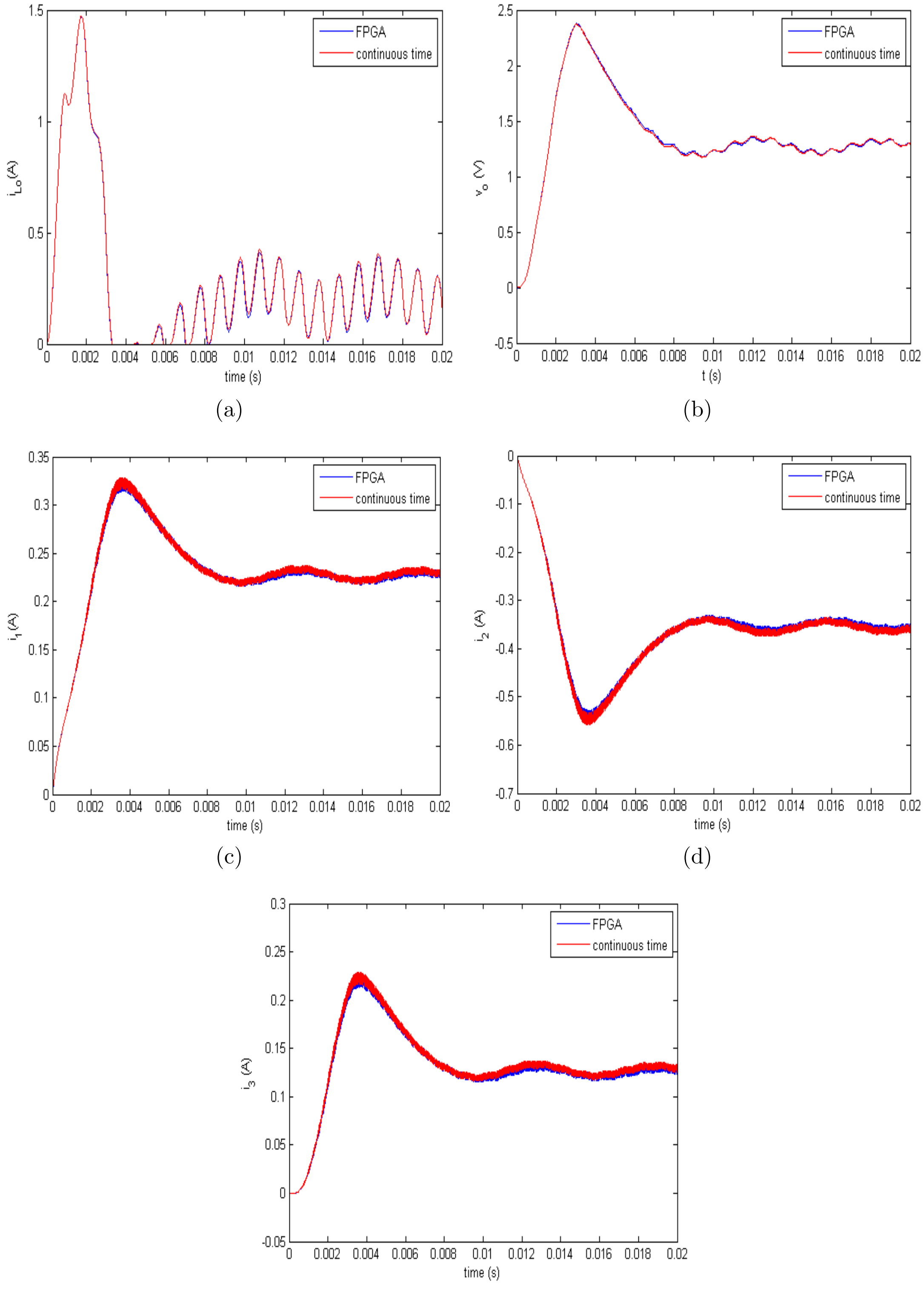
The influence of the branches on the capacitor voltage is modeled by the current drawn by the rectifier

$$J_o(k) = i_{b1}(k-1) + i_{b2}(k-1) + i_{b3}(k-1) \tag{5.47}$$

These sources are computed at each iteration. Unlike the internal controlled sources of the inverter, these control laws are static and have a very small footprint on the resource usage.
To simulate the whole circuit, we have selected the following parameters:

$$\begin{aligned} T_s &= 640ns \\ V_{in} &= 1V\sin(\omega t) \\ \omega &= 2\pi 500 \\ L_o &= 1mH \\ C_o &= 1mF \\ R_p &= 1k\Omega \\ R &= 2\Omega \\ L &= 1mH \\ R_s &= 1560\Omega \\ E_1 &= -0.2V \\ E_2 &= 0.2V \\ E_3 &= 0V \end{aligned} \tag{5.48}$$

The inverter is controlled by a $20kHz$ PWM on the switches $s_6, s_7, s_{10}$. The duty-cycle is equal to 80%. The first 20ms of the starting transient are represented in Figure 5.39, with zooms in Figure 5.40. These zooms are taken at the points where the relative error between the value computed by the FPGA and the expected value obtained using *Plecs* is the largest. As expected, the simulation is close to the expected value for most signals, with a relative error (corresponding to the absolute error divided by the maximum value of the signals) typically less than 2%. Unexpectedly, the largest error is found on the



(a)

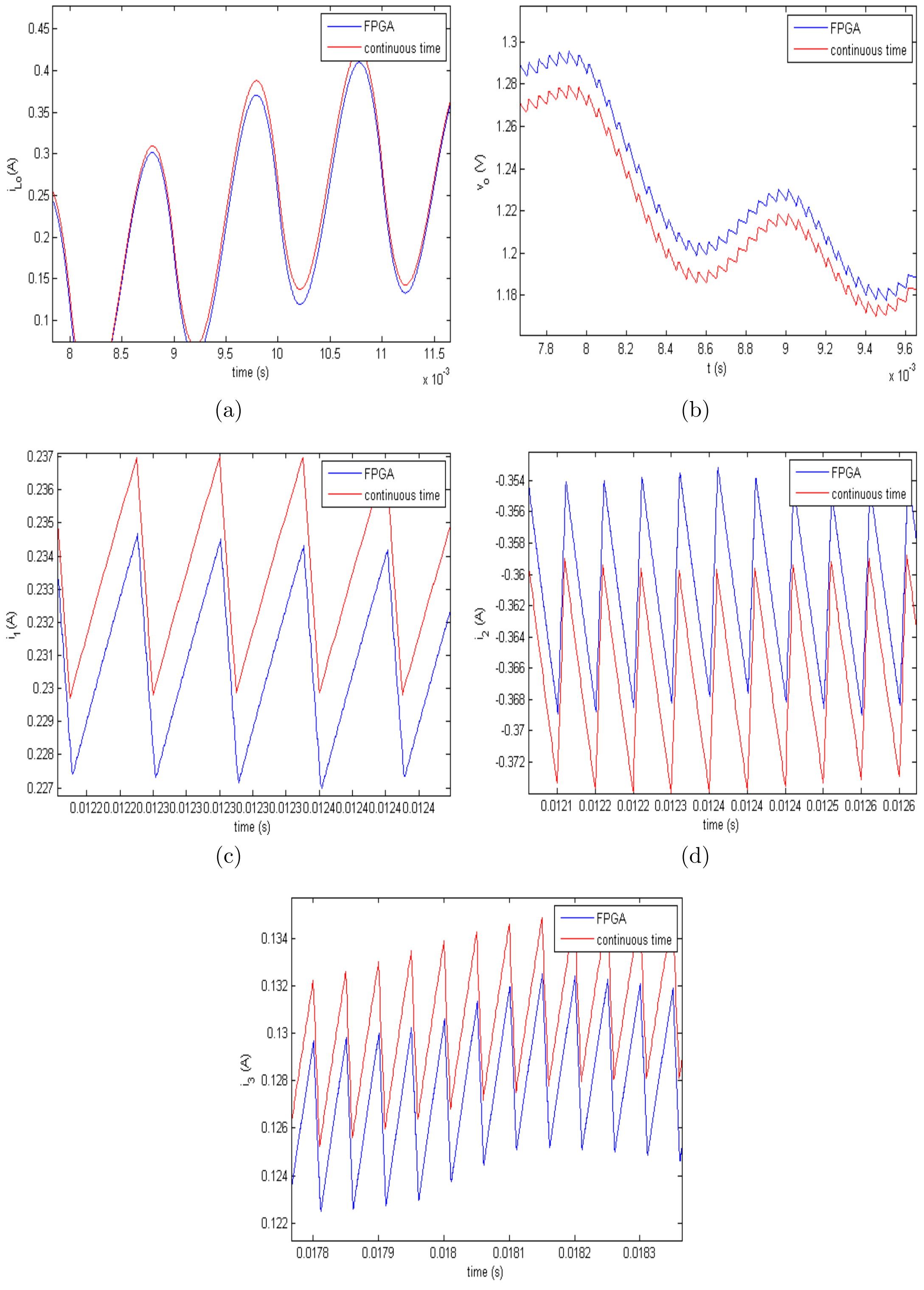
(b)

(c)

(d)

(e)

**Figure 5.39: Real-time simulation of the three-phase AC to AC circuit of Figure 5.34, with the parameters given by (5.48) and using a 20kHz sawtooth PWM to control the switches $s_2$, $s_3$ and $s_6$. The plots show the signals obtained by the FPGA and an offline high resolution simulation.**



(a)

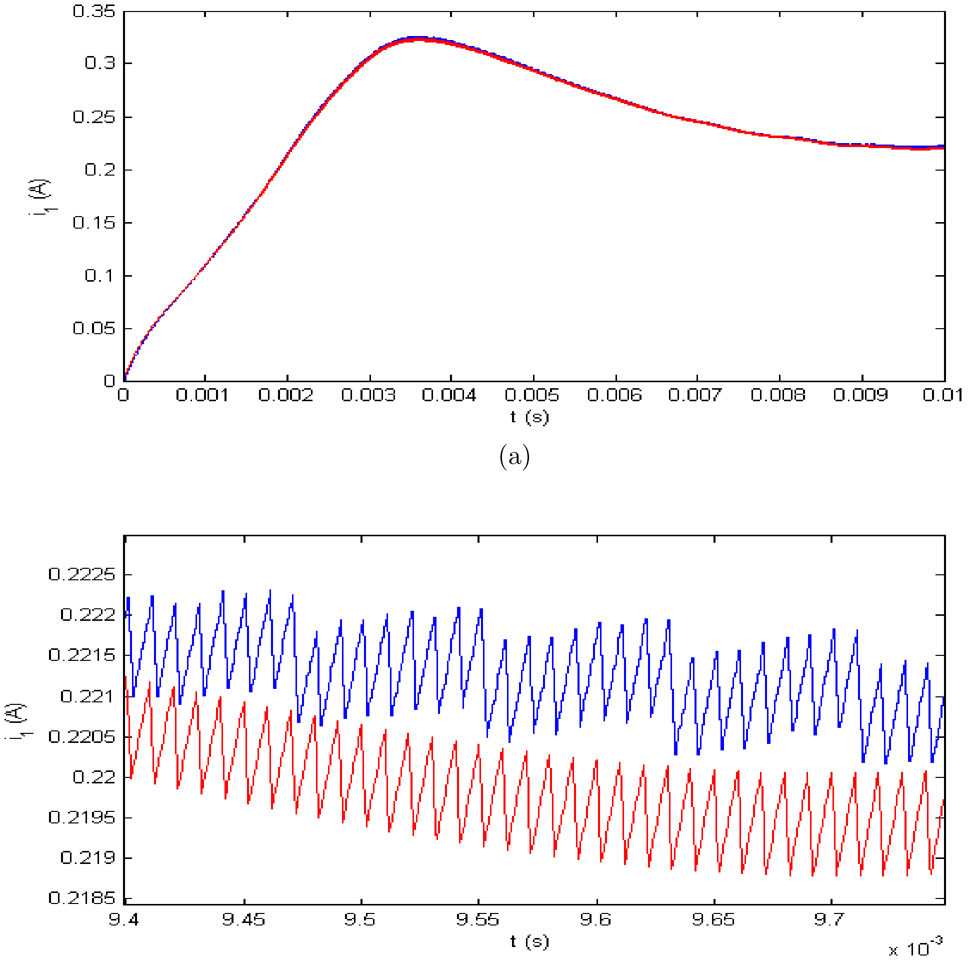
(b)

(c)

(d)

(e)

**Figure 5.40: Zooms on the plots of Figure 5.39, taken at the points with the maximum error between the FPGA signals and their expected value. The maximum error for the output currents is between 1% and 2.5%, while the error on $i_{Lo}$ reaches 5%.**



(a)

(b)

**Figure 5.41: Simulation of the two-stages circuit with a PWM frequency of 100kHz. The lack of averaging on the controlled sources introduces an additional periodic distortion**

input current $i_{Lo}$, where it can reach 5%. This is partially due to the large delay between the modification of a state variable and its effect on a signal placed at the other partition of the circuit. This is particularly true when a switch is commutated: to perceive the effect on $i_{Lo}$, we need to compute the following signals in sequence: $E_{s123}$, $i_{L123}$, $J_{s123}$, $J_o$ and finally $i_{Lo}$, for a total lag of four time steps. Nevertheless, these results should be good enough for most applications, even for high-frequency control. Indeed, while the signals may present an error of a few percents, the peak-to-peak variations of the signals due to the PWM switching are almost the same for the FPGA signals and their expected counterparts. In fact, these variations also present an error of around 1%. Let us now assume that the switching frequency is raised to 100kHz instead of 20kHz. The evolution of the current $i_1(t)$ is shown on Figure 5.41a, with a zoom available on Figure 5.41b. While the long term evolution remains very good, an interesting side-effect can be observed by looking at the current ripples. Indeed, there is now a visible periodic distortion on the current. This effect is explained by the fact that while the state matrices are averaged along the time-step to better represent the signals, the controlled sources are not. If a switch is changed right before the start of the new time step, only the new value will be taken into account even though its effect should have been minimal. Averaging the sources is certainly possible, and would be a welcome improvement to the platform. Nevertheless, we expect that the current results will be sufficient for most applications. Now that the functional performance has been validated, the last remaining step is to take a look at the timing performance and the spatial occupation on the FPGA.

Let us start by looking at the amount of time required by each module. The partial results are written in Table 5.10. These numbers are almost the same as those obtained for the inverter partition alone, showing again that our simulator is mostly independent of the size of the circuit.

| Module | Time (ns) |
|---|---|
| State solver | 65 |
| Natural switching module : solver | 65 |
| Natural switching module : selectors | 35 |
| Selection of the CD matrices | 30 |
| Output solver | 65 |
| Complete Loop (without the sources) | 260 |
| Selection of the sources matrices | 30 |
| Computing the sources | 65 |
| Total | 365 |

**Table 5.10: Latencies of the modules for the emulation of the complete circuit**

| Module | Slices | DSP48 |
|---|---|---|
| State solver | 2500 | 20 |
| Natural switching module Solver | 2600 | 11 |
| Output solver | 2200 | 16 |
| Forced switching module solver | 800 | 24 |
| Sources computation | 2100 | 13 |
| Averaging of the state matrices | 400 | 0 |
| All modules | 12000 | 84 |
| Available on FPGA | 23000 | 180 |

**Table 5.11: Resources occupation of the largest modules for the emulation of the three-phases inverter : around 50% of the resources are needed**

The same conclusion cannot be drawn for the utilization of the resources, as each partition requires its own selectors and dot products. The total resource usage should be close to the sum of the resources required by each partition, except for the more complex laws governing the controlled sources [15]. This is illustrated by the results of Table 5.11. The emulation of the complete circuit requires a bit less than 50% of the slices and multipliers. This design passes all timing tests, which means that it can be implemented on the platform and work as expected.

[15] Another source of variation is the HDL synthesizer itself. Indeed, this tool can decide to copy a single logical block multiple times to reduce the length of the data path.

While this means that even bigger circuits can be emulated with the platform, we must still be careful as larger designs also lead to complex logical circuits characterized by long propagation delays between its nodes. This event can be mitigated by adding buffers in the data paths, which is possible since the total computation time of the loop (365 ns) is lower than the time-step (640 ns). Of course, another solution is to simply use a faster FPGA.

### 5.5.6 Results analysis

The results obtained in the previous tests have shown that we are able to correctly emulate the behavior of two very different circuits. We will now analyze these results as a whole.

Let us first take a look at the precision of the emulated signals. We have shown that a relative error of less than 1% is perfectly reachable for circuits composed of a single partition, thanks to the use of the trapezoidal solver in combination with our switching algorithms. While the precision is lowered when multiple partitions must be interconnected using controlled sources, the error never went higher than 5%. Since the relative tolerance on many power components can be as high as 20% (especially for high-value capacitors and inductors), the error should be acceptable for most applications.

Some very interesting conclusions can be drawn from the comparison of the Tables 5.6 and 5.10, showing the latencies of each submodule contained in the main solver [16]. A first observation is the latency of the loop itself (without the computation of the sources) does not depend much on the circuit complexity and is only increased from 200ns to 260ns as we go from the boost converter to the AC-to-AC circuit. This is a direct consequence of the high parallelism developed for our algorithm. Indeed, thanks to the parallel computation of many operations and the high amount of buffering between the main modules, the time taken to compute (for example) all partial products of the dot products is independent of their amount. The summation of the partial product, made using an adder tree, is the only part of the algorithm impacted by the complexity of the converter. Even then, the use of four-way adders leads to a delay of $\log_4 n$ clock cycles for the summation of $n$ partial products. The reason why the output solvers and the natural switching solver take more time for the AC-AC than for the boost is that, in the boost, all multipliers could be replaced by a simplified version (as studied in section 5.3.3), resulting in a lower latency.

The use of many partitions forces us to additionally compute the controlled sources, which takes about 100ns more. In the current version of the simulator, these sources are computed before the start of each cycle, adding to the latency. However, it seems possible to modify the algorithm to compute these sources in parallel with the state matrices (i.e. in parallel with the same solver), which would solve this issue.

Nevertheless, this result is promising and shows that our algorithm is in theory able to handle larger circuits. The total latency could also be decreased by using a faster FPGA. For instance, we have shown in Table 5.2 that the multipliers can run between 30% and 100% faster when implemented on a FPGA belonging to the Artix 7 family. A discussion about the benefits of resorting to a high-performance FPGA is given in section 5.6.1. If

[16] Since the forced switching module runs in parallel with the solver, its latency does not count toward the total delay incurred by the system

we assume that all logic modules follow this pattern, the simple fact of using a faster platform would drastically improve the maximum sample rate.

The resources occupation of both circuits is shown of Tables 5.7 and 5.11. If the circuit contains a single partition, the complexity of the equations (and hence the resource usage) grows quadratically with the amount of states and sources (corresponding to the size of the matrix products), while the complexity of the selectors grows exponentially with the amount of switches (i.e. a circuit containing $n$ switches possesses $2^n$ topologies before reduction). However, the use of the partitioning essentially reduces this to the sum of the partial occupations of each partition, as well as the computation of the interconnection sources. If all partitions have roughly the same resource occupation, the total requirements are a linear function of the amount of partitions in the system. Notably, this means that going from a three-phase inverter to a five-phase inverter or to multiple parallel inverters fed by the same DC bus would not lead to a a drastic augmentation of resource usage. We should note however that very large circuits could require most of the resources of the FPGA, which complicates the place and routing process. In the worst case, it could even be impossible to route the FPGA. In that case, adding buffers in key places should be sufficient to properly reduce the routing delays.

### 5.5.7 Conclusions

In this section, we have successfully demonstrated the emulation of the two-stage converter of Figure 5.34. Because of its complexity, we have chosen to split it in five partitions (sub-circuits) which are simulated in parallel. We have shown that the emulated signals are very close to their expected value (obtained using an offline simulation). In many tests, the relative error is inferior to 1%, with the notable exception of the inductor current $i_{Lo}$ in Figure 5.40 presenting an error of 5% at times. Furthermore, we also have shown that the error on the amplitude of the ripples due to the PWM control is also very small for a switching frequency of 20kHz. A test with a frequency of 100kHz provides correct results but the lack of averaging on the controlled sources leads to additional distortion.

This is a crucial result which enables the use of control systems that directly drive the switches by measuring instantaneous (i.e. which are not filtered or averaged) values of signals) like sliding-mode controllers. An example was provided in section 5.5.3 with the sliding-mode (hysteresis) control of the input current of a boost converter.

This test highlighted one of the limitations of the digital emulation of any circuit: because of the discrete time-step, we are unable to find the exact time at which the current crosses the hysteresis threshold, and an over(under)shoot outside of the specified limits may appear. This implies that the closed-loop control of the circuit will not be an exact representation of the system. This distortion is not a result of our specific algorithm, and is present in any emulator characterized by a discrete time step. Similar effects can be observed on a real converter when opto-couplers, which also add a delay in the measurement, are used to insulate the power circuit from its control system.

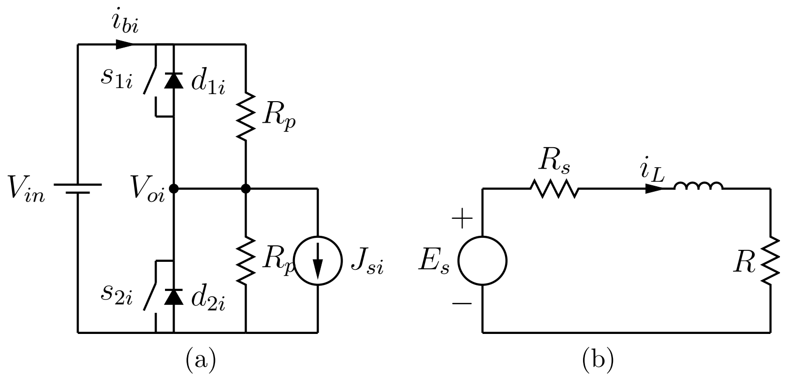


**Figure 5.42: A single-phase inverter partitioned into two sub-circuits. Because of the law controlling $E_s$, this source can increase to $1000V$ even if $V_{in} = 10V$ and $i_L = 1A$**

# 5.6 Possible improvements

The results provided in this chapter show that the real-time platform is a useful tool. Nevertheless, there are a number of improvements that could be brought to enhance its performance and/or its features.

## 5.6.1 Performance and accuracy improvement

### Floating point arithmetic

In its current version, the real-time platform uses fixed-point arithmetic to compute and store all signals. This procedure has the advantage of being fully compatible with the DSP slices of most FPGAs, and only requires an additional clock cycle to shift the result by the correct amount of bits (see section 5.3.1). However, it also introduces a number of limitations in the system.

The most severe limitation is the reduced range of values for the internal signals. Because the amount of bits assigned to the fractional and integer part of the signals is constant, we cannot have a high resolution and high values stored for the same signal.

If we want to make the best use of the 35 bits, we have to know their range beforehand, which means that the Q-factor must be recalculated when the application is changed. For some signals, this range is not evident when looking at the circuit. For instance, let us look at the partitioned circuit of Figure 5.42. As a reminder, the resistance $R_s$ is added to ensure a fast decay of the current when the switching branches turn off (see section 4 for the complete partitioning theory), and is chosen to verify the following equality:

$$T_s = \frac{L}{R_s + R} \tag{5.49}$$

where $T_s$ is the system time-step. This choice ensures the fastest possible convergence. When either switch of the the branch is on, the effect of the resistance is counteracted by applying the following law for the controlled source $E_s$

$$E_s(k) = V_o(k-1) + R_s i_L(k-1) \tag{5.50}$$

Assuming $L = 1mH$ and $T_s = 1\mu s$, we have $R_s = 990\Omega$. Let us further assume that $V_{in} = 10V$ and $i_L(k-1) = 1A$. According to (5.50), a value of $1000V$ is expected for $E_s$. This high value must be taken into account when selecting the Q factor of the signals.
For these reasons, transforming all computations to their floating-point equivalent would lead to a better compatibility without requiring to compute the range of the signals and matrix coefficients. As explained in section 5.3.1, the downside of this representation lies in the additional clock cycles and resources required to perform the additions and multiplications. If sacrificing these extra clock cycles is a viable option (i.e. if we have a sufficient amount of time between the end of the computation of a time-step and the beginning of the next step), converting to floating-point should not have any negative impact on the latency.
It should also be noted that FPGAs with integrated hardware floating-point units have started to appear on the market in 2014 [93], and should be standard in a few years. Hence, converting the design to floating-point will allow us to quickly use these new devices to the best of their capacity.

### Using a faster device

Of course, the easiest way to have faster design is to use a high-end up-to-date FPGA instead of the *Spartan 6* FPGA used in this chapter. We benchmarked the performance of the multiplier implementation for the Spartan 6 and more recent Artix 7 FPGA in section 5.3.2, with the results available on Table 5.2. A speedup of up more than 50% is observed when using a fully pipe-lined design. This result means that we are able to use a faster clock to feed the device, which in turn reduces the total time required to compute a single step, even if the amount of clock cycles remains the same.
These devices also typically integrate a larger amount of logic and DSP slices, which in theory allow us to emulate larger circuits. For example, the largest Artix 7 devices hold up to 215.000 slices, nine time as many as those present in the Spartan 6 [44]. And we should also remember that the Artix line is actually *at the low-end* of the current line-up proposed by the multiple FPGA vendors. While the maximum theoretical speed of the components is the same for both the Artix 7 and the top-of-the line Virtex 7 family (540 MHz for the DSP slices, 530 MHz for the IO and RAM slices and 1.8GHz for the LUT), the higher density and the faster inteconnection network of the Virtex family means that the delays due to the communication between the modules are reduced. In practice, we can expect to reduce the latency by 25% to 50% by switching to a high-end FPGA.
Another feature proposed by modern devices is the integration of a microprocessor in the same chip as the FPGA. The addition of a generic purpose processing unit (typically based on a ARM controller) would enable us to add many nice-to-have features, such as complex source generators, time-based triggers or an on-board memory controller (to save the waveforms on a SD card for example) in far less time that we would have needed to implement them in HDL.

### Averaging the controlled sources

In section 5.4.4, we introduced the averaging of the state matrices as a solution to avoid the loss of precision due to the occurrence of switching events inside a time-step. We have shown that this additional step allow us to correctly model the effects of high frequency (i.e. up to about one tenth of the sampling frequency) switching at the cost of additional logic to perform the averaging.

In principle, the same algorithm could be used to obtain a better representation of the controlled sources that are used to model the dependencies between the partitions of the circuit (see chapter 4). For the moment, no additional treatment is performed beyond selecting the correct control law according to the current mode of the switches.

As shown during the emulation of the three-phase AC/AC circuit in section 5.5.5, and more particularly in Figure 5.41, this lack of averaging induces a distortion in the signals. The effects become more pronounced when the switching frequency becomes higher.

Adding an averaging step during the evaluation of the sources should prevent this error, or at least reduce its effects.

#### Extended parallelism

The biggest strength of the FGPAs lies in their extensive capabilities as parallel-computing platforms. Indeed, there are basically no limitations on the amount of slices that can operate at the same time beyond the need of ensuring that all the processing is done before the start of a new clock cycle.

Since the number of computations required for the time-step is fixed, it makes sense that performing all operations in parallel reduces the total latency, and consequently allows for a shorter time-step. We have already discussed this point when we introduced the two different software architectures in section 5.2, but a deeper analysis should be performed to analyze the effects of the two structures on

- the total latency
- the resource usage
- the quality of the waveforms

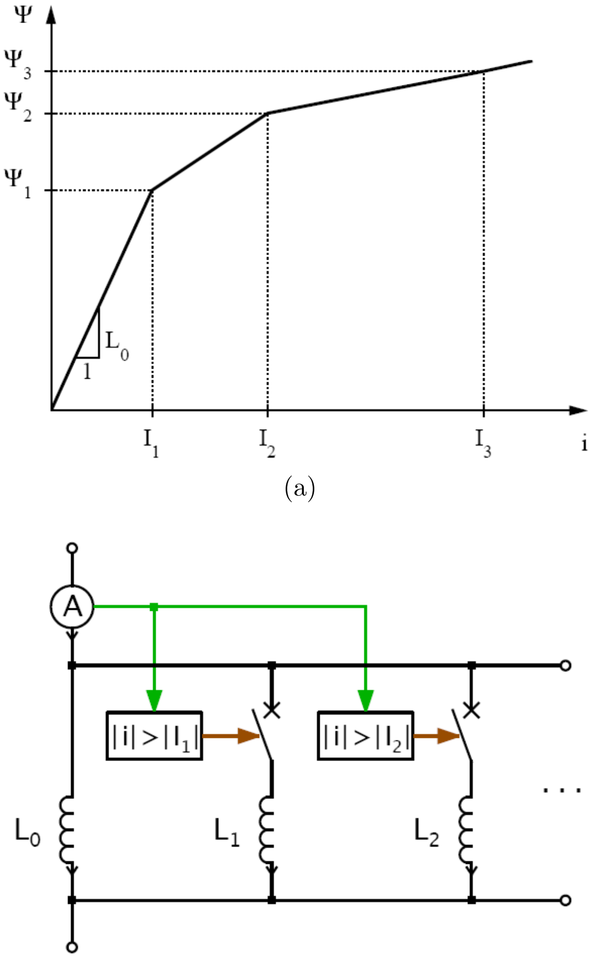
The disadvantage of performing additional steps in parallel (for instance, computing the outputs in parallel with the states) is that we have we do not have access to the most value of the signals, and have to use their previous value instead. Since the state of the diodes partially depend on the outputs of the system, this lag could introduce distortions in the signals since there is a possibility that the new value of the states is computed for an incorrect topology. Again, this effect should be reduced when the time-step is decreased, which is fortunately helped by the use of a higher degree of parallelism.

### 5.6.2 Additional features

#### Handling non-linear equations

In this thesis, we have worked under the assumption that the converters may be represented by piecewise-linear equations. For these systems, we can effectively split their dynamics into different modes linked by the hybrid automaton introduced in 2.5.

However, some elements cannot be put in this form. One example is the AC drive studied in section 2.4.5, which introduces trigonometrical variables in the state-space equations. Since these variables must be evaluated in real-time, we have to introduce a non-linear



(a)

(b)

**Figure 5.43: A saturable inductor transformed into a piecewise-linear model. (a) Flux-current relationship (b) electrical equivalent (taken from [101])**

solver if we ever want to emulate the dynamics of the AC machines. These kinds of solvers are much more complex than linear ones, and often have convergence and stability problems. They also typically require multiple passes at each time step in order to converge.

Some non-linear components can be modeled using piecewise-linear relations. For example, the relationship between the current and the flux inside a saturable inductor may be represented by the piecewise-linear function described in Figure 5.43a. Since the voltage on an inductor is given by $v(t) = \frac{\mathrm{d}\Psi}{\mathrm{d}t} = L\frac{\mathrm{d}i}{\mathrm{d}t}$, this relation also defines the effective inductance of the component. The saturable inductor can be transformed into an equivalent circuit, as depicted by Figure 5.43b.

### Modifying the components at run-time

One of the limitations of our system is that the value of the components have to be fixed before the analysis and the compilation of the circuit, and cannot be changed afterward. This means that changing the value of a single resistor implies that we have to go through the whole design flow again. Giving the user the possibility of changing the value of the electrical components *after* the synthesis, or better yet during the simulation, would provide a faster feedback loop allowing him or her to perform more tests in any given time. Of course, adding this capability is probably easier said that done since changing any component has a good chance of having an impact on all parts of the emulator (i.e. its value intervenes in most of the system matrices). Hence, adding the on-line modification of the circuit would require an updating mechanism able to refresh all the matrix multiplexers at once [17].

A more difficult problem to solve is that the coefficients, which were previously constants, must now be considered as variable signals. When a signal is constant, the synthesizer will typically directly assign each of its signals to a fixed one or zero, according to its binary value. Such an assignation is made locally and as close as possible to the slices where the signal is used, which means that there is virtually no transmission delay to propagate its value.

However, storing a variable signal requires the use of a register or a memory. Additionally, some of the registers could be duplicated at multiple places by the synthesizer to reduce the length of the transmission paths, which means that, at the end, we might have to change multiple signals in various places of the design. These added registers could also have an impact of the global performance of the circuit, since more signals will have to be routed across the FPGA.

Still, adding the capability to easily change the configuration of the circuit would provide a very useful enhancement to the platform, and should be one of the main paths to follow.

### High-speed data link

As a corollary of the previous point, a good way to enhance the integration of the platform would be to provide some sort of high-speed transmission channel between the FPGA and the outside world. As discussed previously, this data link could provide a way to modify some of the parameters on the fly, but many applications could be developed. A first example would be to provide a digital oscilloscope or data-logger by sending the signals to a computer. While these values are already available at the analog outputs through the digital-to-analog converters, this provides a much easier way to store the data for later usage (debug, access to intermediate values, post-processing, black-box,...).

Another way to use this data link would be to interconnect multiple FPGAs, each of them emulating one partition of a larger circuit. This concept can be seen as a natural evolution of the partitioning algorithm accross multiple boards. If we are able to compute and send the signals in a single clock cycle, the results would be exactly the same as the traditional partitioning.

This way, we could store each partition on a different board, and interconnect them when needed.

[17] However, we could probably require that such a change would only be effective after a restart of the system, which solves some of the synchronization issues

## 5.7 Conclusions on the real-time platform

In this chapter, we have exposed the different facets of the real-time platform built in the frame of this thesis. Throughout the design, we have used the extensive parallelism proposed by FPGAs in order to reduce the latency (i.e. the minimal solver time-step) of the system as much as possible. We have also shown that using the parallelism in all parts of the design results in a latency that is only marginally dependent on the size of the circuit as long as the synthesis tools are able to reach timing closure, which means that the output rate can be sustained even for circuits containing many elements and switches.

We have presented the optimizations performed at each step to reduce the resource usage. In section 5.3.2, we have shown that even a simple multiplier can be significantly optimized depending on the situation, and different implementations of the selection tables have been studied in section 5.3.4.

To improve the accuracy of the simulation when the switching frequency is not significantly lower than the sampling rate, we have introduced in section 5.4.4 the oversampling of control signals and the time-averaging of the state matrices to take into account switching events occurring inside the time step. The improvement is significant: a 10/1 ratio can be reached between the computing frequency (i.e. the inverse of the time-step) and the switching frequency, and results have been shown for a switching frequency of 100kHz

Finally, we have validated our platform through multiple tests in section 5.5. These tests have allowed us to evaluate the performance of the emulator in open-loop as well as in closed-loop. In open-loop, we have validated the system by first using the simple boost converter in section 5.5.3, then by implementing a much more complex three-phase AC-to-AC converter 5.5.5. This circuit was built using the partitioning algorithm introduced in chapter 4, and we have shown that even a low-cost FPGA can emulate a relatively large circuit with a typical relative error (defined as the the absolute error divided by the maximum value of the signal) of 1% to 2% provided the ratio between the sampling frequency and the switching frequency does not drop below 10 for a non-partitioned circuit, or below 30 for a partitioned circuit operating in continuous-conduction mode. The ripple due to PWM switching is quite similar to what can be obtained with off-line simulators dedicated to power converters. The benefits of our optimizations were also made manifest during the same test, since the test circuit requires less than one half of the resources provided by the FPGA. We can infer that much larger circuits can be emulated on the more modern lines of FPGAs, which typically integrates ten times as many resources.

The current system still has its limitations, as the error increases for frequencies of 100kHz when using the partitioning algorithm. Because most of the loss in accuracy happens when we split the power converter in multiple partitions, optimizing this part of the algorithm (for instance, by also averaging the controlled sourced) would be an improvement.

Thanks to the analysis of the results performed in section 5.5.6, we have shown that the total latency of the system does not vary much with the size of the emulated circuit. Hence, we can reasonably expect that the emulator will be able to emulate many circuits with the same sample rate. However, the size of the circuit has a much more direct impact on the amount of required FPGA resources with, which is fortunately reduced by the partitioning procedure to a sum of the individual contributions of each partition.

While there are many ways to improve the platform, which have been highlighted in section 5.6, we can confidently conclude that our current design can be used to emulate a wide range of power converters with a resolution of 1 mega-samples per seconds or more and with relative error of few percents on the emulated signals. This important result will now allow us to proceed to the global conclusions of this thesis.

# Chapter 6

# Conclusions

Throughout this work, we have developed a new framework that allows the rapid prototyping of a wide range of power converters by translating a circuit drawn on a computer to a real-time equivalent set of equations and conditions which is processed by an FPGA. This framework takes the form of a tool-chain (see Figure 6.1) that successively

- Extracts the basic information (state matrices, types of switches and sources, ...) from the initial circuit drawn in a typical electronics simulation program. Currently, the framework uses the offline circuit simulator *Plecs*® for these first operations.
- Analyzes and simplifies the dynamic changes that may appear in the power circuit when the internal signals and the inputs and/or modified
- Compiles the result into VHDL source code
- Makes the synthesis of the FPGA and transfers the binary code to the real-time platform

Each of these steps will be reintroduced and concluded on in the following sections.

The simulation of electronic circuits is not a new subject, and many works have paved the way before us. This allowed us to envisage multiple ways to put a circuit into equations in chapter 2. During this study, we extensively compared the two main mathematical representations of linear circuits, the modified nodal analysis (MNA) and the state-space analysis (SSA). Since power converters typically include diodes, transistors and other binary switches leading to sudden changes in their topology, we also made a survey of the different ways of improving the linear representation in order to take these transitions into account. The conclusion was that both methods had a number of advantages and drawbacks:

- The modified nodal analysis, augmented by the current source model for the switches, provides a very simple framework since any topological change is performed by simply changing the control law of the source while keeping the system matrices constant. On the other hand, we have also shown that this representation is far from a minimal model of the system, an intrinsic drawback of the MNA which is even aggravated by the fact that each switch is essentially modeled by a capacitor

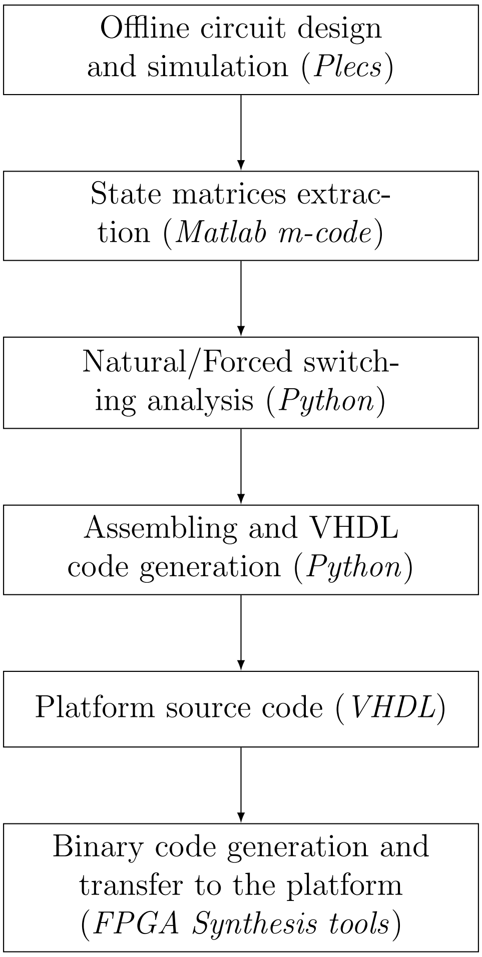

Offline circuit design and simulation (*Plecs*)
State matrices extraction (*Matlab m-code*)
Natural/Forced switching analysis (*Python*)
Assembling and VHDL code generation (*Python*)
Platform source code (*VHDL*)
Binary code generation and transfer to the platform (*FPGA Synthesis tools*)

**Figure 6.1: The complete design flow of the quick prototyping procedure. Steps 1 and 2 are described in chapter 2 while step 3 is the result of chapters 3 and 4. Finally, steps 4 and 5 were the main subject of chapter 5**

or an inductor depending on its state (on or off). Hence, even moderately large circuits will require a large amount of resources (multipliers, memory elements, . . . ) to properly emulate its behavior.

- The state-space analysis allows us to use ideal on/off switches in the circuit, which drastically reduces the complexity of the underlying equations. Furthermore, the inherent compactness of the SSA leads to a reduced set of equations which are easier to compute. The addition of the hybrid automaton concept to the SSA provides an elegant framework to accurately track the dynamics of the circuit in response to both continuous changes (i.e. induced by the state equations) and discrete ones (due to a change in the control signal of the switches). On the other hand, this representation is penalized by the exponential growth of its complexity as a function of the amount of switches in the circuit. While storing all these variations and selecting the correct one in real-time is certainly feasible for smaller circuits, it becomes quickly impossible to handle when the circuit grows in size.

A solution had to be chosen and we estimated that the advantages SSA made it more-suited for a real-time implementation. Our objective then became to optimize it as much

as possible, to identify its limits and try to overcome them. This lead us to our first major contribution to the state of the art: a way to automatically analyze the hybrid automaton associated to the circuit in order to reduce its complexity as much as possible while keeping all the necessary information intact (see chapter 3).
We introduced two tools, based on similar principles, to study

- the natural changes in the state of the diodes due to the evolution of the continuous signals (for example, a diode switching off when its current becomes negative)
- the forced changes that happen in reaction to the commutation of a controlled switch such as a MOS transistor. A frequent example is the free-wheeling diodes, which have to switch on to provide a path for an inductive current.

The aim of both tools is to answer the same question : *Knowing the current state of the system, what are the different possible topologies reachable in the immediate future ?* This question, known as a reachability problem, has no universal resolution, hence the introduction of our own solution.
The first or our two tools, the natural switching module, parses the transition graph of the automaton and finds all the topologies that can be reached when the system starts from a given topology. Since the original graph can be relatively large, we also introduced multiple ways to reduce the number of acceptable branches by studying the conditions (or guards) associated with each transition. By eliminating all the changes whose guard cannot be physically verified, of by fusing together equivalent branches, we have shown that we can drastically reduce the amount of nodes and transitions that are available for each current topology. Many of these simplifications are based on the solution of a positive linear combination of equation, solved thanks to the introduction of the two-step Simplex algorithm.
After performing this study for each acceptable starting topology, remains only the minimal amount of information needed to pinpoint the new topology taken by the circuit according to a defined set of laws. This information takes the form of a set of inequalities corresponding to some diode currents and voltages (but not necessarily corresponding to all diodes and not always found in the same topology) that must be evaluated in real-time. These sets are specific to a single topology. A big advantage of this method is that the new state is found in a single step, even if multiple diodes must be switched in sequence due to a chain reaction of events. In our state of the art, we found that simulators based on the nodal analysis and most tools based on the state-space analysis are not be able to properly analyze these cascade of events and only switch a subset of the diodes, leading to unwanted transient effects in the simulation.
The second analysis method, called forced switching module, is designed to study the transitions that occur due to changes in the control signals of the switches; it obeys similar rules. More precisely, it is based on the observation that the value of state variables cannot change instantaneously. This principle allows our tool to eliminate all topologies that would lead to any brutal change in a state variable. A last step is performed to further reduce the quantity of information required to select the correct topology among the remaining ones. Again, this novel analysis tool allows us to find the correct state of the system in a single step and without the need of analyzing the circuit during the real-time emulation.

Through numerous examples, we have shown that both tools lead to a vast reduction of the amount of calculations required at run time, compared to the standard hybrid automaton *and* to the traditional MNA-based systems. Notably, we have shown that, thanks to the global optimization provided by our tools, simulating a typical AC-to-DC converter composed of 12 switches could require 80% less resources when compared to the MNA solver [1]. This significant result means that large power converters can be emulated on low-power digital devices.

Since our optimization tools work by systematically analyzing all possible paths, they are still susceptible to the combinatorial explosion of the number of topologies and transitions, making them impracticable for the study of large circuits such as three-phase, three-level neutral point clamped (NPC) inverters. While using powerful platforms optimized for number crunching can be a solution in some cases, there will always be a limit to the memory and power of these computing platform, limiting their usage.
Instead, we introduced a solution by *partitioning* the converter in multiple sub-circuits. Since each part can be analyzed separately, the time needed to completely study the circuit is vastly reduced. The interconnection between the partitions is modeled using controlled voltage or current sources.
While most circuits can be sliced in parts that are relatively independent, basic partitioning methods will fail if the state of a switch is directly controlled by a signal created in another partition. An example of this would be a branch composed of a diode in series with an inductor, while these components do not belong to the same partition. To solve this problem, we introduced a novel way of interconnecting the partitions using sources whose law depends on the topology of the circuit. By using variable Thevenin equivalents for the sources instead of pure current/voltage sources, we are able to force the correct behavior in all partitions at once. These sources are updated at the beginning of each time-step, using the outputs computed during the previous step.
We have shown that, by properly calibrating the sources, we are able to emulate even the largest circuits by solving each partition separately.
The multiple test-circuits studied in section 4.4 have allowed us to conclude that the partitioning framework performs well for those examples. A slight degradation of the emulated signals is present, but the error remains inferior to a few percent when the partitions are completely decoupled. The signals resulting of the simulation of circuits with coupled partitions present distortions when the incriminated elements are switched on or off, reducing the accuracy of the emulation. However, these distortions are mostly present in a subset of the signals (typically those directly impacted by the switching behavior of the diodes) and the others are less disturbed.
Still, this degree of performance will be acceptable in most cases. It will keep on improving in the future since the distortions, essentially due to the delays between the calculation of a signal and its effects on other partitions, will be reduced because the performances of the digital devices used to implement the emulator are growing at a high rate.

The implementation of the real-time algorithm required the development of a dedicated digital platform, which we presented in chapter 5. This platform had to be able

[1] In this context, the *resource usage* corresponds to the amount of mathematical operations required at each time-step

to handle high clock rates and provide a low computation time for the solver. For this purpose, we compared the many ways of implementing the key elements of the solvers (matrix products, selection tables, . . . ), and proposed an efficient solution for each case. In particular, our goal was to reduce the resource usage as much as possible, leading to smaller designs, a faster routing process and higher speed. From these fundamental modules, we have shown how to develop the main components of the solver.

While the original solver only sampled the external signals controlling the transistors once per time-step, we have shown that this basic procedure could lead to quick divergence between the emulated signal and its expected value. Hence we introduced a new way to handle the control signals by oversampling them and by averaging the state matrices over the emulation time-step. This improvement has provided very good results, completely removing the divergence.

We have tested our platform by running the analysis process on two different circuits, a boost converter and a three-phases AC to AC converter, and implemented the resulting modules on the FPGA. From this tests, we were able to conclude that the system, in its current state, is able to handle these two circuits easily. In open-loop, we have shown that the waveforms obtained in real-time are almost exactly the same that were obtained during the offline simulations. This good performance was of course expected, since there is functionally no difference between computing the signals in the FPGA or on a computer as long as all timing criteria are respected. In particular, the AC-AC converter was simulated with a time step of 640ns without any problem. Furthermore, since the actual latency was closer to 400ns, this system could easily go faster if fitted with better digital-to-analog converters. While the signals were slightly degraded when the emulator is used in closed-loop control with a sliding-mode controller operating at 100kHz, this is not due to a limitation of our system but rather to a side-effect of the sampling of the signals.

Another remarkable result is that the latency is mostly independent of the size of the circuit, meaning that these speeds can be reached for a wide variety of converters. Thus, we are very confident that our solver will scale well as better FPGAs become available on the market. We have shown that a speed improvement of up to 50% could already be obtained today simply by switching to an entry-level FPGA from the newest generation.

There is, obviously, still room for improvements, both from the analysis side and the solver side, and we highlighted some of them in section 5.6. The partitioning algorithm can certainly be improved to ensure a better stability of the system, and to avoid the distortions that happen when the partitions can go in discontinuous conduction mode. If such an improvement is made, this algorithm could truly allow our solver to work for almost all applications. Another weakness of this process is its current lack of automation, requiring the user to write him- or herself the laws ruling the values of the sources used to interconnect the partitions. Developing an interface allowing the user to quickly give directives as how the partitions are connected would certainly be nice in terms of productivity an ergonomy.

As for the platform itself, many side-features could be added to improve its usability. We mentioned, among other things, the addition of a data channel between the FPGA and a computer.

Looking at the whole picture, it seems clear that we have met our main objectives, which were to propose a complete and cost-effective system able to process as many power converter as possible and translate them into a real-time model. We have shown that this procedure is certainly possible, and that our framework is able to handle circuits composed of twenty switches without any problems. Furthermore, the addition of the partitioning has allowed us to completely circumvent the complexity limitation inherent in our method, and is perfectly integrated into our custom-made real-time platform. Thanks to its great scalability and its high performance, we are confident that it could be used in a variety of cases and help many engineers develop their power systems. Didactical applications where student could study the control without any risks could also be quite useful.

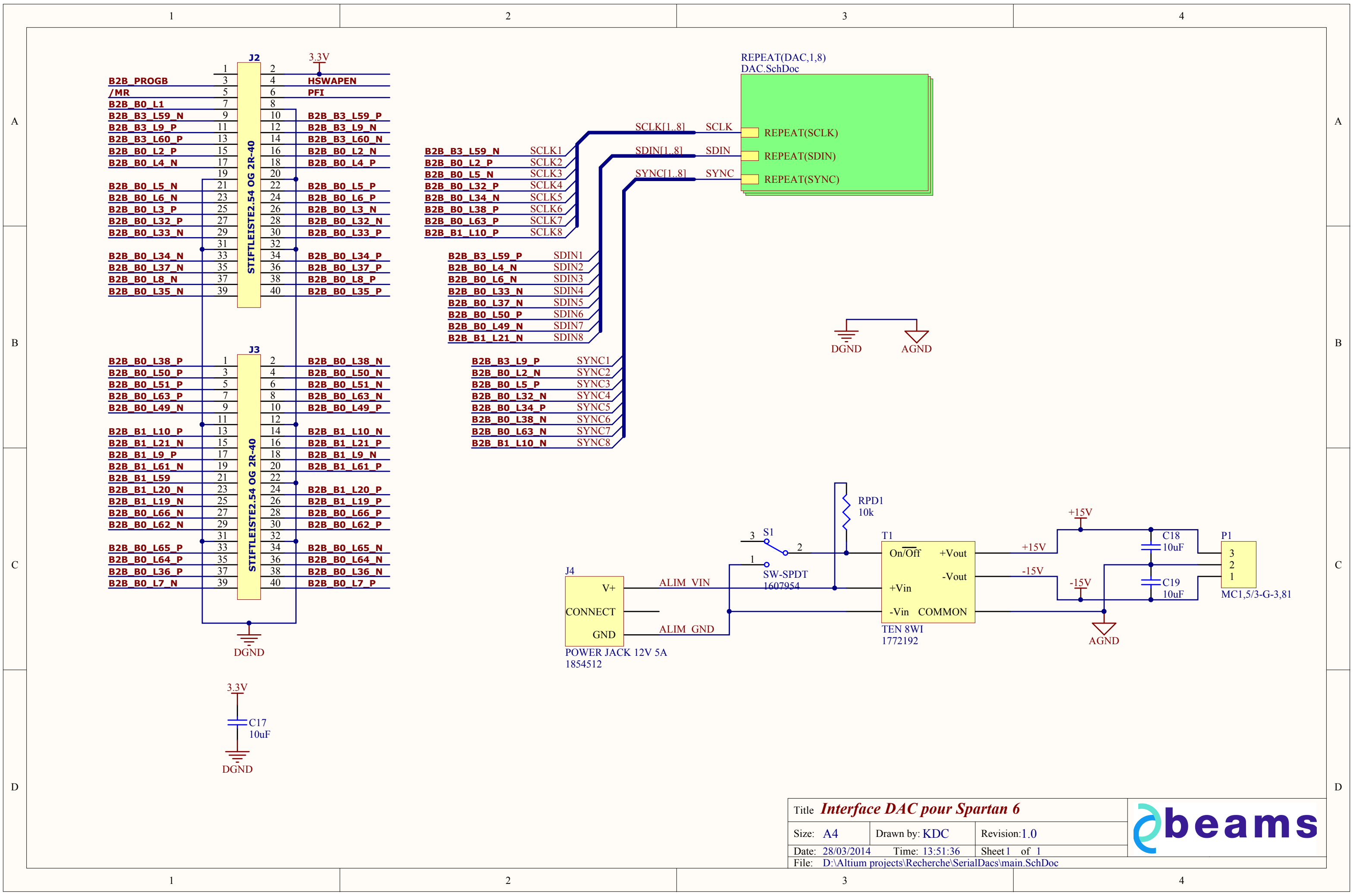
# List of Figures

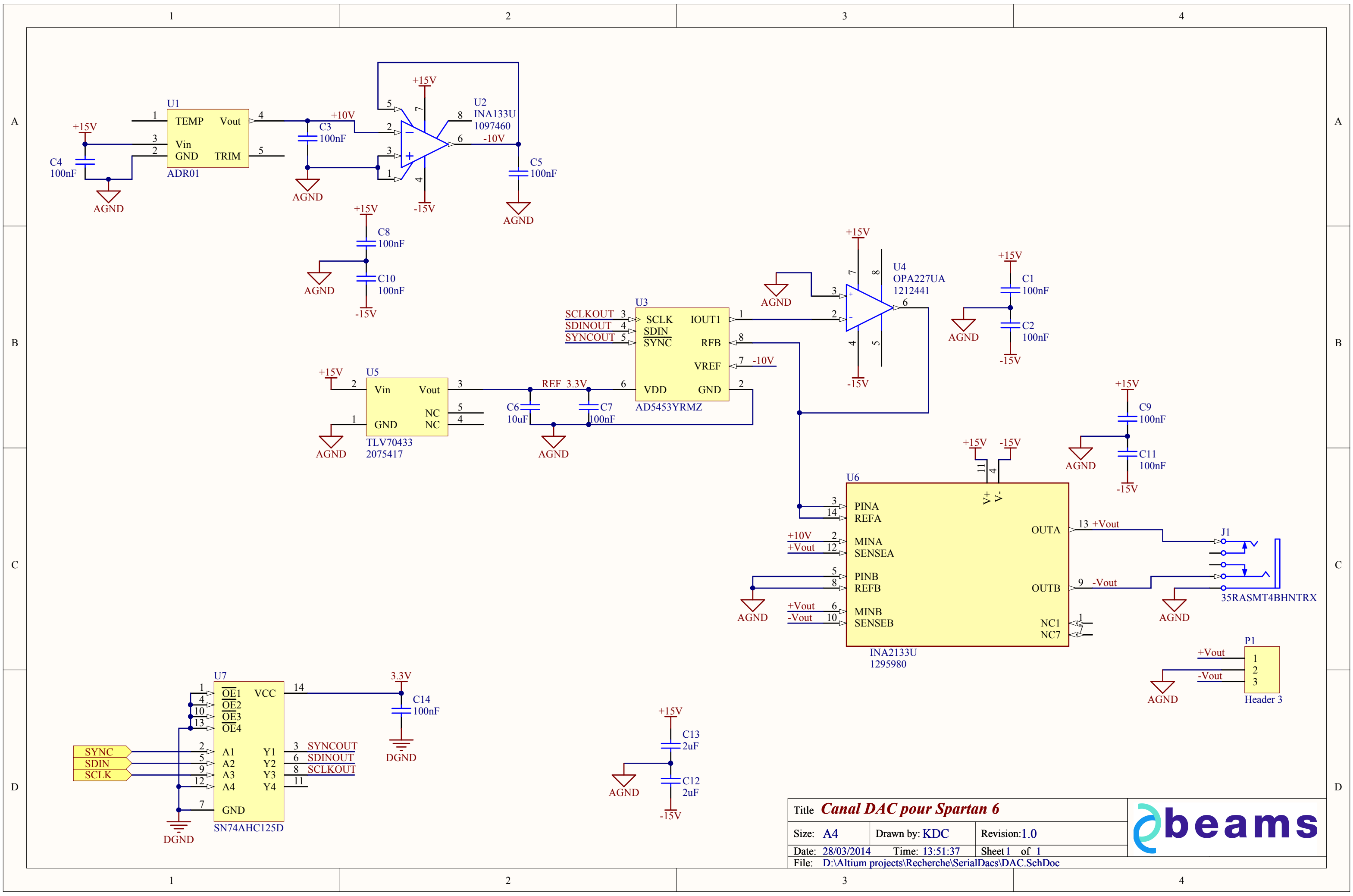
# List of Tables

# Appendix A

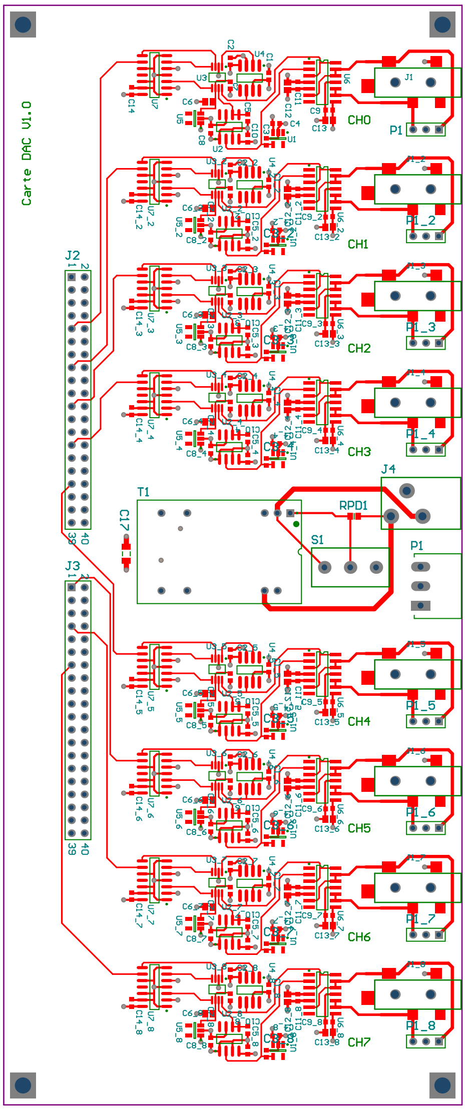
# Schematics of the Digital-to-analog board

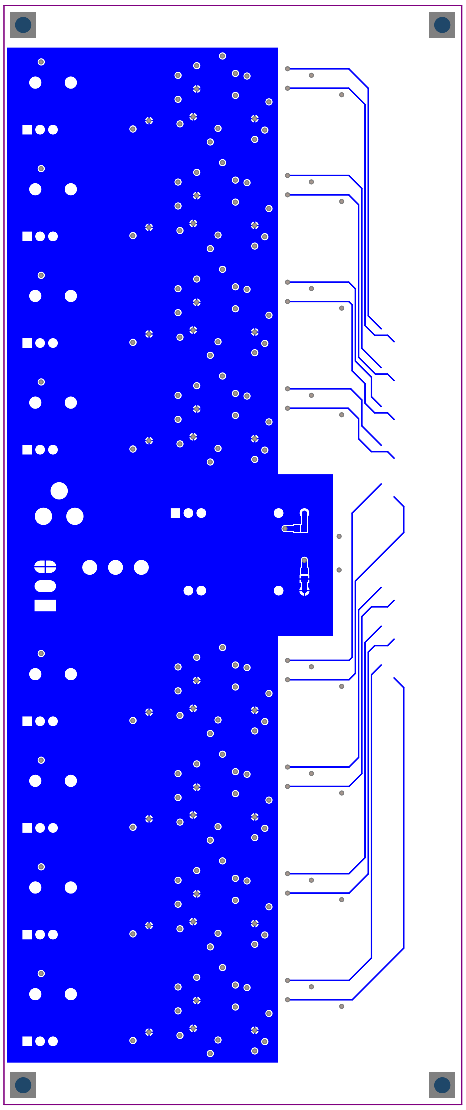
The next pages present the schematics of the custom-made board connected to the *TE0303* FPGA main board. This board contains the eight digital-to-analog converters which provide the analog measurements.


1
2
3
4
A
B
C
D
J2
3.3V
1 2
B2B_PROGB 3 4 HSWAPEN
/MR 5 6 PFI
B2B_B0_L1 7 8
B2B_B3_L59_N 9 10 B2B_B3_L59_P
B2B_B3_L9_P 11 12 B2B_B3_L9_N
B2B_B3_L60_P 13 14 B2B_B3_L60_N
B2B_B0_L2_P 15 16 B2B_B0_L2_N
B2B_B0_L4_N 17 18 B2B_B0_L4_P
19 20
B2B_B0_L5_N 21 22 B2B_B0_L5_P
B2B_B0_L6_N 23 24 B2B_B0_L6_P
B2B_B0_L3_P 25 26 B2B_B0_L3_N
B2B_B0_L32_P 27 28 B2B_B0_L32_N
B2B_B0_L33_N 29 30 B2B_B0_L33_P
31 32
B2B_B0_L34_N 33 34 B2B_B0_L34_P
B2B_B0_L37_N 35 36 B2B_B0_L37_P
B2B_B0_L8_N 37 38 B2B_B0_L8_P
B2B_B0_L35_N 39 40 B2B_B0_L35_P
STIFTLEISTE2.54 OG 2R-40
J3
B2B_B0_L38_P 1 2 B2B_B0_L38_N
B2B_B0_L50_P 3 4 B2B_B0_L50_N
B2B_B0_L51_P 5 6 B2B_B0_L51_N
B2B_B0_L63_P 7 8 B2B_B0_L63_N
B2B_B0_L49_N 9 10 B2B_B0_L49_P
11 12
B2B_B1_L10_P 13 14 B2B_B1_L10_N
B2B_B1_L21_N 15 16 B2B_B1_L21_P
B2B_B1_L9_P 17 18 B2B_B1_L9_N
B2B_B1_L61_N 19 20 B2B_B1_L61_P
B2B_B1_L59 21 22
B2B_B1_L20_N 23 24 B2B_B1_L20_P
B2B_B1_L19_N 25 26 B2B_B1_L19_P
B2B_B0_L66_N 27 28 B2B_B0_L66_P
B2B_B0_L62_N 29 30 B2B_B0_L62_P
31 32
B2B_B0_L65_P 33 34 B2B_B0_L65_N
B2B_B0_L64_P 35 36 B2B_B0_L64_N
B2B_B0_L36_P 37 38 B2B_B0_L36_N
B2B_B0_L7_N 39 40 B2B_B0_L7_P
STIFTLEISTE2.54 OG 2R-40
DGND
3.3V
C17
10uF
DGND
B2B_B3_L59_N SCLK1
B2B_B0_L2_P SCLK2
B2B_B0_L5_N SCLK3
B2B_B0_L32_P SCLK4
B2B_B0_L34_N SCLK5
B2B_B0_L38_P SCLK6
B2B_B0_L63_P SCLK7
B2B_B1_L10_P SCLK8
B2B_B3_L59_P SDIN1
B2B_B0_L4_N SDIN2
B2B_B0_L6_N SDIN3
B2B_B0_L33_N SDIN4
B2B_B0_L37_N SDIN5
B2B_B0_L50_P SDIN6
B2B_B0_L49_N SDIN7
B2B_B1_L21_N SDIN8
B2B_B3_L9_P SYNC1
B2B_B0_L2_N SYNC2
B2B_B0_L5_P SYNC3
B2B_B0_L32_N SYNC4
B2B_B0_L34_P SYNC5
B2B_B0_L38_N SYNC6
B2B_B0_L63_N SYNC7
B2B_B1_L10_N SYNC8
SCLK[1..8]
SDIN[1..8]
SYNC[1..8]
SCLK
SDIN
SYNC
REPEAT(DAC,1,8)
DAC.SchDoc
REPEAT(SCLK)
REPEAT(SDIN)
REPEAT(SYNC)
DGND
AGND
RPD1
10k
S1
3
2
1
SW-SPDT
1607954
J4
V+
CONNECT
GND
POWER JACK 12V 5A
1854512
ALIM_VIN
ALIM_GND
T1
On/Off
+Vout
-Vout
+Vin
-Vin
COMMON
TEN 8WI
1772192
+15V
-15V
+15V
-15V
AGND
C18
10uF
C19
10uF
P1
3
2
1
MC1,5/3-G-3,81
Title Interface DAC pour Spartan 6
Size: A4
Drawn by: KDC
Revision: 1.0
Date: 28/03/2014
Time: 13:51:36
Sheet 1 of 1
File: D:\Altium projects\Recherche\SerialDacs\main.SchDoc
beams



U1
TEMP
Vout
Vin
GND
TRIM
ADR01
+15V
C4
100nF
AGND
C3
100nF
+10V
U2
INA133U
1097460
-10V
C5
100nF
+15V
-15V
C8
100nF
C10
100nF
U3
SCLKOUT
SDINOUT
SYNCOUT
SCLK
SDIN
SYNC
IOUT1
RFB
VREF
VDD
GND
AD5453YRMZ
REF 3.3V
U4
OPA227UA
1212441
C1
100nF
C2
100nF
U5
Vin
Vout
NC
NC
GND
TLV70433
2075417
C6
10uF
C7
100nF
C9
100nF
C11
100nF
U6
PINA
REFA
MINA
SENSEA
PINB
REFB
MINB
SENSEB
OUTA
OUTB
NC1
NC7
V+
V-
INA2133U
1295980
+Vout
-Vout
J1
35RASMT4BHNTRX
P1
Header 3
U7
OE1
OE2
OE3
OE4
VCC
A1
A2
A3
A4
Y1
Y2
Y3
Y4
GND
SN74AHC125D
SYNC
SDIN
SCLK
3.3V
C14
100nF
DGND
C13
2uF
C12
2uF
Title Canal DAC pour Spartan 6
Size: A4
Drawn by: KDC
Revision: 1.0
Date: 28/03/2014
Time: 13:51:37
Sheet 1 of 1
File: D:\Altium projects\Recherche\SerialDacs\DAC.SchDoc
beams


Carte DAC V1.0
J2
J3
T1
C17
RPD1
S1
J4
P1
J1
CH0
CH1
CH2
CH3
CH4
CH5
CH6
CH7
P1_2
P1_3
P1_4
P1_5
P1_6
P1_7
P1_8



# Bibliography

[1] H. Doi, M. Goto, T. Kawai, S. Yokokawa, and T. Suzuki, “Advanced power system analogue simulator,” *Power Systems, IEEE Transactions on*, vol. 5, no. 3, pp. 962–968, Aug 1990.

[2] G. Nimmersjo, O. Werner-Erichsen, B. Hillstrom, and G. Rockefeller, “A digitally-controlled, real-time, analog power-system simulator for closed-loop protective relaying testing,” *Power Delivery, IEEE Transactions on*, vol. 3, no. 1, pp. 138–152, Jan 1988.

[3] M. Kayal, R. Cherkaoui, I. Nagel, L. Fabre, F. Emery, and B. Rey, “Toward a power system emulation using analog microelectronics solid state circuits,” in *Power Tech, 2007 IEEE Lausanne*. IEEE, 2007, pp. 726–730.

[4] E. Kuh and R. Rohrer, “The state-variable approach to network analysis,” *Proceedings of the IEEE*, vol. 53, no. 7, pp. 672 – 686, July 1965.

[5] R. Newcomb, *Network Theory: the State-space Approach*, ser. Network Theory: the State-space Approach. Librarie universitaire, 1968.

[6] C.-W. Ho, A. E. Ruehli, and P. A. Brennan, “The modified nodal approach to network analysis,” *Circuits and Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 504–509, 1975.

[7] A. Willson, I. Circuits, and S. Society, *Nonlinear Networks: Theory and Analysis*, ser. IEEE Press selected reprint series. IEEE Press, 1975. [Online]. Available: https://books.google.be/books?id=kuceAQAAIAAJ

[8] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design.* Van Nostrand Reinhold Company Inc., 1983.

[9] J. Choma, *Electrical networks: theory and analysis*, ser. A Wiley Interscience publication. Wiley, 1985. [Online]. Available: http://books.google.be/books?id=LuZSAAAAMAAJ

[10] L. O. Chua, C. A. Desoer, and E. S. Kuh, *Linear and Nonlinear Circuits.* McGraw-Hill Companies, 1987.

[11] G. C. Verghese, M. E. Elbuluk, and J. G. Kassakian, “A general approach to sampled-data modeling for power electronic circuits,” *Power Electronics, IEEE Transactions on*, no. 2, pp. 76–89, 1986.

[12] P. Pejovic and D. Maksimovic, "A method for fast time-domain simulation of networks with switches," *Power Electronics, IEEE Transactions on*, vol. 9, no. 4, pp. 449–456, 1994.

[13] C. Hsiao, R. Ridley, H. Naitoh, and F. Lee, "Circuit-oriented discrete-time modeling and simulation for switching converters," in *PESC'87-Annual IEEE Power Electronics Specialists Conference*, vol. 1, 1987, pp. 167–176.

[14] P. O. Lauritzen, "Simulation and modeling for power electronics," in *Computers in Power Electronics, 1988., IEEE Workshop on.* IEEE, 1988, pp. 37–42.

[15] L. W. Nagel and D. Pederson, "Spice (simulation program with integrated circuit emphasis)," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M382, Apr 1973. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html

[16] J. H. Allmeling and W. P. Hammer, "Plecs - piece-wise linear electrical circuit simulation for simulink," in *IEEE 1999 International Conference on Power Electronics and Drive Systems.* PEDS, 1999.

[17] R. Kuffel, J. Giesbrecht, T. Maguire, R. Wierckx, and P. McLaren, "Rtds-a fully digital power system simulator operating in real time," in *WESCANEX 95. Communications, Power, and Computing. Conference Proceedings., IEEE*, vol. 2, 1995, pp. 300–305 vol.2.

[18] J. B. Simon Abourida, Christian Dufour, "Real-time and hardware-in-the-loop simulation of electric drives and power electronics: Process, problems and solutions," in *The 2005 Power Electronics Conference*, 2005.

[19] C. Dufour, S. Cense, and J. Belanger, "Fpga-based switched reluctance motor drive and dc-dc converter models for high-bandwidth hil real-time simulator," in *Power Electronics and Applications (EPE), 2013 15th European Conference on*, Sept 2013, pp. 1–8.

[20] M. Matar and R. Iravani, "Fpga implementation of the power electronic converter model for real-time simulation of electromagnetic transients," *Power Delivery, IEEE Transactions on*, vol. 25, no. 2, pp. 852 –860, 4 2010.

[21] A. Myaing and V. Dinavahi, "Fpga-based real-time emulation of power electronic systems with detailed representation of device characteristics," in *Power and Energy Society General Meeting, 2011 IEEE*, 2011, pp. 1–11.

[22] H. Blanchette, T. Ould-Bachir, and J.-P. David, "A state-space modeling approach for the fpga-based real-time simulation of high switching frequency power converters," *Industrial Electronics, IEEE Transactions on*, vol. 59, no. 12, pp. 4555–4567, 2012.

[23] T. OuldBachir, H. Blanchette, and K. Al-Haddad, "A network tearing technique for fpga-based real-time simulation of power converters," *Industrial Electronics, IEEE Transactions on*, vol. 62, no. 99, pp. 1–1, 2014.

[24] C. Dufour, J. Mahseredjian, and J. Belanger, "A combined state-space nodal method for the simulation of power system transients," *Power Delivery, IEEE Transactions on*, vol. 26, no. 2, pp. 928–935, 2011.

[25] SIMetrix, "How spice works," SIMetrix, Tech. Rep., 2003.

[26] *Electromagnetic Transient Program (EMTP) Rule Book.*

[27] D. Par, G. Turmel, J.-C. Soumagne, V. Q. Do, S. Casorie, M. Bissonnette, B. Marcoux, and D. McNabb, "Validation tests of the hypersim digital real time simulator with a large ac-dc network," in *International Conference on Power Systems Transients - IPST 2003 in New Orleans, USA*, 2003.

[28] A. I. Zecevic, "Fundamentals of computer-aided circuit simulation," lecture Notes.

[29] L. Zadeh and C. Desoer, *Linear system theory: the state space approach*, ser. McGraw-Hill series in system science. McGraw-Hill, 1963.

[30] S. Natarajan, "A systematic method for obtaining state equations using mna," *Circuits, Devices and Systems, IEE Proceedings G*, vol. 138, no. 3, pp. 341–346, 1991.

[31] Y. Kang, "Systematic method for obtaining state-space representation of nonlinear dynamic circuits using mna," *Electronics Letters*, vol. 28, no. 21, pp. 2028–2030, 1992.

[32] E. Hairer, S. P. Nrsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, 2nd ed., ser. Springer series in computational mathematics 8, 14. Berlin ; New York: Springer-Verlag, 1993, 93007847 E. Hairer, S.P. Nrsett, G. Wanner. ill. ; 25 cm. Vol. 2 by E. Hairer, G. Wanner. Includes bibliographical references and indexes. 1. Nonstiff problems – 2. Stiff and differential-algebraic problems.

[33] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems (Springer Series in Computational Mathematics).* Springer, 2004.

[34] L. T. Biegler, "Differential-algebraic equations (daes)," lecture Notes.

[35] A. Sedra and K. Smith, *Microelectronic Circuits*, ser. Microelectronic Circuits. Oxford University Press, 1998, no. vol. 1.

[36] D. Bedrosian and J. Vlach, "Time-domain analysis of networks with internally controlled switches," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 39, no. 3, pp. 199 –212, mar 1992.

[37] A. Massarini and U. Reggiani, "Computer-aided time-domain large-signal analysis of networks with switches," in *Industrial Electronics, 1996. ISIE '96., Proceedings of the IEEE International Symposium on*, vol. 2, jun 1996, pp. 567 –572 vol.2.

[38] A. Massarini, U. Reggiani, and M. Kazimierczuk, "Analysis of networks with ideal switches by state equations," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 44, pp. 692 – 697, 1997.

[39] P. Pejovic and D. Maksimovic, "A new algorithm for simulation of power electronic systems using piecewise-linear device models," *Power Electronics, IEEE Transactions on*, vol. 10, no. 3, pp. 340–348, 1995.

[40] D. Majstorovic, I. Celanovic, N. Teslic, N. Celanovic, and V. Katic, "Ultralow-latency hardware-in-the-loop platform for rapid validation of power electronics designs," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 10, pp. 4708–4716, 2011.

[41] M. Matar and R. Iravani, "The reconfigurable-hardware real-time and faster-than-real-time simulator for the analysis of electromagnetic transients in power systems," *Power Delivery, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2012.

[42] T. Kato, K. Inoue, T. Fukutani, and Y. Kanda, "Multirate analysis method for a power electronic system by circuit partitioning," *Power Electronics, IEEE Transactions on*, vol. 24, no. 12, pp. 2791–2802, 2009.

[43] M. Dagbagi, L. Idkhajine, E. Monmasson, and I. Slama-Belkhodja, "Fpga implementation of power electronic converter real-time model," in *Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), 2012 International Symposium on*, 2012, pp. 658–663.

[44] Xilinx, "7 series fpgas overview," July 2013.

[45] G. semiconductors, *S300Y thru S300YR Silicon Standard Recovery Diode*, Genesic semiconductors.

[46] N. Mohan, T. M. Undeland, and W. P. Robbins, *Power Electronics: Converters, Applications, and Design.* Wiley, 2002.

[47] R. Burkel and T. Schneider, *Fast Recovery Epitaxial Diodes (FRED) Characteristics - Applications - Examples*, Ixys, 1999, iXAN0044.

[48] NXP, *Ultrafast power diode*, NXP Semiconductors, 1 2014.

[49] P. O. Lauritzen and C. Ma, "A simple diode model with reverse recovery," *Power Electronics, IEEE Transactions on*, vol. 6, no. 2, pp. 188–191, Apr 1991.

[50] N. Krihely and S. Ben-Yaakov, "Modeling and evaluation of diode reverse recovery in discrete-transition simulators," in *Energy Conversion Congress and Exposition (ECCE), 2010 IEEE*, Sept 2010, pp. 4514–4520.

[51] A. Dastfan, "A new macro-model for power diodes reverse recovery," in *Proceedings of the 7th WSEAS International Conference on Power Systems*, 2007.

[52] C. Blake and C. Bull, *IGBT or MOSFET: Choose Wisely*, International Rectifier.

[53] F. de Leon and J. Martinez, “Dual three-winding transformer equivalent circuit matching leakage measurements,” *Power Delivery, IEEE Transactions on*, vol. 24, no. 1, pp. 160–168, Jan 2009.

[54] E. Delaleau, J.-P. Louis, and R. Ortega, “Modeling and control of induction motors,” 2001.

[55] J. Lygeros, “Lecture notes on hybrid systems,” department of Electrical and Computer Engineering.

[56] G. Samad, T. ; Balas, *Software-Enabled Control:Information Technology for Dynamical Systems.* Wiley-IEEE Press, 2003, ch. Hybrid Systems: Review and Recent Progress, pp. 273 – 298.

[57] M. Senesky, G. Eirea, and T. Koo, “Hybrid modelling and control of power electronics,” in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, O. Maler and A. Pnueli, Eds. Springer Berlin Heidelberg, 2003, vol. 2623, pp. 450–465. [Online]. Available: http://dx.doi.org/10.1007/3-540-36580-X_33

[58] T. Geyer, G. Papafotiou, and M. Morari, “Model predictive control in power electronics: A hybrid systems approach,” in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC ’05. 44th IEEE Conference on*, Dec 2005, pp. 5606–5611.

[59] M. Mirzaei and A. A. Afzalian, “Hybrid modelling and control of a synchronous dc-dc converter,” *International Journal of Power Electronics*, vol. 1, no. 4, pp. 414–433, 2009.

[60] S. Mariethoz, S. Almer, M. Baja, A. Beccuti, D. Patino, A. Wernrud, J. Buisson, H. Cormerais, T. Geyer, H. Fujioka, U. Jonsson, C.-Y. Kao, M. Morari, G. Papafotiou, A. Rantzer, and P. Riedinger, “Comparison of hybrid control techniques for buck and boost dc-dc converters,” *Control Systems Technology, IEEE Transactions on*, vol. 18, no. 5, pp. 1126–1145, Sept 2010.

[61] M. Kinsy, O. Khan, I. Celanovic, D. Majstorovic, N. Celanovic, and S. Devadas, “Time-predictable computer architecture for cyber-physical systems: Digital emulation of power electronics systems,” in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, 29 2011-dec. 2 2011, pp. 305 –316.

[62] M. A. Kinsy, I. Celanovic, O. Khan, and S. Devadas, “Martha: Architecture for control and emulation of power electronics and smart grid systems,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 519–524.

[63] J. Lygeros, C. Tomlin, and S. Sastry, “Hybrid systems: Modeling, analysis and control,” December 2008, draft.

[64] M. Vekic, S. Grabic, D. Majstorovic, I. Celanovic, N. Celanovic, and V. Katic, "Ultralow latency hil platform for rapid development of complex power electronics systems," *Power Electronics, IEEE Transactions on*, vol. 27, no. 11, pp. 4436–4444, 2012.

[65] P. Raoji, D. Reilly, and R. Adair, "Design review: 150 watt current-mode flyback," tech Report by Unitrode.

[66] D. Murthy-Bellur and M. K. Kazimierczuk, "Two-switch flyback pwm dcdc converter in discontinuous-conduction mode," *International Journal of Circuit Theory and Applications*, vol. 39, no. 8, pp. 849–864, 2011. [Online]. Available: http://dx.doi.org/10.1002/cta.672

[67] K. De Cuyper, M. Osee, F. Robert, and P. Mathys, "A digital platform for real-time simulation of power converters with high switching frequency," in *Power Electronics and Applications (EPE 2011), Proceedings of the 2011-14th European Conference on*, 30 2011-sept. 1 2011, pp. 1 –10.

[68] ——, "A fast, state-graph-based diode switching algorithm for real-time power converter emulators," in *Control and Modeling for Power Electronics (COMPEL), 2012 IEEE 13th Workshop on*, 2012, pp. 1–7.

[69] A. D. Pathak, "mosfet/igbt drivers theory and applications," IXYS, Tech. Rep., 2001.

[70] B. Neidorff, "New integrated circuit produces robust, noise immune system for brushless dc motors," Unitrode, Tech. Rep., 2008.

[71] P. Santesso and M. Valcher, "Reachability properties of discrete-time positive switched systems," in *Decision and Control, 2006 45th IEEE Conference on*, Dec 2006, pp. 4087–4092.

[72] C. Le Guernic, "Calcul datteignabilite des systemes hybrides a partie continue lineaire," Ph.D. dissertation, Universit Joseph Fourier, 2009.

[73] S. Vestal, "A new linear hybrid automata reachability procedure."

[74] J. Hefferon, *Linear Algebra*, ser. VCU mathematics textbook series. Department of Mathematics & Applied Mathematics/Virginia Commonwealth University, 2009. [Online]. Available: http://books.google.be/books?id=UtdGPwAACAAJ

[75] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: http://dx.doi.org/10.1007/BF01386390

[76] J. Bang-Jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*, ser. Springer Monographs in Mathematics. Springer, 2009.

[77] J. Y. Yen, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quart. Applied Math*, vol. 27, pp. 526–530, 1970.

[78] N. Femia and M. Vitelli, "Time-domain analysis of switching converters based on a discrete-time transition model of the spectral coefficients of state variables," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 50, no. 11, pp. 1447–1460, Nov 2003.

[79] I. Rectifier, *IRFH8303PbF - HEXFET Power MOSFET*, International Rectifier, 10 2013.

[80] M. Frisch, *Advantages of NPC Inverter Topologies with Power Modules*, Vincotech, July 2009.

[81] B. Velaerts, P. Mathys, E. Tatakis, and G. Bingen, "A novel approach to the generation and optimization of three-level pwm wave forms for induction motor inverters," in *Power Electronics Specialists Conference, 1988. PESC '88 Record., 19th Annual IEEE*, April 1988, pp. 1255–1262 vol.2.

[82] T. Kato, "Multi-rate transient analysis of power electronic circuits by the envelope-following method with sensitivities of switch timings," in *Power Electronics Specialists Conference, PESC '94 Record., 25th Annual IEEE*, Jun 1994, pp. 1277–1281 vol.2.

[83] T. Kato, K. Inoue, Y. Kotani, and T. Ogawa, "Generalization of parallel analysis for a power electronic system by circuit partitioning," in *Control and Modeling for Power Electronics (COMPEL), 2013 IEEE 14th Workshop on*, June 2013, pp. 1–7.

[84] H.-J. Wu and W.-S. Feng, "Efficient simulation of switched networks using reduced unification matrix," *Power Electronics, IEEE Transactions on*, vol. 14, no. 3, pp. 481–494, May 1999.

[85] S. Hui, K. K. Fung, and C. Christopoulos, "Decoupled simulation of dc-linked power electronic systems using transmission-line links," *Power Electronics, IEEE Transactions on*, vol. 9, no. 1, pp. 85–91, Jan 1994.

[86] S. Hui and K. K. Fung, "Fast decoupled simulation of large power electronic systems using new two-port companion link models," *Power Electronics, IEEE Transactions on*, vol. 12, no. 3, pp. 462–473, May 1997.

[87] G. F. Franklin, M. L. Workman, and D. Powell, *Digital Control of Dynamic Systems*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[88] M. Aguirre, L. Calvino, and M. Valla, "Multilevel current-source inverter with fpga control," *Industrial Electronics, IEEE Transactions on*, vol. 60, no. 1, pp. 3–10, Jan 2013.

[89] J. Liang, A. Nami, F. Dijkhuizen, P. Tenca, and J. Sastry, "Current source modular multilevel converter for hvdc and facts," in *Power Electronics and Applications (EPE), 2013 15th European Conference on*, Sept 2013, pp. 1–10.

[90] N. Intruments, "Fpga fundamentals," 2003.

[91] Xilinx, *Floating-Point Operator v5.0*, 2009.

[92] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of high-performance floating-point arithmetic on fpgas," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 149.

[93] Altera. (2014, 08) The industrys first floating-point fpga.

[94] Xilinx, *Xilinx UG389 Spartan-6 FPGA DSP48A1 Slice, User Guide*, 2009.

[95] ——, *XtremeDSP for Virtex-4 FPGAs*, may 2008.

[96] ——, *Xilinx DS717 Multiply Adder v2.0*, 2009.

[97] J. Hayes, *Introduction to Digital Logic Design*. Addison-Wesley, 1993.

[98] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984.

[99] M. Faruque, V. Dinavahi, and W. Xu, "Algorithms for the accounting of multiple switching events in digital simulation of power-electronic systems," *Power Delivery, IEEE Transactions on*, vol. 20, no. 2, pp. 1157–1167, 2005.

[100] J. E. Volder, "The cordic trigonometric computing technique," *Electronic Computers, IRE Transactions on*, vol. EC-8, no. 3, pp. 330–334, Sept 1959.

[101] Plecs, *PLECS user manual*, 3rd ed., Plecs gmbh, 2014.