Digital Control Techniques for DC/DC Power Converters

Javier Martínez García-Tenorio

August 2009

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author: Javier Martínez García-Tenorio

Name of the thesis: Digital Control Techniques for DC/DC power converters

Date: 1.9.2009 **Number of pages:** xx

Faculty: Electronics, Communications and Automation

Professorship: Power Electronics

Supervisor: Prof. Jorma Kyyrä

Instructor: D.Sc. Adan Simon Muela

The newest generation of processors for embedded applications requires high performance power supplies. As a matter of fact, the new generation of power supplies are based on high-speed switching DC/DC converters. The latest constraint makes very difficult the task to implement a real time control system based on low-performance controllers. This is the reason why FPGAs are becoming more and more popular for the construction of real time control systems. Nevertheless, the main problem of using FPGAs is the requirement of having a good knowledge of the internal architecture and the associated languages for the development of applications. Thus, the main motivation of this work is intended to develop a generic methodology for the development of control systems using FPGAs. The system takes as an example the digital control techniques associated in the DC/DC converters domain. The work could be divided into three parts. The first part is introducing all the linear models associated with the behaviour of Buck converters and the design of regulated control system for these power supplies. The second part corresponds to the development of all the theoretical material to develop a fixed-point architecture for linear controllers. Finally, the third part specifies and designs a reusable architecture inside of the FPGA for the development of digital control systems.

Keywords: Buck converter, Linear Model, DPWM, Control system FPGA, VHDL, Coefficient Sensitivity, Fixed-Point

Preface

This document has been the result of my work experience at *LAAS*,*Laboratoire d'Analyse et d'Architecture des Systèmes*, during the period of time comprised between February 2008 and April 2009. *LAAS* are one of the facilities that *CNRS* (Centre National de la Recherche Scientifique) owns in the surrounding areas of Toulouse.

In this doccument you are going to find all the clues related with the design and development of digital control systems using FPGAs. The doccument takes, as an example, the case study of a Buck converter used for the newest generation of Intel CPU cores. Nevertheless, the discussion tries to settle a theoretical mathematical background to be rigorous and highlight the main problems of using reconfigurable architectures.

This work would have never been possible without the help and guidance of my instructor, D. Sc. Adan Simon Muela. I would like also to thank the support of Prof. Corinne Alonso and Prof. Bruno Estibals and all my work colleagues. Finally, I would like to thank the course that Prof. Eric Monmasson for the training course held at the Université de Cergy-Pontoise.

Contents

List of Figures

List of Tables

List of symbols

α	Attenuation parameter
β	Exponent
$\Delta V_{fs\ ADC}$	ADC full-scale voltage
$\Delta V_{out\ max}$	Maximal variation allowed in the measured variable
$\Delta V_{q\ DPWM}$	DPWM quantification level
$\Delta V_{q\ max}$	ADC voltage quantification level
η_{ADC}	ADC bits number
$\hat{c}_i$	i_{th} quantized coefficient
$\hat{p}_i$	i_{th} quantized pole
$\mathbb{R}$	Set of real numbers
$\mathbb{Z}$	Set of integer numbers
$\hat{\mathbf{u}}$	Small-signal input vector
$\hat{\mathbf{x}}$	Small-signal state vector
$\hat{\mathbf{y}}$	Small-signal output vector
$\mathbf{C}$	Coefficient matrix
$\mathbf{P}$	Pole matrix
$\mathbf{u}$	Input vector
$\mathbf{V}$	Vandermonde matrix
$\mathbf{x}$	State vector
ω_0	Natural frequency
$\sigma^2(e)$	Variance of the statistical noise function

ε	Error coefficient
ζ	Damping ratio
c_i	i_{th} coefficient
c_n	Fourier Coefficient
C_{in}	Input capacitance
$d(t)$	Control signal
$E(e)$	Mean of the statistical noise function
$e_d[n]$	DPWM noise signal
$e_u[n]$	Control noise signal
$e_v[n]$	Voltage error
$e_{pu}[n]$	Per-unit control error
$e_{v\ out}[n]$	Output voltage noise
F_{factor}	Form factor function
$f_{sw\ mp}$	Switching frequency multi-phase converter
G_{ADC}	ADC transfer function
G_{buck}	Buck converter transfer function
$G_{controller}$, G_c	Controller transfer function
G_{DPWM}	DPWM transfer function
$G_{id}(s)$	Control to Inductor Current
$G_{ii}(s)$	Input Voltage to Inductor Current
$G_{ilio}(s)$	Output Current to Inductor Current
$G_{od}(s)$	Control to Output Voltage
$G_{oi}(s)$	Input Voltage to Output Voltage
$H(s)$	Transfer function of the input filter
H_v	Sensor gain
$I_1(\omega)$	Current spectrum of a single-phase converter
$i_k(t)$	k-th inductor current

i_L	Inductance current
I_{max}	Maximum Current
I_{min}	Minimum Current
I_{nh}	n^{th} current harmonic of the input current waveform
I_{ns}	Filtered n^{th} current harmonic
i_{out}	Output current
K_c	Critical gain
k_d	Derivative gain
k_i	Integrative gain
k_p	Proportional gain
m	Mantissa
$N_{coefficents}$	Coefficient wordlenght
n_{DPWM}	Number of bits of DPWM
$N_{integer}$	Coefficient integer wordlenght
N_{real}	Coefficient fractional wordlenght
p_i	i_{th} pole
PDF	Probability density function
Q	Quantizer block
r_i	i^{th} residue
R_{ESR}	Equivalent Series Resistance in capacitor
$R_{inductance}$	Parastic series resistance
T_{cycle}	Required time to perform the calculation of a new duty cycle
$u[n]$	Control voltage
$u_{pu}[n]$	Per-unit control signal
v_c	Capacitance voltage
V_g	Input voltage

V_{adc}	ADC reference voltage
v_{out}	Output voltage
V_{ref}	Reference voltage
W	Numerical wordlenght
X	Fractional wordlenght
$Z_{out}(s)$	Open-Loop to Output Impedance
ADC	Analog Digital Converter
AMS	Analog Mixed-Signal Extensions
C	Capacitance
CPU	Core Processing Unit
D	Duty cycle
DPWM	Digital Pulse Width Modulator
DSC	Digital Signal Controller
EDA	Electronic Design Automation
ESR	Equivalent Series Resistance
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Design Language
IBIS	Input/Output Buffer Information Specification
ICEM	Integrated Circuits Electromagnetic Model
KCL	Kirchhoff Current Law
KVL	Kirchhoff Voltage Law
L	Inductance
LSB	Least Significant Bit
MSB	Most Significant Bit
N	Number of commutation
PCB	Printed Circuit Board

PID	Proportional Integrative Derivative
VRM	Voltage Regulation Mode
XST	Xilinx Synthetisable Tool

Chapter 1

Introduction

Embedded applications have emerged appreciably during the past few years due to the considerable increase of nomad and traveller ways of life. These itinerant lifestyles induce the apparition and development of more and more portable and autonomous systems. As a consequence, energy sources, their corresponding storage devices and power management control systems should be improved substantially to obtain an optimal and long-lasting working operation. Consequently, new issues in the quest for longer power autonomy have appeared recently. Furthermore, embedded products own strict power supplies design requirements which are becoming more and more critical with the increasing complexity of the functionalities proposed by these new portable devices.

In this context, power management discipline has acquired an increasing interest in the design and manufacture of embedded systems for automotive, consumer and other industrial markets. Therefore, one of the current challenges in the domain of power electronics is the design and implementation of power supplies for the new generation of high-performance CPUs. The characteristics of the new families of processors may require from 40 to 100 watts of power. The load current for such power supplies must have a good dynamic respone. For instance, the slew rate parameter is in the order of 50 $\frac{A}{\mu s}$. On the other hand, this fast current response should still keep the output voltage within tight regulation and response time tolerances. To manage such a big number of watts, it is necessary to feed CPUs by following some strict specifications related to the stability of the input voltage and current. Therefore, the creation of several control techniques is necessary to accomplish the specification of the manufacturer.

The first digital architecture in charge to implement the control techniques for the newest generation of power supplies was the Freescale DSC56800 core processor. This technology was not fast enough to implement a real-time control system. The need of controlling a Buck converter, whose switching frequency is 1 MHz, requires fast digital reconfigurable systems. FPGAs were the best solution to solve the problem concerning the computational speed.

Nevertheless, the implementation of the previous control techniques using a DSC had to be radically changed. The internal architecture of FPGAs have nothing in common with the typical processor architectures. In addition to the previous problem, the programmers used to the use of programming languages such as C, should firstly understand that Hardware Description Languages are intended to configure the hardware of an FPGA. The drawbacks specified earlier came up with the need of developping a generic methodology to program FPGAs.

As a result, to face these new challenges in the control of DC/DC power supplies area, our work takes part in the LISPA (Laboratoire pour l'Intégration des Systèmes de Puissance Avancés) which is a French (Midi-Pyrenean) regional project including the LAAS-CNRS and Freescale Semiconductor.

The main motivation of this chapter is to introduce the reader in the early stages of control systems design for embedded applications. This Master's Thesis takes, as an example, the case of output voltage regulation for power converters applying a generic methodology. Figure 1.1 shows briefly the steps and procedures to create a digital control systems using a FPGA as a digital controller for embedded aplications. Most of the layers in the stack are connected with one of the chapters appearing in this report.

First step in the design methodology, and the most essential, is the definition of the problem to be solved. This is not a straightforward step, but it is crucial for the development of the control system. Engineers must have a clear idea of the problem they want to face, the underlying specifications they should accomplish and finally the propierties of the technology employed. The second step requires the task of an specialist whose expertise domain is associated with the creation of behavioral models of a physical process. Every dynamic system or physical process can be described nowadays by means of mathematical tools, for example differential equations. State-space representation allow us to predict system behaviour when a given input is aplied. As soon as the mathematical model of the system is understood and validated, a control engineer is in charge to develop and optimize control laws in order to achieve the desired specifications defined in point 1. The validation of the whole control system, including the physical system and the control law, in tools such as *Matlab* or *Simulink* is necessary before starting the electronic design and implementation. The first three steps are explained detailed in Chapter 2. The first half of this chapter describes the state-space representation of our Buck converter. The second half describes every block included in a control for the output voltage regulation of a Buck converter.

The bottom of Figure 1.1 is associated to a large extent with the fabrication of the electronic system and also with the translation of our control algorithms into the core of the FPGA. Translation of control law algorithms deals with the problem of the coefficient quantization. It is also important to know the wordlength required to represent the input and output signals in the controller. Another important parameter is the roundoff error provoked

by the truncation at the ouput of arithmetic circuits. Chapter 3 deals with all the previous problems and the results of this part of the work are essential to the design of the controller inside of the FPGA.

Last steps are essential before the industralization of our design. First of all, we need to represent, using HDL *(Hardware Description Languages)*, the whole control system including physical system and control laws. This step is very important to validate the previous step. If the results obtained in this step are matching the simulation obtained in *Simulink*, we can almost be sure that future problems will be related to the malfunction of the electronic system. Nowadays, HDL allow the possibility of modelling physical systems as a continous process thanks to the AMS *(Analog Mixed Signal)* extension of VHDL. Chapter 4 discusses the design of the synchronous machine that controls the behaviour of the Buck converter.

Finally, the creation of the electronic circuitry containing our digital system is the last step before the industralization of the electronic product. The existence of EDA tools *(Electronic Design Automation)*, allow the engineers to quickly design the layout. But let's remember that the domain of EDA is not only the creation of software for the PCBs fabrication. The EDA also develops and standarizes models such as ICEM *(Integrated Circuit Electromagnetic Model)* oriented to the modelling of radiated and conducted parasitic emissions of integrated circuits on printed circuit boards (PCB). IBIS [1] and SPICE model are also useful for simulating components and electronic systems. Validation and testing the whole system should be the last step before the industrialization of the system. Last chapter shows experimental results of the output voltage regulation. This chapter validates the methodology for digital control systems using FPGAs.

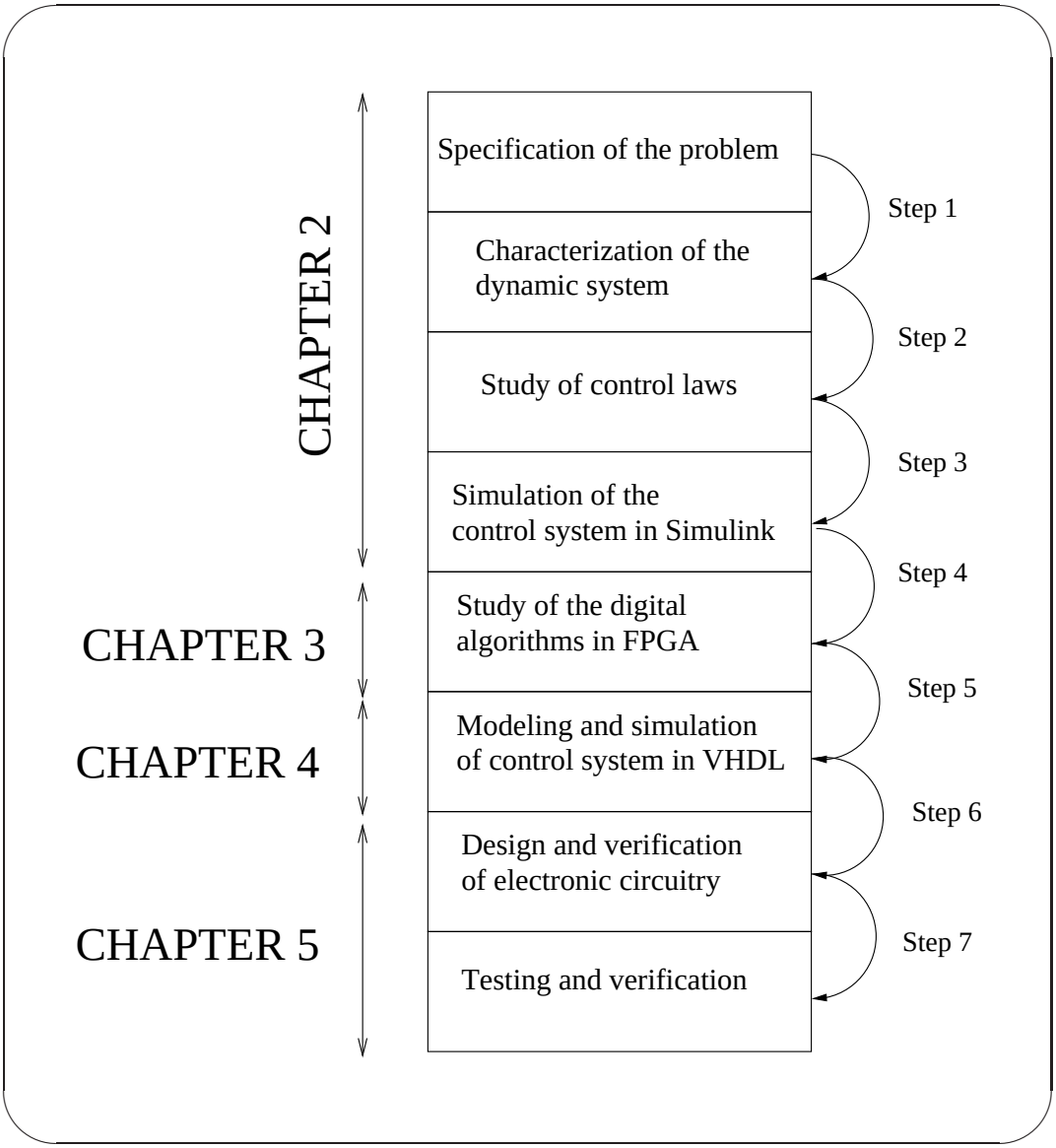


Figure 1.1: Steps and procedures to create a digital control system

Chapter 2

Modeling of the digitally-controlled power converter

2.1 Introduction

In particular, this chapter is based on the study of the first three levels of our chain. DC/DC power converters are the study case of our generic methodology to create embedded digital control systems using FPGAs as processing cores. The target is to deliver the most appropiated output voltage to the load in whichever work conditions.

In the first section of this chapter, the state-space representation of one-phase Buck converters can be found and also the input current characterization of interleaving multiphase Buck converters. After the presentation of the linear model of a switching power converter, the efforts are focussed on the assertion of the electronic systems to control digitally the power converter.

For the time being, DC/DC converters can be controlled by means of digital architectures. Digital domain creates the possibility of making an embedded design of the converter and allows performing difficult mathematical operations inside of an unique package. For example, non-linear control can be implemented more easily by means of digital computers.

Digital systems consists of several blocks as it is ilustrated in Figure 2.1a. First, an interface between analogue and discrete worlds. Next, a digital controller in charge of the management of the digital system. Finally, another interface between digital and analogue world.

In our case, acquisition is made by means of an ADC converter. After that, the control law generation is achieved by a digital controller. And finally, a duty cycle signal following a PWM pattern is generated. In digitally-controlled power converters, DPWM block is considered as a DAC. Figure 2.1b shows the reason for such analogy. The input of the DPWM, $u[n]$ is a

binary word coming from the digital controlle. The output, $d(t)$ is a PWM pattern

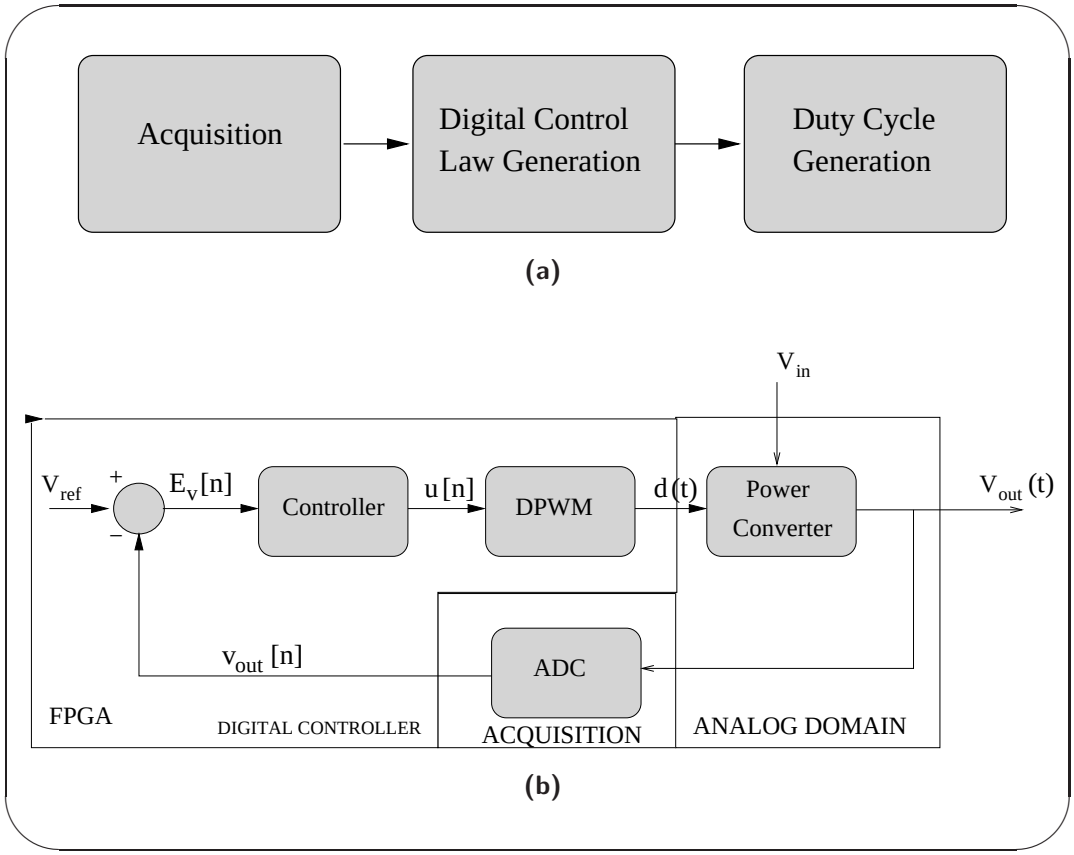


Figure 2.1: Implemented control system. (a) shows the schema of a digital system. (b) shows a negative-feedback control system

Thus, our system is composed of the following elements:

- Power converter: one-phase Buck converter
- ADC: samples the analogue signal to be read by the FPGA
- Reference : internal value placed inside of the digital system necessary to assure that the output voltage of the power converter tends to a value specified by the specifications of the problem. The value in figure 2.1b is represented by V_{ref}
- Substracter: generates the voltage error $E_v[n]$
- Linear controller: processes error voltage generating the command reference, $u[n]$
- DPWM *(Digital Pulse Width Modulation)*: it generates the duty cycle signal, $d(t)$. As previously explained, the DPWM is considered as a DAC.

2.2 Power converter

2.2.1 One-phase Buck converter

The easiest way to represent DC/DC converters is by means of a state space model representation. State space of a system or of a mathematical process is the minimum set of equations (called state variables). These variables contain sufficient information about the history of the system or process to allow computation of the future system behaviour.

To express precisely the definition of state, in mathematical terms, some notation is needed. Let the system be denoted by S. Associated to S, there is a set of input variables $u_1, u_2, ..., u_p$ and a set of output variables $y_1, y_2, ..., y_m$. For convenience, we let the input variables to be represented by an input vector.

$$u = [u_1 u_2 ... u_p]^T \tag{2.1}$$

and the output vector is represented as a m-dimensional vector

$$y = [y_1 y_2 ... y_m]^T \tag{2.2}$$

The input and output vectors are assumed to be functions of time t. To indicate this explicitly, we write u(t) and y(t), denoting respectively, the value of u and y at time t.

After that, we designate a set of state space variables by $x_1, x_2, ..., x_n$. Then, we are led to define the state vector as the n-dimensional vector $x = [x_1 x_2 ... x_n]^T$.

We are now in a position to express a definition of state depending on the state vector, the input vector and the output vector. For linear differential systems the state space model takes the simplified form of

$$\begin{cases} Kx'(t) = Ax(t) + Bu(t) \\ \quad y(t) = Cx(t) + Eu(t) \end{cases} \tag{2.3}$$

In case of a Buck converter, two state variables allows us to know the situation of the system, for example the inductor current and capacitor voltage are the aforementioned states.

The operation of a Buck is divided into two moments associated to the switch status as it can be observed in Figures 2.2 and 2.3. *"On-state"* operation will define the first set of equations and consequently *"off-state"* operation will define the second one. Finally, the behavioral model of the buck converter is the weighted mean of the two previous equations.

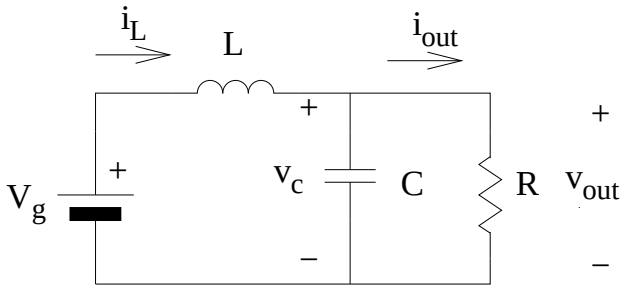


Figure 2.2: Switch ON Buck converter

2.2.1.1 State-space averaging

Switch On Considering an ideal switch is ideal, inductance and capacitor, we define the first set of equations when the switch is in the *"on-state"* as follows:

$$\begin{cases} V_g = L\dfrac{di_L}{dt} + v_c \\ i_l = C\dfrac{dv_c}{dt} + \dfrac{v_c}{R} \end{cases} \tag{2.4}$$

$$\begin{cases} i_{out} = \dfrac{v_c}{R} \\ v_{out} = v_c \end{cases} \tag{2.5}$$

For the first equation in the system, we are using *KVL* (Kirchoff Voltage Law), that is, the sum of voltages in a mesh is equal to zero. Similarly, the second equation is formulated using *KCL* (Kirchoff Current Law), which is equivalent to say that the sum of currents entering and leaving a node is equal to zero.

Defining set of vectors :

$$\mathbf{u} = \begin{pmatrix} V_g \\ I_g \end{pmatrix} \qquad \mathbf{x} = \begin{pmatrix} i_L \\ v_c \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} v_{out} \\ i_{out} \end{pmatrix} \tag{2.6}$$

Replacing (2.4) and (2.5) in (2.3), we obtain:

$$\begin{cases} \begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \mathbf{x}' = \begin{pmatrix} 0 & -1 \\ 1 & \frac{1}{R} \end{pmatrix} \mathbf{x} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \mathbf{u} \\ \mathbf{y} = \begin{pmatrix} 0 & 1 \\ 0 & \frac{1}{R} \end{pmatrix} \mathbf{x} \end{cases} \tag{2.7}$$

where,

$$\mathbf{K} = \begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \qquad \mathbf{A_1} = \begin{pmatrix} 0 & -1 \\ 1 & \frac{-1}{R} \end{pmatrix} \qquad \mathbf{B_1} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad \mathbf{C_1} = \begin{pmatrix} 0 & 1 \\ 0 & \frac{1}{R} \end{pmatrix}$$

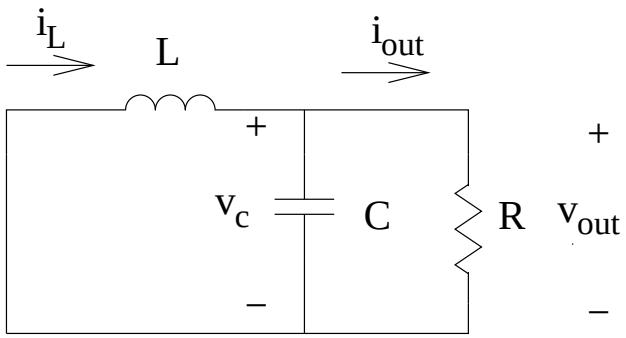


Figure 2.3: Switch OFF Buck converter

Switch Off Similarly to the previous case, the equations when the switch is in the *off-state*,using *KVL* and *KCL* as follows:

$$\begin{cases} 0 = L\dfrac{di_L}{dt} + v_c \\ i_l = C\dfrac{dv_c}{dt} + \dfrac{v_c}{R} \end{cases} \tag{2.8}$$

$$\begin{cases} i_{out} = \dfrac{v_c}{R} \\ v_{out} = v_c \end{cases} \tag{2.9}$$

Similarly than in the previous case, we replace Equations (2.8) and (2.9) into (2.3).

$$\begin{cases} \begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \mathbf{x}' = \begin{pmatrix} 0 & -1 \\ 1 & \frac{1}{R} \end{pmatrix} \mathbf{x} \\ \mathbf{y} = \begin{pmatrix} 0 & 1 \\ 0 & \frac{1}{R} \end{pmatrix} \mathbf{x} \end{cases} \tag{2.10}$$

$$\mathbf{K} = \begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \qquad \mathbf{A_2} = \begin{pmatrix} 0 & -1 \\ 1 & \frac{-1}{R} \end{pmatrix} \qquad \mathbf{B_2} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad \mathbf{C_2} = \begin{pmatrix} 0 & 1 \\ 0 & \frac{1}{R} \end{pmatrix}$$

Averaging over one period, the resulting state space equations are:

$$\begin{cases} A = A_1 D + A_2(1-D) \\ B = B_1 D + B_2(1-D) \\ C = C_1 D + C_2(1-D) \end{cases} \tag{2.11}$$

in the Buck Converter, $A_1 = A_2$, $B_2 = 0$, $C_1 = C_2$ we have that state matrix are:

$$A = A_1 \quad B = B_1 D \quad C = C_1$$

Finally, the average model in one switching period is:

$$\begin{cases} \begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \mathbf{x}' = \begin{pmatrix} 0 & -1 \\ 1 & \frac{1}{R} \end{pmatrix} \mathbf{x} + \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} \mathbf{u} \\ \mathbf{y} = \begin{pmatrix} 0 & 1 \\ 0 & \frac{1}{R} \end{pmatrix} \mathbf{x} \end{cases} \tag{2.12}$$

2.2.1.2 Steady-state simulations

Previous system is only useful to calculate and parametrize the Buck behaviour at a constant duty cycle ratio, D. Therefore, the buck is considered to be working under steady-state conditions whilst D is constant over the time. A *Matlab* script has been written to validate the state space average process developped in previous points. By choosing the following parameters, whose nomenclature is related to Figures 2.2 and 2.3 we can calculate the step response.

Table 2.1: Parameters of the single-phase converter designed in our laboratory for a 1 MHz switching frequency

System Parameter	Value
L	$1,2\mu H$
C	$240\mu F$
R	2Ω
V_g	$5V$
D	$0,2$

2.2.1.3 Control-to-output transfer function

Once, the average model of the Buck converter is calculated, we should calculate the transfer function that relates the output voltage with the duty cycle solving Equation (2.3). The reason why the relationship between v_{out} and d is calculated is because we are willing to implement an output voltage control system. As it could be appreciated in the formulation of the state-space averaging system derivation, the duty cycle, D is closely related to the average energy entering to the system per switching period.

The solution of this system can be found by means of *Laplace* transform. After applying the *Laplace* operator in Equation (2.12), the system can be solved using clasical matrix computations. To calculate the output voltage, we only need the first Equation of (2.12) because in the Buck converter

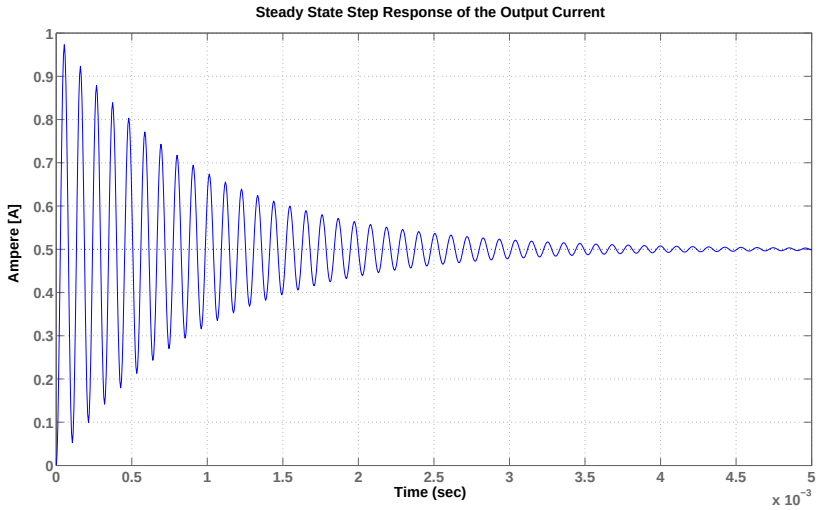


Figure 2.4: Step response of the output current of lossless static buck model. The response shows big overshoots during first transients because there is an energy exchange without losses between the inductance and the capacitance

the output voltage in the load and the voltage capacitance are the same (if parastic elements are not included).

$$\left(\begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} s - \begin{pmatrix} 0 & -1 \\ 1 & \frac{1}{R} \end{pmatrix}\right) X(s) = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} V_g \tag{2.13}$$

$$\begin{pmatrix} Ls & 1 \\ -1 & Cs - \frac{1}{R} \end{pmatrix} = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} V_g \tag{2.14}$$

The final step required to work out the value of the output voltage is the calculation of the inverse matrix in Equation (2.14). The transfer function that relates the output voltage when the duty cycle is constant and having a value of D is

$$G_{OD}(s) = \frac{RCs + 1}{s^2 + \frac{1}{RC}s + \frac{1}{LC}} \cdot \frac{V_g}{LC} \tag{2.15}$$

Using the values listed in Table 2.1 and applying them in Equation (2.15), the transfer function turns out into the form :

$$G_{OD} = \frac{0.00048s + 1}{s^2 + 2083.33s + 3.472 \cdot 10^9} \cdot 3.472 \cdot 10^9 \tag{2.16}$$

As it is known, every second-order transfer function has the main form

$$G(s) = \frac{b_2 s^2 + b_1 s + b_0}{s^2 + 2\zeta\omega_0 s + \omega^2} \tag{2.17}$$

The parameter ω_0, also called the *natural frequency* of the system, will determine how fast the system oscillates during any transient response. On the other hand, the parameter ζ, also called *damping ratio*, will determine how much the system oscillates as the response decays toward steady state. In the given example of this section, the natural frequency and damping ratio are 58.92 kHz and 0.0176768 respectively. In [2], we can see that the system tends to be more oscillatory whenever the damping ratio is comprised between zero and the unity. This propierty has a physical sense in an LC network. In the model of the Buck converter, the ESR (Equivalent Series Resistance) and the inductor copper losses were not included. Therefore, the energy exchange between the inductance and capacitance is lossless and the oscillation in the transient response is remaining longer over the time.

Using a model that includes more realistic parameters, we can contrast the limitations of a lossless model. The state-space representation of a model can be written using the procedure of *State-Space Averaging Method*. The equations are based on Figure (2.5):

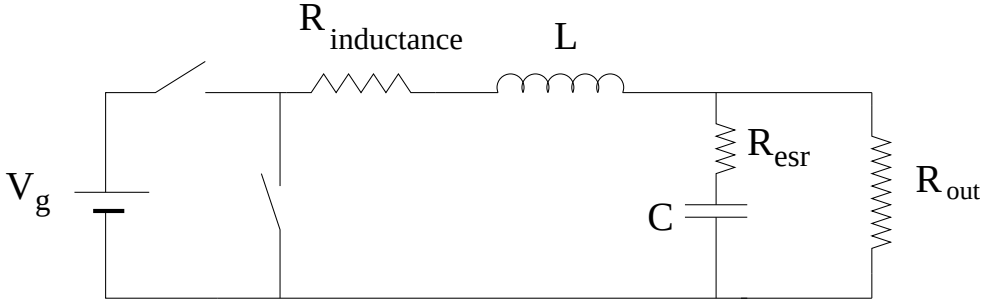


Figure 2.5: Buck model including losses

The demonstration to obtain the state-space representation of Equation (2.18) can be found in the Appendix B.

$$\begin{cases} \begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \mathbf{x}' = \begin{pmatrix} -\left(R_{inductance} + \frac{R_{out}R_{esr}}{R_{out}+R_{esr}}\right) & -\frac{R_{out}}{R_{out}+R_{esr}} \\ -\frac{R_{out}}{R_{out}+R_{esr}} & \frac{1}{R_{out}+R_{esr}} \end{pmatrix} \mathbf{x} + \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} \mathbf{u} \\ \mathbf{y} = \begin{pmatrix} \frac{R_{out}R_{esr}}{R_{out}+R_{esr}} & \frac{R_{out}}{R_{out}+R_{esr}} \\ \frac{R_{esr}}{R_{out}+R_{esr}} & \frac{1}{R_{out}+R_{esr}} \end{pmatrix} \mathbf{x} \end{cases} \tag{2.18}$$

Two simulations of the new model are performed in *Matlab* to show the divergence with the lossless model. There are defined two type of parameters to show how the damping effect is affected by the addition of series resistances both in the inducance and capacitance. In the first case, the values of R_{esr} and $R_{inductance}$ are 0.4853 Ω and 2.6 Ω respectevily. The transfer function $G_{OD}(s)$ is calculated in Matlab, showing a result where the coefficients are rounded:

$$G_{OD}(s) = \frac{3,254 \cdot 10^5 + 2,994 \cdot 10^9}{s^2 + 3.453 \cdot 10^4 s + 2.304 \cdot 10^9} \tag{2.19}$$

The natural frequency, ω_0, and the damping factor, ζ are in Equation (2.20). The step response of the system is shown in Figure 2.6

$$\begin{cases} \omega_o = 48\ kHz \\ \zeta = 0.36 \end{cases} \tag{2.20}$$

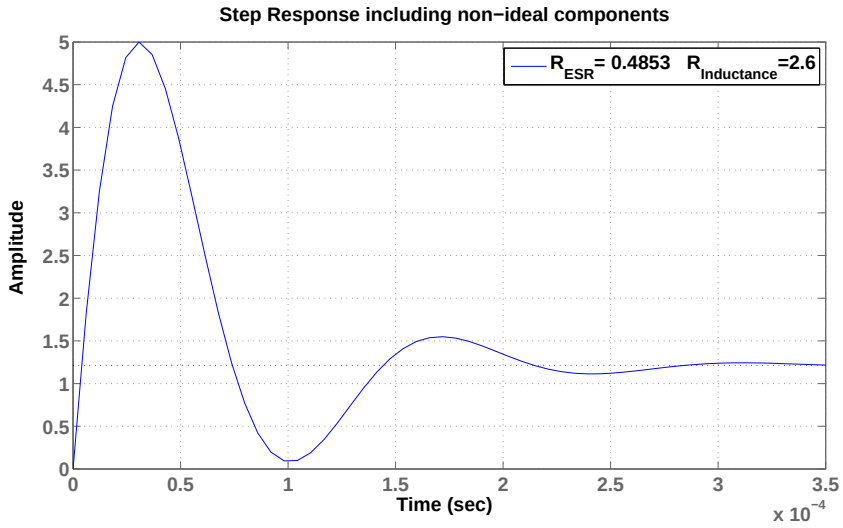


Figure 2.6: Step response of the model when $R_{esr} = 0.4853\,\Omega$ and $R_{inductance} = 2.6\ \Omega$. The step response is classified as underdamped because $\zeta = 0.36$

The second considered situation happens when the values of R_{esr} and $R_{inductance}$ are 9Ω and 2.5Ω respectively. In this case the transfer function calculated in Matlab is showed in Equation (2.21). The natural frequency and damping factor are $\omega_0 = 26.665\ kHz$ and $\zeta = 29.5228$. The step response, shown in Figure 2.7, is not exhibiting any oscillation during the transient. The physical reason why our dynamic system, is not showing any oscillation is because of the exchange of energy between the inductance and capacitance absorbed in R_{esr}.

$$G_{OD}(s) = \frac{1.364 \cdot 10s + 6.313 \cdot 10^8}{s^2 + 1.574 \cdot 10^6 s + 7.11 \cdot 10^8} \tag{2.21}$$

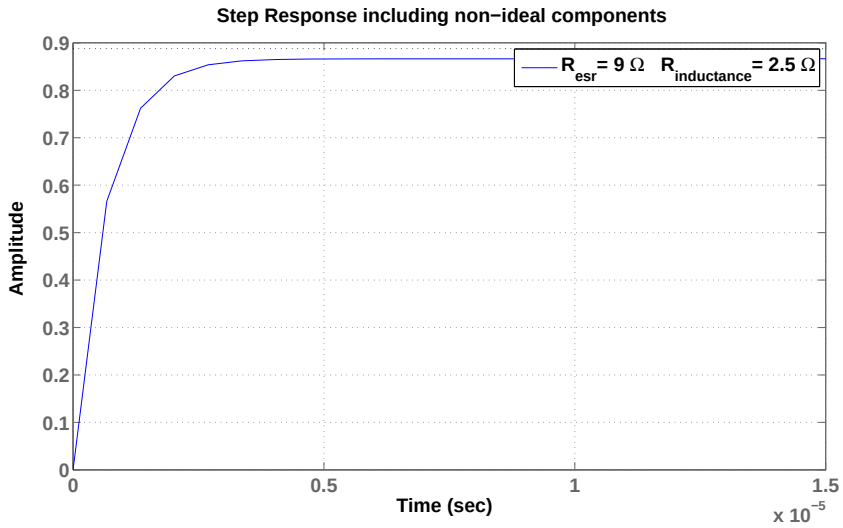


Figure 2.7: Step Response of the model when $R_{esr} = 9\Omega$ and $R_{inductance} = 2.6\Omega$. The step response is classified as overdamped because $\zeta = 29.5228$.

2.2.1.4 Small-signal model

The developed state space model of the power converter seems to be accurate enough for steady-state working operation, but the emerging question is how our system behaves when there is a disturbance in the system. Disturbances in the output voltage over the load might be originated because of many different situations such as disturbances in the input voltage, output current or uncertainties in the duty cycle. To understand the behavioural model of a Buck converter, it is necessary to create a small-signal model. What is the reason to be called like this?. The answer is easy. The output response in a power converter, as a linear system, can be divided into the sum of multiple inputs. That is, the output in a Buck can be expressed by superposition using the sum of a steady-state response and a small-signal disturbance.In mathematical terms, it can be expressed as Equation (2.22):

$$\begin{cases} K\left(x+\hat{x}\right)' & = A\left(x+\hat{x}\right)+B\left(u+\hat{u}\right) \\ \left(y+\hat{y}\right) & = C\left(x+\hat{x}\right)+E\left(u+\hat{u}\right) \end{cases} \tag{2.22}$$

State-space matrices are also affected by duty cycle disturbances. For example, state-space A-matrix is

$$\begin{cases} A+\hat{a} & = A_1\left(D+\hat{d}\right)+A_2\left(1-D-\hat{d}\right) \\ & = A_1D+A_2\left(1-D\right)+A_1\hat{d}-A_2\hat{d} \end{cases} \tag{2.23}$$

In a similar way, we can define state matrix B, C and E. The resulting state space is

$$\begin{cases} K\left(x' + \hat{x}'\right) & = (A + \hat{a})\left(x + \hat{x}\right) + \left(B + \hat{b}\right)(u + \hat{u}) \\ y' + \hat{y}' & = (C + \hat{c})\left(x + \hat{x}\right) + (E + \hat{e})(u + \hat{u}) \end{cases} \tag{2.24}$$

Operating the previous expression, neglecting second-order differential terms, eliminating the steady-state operation terms and rearranging terms, we have that small-signal state space model is

$$\begin{cases} K\hat{x}' & = (A\hat{x} + B\hat{u}) + \left((A_1 - A_2)\, x + (B_1 - B_2)\, u\right)\hat{d} \\ & = A\hat{x} + B_1 D\hat{u} + B_1 u\hat{d} \\ \hat{y}' & = (C\hat{x} + E\hat{u}) + \left((C_1 - C_2)\, x + (E_1 - E_2)\, u\right)\hat{d} \\ & = C\hat{x} \end{cases} \tag{2.25}$$

Small-signal simulations A *Matlab* simulation is used to highlight the results of Equation (2.25). To study the small-signal behavior of the Buck converter, the parameters of the studied Buck are the same as in subsection 2.2.1.2. Nevertheless, in this report the only disturbance studied is at the duty cycle, $\hat{d}$. One of the reasons of the generation of this perturbation might be caused, for example, because of the finite number of bits used to generate a duty cycle. Blue graph in Figure 2.9 shows the step response of the studied Buck converter when the magnitude of the disturbance is equal to $\hat{d} = 0.1 \cdot D$. On the other hand, green graph corresponds to the output voltage when the disturbance is applied to the steady-state operation. Figure 2.8 might help the reader to highlight the fact that a Buck converter is a linear system. Steady-state value of green graph corresponds to the addition of the steady-state value of Figure 2.4 and the disturbance. The reader might see [3] in order to see more about the disturbance caused at the ouput voltage in a buck converter due to the input voltage or inductor current.

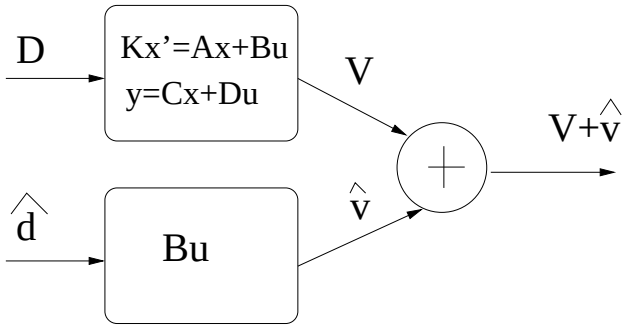


Figure 2.8: Dynamic operation of the Buck converter

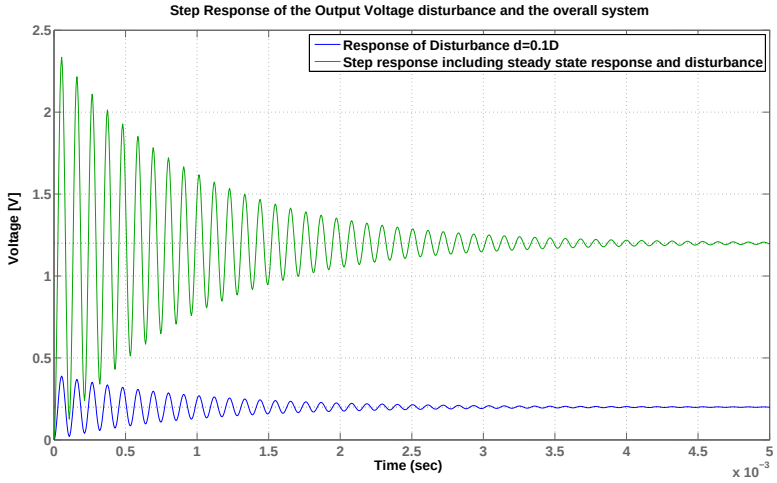


Figure 2.9: Step response of the output voltage in the dynamic Buck model

Small-signal transfer function definitions From small-signal model converter, we can formulate six important transfer functions defined by the following list. The following functions take into consideration all possible disturbances affecting the power converter state variables. As our study case is the output voltage regulation of a Buck converter, we noticed that state variables are inductor current and capacitance voltage. Disturbance in our system can come from the inaccuracy in the resolution of our duty cycle, the poor regulation of the input voltage and also the dynamic variation of the output current. Thus, six transfer functions matching the variable disturbance with the source of origin are used to describe the power converter behaviour.

Input voltage to inductor current

$$G_{ii}(s) = \left.\frac{\hat{I_L}}{\hat{v_{in}}}\right|_{\hat{i_{out}}=\hat{d}=0} \tag{2.26}$$

Input voltage to output voltage

$$G_{oi}(s) = \left.\frac{\hat{V_{out}}}{\hat{V_{in}}}\right|_{\hat{i_{out}}=\hat{d}=0} \tag{2.27}$$

Control to inductor current

$$G_{id}(s) = \left.\frac{\hat{I_L}}{\hat{d}}\right|_{\hat{i_{out}}=\hat{V_{in}}=0} \tag{2.28}$$

Control to output voltage

$$G_{od}(s) = \left.\frac{\hat{V_{out}}}{\hat{d}}\right|_{\hat{i_{out}}=\hat{d}=0} \tag{2.29}$$

Open-loop to output impedance

$$Z_{out}(s) = \left.\frac{\hat{V_{out}}}{\hat{I_{out}}}\right|_{\hat{d}=\hat{V_{in}}=0} \tag{2.30}$$

Output current to inductor current

$$G_{ilio}(s) = \left.\frac{\hat{I_L}}{\hat{I_{out}}}\right|_{\hat{d}=\hat{V_{in}}=0} \tag{2.31}$$

Variable resistive loads have not been considered in this report, because the development of our converter model is simplified.Therefore, the set of first four transfer function for our model are listed as:

Input Voltage to Inductor Current

$$G_{ii}(s) = \frac{D}{LRC}\frac{RCs+1}{s^2+\frac{s}{RC}+\frac{1}{LC}} \tag{2.32}$$

Input Voltage to Output Voltage

$$G_{oi}(s) = \frac{D}{LC}\frac{1}{s^2+\frac{s}{RC}+\frac{1}{LC}} \tag{2.33}$$

Control to Inductor Current

$$G_{id}(s) = \frac{V_g}{LRC}\frac{RCs+1}{s^2+\frac{s}{RC}+\frac{1}{LC}} \tag{2.34}$$

Control to Output Voltage

$$G_{od}(s) = \frac{V_g}{LC}\frac{RCs+1}{s^2+\frac{s}{RC}+\frac{1}{LC}} \tag{2.35}$$

2.2.2 Multiphase model converter

2.2.2.1 Introduction

Parallel or multiphase architectures for DC/DC converters are useful for DC loads that requires high output current and a high slew rate such as the newest CPUs cores.

Multiphase Buck converters are composed by a number of N commutation cells or phases, where every cell is composed by a switching device and a single inductance. Each cell is always connected to a common capacitance and to a load. The topology for a *N-phased Buck Converter* shown in Figure 2.10:

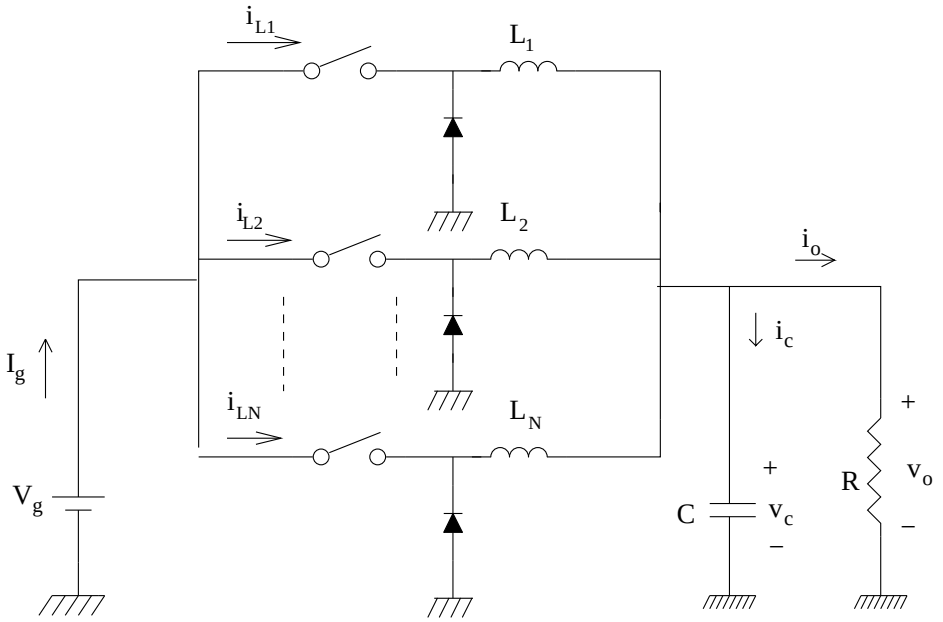


Figure 2.10: Multiphase Buck converter

2.2.2.2 Input current characterization

Single-phase input curent spectrum In steady state behavior, the multiphase model converter can be described by means of the complex Fourier series thanks to the periodicity of the input current waveform. Input current for a buck converter is illustrated in Figure 2.11.

In general terms, every periodic waveform cycle f(t) can be represented by an averaged set of coefficients "c_n" as follows [4]:

$$f(t) = \sum_{i=-\infty}^{i=\infty} c_n e^{jn\omega_0 t} \tag{2.36}$$

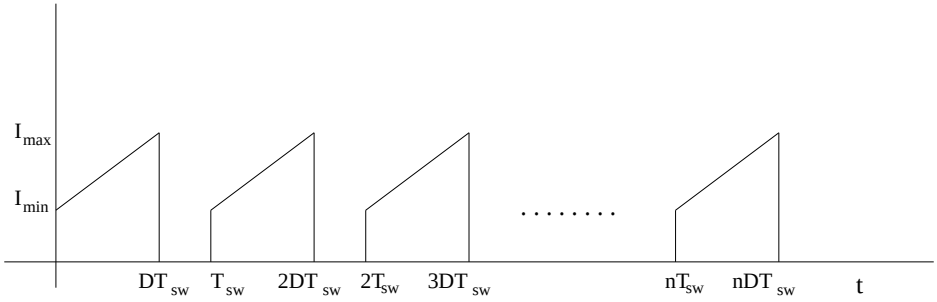


Figure 2.11: Input current of a Buck converter

being "c_n",

$$c_n = \frac{1}{T_{sw}} \int_0^{T_s} f(t) e^{-jn\omega_0 t} dt \tag{2.37}$$

with:

T_{sw}	switching frequency of the buck converter
ω_0	fundamental frequency

According to the previous formulation, the coefficients "c_n" for the input current waveform in the single-phase converter case can be calculated by:

$$\begin{aligned} c_0 &= \frac{I_{max} - I_{min}}{2} D \\ c_n &= \frac{I_{max} - I_{min}}{D\left(2\pi n\right)^2} 2\sin(\pi n D) e^{-j\left(\pi n D + \frac{\pi}{2}\right)} \end{aligned} \tag{2.38}$$

with:

I_{max}:	maximal input-current
I_{min}:	minimal input-current
D:	steady-state duty-cycle

Therefore, the time domain function, for the input current waveform in Buck converters, can be represented by the infinite-Fourier series shown by

$$\begin{aligned} f(t) &= \frac{\left(I_{max} - I_{min}\right) D}{2} \cdot \\ &\cdot \left(1 + \sum_{n=-\infty n\neq 0}^{+\infty} \frac{\sin(\phi_n)}{\left(\phi_n\right)^2} e^{-j\left(\phi_n + \frac{\pi}{2}\right)} e^{jn\omega_0 t}\right) \end{aligned} \tag{2.39}$$

being

$$\phi_n = \pi n D \tag{2.40}$$

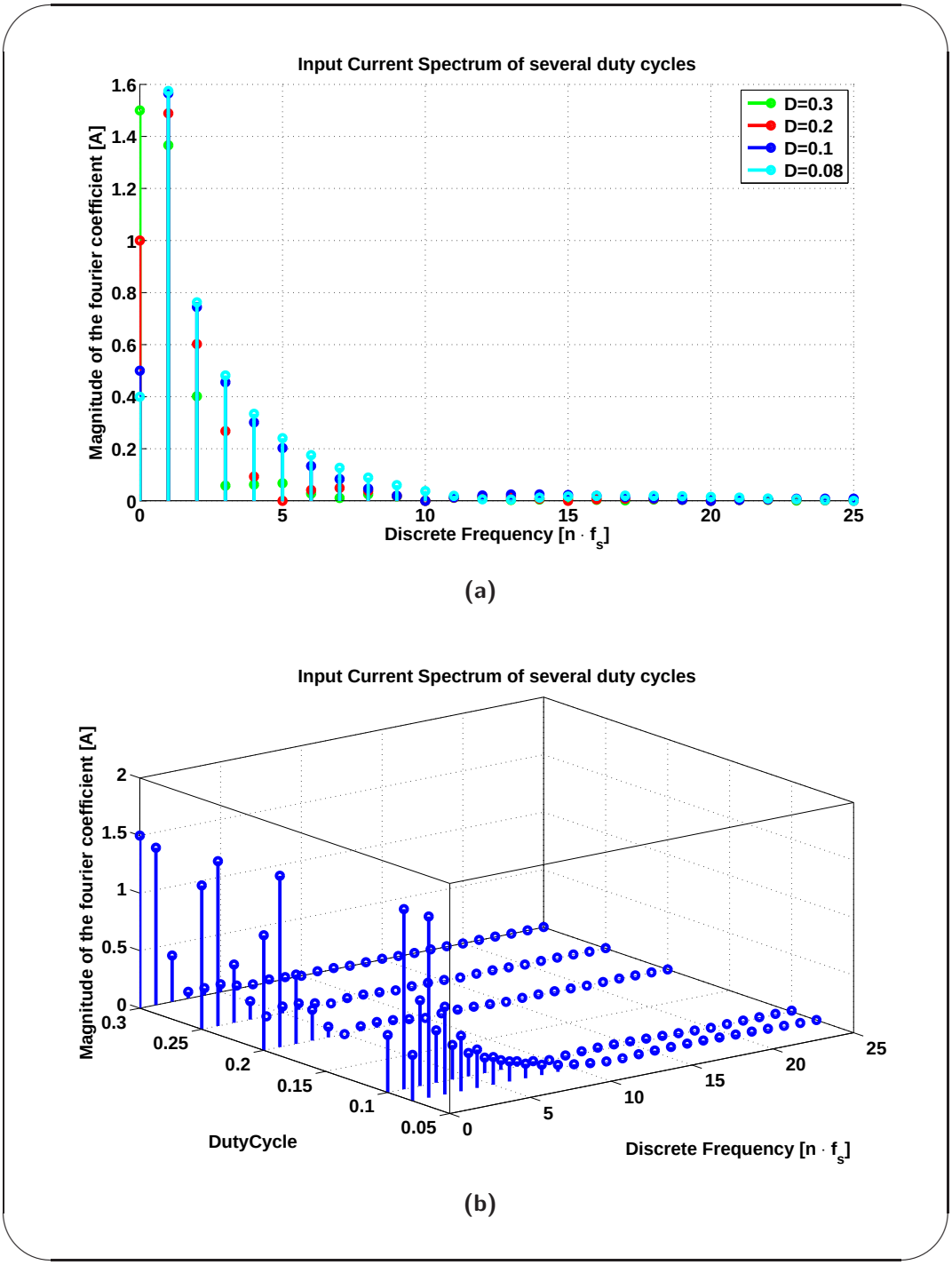


Figure 2.12: Input current spectrum for a single-phase Buck converter. The spectrum is calculated setting I_{max} and I_{min} to the values $20A$ and $10A$ respectively. (a) shows a 2D representation of the current spectrum for several duty cycles. (b) shows the same graph in a 3D graph

As it can be observed in Figures 2.12a and 2.12b, the harmonic content is concentrated on the fundamental power converter frequency. Obviously, in the case of the single-phase Buck converter, the fundamental frequency is the power converter switching frequency. After it, main harmonics are distributed in integer multiples of this fundamental frequency. In last figures, it can be also observed how for very low duty cycles, the harmonic content is centered in the fundamental frequency and first harmonics. Nevertheless, the harmonic content distribution, roughly speaking, tends quickly to zero as soon as high frequency values are achieved. This decay factor is equal to 2 as shown in Equation (2.38).

Multi-phase input current spectrum Input current for multiphase DC/DC converters feeds to the load using the so-called interleaving mode. In this mode of operation, the *i-switch* (see Figure 2.10) is activated every fraction of the switching period. The operation of interleaving multiphase converter is seen to be working to a frequency of $N \cdot f_s$, as it will be demonstrated along this section.

The architecture can be modeled assuming that all input current waveforms are identical and shifted one fraction of the switching period, T_{sw}, of the power converter commutation cell as shown in :

$$\frac{T_{sw}}{N} \tag{2.41}$$

N being the number of phases of the power converter.

Using the Kirchoff's law (KCL), the input current can be obtained by the addition of each commutation cell output current as follows:

$$i_{mp}(t) = i_1(t) + i_2(t) + ... + i_N(t) = \sum_{j=1}^{N} i_j(n) \tag{2.42}$$

Considering that the time series for each input current commutation cell are shifted versions of the fundamental one, the following relationship can be obtained:

$$i_1(t) = i_2\left(t - \frac{T_s w}{N}\right) = ... = i_N\left(t - \frac{N-1}{N} \cdot T_{sw}\right) \tag{2.43}$$

Next step is to apply the Fourier series operator in order to find the input current spectrum for the interleaved multiphase Buck converter:

$$I_g(\omega) = I_1(\omega) + I_2(\omega) \cdot e^{j\omega \frac{T_{sw}}{N}} + ... + I_{N-1} \cdot e^{j\omega \frac{N-1}{N} T_{sw}} \tag{2.44}$$

$$I_g(\omega) = I_1(\omega) \sum_{k=0}^{N-1} e^{j\omega k \frac{T_{sw}}{N}} \tag{2.45}$$

$$= I_1(\omega) \cdot F_{factor}(\omega) \tag{2.46}$$

$F_{factor}(\omega)$ being the following function

$$F_{factor}(\omega) = \frac{\sin\left(\omega\frac{T_{sw}}{2}\right)}{\sin\left(\omega\frac{T_{sw}}{2N}\right)} \cdot e^{j\omega T_{sw}\frac{N-1}{2N}} \tag{2.47}$$

As it can be observed in previous equations, the spectrum of an interleaved multiphase power converter is strongly influenced by the spectrum of the single-phase one associated to an additional second term so-called Form Factor. The magnitude of this last discrete function owns several peculiarities. The analysis of this discrete function will be focussed on the search for its its critical points. The first motivation for a consequent analysis is the placement of the zeros of the function, demonstrated as follows:

$$\begin{aligned} \sin\left(\omega\frac{T_{sw}}{2}\right) &= 0 \\ \omega\frac{T_{sw}}{2} &= k\pi \\ \omega &= \frac{2\pi k}{T_{sw}} = \omega_{sw}k \end{aligned} \tag{2.48}$$

Equation (2.48) shows that zeros of form factor function are zero at multiples of f_{sw}. Nevertheless, Form-factor function has a critial point at $N \cdot f_{sw}$, because denominator of *Form-factor function* is equal to zero. Thus, the only way to solve this uncertainity is by means of the *L'Hôpital* theorem:

$$\lim_{\omega\to N\omega_{sw}} F_{factor}(\omega) = \lim_{\omega\to N\omega_{sw}} \frac{\sin\left(\omega\frac{T_{sw}}{2}\right)}{\sin\left(\omega\frac{T_{sw}}{2N}\right)} = \underbrace{\frac{0}{0}}_{uncertainity} \tag{2.49}$$

$$\begin{aligned} \lim_{\omega\to N\omega_{sw}} F_{factor}(\omega) &= \lim_{\omega\to N\omega_{sw}} \frac{\sin\left(\omega\frac{T_{sw}}{2}\right)}{\sin\left(\omega\frac{T_{sw}}{2N}\right)} \\ &= \lim_{\omega\to N\omega_{sw}} \frac{\frac{T_{sw}}{2}cos\left(\omega\frac{T_{sw}}{2}\right)}{\frac{T_{sw}}{2N}cos\left(\omega\frac{T_{sw}}{2N}\right)} \\ &= \frac{\frac{T_{sw}}{2}\cos\left(\frac{N\omega_s T_{sw}}{2}\right)}{\frac{T_{sw}}{2N}\cos\left(\frac{N\omega_s T_{sw}}{2N}\right)} \\ &= \frac{\frac{T_{sw}}{2}\cos\left(\frac{N\frac{2\pi}{T_{sw}}T_{sw}}{2}\right)}{\frac{T_{sw}}{2N}\cos\left(\frac{N\frac{2\pi}{T_{sw}}T_s}{2N}\right)} \\ &= \frac{\frac{T_{sw}}{2}\cos\left(N\pi\right)}{\frac{T_{sw}}{N}\cos\left(\pi\right)} = N\cdot(-1)^N \end{aligned} \tag{2.50}$$

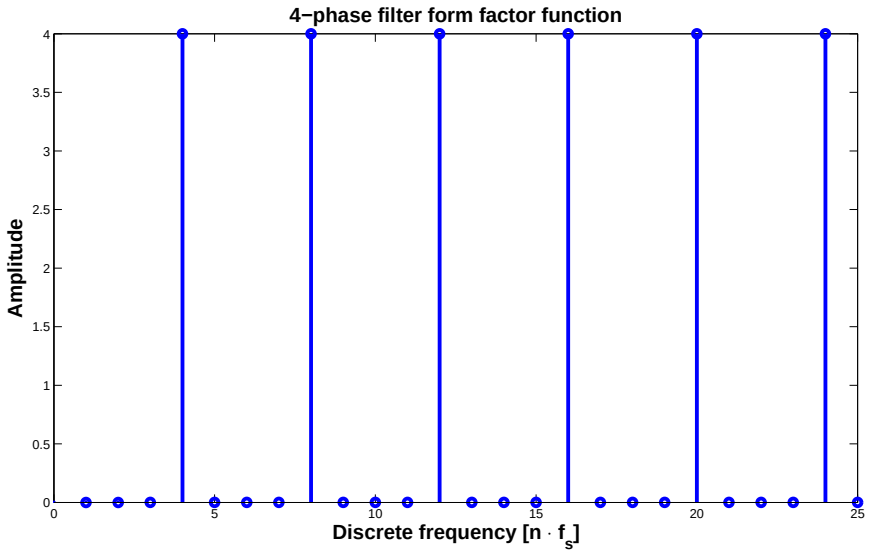


Figure 2.13: Form factor function for a four-phased Buck converter

The Form factor is represented in Figure 2.13 for a four-phased power converter case. As it was expected, harmonics are found in the multiples of the fundamental frequency and their value is equal to N since the number of phases is even. On the other hand, i.e. for an odd number of phases, their magnitude will be $-N$. As a consequence of this Form Factor, the harmonic spectrum is concentrated over the multiples of $N \cdot f_{sw\_mp}$. Otherwise, harmonics are equal to zero. The reason for this frequency behavior is because the spectrum for multi-phase converters, as seen in Figures 2.14a , is composed by the same spectrum as the single-phase converter multiplied by the Form Factor. Thanks to last multi-phase Buck converter skill, its input current inherites all the frequency propierties coming from the single-phase converters. Last conclusions can be observed in Figures 2.14a and 2.14b where the input-current spectrum is shown for a four-phased power converter.

Previous Figures 2.12a and 2.14a compare the single-phase and multi-phase Buck converter input-current harmonic contents. In these figures, it is worthy to notice the discrete behavior of the power converter input-current spectrum due to the periodicity of the signal. Moreover, some conclusions can be extracted associated to the relationship existing between the harmonic amplitudes and the power converter duty cycle. Thus, this relation is inversely proportional since the duty cycle is placed in its denominator, as it can be seen in previous Figures. Therefore, the smaller duty cycle is, the bigger harmonic magnitude for the multiples of the fundamental frequency are. On the other hand, the DC harmonic amplitude owns a direct relationship between the average value and the duty cycle.

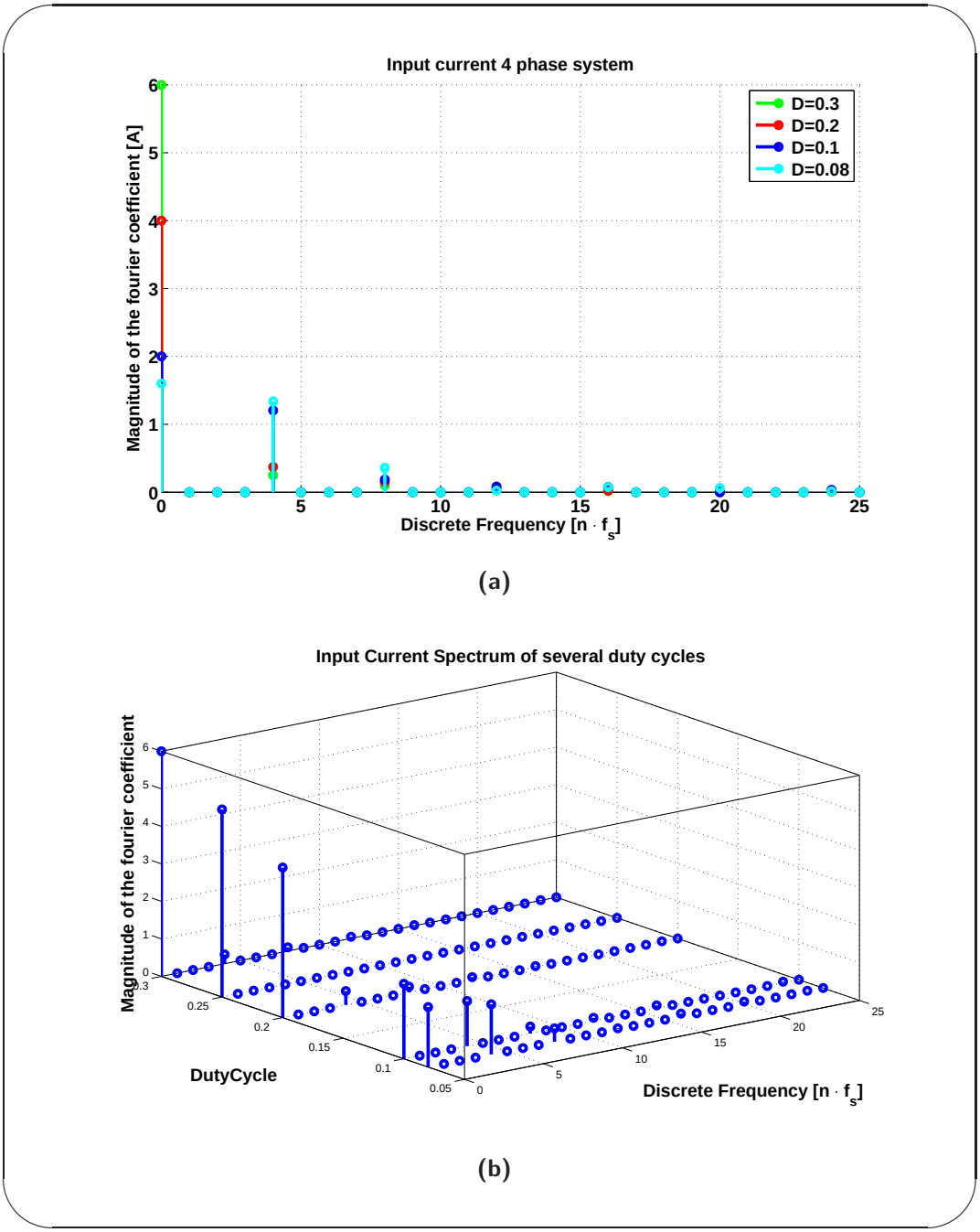


Figure 2.14: Input current spectrum for a multi-phase Buck converter. The spectrum is calculated setting I_{max} and I_{min} to the values $20A$ and $10A$ respectively. The number of phases is 4. (a) is a 2D representation showing the current spectrum for several duty cycles. (b) represents the same information in a 3D graph

2.2.2.3 Input-capacitor filter design of interleaved multiphase converters

Introduction Last conclusions are quite important in the design of input-filters for VRMs, in particular, for the input capacitor. As a matter of fact, the widespread trend for the input capacitor design is based on the maximal allowed input voltage drops. This is why the quantity of input capacitors in VRMs is often quite important to absorb correctly the drops in the input power supply. Nevertheless, these input capacitors take a lot of place and make the final design more expensive. As a result and in order to reduce the aforementioned problem, interleaved multiphase converters allow us to reduce the AC frequency components. Then, they are cancelled thanks to the analysis of zeros in the Form Factor function. In short, all frequency components that are different to zero are now placed at multiples of $N \cdot f_{sw\_mp}$. This allows an easier design of the input-filter design with a lower capacitor number and obtaining a reduced harmonic content. The magnitude of the input capacitance must be properly calculated to assure enough attenuation for the selected harmonics. Nevertheless, the value of the input capacitance could be a source of unstabilities in the switching regulator if it is not calculated correctly.

Single-phase converters Figure 2.15 shows the typical arrangement of an electronic load fed by a switching converter. The system can be divided into three parts. The *power source* is composed by a voltage source, its output resistance and, finally, the inductance associated to the cables. The power stage, composed by a single-phase Buck converter, is feeding a load. The system requires an input filter in order not to propagate back to the power source the harmonics produced by the input current waveform. It can be seen in Figure 2.15 that the input capacitance in a power converter has the task to become a low-impedance path for all the high frequency harmonics coming from the load and returning to the ground.

Therefore, the value of the input capacitance in a single-phase converter can be calculated imposing a desired attenuation for the selected harmonics as follows:

$$I_{ns} = H(s)I_{nh} = \alpha I_{nh} \tag{2.51}$$

with :

I_{ns}	Filtered n^{th} current harmonic
α:	attenuation parameter
I_{nh}:	n^{th} harmonic of the input current waveform
$H(s)$:	Transfer function of the input filter

Considering that the load effect of the source resistance and inductance in Figure 2.15 are small, the high frequency impedance is mainly due to the

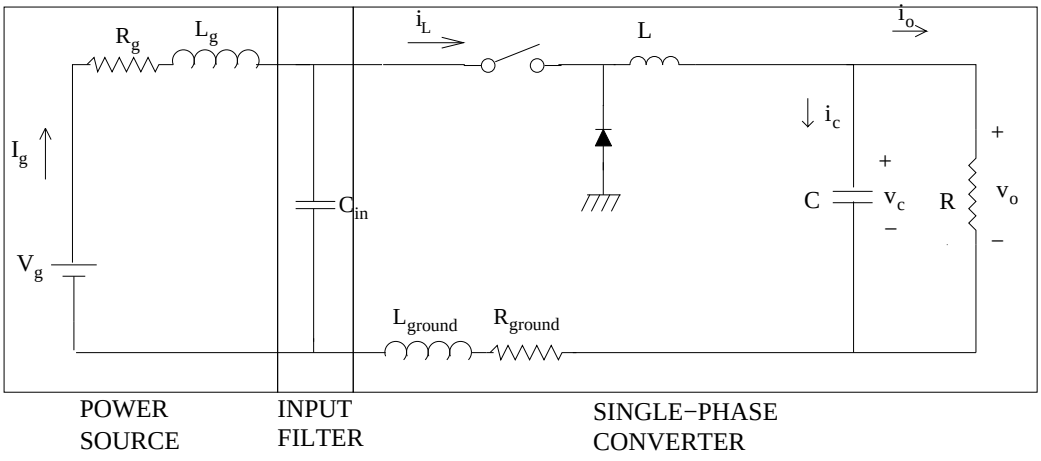


Figure 2.15: Power system model based on an single-phase Buck converter

effect of the capacitance. In fact, looking at Figure 2.15, the capacitance is a low path impedance for high frequency current harmonics. Therefore, the input filter transfer function at high frequencies can be approximated by the capacitance element only :

$$|H(j\omega)| = \frac{1}{C_{in1}\omega} = \alpha(\omega) \tag{2.52}$$

Using Equation (2.52) into (2.51) and (2.38), we have:

$$|I_{ns}| = \underbrace{\frac{1}{2\pi f C_{in1}}}_{\text{transfer function}} \cdot \underbrace{\left(\frac{I_{max} - I_{min}}{2(\pi n)^2 D}\sin(\phi_n)\right)}_{\text{Single-phase current spectrum}} \tag{2.53}$$

The design goal is to be able to attenuate the switching frequency, that is when $n = 1$ and $f = f_{sw}$ in Equation (2.53)

$$C_{in1} = f(I_{1s}, D) = \frac{(I_{max} - I_{min})\sin(\phi_1)}{2(\pi)^2 D I_{1s} f_{sw}} \tag{2.54}$$

Using Equation (2.53), we can define the parameter I'_{1h} as the magnitude of the n^{th}-harmonic if the duty cycle was 1. Thus, Equation (2.53) is

$$|I'_{ns}| = \frac{1}{2\pi f C_{in1}} \cdot \frac{I'_{1h}}{D} = \alpha'(f) \cdot \frac{I'_{1h}}{D} \tag{2.55}$$

being $\alpha'(f)$,

$$\alpha'(f) = \frac{I'_{nh}}{2\pi f C_{in1}} \tag{2.56}$$

Applying logarithms to both sides of Equation (2.55), we can see that the relationship is now:

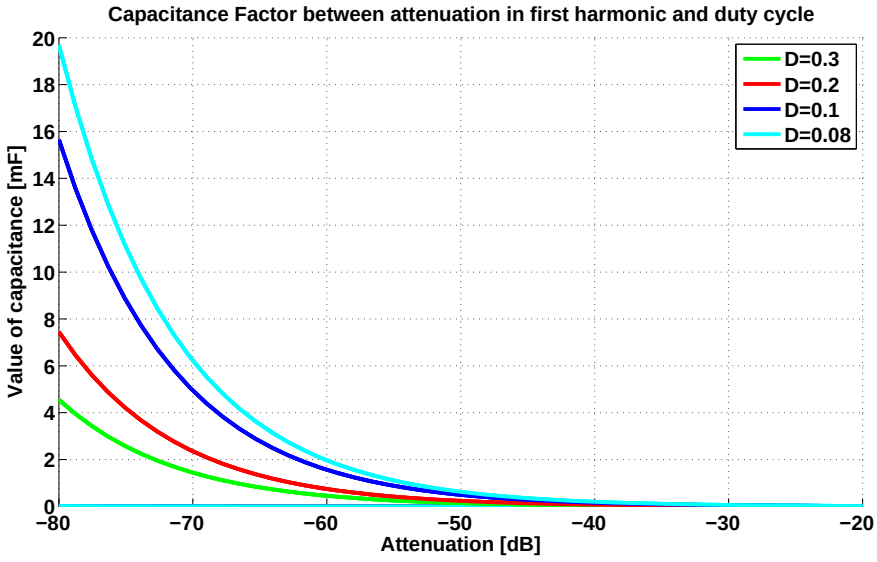


Figure 2.16: Input capacitance value for a single-phase Buck converter depending on the duty cycle when an attenuation of -40 dB is applied in the fundamental frequency

$$|I_{ns}|_{dB} = \alpha'_{dB}(f) - 20 \cdot \log(D) \tag{2.57}$$

Last equation explains how the duty cycle degrades the attenuation of the switching frequency filter in a single-phase converter. Figure 2.16 shows that the capacitance must be bigger when the duty cycle is lower.

Multi-phase converters For multi-phase power converters, the procedure to calculate the value of the input capacitance is basically the same. The main difference strives in the cancellation effect of the *Form-Factor* function. Looking at Figure 2.17, the model of the input current capacitance in multi-phase converters can be seen to be formed by the same blocks than in Figure 2.15. The block *Form-Factor* function is considered to be part of the input filter because it cancels all the harmonics of the input current except those multiple of N (number of phases).

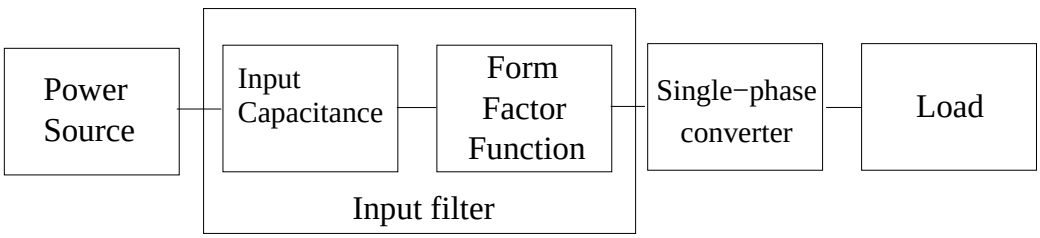


Figure 2.17: Input filter for multi-phase converters

The value of the input capacitance must be calculated forcing that the first harmonic, k, in the multi-phase power converter is attenuated in a factor α. Therefore, Equation (2.53) is now:

$$|I_{ks}| = \underbrace{\frac{1}{2\pi f C_{inN}}}_{\text{transfer function}} \cdot \underbrace{\left(\frac{I_{max} - I_{min}}{2(\pi k)^2 D} \sin(\phi_k)\right)}_{\text{Multi-phase current spectrum}} \tag{2.58}$$

Then, thanks to the effect of the form factor function, the first harmonic in multi-phase power converters is placed at:

$$k = N \cdot n \tag{2.59}$$

with :

k: k^{th} harmonic of the multi-phase converter
n: n^{th} harmonic of the input-phase converter
N: Number of phases

The *Form-Factor* function also multiplies the value of each harmonic by a constant *N*. Finally, the value transfer function must be calculated when

$$f_{sw\_mp} = N \cdot f_{sw} \tag{2.60}$$

Substituing Equations (2.59) and (2.60) into (2.61), the value of the input capacitance in multi-phase converters is

$$C_{inN} = f(I_s, D, N) = \frac{(I_{max} - I_{min}) \sin(\phi_{N \cdot n})}{2(\pi N n)^2 D I_{1s} f_{sw}} \tag{2.61}$$

In a similar way, (2.58) can be splitted defining again the parameter I'_{kh}, as the value of the k^{th}-harmonic when the converter is single and the duty cycle is equal to 1. (2.58) turns into

$$|I_{ks}| = \frac{1}{2\pi f C_{in}} \cdot \frac{I'_{kh}}{N \cdot D} = \alpha'(f) \cdot \frac{1}{N \cdot D} \tag{2.62}$$

with $\alpha'(f)$ defined in (2.56). Applying logarithms to (2.62), the relationship of the attenuation, the number of phases and duty cycle for a *N-phased* converter is calculated like

$$|I_{ks}|_{dB} = \alpha'(f)_{dB} - 20\log(N) - 20\log D \tag{2.63}$$

Table 2.2 summarizes the required value for C_{in} to obtain an attenuation of 40 dB in the first harmonic.

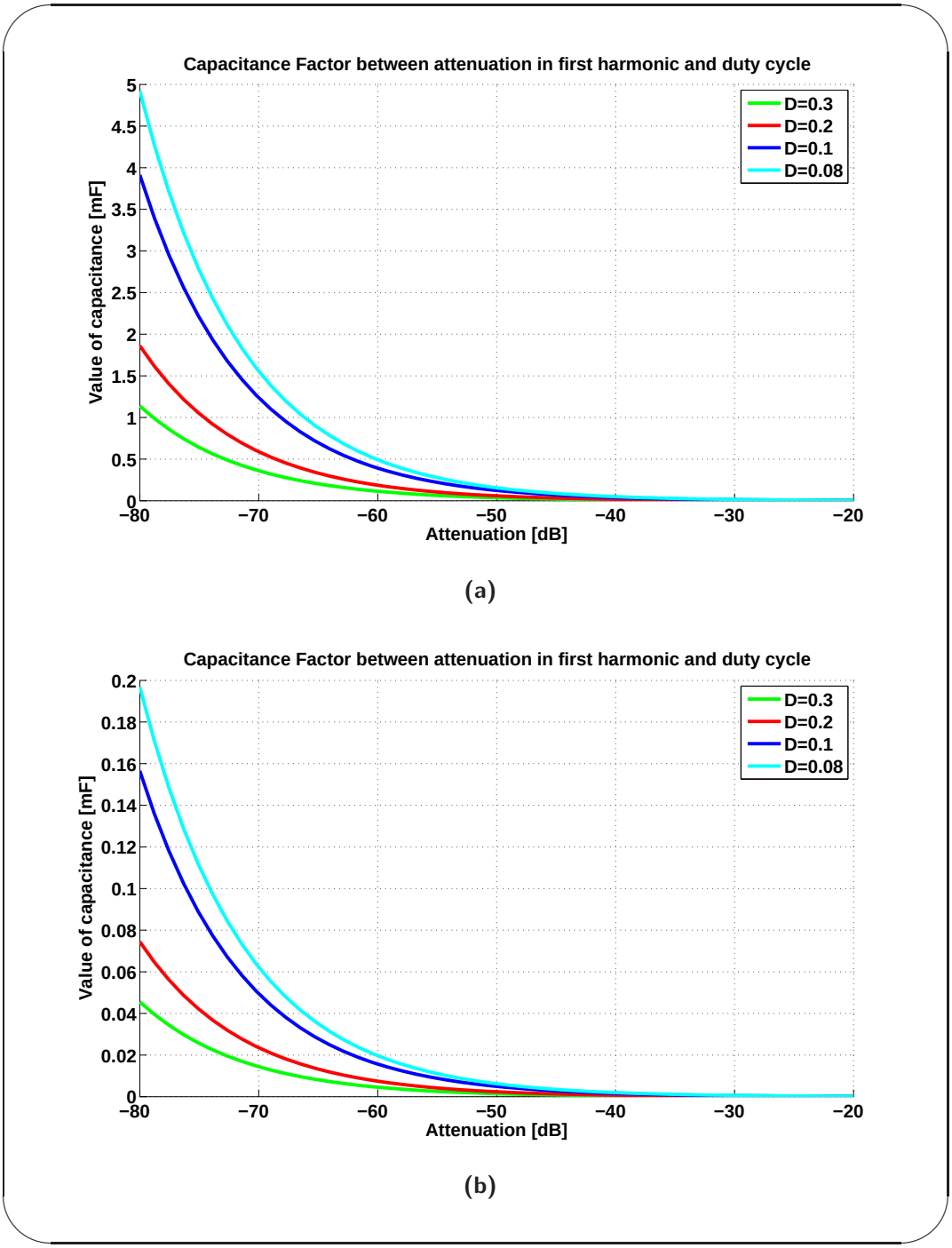


Figure 2.18: C_{in} value required to attenuate first harmonic when an attenuation of -40 dB must be applied to the first harmonic in a N-phased Buck converter. (a) shows the capacitor values when the multiphase converter is composed by 2 phases. (b) shows the capacitor values when the number of phases is 10

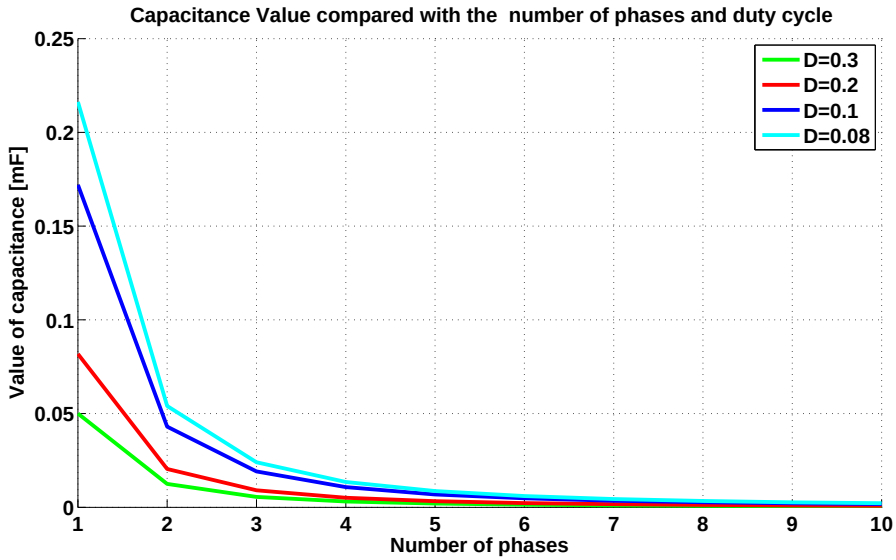


Figure 2.19: Input Capacitance value for multi-phase converters depending on the number of phases

Table 2.2: Capacitor values depending on the phase and duty cycle

Number of phases	D	C_{in} Value $[\mu F]$
1	0.3	50.0267
	0.2	81.7797
	0.008	216,3
2	0.3	12.5067
	0.2	20.445
	0.008	54.064
4	0.3	3.1267
	0.2	5.11
	0.008	13.59
10	0.3	0.5
	0.2	8.2
	0.008	2.2

2.3 Acquisition module

The function of the acquisition module is to sample and to quantize the information delivered by our analogue power converter into digital data useful for the digital controller. The device in charge of this important task is the ADC.

It is worth to note the number of bits required to code efficiently the output voltage according to the desired specification. Calculations for the I/O resolutions have been given in [3] in order to respect Intel design specifications.

The first main target is to assure that our system has enough resolution to track accurately the analogue magnitude that should be converted in the digital domain. It was proved in [3] that the minimal number of bits for our case is eight bits, following Equations (2.64):

$$\Delta V_{out\ max} > \Delta V_{q\ ADC} > \Delta V_{q\ DPWM} \tag{2.64}$$

$$\underbrace{\Delta V_{out\ max}}_{\text{maximum voltage variation}} \geq \underbrace{\frac{V_{fs\ ADC}}{2^{n_{ADC}} \cdot H_v}}_{\Delta V_{q\ ADC}\ =\ \text{voltage quantification error}} \tag{2.65}$$

being,

- $\Delta V_{out\ Max}$ = maximal variation allowed in the measured variable
- $\Delta V_{q\ ADC}$ = ADC voltage quantification level
- $\Delta V_{q\ DPWM}$ = DPWM quantification level
- $\Delta V_{fs\ ADC}$ = ADC full-scale voltage
- n_{ADC} = ADC bits number
- H_v = sensor gain

In our study case, we use a pipeline ADC architecture with parallel data ouput interfaced with the FPGA. ADC in the given system was selected to be ADC08060 [5]. It is a low-power, 8-bit, monolithic converter with an on-chip track-and-hold circuit. This ADC topology presents the best trade-off between resolution and conversion times introducing a reduced delay to our system which is acceptable for the given power converter. It is important to notice that digital output is not acquired instantaneously in the selected ADC. As [5] shows, digital samples are available after 6 clock pulses. It is going to be important to synchronize our FPGA with our acquisition system.

2.4 Digital control law generation

2.4.1 Introduction

Feedback control systems are used in everyday life to assure that the output response in a physical system called "plant" corresponds to our desired specifications. Examples of a control system in nature are in warm-bloodedness animals where blood temperature is kept constant even if external conditions are extreme for life.

In our study case, the system plant is a one-phase synchronous Buck converter. The main parameter to be controlled is the power converter output voltage depending on load conditions, input voltage and command disturbances. Therefore, the output voltage must be always remaining constant over the time and the mentioned previous conditions.

In order to achieve a stable response in the system, power converters are operated in a negative-feedback architecture (see Figure 2.1b), respecting some design guidelines.

By introducing a linear controller before the power converter, we can modify the dynamics according to the classical relationships in the field of control systems. Nevertheless, the control could also be non-linear. For example, a popular non-linear control technique could be the sliding control method. The reason why a linear controller was chosen instead of the sliding control is due to a better steady-state error operation. Thus, the problem of designing a controller into a digital architecture corresponds to the problem of implementing the digital response of a LTI system.

Figure 2.1b allows us to derive the relationship between V_{ref} and V_{out}. The knowledge of the transfer function of our system including the power plant is essential to define the transfer function of our controller. A guide explaining how to place efficiently poles and zeros in this kind of systems, by means of the pole-zero matching technique, can be found in [3]. For example, if one of our requirements is that the steady-state error of the output is zero, then, one integrator element is needed to be implemented in the controller.

$$\frac{V_{out}(s)}{V_{ref}(s)} = \frac{G_{controller}(s)G_{buck}(s)G_{DPWM}(s)}{1 + G_{controller}(s)G_{buck}(s)G_{DPWM}(s)G_{ADC}(s)} \tag{2.66}$$

Because the resulting transfer function of the controller is a linear system, we are able to use all the mathematical tools related to the analysis of linear systems.

2.4.2 Digital filter

2.4.2.1 Introduction

The controller block, should be implemented in the FPGA. This block is designed to perform the desired static and dynamic performances in terms of steady-state error, overshoot or rise time, delivering the command introduced to the DPWM by means of a given digital control law.

Since the first time PIDs were used, there has been many works related to the configuration of its parameters and architectures. For example, Ziegler-Nichols methods were useful to tune the parameters of PIDs thanks to the knowledge of the power converter dynamics and also the final specification of the problem [6]. Another characteristic of the PID is the possibility to be implemented in different manners. Parallel and serial PID are the most popular forms of the same idea. The main advantage why in industry PID has been so succesful along the years was because of the easiness of implementation as an analogue controller.

Later on, with the inclusion of digital computers and digital reconfigurable architectures in every day life, the PID became a less interesting implementation due to the new computation power. The possibility to implement very complex controllers in a tiny package offering better dynamics for the response of the power converter took the place of PID controllers in industry. The main problem, as it will be explained in the next chapter, is the requirement of a profound analysis when fixed-point arithmetics are used for the implementation of the filter.

2.4.2.2 PID

Classical digital filters are based on PID architectures. PID acronym is due to words: Proportional, Integral and Derivative. The compensation effects caused by each one of the filter branches are:

- Proportional: multiplies current error
- Integral : performs accumulation of current and past error
- Derivative : asses derivation

Table 2.3 shows how we can tune all elements according to the desired effect we want to obtain in the output system response. Classical tunning methods are based on the Ziegler-Nichols formulation [7]. In this tunning technique the I and D gains are first set to zero. The "P" gain is increased until it reaches the "critical gain", K_c, at which the output of the loop starts to oscillate. K_c and the oscillation period P_c are used to set the gains as shown in Table 2.3

PID architecture can be implemented by several ways. It can be implemented by means of either a parallel architecture, where all blocks perform

Table 2.3: Ziegler - Nichols method

Control Type	K_p	K_i	K_d
P	$0,5K_c$	-	-
I	$0,45K_c$	$1,2K_p/P_c$	-
D	$0,6K_c$	$2K_p/P_c$	$K_pP_c/8$

every function or a serial architecture, where the PID function is implemented in a implicit form taking the difference of a parallel form.

$$u[n] = K_p e_v[n] + K_i \sum_{i=0}^{+\infty} e_v[n] + K_d \Delta e_v[n] \tag{2.67}$$

where

- $u[n]$ is the output of the PID controller
- $e_v[n]$ is the input of the PID controller
- K_p is the gain of the proportional branch
- K_i is the gain of the integrative branch
- K_d is the gain of the derivative branch

To obtain the series equivalent architecture, we take difference of input we have :

$$\Delta u[n] = K_p \Delta e_v[n] + K_i e_v[n] + K_d \Delta^2 e_v[n] \tag{2.68}$$

$$\Delta e_v[n] = e_v[n] - e_v[n-1] \tag{2.69}$$

$$\Delta^2 e_v[n] = \Delta(\Delta e_v[n]) = e_v[n] - 2e_v[n-1] + e_v[n-2] \tag{2.70}$$

substituing

$$\Delta u[n] = (K_p + K_i + K_d)e_v[n] - (K_p + 2K_d)e_v[n-1] + K_d e_v[n-2] \tag{2.71}$$

$$\Delta u[n] = K_0 e_v[n] + K_1 e_v[n-1] + K_2 e_v[n-2] \tag{2.72}$$

finally the expression of the series output PID is:

$$u[n] = u[n-1] + K_0 e[n] + K_1 e[n-1] + K_2 e[n-2] \tag{2.73}$$

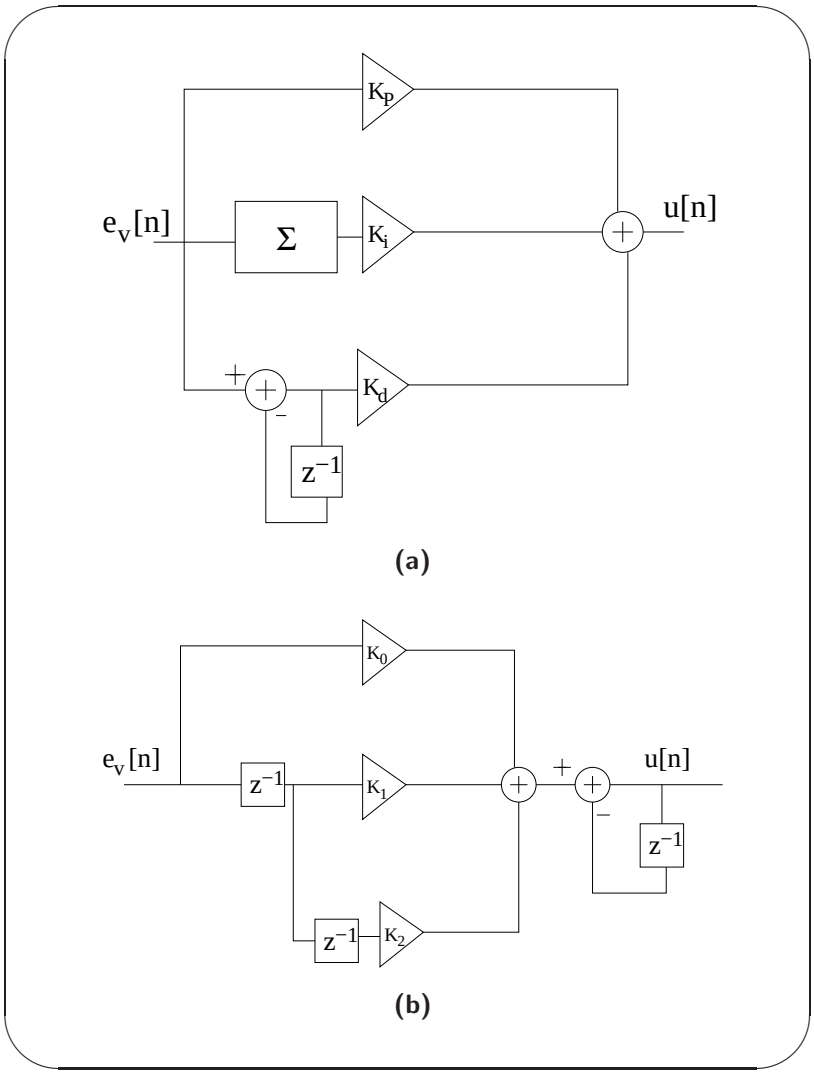


Figure 2.20: PID implementations. (a) shows the parallel form. (b) shows the serial form

$$K_0 = (K_p + K_i + K_d) \tag{2.74}$$
$$K_1 = (K_p + 2K_d)$$
$$K_2 = K_d \tag{2.75}$$

Parallel PID architectures owns an advantage over series one. The integrative term is made directly cycle by cycle. On the other hand, it is difficult to calculate maximum values over time and this is something very important in order to avoid nonlinearities such as overflow or wrap. For that reason, an antiwind-up filter is required in the parallel branch.

Since both PID are equivalent it is reasonable that both are having the same transfer function.

$$G_c(z) = \frac{K_0 + K_1 z^{-1} + K_2 z^{-2}}{1 - z^{-1}} \tag{2.76}$$

Based on the aforementioned PID tunning technique, called pole-zero matching, we present an example of our study case obtaining the next set of coefficients for the Buck converter in Table 2.1. The design is explained in [3].

$$K_0 = 1.5127 \quad K_1 = -2.2123 \quad K_2 = 0.761 \tag{2.77}$$

Using Equation (2.77) and the *Matlab* toolbox *ltiview*, we can see impulse response and Bode plot in Figures 2.21a and 2.22a. The impulse reponse filter is used as a pattern to validate our programming method in VHDL. Therefore, once our filters are programmed, the impulse response calculated in Matlab and the VHDL result should match except for the errors due to the numerical accuracy between previous systems (Floating-point and fixed-point number representations).

2.4.2.3 Structures of 3^{rd} order

On the other hand, to improve the dynamics of the output voltage, the following 3-rd order filter has been calculated [3]. The transfer function presented in this Master's Thesis, has been written rounding the coefficients in order to make easier the interpretation. Nevertheless, all decimals are considered for computing our algorithms.

$$G_c(z) = \frac{5z^2 - 9.789z + 4.79}{z^3 - 1.01066z^2 - 0.24466z + 0.2553} \tag{2.78}$$

2.4.2.4 Comparaison between PID and 3^{rd} order

Figures 2.21a and 2.21b represent the impulse response for the PID controller and the third-order filter respectively. We can appreciate in Figure 2.21a that the impulse response of the PID has an abrupt dynamic. That is, the number of samples required to reach the steady-state value of the PID is less than for a 3^{rd}-order filter. To demonstrate why it changes so quickly, we can use Equation (2.76) to invert the z-transform. The temporal sequence is:

$$u[n] = K_0\delta[n] + (K_0 + K_1)\,\delta[n-1] + (K_0 + K_1 + K_2)\,u[n-2] \tag{2.79}$$

being $\delta[n]$ the unitary impulse function and $u[n]$ the step function. Using the value of the coefficients given in Equation (2.77), the previous equation matches completely with Figure 2.21a. We can see in Equation (2.79) and

Figure 2.21a that the number of samples to reach the steady-state value are three.

On the other hand, the impulse reponse of the 3^{rd} order controller has an oscillatory transient and the best way to analyze it is using the decomposition of the transfer function using the residue method. This method is going to be used along the chapters, this can be found in Appendix A and in [8]. The residue method is very powerful to decompose a rational function where its polynomials have a high degree into a sum of simple rational functions. After applying the residue method, Equation (2.78) changes to the following form (coefficients are rounded) :

$$G_c(z) = \frac{0.001493}{z-1} + \frac{2.214852}{z-0.51066} + \frac{7.2134}{z+0.5} \tag{2.80}$$

To obtain the z-inverse function of the 3^{rd} order function controller, we can use Equation (2.80). The resulting time

$$\begin{aligned} u[n] =& 0.001493 \cdot u[n-1] + 2.214852 \underbrace{(0.51066)^n \cdot u[n-1]}_{\text{decreasing exponential}} + \\ &+ \underbrace{7.72134\,(-0.5)^n \cdot u[n-1]}_{\text{decreasing oscillatory exponential}} \end{aligned} \tag{2.81}$$

Equation (2.81) shows finally why the impulse response of the 3^{rd} order function has an oscillatory behavior. The last term is an exponential that changes of sign every sample. Therefore, the conclusion is that all poles located $-1 \leq \mid z \mid \leq 0$ have an oscillatory trend. Finally, thanks to the residue method, it can be seen that the number of poles matches with the number of exponentials forming the time-domain waveform.

On the other hand, Figures 2.22a and 2.22b represent both the bode chart for the PID controller and third-order filter.

For the PID controller, we can see how the integrative branch (see denominator of Equation (2.76)) eliminates the steady state error. The frequency response chart allow to check the existance of two zeros (number of solutions of the numerator in Equation (2.76)). One of the zeros is located at the natural frequency of the power converter in order to cancel one of the complex poles located in the natural frequency of the power converter, ω_0. The purpose of the second zero is to maximize the open-loop gain and to enlarge stability margins.

The third-order filter has also an integrative behaviour to cancel the steady-state error. The main difference with respect to the PID controller is that the zeros are now complex. This property makes it possible to cancel both complex poles of the power converter placed at $\frac{1}{\sqrt{LC}}$ making the control system less oscillatory.

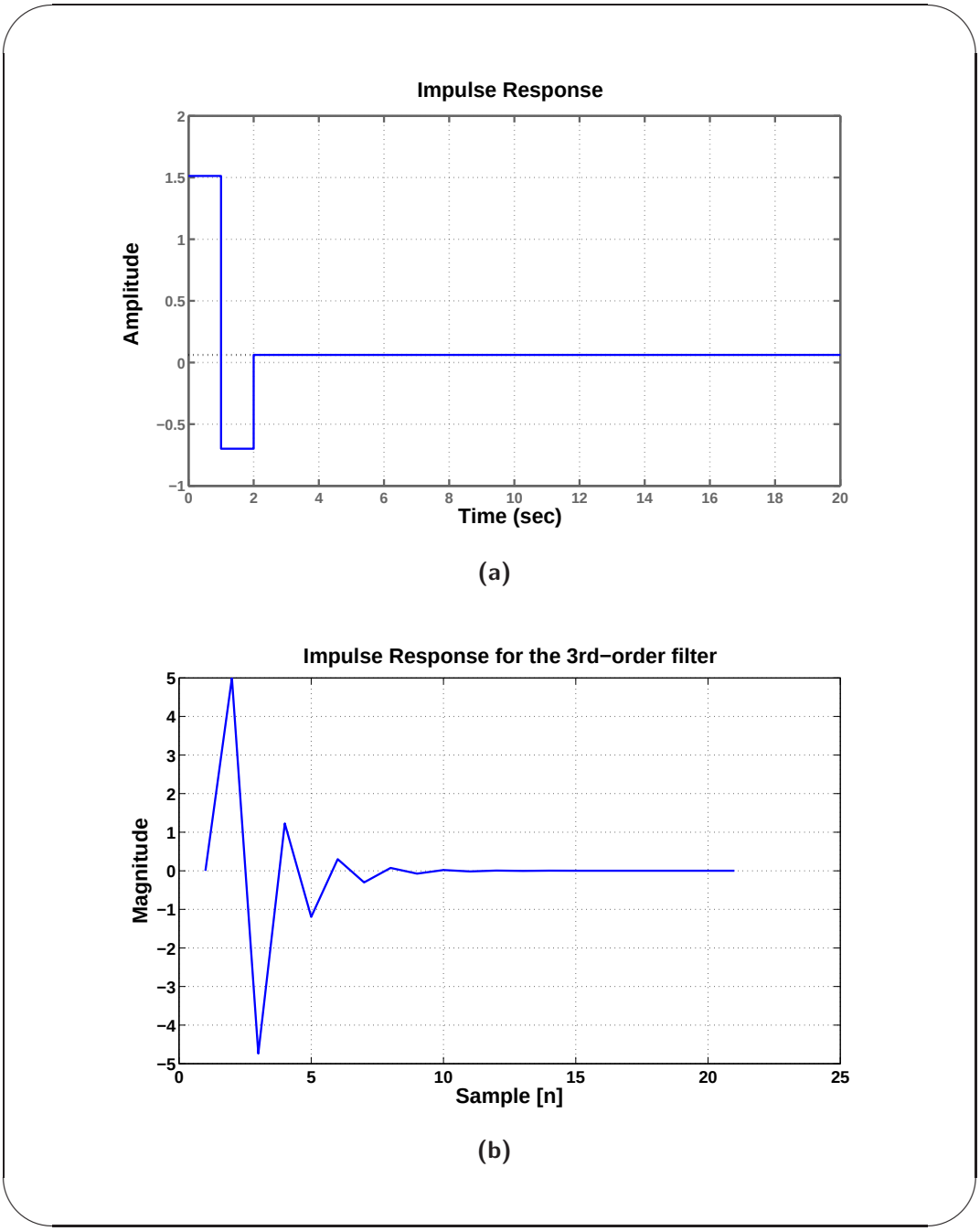


Figure 2.21: Impluse response of the linear controllers. (a) shows the PID impulse response, see Equation (2.79). (b) shows the 3^{rd}-order filter impulse response, see Equation (2.81).

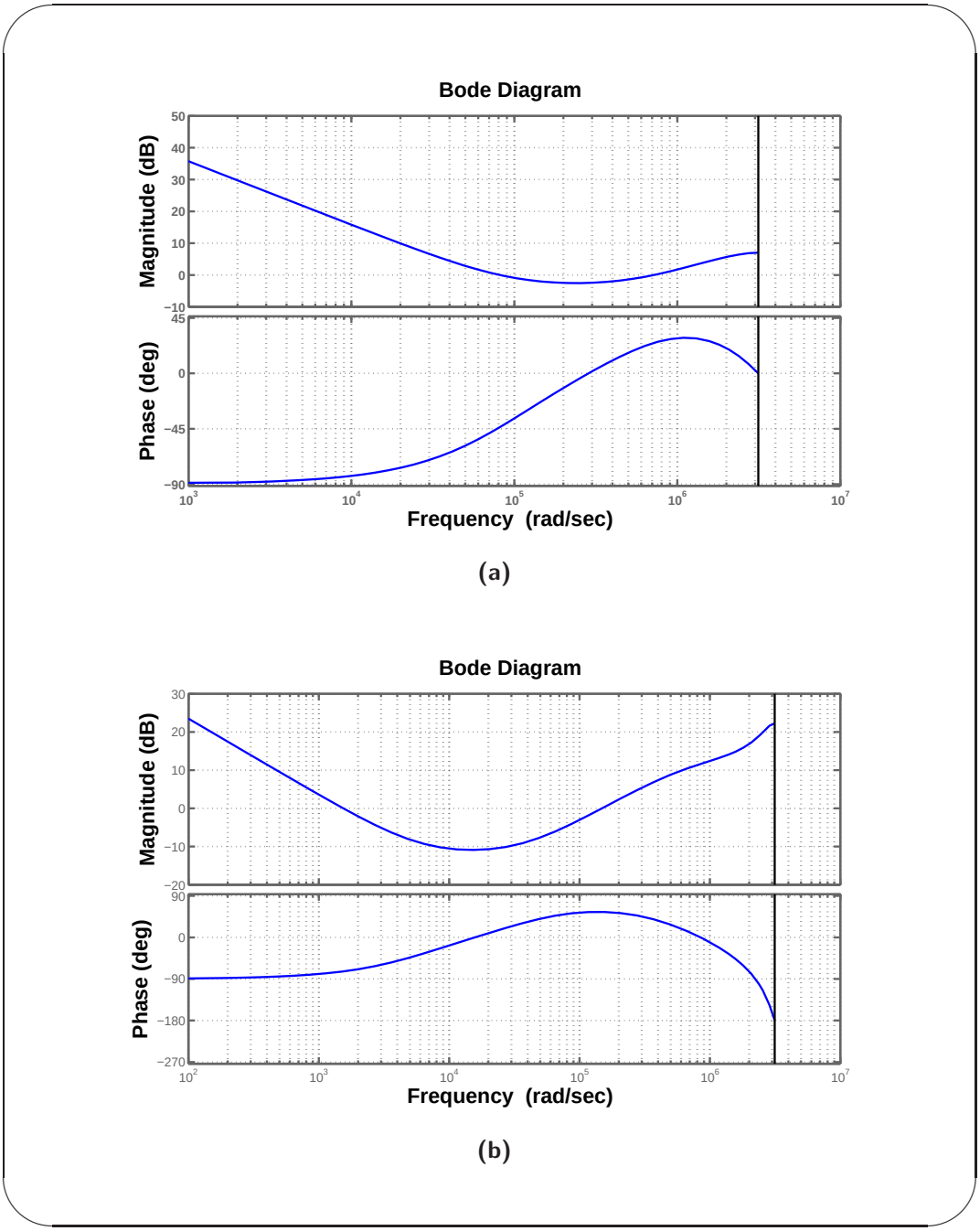


Figure 2.22: Bode plots of the linear controllers. (a) shows the PID Bode plot. Transfer function corresponds to Equation (2.76). (b) shows the 3^{rd}-order filter Bode plot. Transfer function corresponds to Equation (2.78).

2.5 Duty cycle generation

2.5.1 Introduction

This module performs the interface between digital and analogue world. There are some architectures to create a PWM pattern. The analogue way to do this is by comparing a saw-tooth waveform, $s(t)$, with a signal $u(t)$. The output waveform is then expressed by means of the following formula:

$$d(t) = \begin{cases} 1 & \text{if } s(t) < u(t) \\ 0 & \text{if } s(t) > u(t) \end{cases} \tag{2.82}$$

The digital PWM generation lies on the same principle expressed by Equation (2.82). The main advantage of using this technique is that the hardware performing PWM patterns can be implemented using digital circuits. This fact allows to increase the use of the logic elements included inside of the FPGA and consequently the minituarization of the electronic system. Nevertheless, the FPGA does not include the most optimal hardware to implement an accuarate DPWM. There are mainly three families of DPWM architectures [9- 12].

- Fast-clock counter
- Delay line loop
- Hybrid

The DPWM owns a quantizer effect of a number comprised between 0 and 1, since the controller is having a normalized output between -1 and 1. The more bits are used for the DPWM, the more accurate is our output voltage in VRM and therefore we decrease a source of error in the loop. The quantizer block is represented as the Figure 2.23:

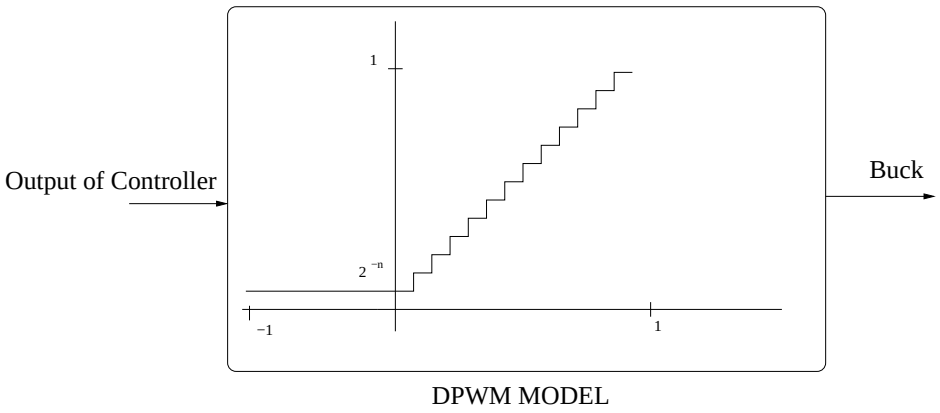


Figure 2.23: DPWM Model Block

The number of elements in the staircase is composed of 2^{-n}, n being the number of total bits. The number of bits required is given in [3] as

$$\begin{aligned} n_{DPWM} &= \left\lfloor n_{ADC} + \log_2\left(\frac{V_{out}H_v}{V_{fsADC}D_{min}}\right)\right\rfloor + 1 \\ &= \left\lfloor \log_2\left(\frac{V_{out}H_v}{\Delta V_{qADC}D_{min}}\right)\right\rfloor = \lfloor x \rfloor \end{aligned} \tag{2.83}$$

being

- V_{out} = Output Voltage
- D_{min} = Minimum duty cycle
- $\lfloor x \rfloor$ is equivalent to the greatest integer greater or equal to x

In our study case, the maximum number of bits that we are using is 10, because of hardware constraint associated to the used FPGA. The number of bits, shown in [3], is 12.

2.5.2 DPWM architecture

2.5.2.1 Fast clock counter

The saw-tooth signal is generated by means of a counter of n bits, where n is the number of bits of our controller. The PWM output waveform comes from the comparison between a counter and the controller output. When the counter is equal to zero, the output of the fast clock counter is set to one, but when the counter reachs the number equal to $u[n]$, the SR latch changes the output of the fast clock counter to zero. The scheme of this architecture is shown in Figure 2.24. For a fast-clock counter architecture owning a generic number of n bits, the input counter frequency is :

$$f_{counter}(n) = 2^n \cdot f_{sw} \tag{2.84}$$

being n, the number of bits introduced at the input of fast clock counter and f_{sw} is the switching frequency of the Buck converter.

Example 2.5.2.1. *For a binary word consisting of 16 bits and a Buck converter switching at $1MHz$, the internal clock frequency of the counter must be equal to*

$$f_{counter}(16) = 65,536 \ GHz \tag{2.85}$$

To respect restrictive constraints of accuracy for controlling our power converter, a high number of bits is necessary to generate a fine DPWM pattern. Thus, the fast clock counter topology requires a high frequency imposed by Equation (2.84). As we know, higher frequencies are more demanding in terms of energy consumption.

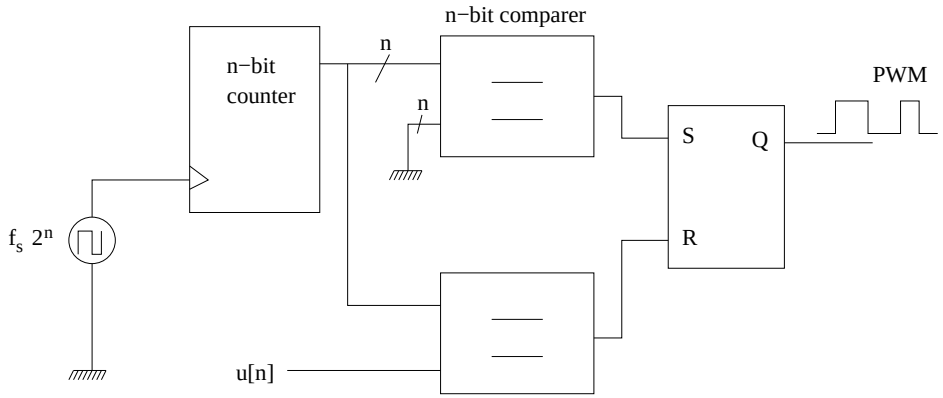


Figure 2.24: Scheme of a fast clock counter

2.5.2.2 Delay-line loop

A delay-line loop consists of an array of buffers placed sequentially. A pulse is generated every T_{sw} at the input of the delay-line. Every buffer has associated a propagation delay Δt between elements. The work principle relies on a comparaison between the digital controller output and the instant delay using a multiplexor as it is ilustrated in Figure 2.25. To achieve this comparison, we need only a latch at the output stage of the Delay-line loop. In comparaison to the previous architecture, the problem arises in the number of buffers in the delay-line. Therefore, the main drawback is in the size of sillicon that this structure requires to be implemented. In fact, the number of delay elements increases exponentially with the accuracy. The propagation delay for every buffer in the delay-line must be equivalent to one MSB of the digital word as stated in Equation (2.86)

$$t_{delay} = \Delta t = 2^{-n} \tag{2.86}$$

n being the number of bits.

2.5.2.3 Hybrids

As normally happens, the best solution consists of a mix of proposed solutions. Hybrids are implemented by a counter and a delay line loop. The main idea is to divide the n-bit word, as it is shown in Figure (2.27), coming from the digital controller into two parts. This allow decreasing drawbacks related to high frequencies or the high number of tap delays. The demonstration requires some definitions.

Definition 2.5.2.1. *The data width of the DPWM hardware consisting of* n *bits can be divided in two parts. Coarse and fine adjustment.*

$$\underbrace{n}_{Data\ width} = \underbrace{a}_{Coarse} + \underbrace{b}_{Fine} \tag{2.87}$$

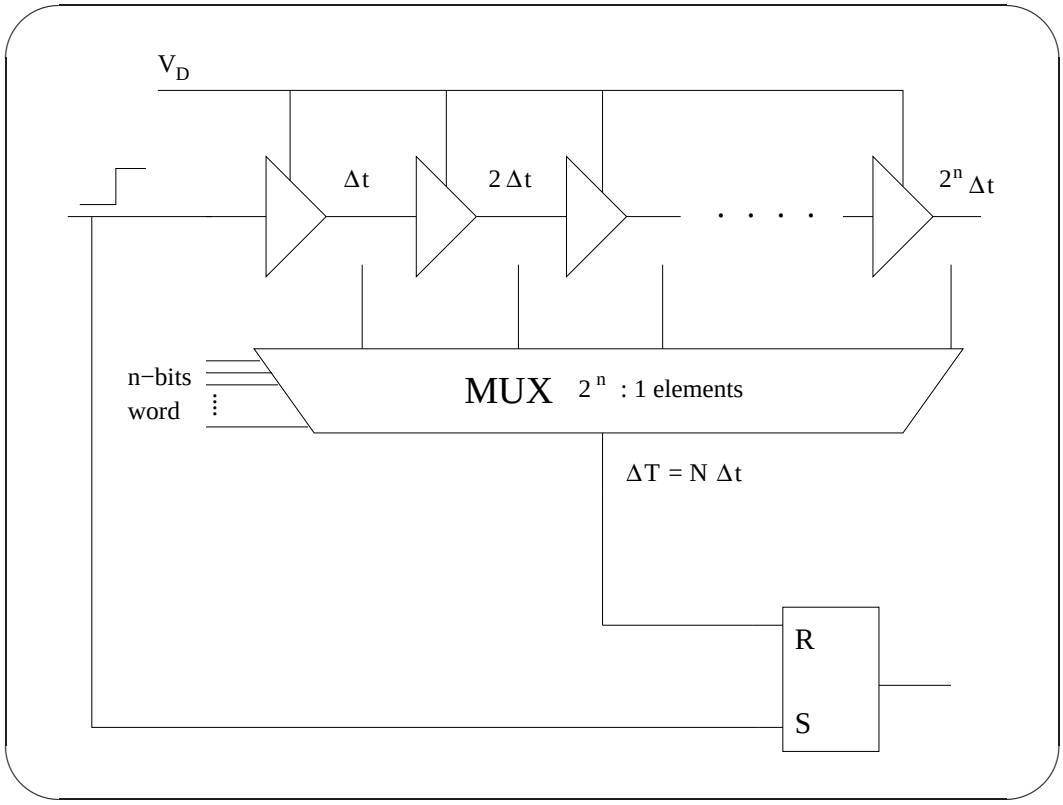


Figure 2.25: Delay-line loop

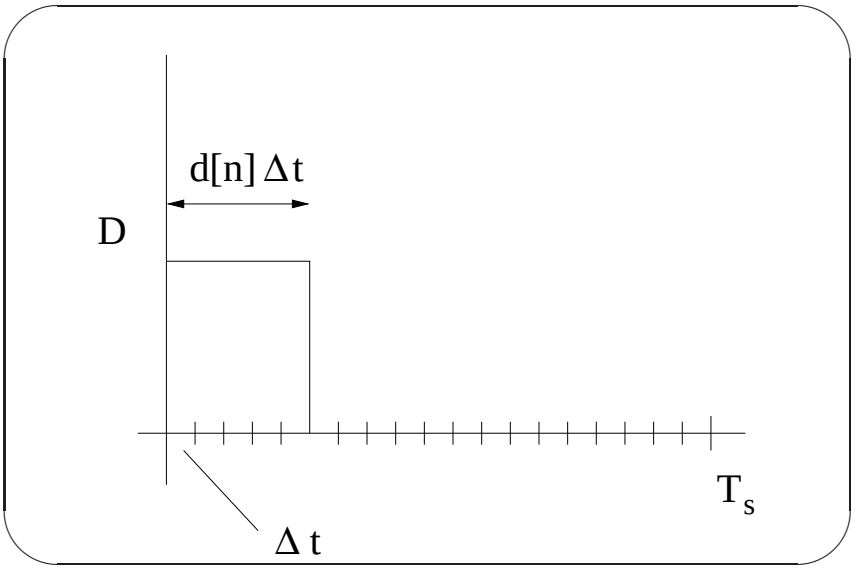


Figure 2.26: Quantizer effect of DPWM

Definition 2.5.2.2. *The duty cycle is the result of multiplying the binary word entering into the DPWM by the quantization step of the DPWM generation block. The more bits are used to generate the digital word, the more accurate is the generation of the duty cycle.*

$$D = \underbrace{d[n]}_{Binary\ word} \cdot \underbrace{\Delta t}_{quantization\ step} \tag{2.88}$$

Definition 2.5.2.3. *The quantization step for the DPWM is inversely proportional to the data width of the binary word. A closed expression to determine the quantization step is given by*

$$\Delta t = \underbrace{2^{-n}}_{LSB\ the\ binary\ word} \cdot \underbrace{T_{sw}}_{switching\ period} \tag{2.89}$$

Figure 2.26 summarizes all the previous definition. It can be easily appreciated how the duty cycle is formed discretizing the switching time, T_s in quantization steps of Δt.

Definition 2.5.2.4. *All binary words can be represented by means of*

$$d[n] = \sum_{i=0}^{n-1} a_i 2^i \tag{2.90}$$

After all the previous definitions, we are ready to prove why a Hybrid architecture can be generated by a delay-line loop and a fast clock counter altogether. The duty cycle can be rewritten using Equations (2.88), (2.90) and (2.89).

$$D = 2^{-n} T_{sw} \sum_{i=0}^{n-1} a_i 2^i \tag{2.91}$$

Equation (2.87) is useful to split the binary word into the coarse and fine adjustment.

$$D = 2^{-n} T_{sw} \left(\underbrace{\sum_{i=b}^{n-1} a_i 2^i}_{\text{coarse adjustment}} + \underbrace{\sum_{i=0}^{b} a_i 2^i}_{\text{fine adjustment}} \right) \tag{2.92}$$

Taking the common factor of 2^b at the coarse part of the binary word, Equation (2.92) results

$$D = 2^{-n} T_{sw} \left(2^b \sum_{i=0}^{n-b-1} a_i 2^{i-b} + \sum_{i=0}^{b} a_i 2^i \right) \tag{2.93}$$

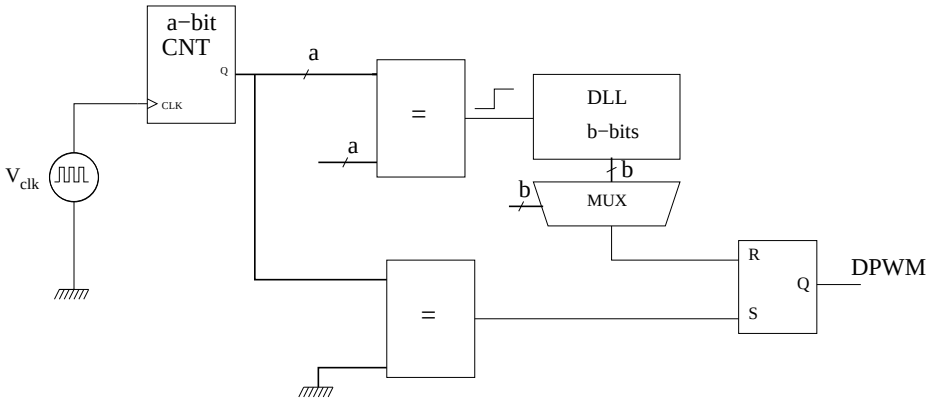


Figure 2.27: Hybrid Scheme

Operating in Equation (2.93) and using $n - b = a$, the duty cycle is now equal to

$$D = \underbrace{2^{-n+b} T_{sw} \sum_{i=0}^{a-1} a_i 2^i}_{coarse\ adjustment} + \underbrace{2^{-n} T_{sw} \sum_{i=0}^{b-1} a_i 2^i}_{fine\ adjustment} \tag{2.94}$$

$$D = \underbrace{2^{-a} T_{sw} \sum_{i=0}^{a-1} a_i 2^i}_{\text{fast clock counter}} + \underbrace{2^{-n} T_{sw} \sum_{i=0}^{b-1} a_i 2^i}_{\text{delay-line loop}} \tag{2.95}$$

$$f_a = \frac{1}{2^{-a} T_{sw}} = 2^{-} a \cdot f_{sw} \tag{2.96}$$

The resulting Equation (2.95) shows the the operation of the hybrid is based on a coarse generated by a fast clock counter and a fine adjustment thanks to the inclusion of a Delay-line loop. The frequency of operation for the counter is given by Equation (2.96).

2.6 Conclusions

In the first part of this chapter the linear models of a switching power converter have been analyzed. These models are accurate enough for frequencies up to $\frac{f_{sw}}{2}$. The origin of these unaccuracies in frequencies higher than the switching frequency is related to the use of the state-space average method. For example, Figures such as 2.5, 2.6 and 2.7 show that all the oscillations appearing in the signals are not having any harmonic. In contrast to the real waveforms in a Buck converter, the inductor current, for example, is triangular-shaped. This shows that if the accuracy in the model is required, then non-linear (chaotic models) can be applied [13]. After the derivation

of a linear model for Buck converters, the small signal model is used for the analysis of the behavior of small disturbances. And finally, a set of transfer functions is defined to see how all the input parameters of a Buck converter affects the output voltage response.

After the study of the linear model of a power converter, an analytical study comparing the input current of a single-phase and a multiphased structure has been developed. Then, the spectrum of an interleaved multiphase power converter has been calculated. It has been shown how its harmonic content is strongly influenced by the spectrum of the single-phase converter plus the addition of a second function, Form Factor function. This discrete function is presented giving the harmonic magnitude value for a given number of phases and it owns several singularities. As a consequence, it can be noted that the Form Factor function is zero for all multiples of the switching frequency. Nevertheless, this function has an uncertainty at the critical point, $N \cdot f_{sw\_mp}$. Therefore, in a single-phase converter all frequency contents are centered over the switching frequency and their first multiples and they are by far larger than DC contents for very low duty cycles. Thus, it has been found that the smaller duty cycle is, the larger AC frequency components are found. On the other hand, the average amplitude for DC current is directly proportional to the duty cycle. It is quite important to determine the working point of the power converter and the final design of the input-filter capacitor value in order to decrease the number of input capacitors and the final cost of the system. Comparing (2.61) and (2.54), we see the value of the input capacitance of a *N-phased* converter is related to the value for *single-phase* converters just dividing the number of phases by N^2. In multi-phase converters, since $N > 1$ and $0 < D \leq 1$, both logarithms have opposite sign in (2.63). This is the reason why for the low duty cycles, it is required to increase the number of phases to lighten the value of C_{in}. Fig. 2.18 shows that for same duty cycles and attenuations, the values of C_{in} are always much lower when the number of phases increases.

Finally, the rest of the chapter ends with a short presentation of the rest of components necessary to perform a digital voltage regulation. First of all, the ADC used for our application is presented. Secondly, all the linear controllers used during master's thesis are presented and analyzed in a time/frequency frame. A derivation of the serial and parallel forms based on the use of difference equations has been done. Finally, the last part shows all the new ways to generate a PWM pattern digitally. It is been proved mathematically why hybrids configurations are the best option.

Chapter 3

Digital control laws implementation by means of reconfigurable architectures

3.1 Introduction

3.1.1 FPGA architectures

In general, FPGAs require three major types of elements:

- combinational logic
- interconnect
- I/O pins

These three elements are mixed together to form an FPGA fabric.

Figure 3.1 shows the basic structure of an FPGA that incorporates these three elements. The combinational logic is divided into relatively small units which may be known as **Logic Elements (LEs)** or **combinational logic blocks (CLBs)**. The LE or CLB can usually form the function of several typical logic gates but it is still small compared to the typical combinational logic block found in a large design. The interconnections between the logic elements are made using programmable interconnections. This interconnection may be logically organized into channels or other units. FPGAs typically offer several types of interconnections depending on the distance between combinational logic blocks. Moreover, clock signals are also provided with their own interconnection networks. The I/O pins may be referred as **I/O blocks (IOBs)**. They are generally programmable to be inputs or outputs and often provide other features as low-power or high-speed connections.

An FPGA designer must rely on pre-designed wiring, unlike a custom VLSI designer who can design wires as needed to make connections. The

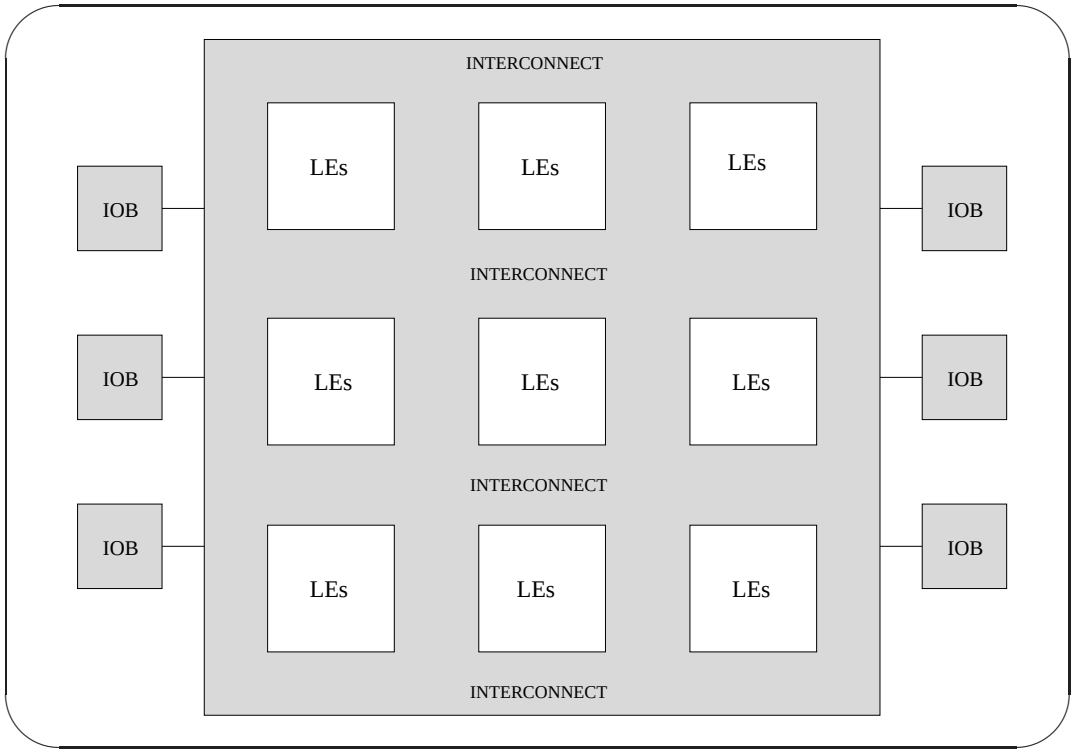


Figure 3.1: FPGA structure

interconnection system of an FPGA is one of its most complex aspect because wiring is a global property of a logic design.

Connections between logic elements may require complex paths since the LEs are arranged in some sort of two-dimensional structure as shown in Figure 3.1. We therefore need to make connections not just between LEs and wires but also between wires themselves. Wires are typically organized in **wiring channels** or **routing channels** that run horizontally and vertically through the chip. Each channel contains several wires. For example in Figure 3.2, the net in the figure starts from the output of the LE in the upper-right-hand corner, travels down vertical channel 5 until it reaches horizontal channel 2, then moves down vertical channel 3 to horizontal channel 3. It then uses vertical channel 1 to reach the input of the LE at the lower-left-hand corner. Therefore, it can be seen that the calculation of the delay is difficult to be calculated and it depends on how the mapping algorithms are mapping the required hardware for the controller application.

All FPGAs need to be **programmed** or **configured**. There are three major circuit technologies for configuring FPGA: SRAM, anti-fuse and flash. SRAM-based FPGAs hold their configurations in static memory. The output of the memory cell is directly connected to another circuit and the state of the memory cell continuously controls the circuit being configured.

Using static memory has several advantages:

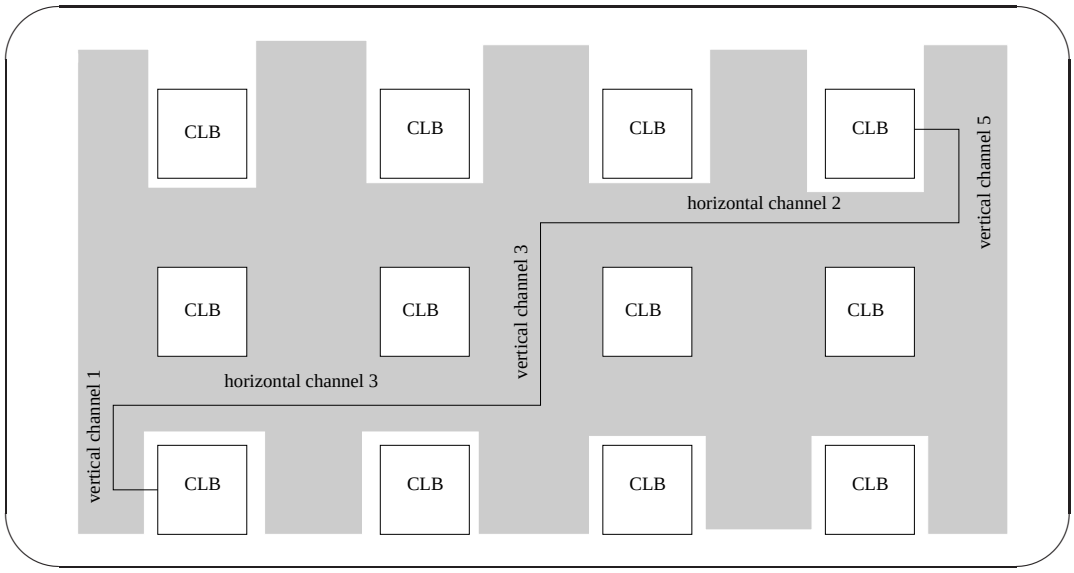


Figure 3.2: FPGA structure

- The FPGA can be easily reprogrammed. Because the chips can be reused, and generally reprogrammed without removing them from circuit, SRAM -based FPGAs are the generally accepted for system prototyping.
- The FPGA can be reprogrammed during system operation, providing dynamically reconfigurable systems.
- The circuits used in the FPGA can be fabricated with standard VLSI processes.
- Dynamic RAM, although more dense, needs to be refreshed, which would make the configuration circuitry much more cumbersome.

SRAM-based FPGA also have some disadvantages:

- The SRAM configuration memory burns a noticeable amount of power, even when the program is not changed.
- The bits in the SRAM configuration are susceptible to theft.

The basic method used to build a **combinational logic block (CLB)** also called a **logic element** or **LE** – in an SRAM-based FPGA is the **Lookup Table (LUT)**. As shown in Figure 3.3 , the lookup table is an SRAM that is used to implement a truth table. Each address in the SRAM represents a combination of inputs to the logic element. The value stored at that address represents the value of the function for that input combination. A *n-input*

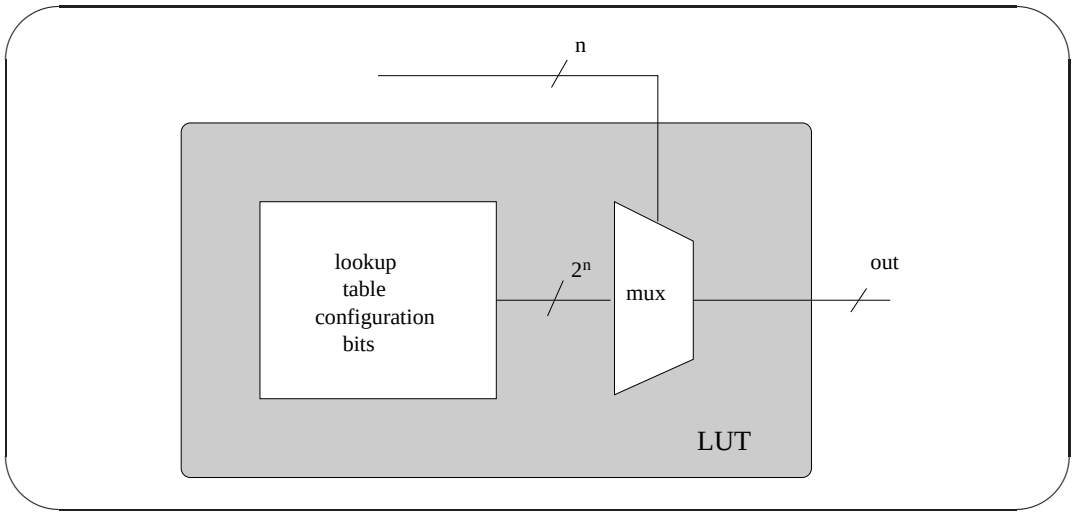


Figure 3.3: Lookup Table

function requires an SRAM with 2^n locations. Since a basic SRAM is not clocked, the lookup table LE operates much as any other logic gate.

Unlike a typical logic gate, the function represented by the LE can be changed by modifying the values of the bits stored in the SRAM. As a result, the *n-input* LE can represent 2^n functions. The delay through the lookup table is independent of the bits stored in the SRAM, so the delay through the logic element is the same for all functions. This means that, for example, a lookup-table based LE will exhibit the same delay for a 4-input XOR and a 4-input NAND. In contrast, a 4-input XOR built with static CMOS logic is considerably slower than a 4-input NAND. Of course, the state logic gate is generally faster than the LE.

Logic elements generally contain registers – flip-flop and latches- as well as combinational logic. A flip-flop or latch is small compared to the combinational logic element. As shown in Figure 3.4, the memory element is connected to the output; whether it stores a given value is controlled by its clock and enable inputs.

More complex logic blocks are also possible. For example, many logic elements also contain special circuitry.

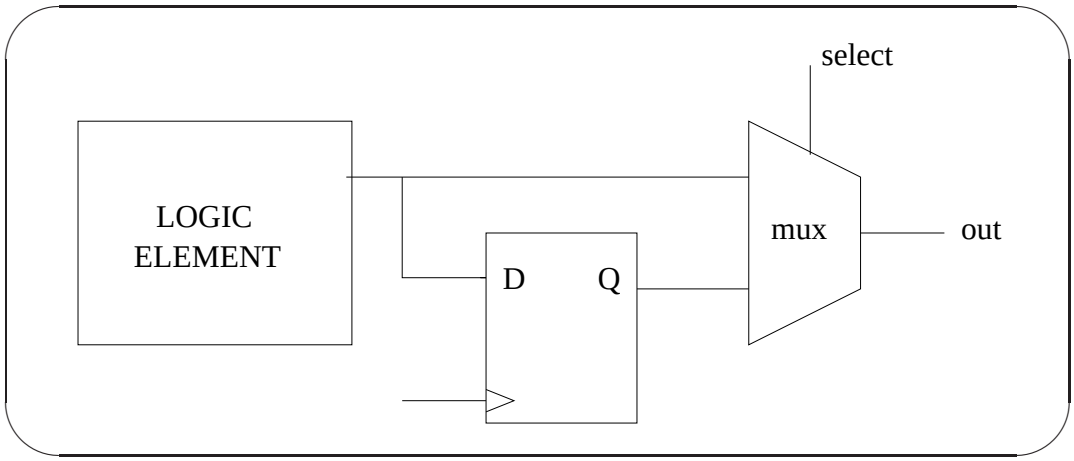


Figure 3.4: Flip Flop with Logic Element

3.1.2 State of the art of hardware description languages

This chapter highlights the historical reasons for the standardisation of HDL languages. Later on, it comments differences among two of the most popular HDL languages, and, finally it discusses the reasons for choosing VHDL.

In the VLSI era, a structured design process is required. In response to this need, considerable efforts have been being expended in the development of design aids. Hardware description languages are a specific example of this and a great deal of effort is being expanded in their development. Actually, the use of these languages is not new. Languages, such as CDL, ISP and AHPL have been used since the 70's. However, their primary application have been the verification of a hardware architecture without a high degree of accuracy and the verification of a design's architecture. That is, their timing model is not precise, and/or their language constructs imply a certain hardware structure. Newer languages, such as Verilog and VHDL, have more universal timing models and they imply no particular hardware structure.

Thus, Hardware Description Languages have two main applications which are documenting a design and modelling it.

In general, all useful hardware description languages are supported by a simulator. Thus, the model inherent in an HDL description can be used to validate a design. Prototyping of complicated systems is extremely expensive. Thus, the goal of those concerned with the development of hardware languages is to replace this prototyping with validation through simulation. Another important use of HDL descriptions in synthesis, that is, the automatic generation of gate-level circuits from the HDL model. HDL models are also used for test generation.

The use of hardware description languages in the design process implies a different approach than those used in the past. Formerly, digital designers were restricted to rely on a combination of word descriptions, block diagrams,

timing diagrams, and logic schematics to describe their diagrams. Also, for a designer to be personally involver in the simulation process was considered to be an *"anathema"*. The attitude was, *"those who can't design, simulate"*. The situation today is markedly different. Designers have a great deal more training in software. They prepare their designs in computer environment, where the design is entered as an HDL source file, as a schematic diagram, or as some combination of both approaches. Simulation is a tool frequently employed by the designers themselves to verify the correctness of the design. Synthesis tools translate the HDL models into gate-level designs.

To illustrate the concepts of an structured design, a specific hardware description language must be used. Until recently, there has not been a standard hardware description language, as there are standard programming languages such as C, C++, and JAVA. However, beginning in 1983, the US Department of Defence sponsored the development of the VHSIC (very high speed integrated circuit) hardware description language (VHDL). The original intent of the language was to serve as a means of communicating designs from one contractor to another in the VHSIC program.

In June 1987, eligible members voted to accept this version of VHDL as the standard one. In december 1987, it became official. From 1988 through 1992, VHDL users suggested minor changes to the language. The proposed changes were balloted in 1993, some of which were incorporated into the language.

3.1.2.1 Differences between most popular HDL languages

For the moment, there are currently available two industry standards related to the development of digital integrated circuits: VHDL and Verilog. As mentioned before, VHDL became IEEE standard 1076 in 1987. Nevertheless, Verilog has been used longer than VHDL and has been used extensively since it was launched by Gateway in 1983. It became IEEE standard 1364 in December 1995.

This section compares both languages depending on some features and tries to orientate the reader in order to justify why VHDL was chosen.

Table 3.1: Levels of hardware description details

Level of detail	Behavioral Domain Representation	Structural Domain Primitive
System	Performance Specification	Purpose of the project
Chip	Algorithms	Output Voltage Regulation
Register	Data Flow	Memory, Counter, Multiplier
Gate	Boolean Equations	AND, OR
Circuit	Differential Equations	Transistor, R, L, C

1. Capability

   - Every language allows us to describe the function of a digital circuit depending on several hierarchy abstraction. The grammar for every language may suit better for several type of hardware descriptions. Hardware description is a set of interconnected elements representing individual functions. Every element inside of a hardware must be represented in a certain level of detail to achieve a whole and more generic performance. Levels of details are divided into six (see Table 3.1). Modelling constructs in VHDL and Verilog cover a slightly different spectrum across the levels of behavioural abstraction.

2. Data Types

   - *VHDL* A multitude of language or user defined data types can be used. This may mean dedicated functions are needed to convert objects from one type to another. The facility of making your own personal set of data types allows to read better the code and also creates a link between real world and application.
   - *Verilog* Compared to VHDL, Verilog data types are simple, easy to use and very much geared towards modelling hardware structure as opposed to abstract hardware modelling. Unlike VHDL, all data types used in a Verilog model are defined by the language and not by the user.

3. Design re-usuability

   - *VHDL* Procedures and functions may be placed in a package so that they are available to any design-unit that wishes to use them.
   - *Verilog* There is no concept of packages in Verilog. Functions and procedures used within a model must be defined in the module.

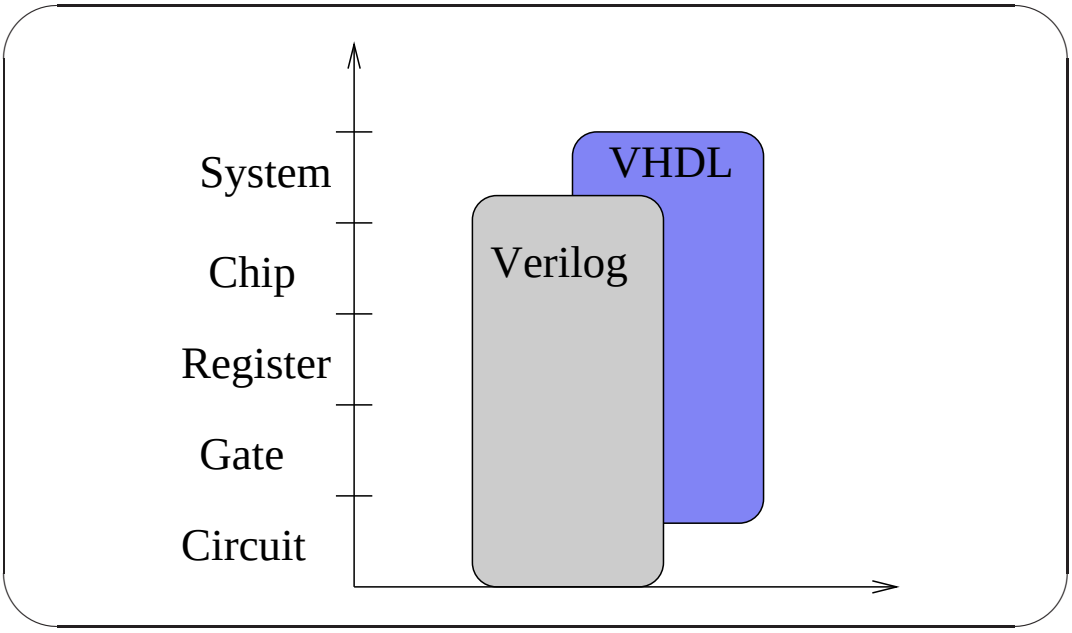


Figure 3.5: Representation of HDL details

4. Easiest to learn
   - Starting with zero knowledge of either language, Verilog is the easiest to learn and understand.
5. High Level Constructs
   - *VHDL* There are more constructs and features for high-level modeling in VHDL than there are in Verilog. Abstract data types can be used along with the following statements:
     (a) package statements for model reuse
     (b) configuration statements for configuring design structure
     (c) generate statements for replicating structure
     (d) generic statements for generic models that can be individually characterized, for example, bit width

     All these language statements are useful in synthesizable models.
   - *Verilog* Except for being able to parameterize models by overloading parameter constants, there is no equivalent to the high-level VHDL modeling statements in Verilog.
6. Libraries
   - *VHDL* A library is a store for compiled entities, architectures, packages and configurations. Useful for managing multiple design projects.

- *Verilog* There is no concept of a library in Verilog. This is due to their origins as an interpretive language.

7. Low Level Constructs
   - *VHDL* Simple two input logical operators are built into the language, they are: NOT, AND, OR, NAND, NOR, XOR and XNOR. Any timing must be separately specified using the after clause.
   - *Verilog* The Verilog language was originally developed with gate level modeling in mind, and so has very good constructs for modeling at this level and for modeling the cell primitives of ASIC and FPGA libraries.
8. Managing large designs
   - *VHDL* Configuration, generate, generic and package statements all help manage large design structures.
   - *Verilog* There are no statements in Verilog that help manage large designs.
9. Operators
   - The majority of operators are the same between two languages.
10. Procedures and tasks
    - VHDL allows concurrent procedure calls. Verilog does not allow it.
11. Structural replication
    - *VHDL* The **generate** statement replicates a number of instances of same design-unit or some sub part of a design.
    - *Verilog* There is no equivalent statement in Verilog.
12. Test harnesses
    - Designers typically spend about 50% of their time writing synthesizable models and the other 50% writing a test harness to verify the synthesizable models. Test harnesses are not restricted to the synthesizable subset and so are free to use the full potential of the language. VHDL has generic and configuration statements that are useful. Focussing on the application, we are willing to develop VHDL is going to be more convenient due to the ability of creating data types called in a closer way to the aplication. Also, it is possible to reuse the code among application. Thus,

we can easily create structural models for different control algorithms. Nevertheless, the difficulty in learning the whole syntax has been the order of several months to get to know the whole set of standard libraries, types of syntax verbose and, finally, the set of instructions for writing proper test benches.

3.2 Modeling linear systems using FPGAs

3.2.1 Introduction

Some common problems are performed in FPGA due to the lack of a specific arithmetic architecture like DSP cores. DSPs normally have a CPU whose architecture is Harvard, a set of RISC instructions to increase the computational speed of algorithms, a rigid ALU having all the arithmetic circuits necessary to perform filters and a C compiler that allows us to start a quick programming avoiding the knowledge of the DSP architecture.

On the other hand, the architecture of FPGAs is defined by the VHDL file that the user is typing. This freedom might be a double-edged sword. The advantage of this flexibility is multiple. First of all, there is an increase of parallelism and, consequently, speed. It is also important to think that when the application needs a custom design to be produced massively, FPGA plays an important role before manufacturing the same desgin into an ASIC. Therefore, the underlying advantage is the possibility to optimize our algorithm in terms of size.

Nevertheless, the lack of a fixed architecture inside of the FPGA package and the non-existence of a set of data types for synthesis can make the programming of the algorithm a random success. For that reason, the development of a generic algorithm in VHDL, based on a previous mathematical analysis, is interesting in order to speed up the development of algorithms.

3.2.2 Rational number system on FPGA

The numbers representing the signals in a control system belongs to the set of $\Re$ (Real Numbers). In an FPGA, the numerical representation is based on busses representing integer numbers, belonging to the set of Z. Therefore, it is necessary a systematic way to design and represent real numbers inside of the FPGA. Nowadays, there has been some attempts to perform some VHDL libraries representing rational numbers. The *fixed-point* and *floating-point* IEEE proposed libraries are able to synthetize hardware with the inconvenient of having low-speed architectures. Finally, after the study of the use of the fixed point libraries, we decided to implement our syntethisable VHDL code for the representation of rational numbers.

3.2.2.1 Fixed-point

The representation of a rational number in fixed point consists of the division of the number into two parts as it can be seen in the Figure 3.6. This representation is exactly the same as the used performing arithmetic by hand. The point that separates the integer part and the decimal part is not represented in the FPGA. For that reason, we follow in this report the WQX nomenclature which is shown in the Figure 3.6. This means that every rational registers have W bits where X bits of the length are decimal.

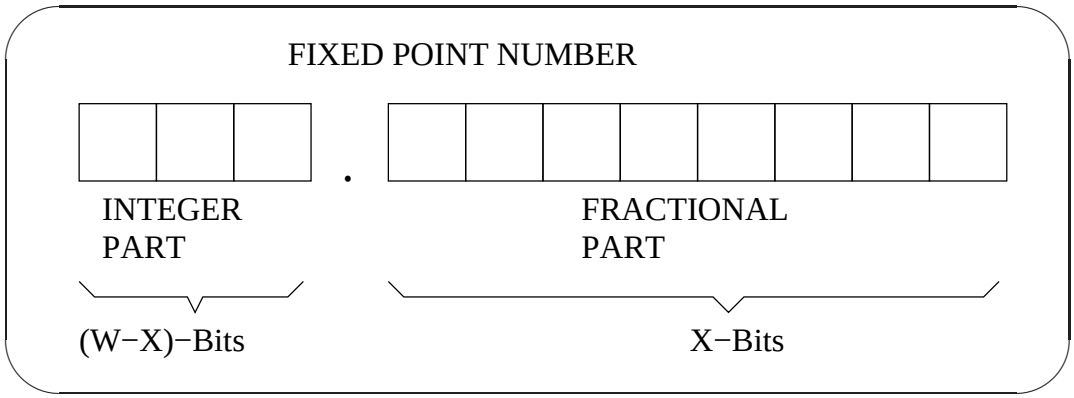


Figure 3.6: NQM Fixed-Point Structure

The range and accuracy of a WQX representation is given by

$$Range = \left[-2^{W-X-1}, 2^{W-X-1} - 2^{-X}\right] \tag{3.1}$$

$$Accuracy = 2^{-X} \tag{3.2}$$

Example 3.2.2.1. *Consider a rational number coded using a 11Q8 pattern. The range and accuracy of the codification is*

$$Range = [-4, 3, 99609375]$$

$$Accuracy = \frac{1}{256}$$

3.2.2.2 Floating-point

Floating-Point numbers are divided by two parts: mantissa and exponent. Every value can be translated into fixed-point by the means of the following relationship:

$$y = (-1)^S \cdot m \cdot \beta^{e-bias} \tag{3.3}$$

where m is an integer comprised between $0 \leq m \leq \beta^t - 1$ and being called *mantissa*. The sign is represented by s. The parameter β is the base representation of our digital architecture. It might be a numerical representation either binary, octal or hexadecimal. For the case of using a binary representations $\beta = 2$. Finally, if the exponent is represented using e bits, then the bias is selected to be

$$bias = 2^{e-1} - 1 \tag{3.4}$$

The *range* of the nonzero floating-points is given by

$$\beta^{e_{min}} \leq |y| \leq \beta^{e_{max}} \left(1 - \beta^{-t}\right) \tag{3.5}$$

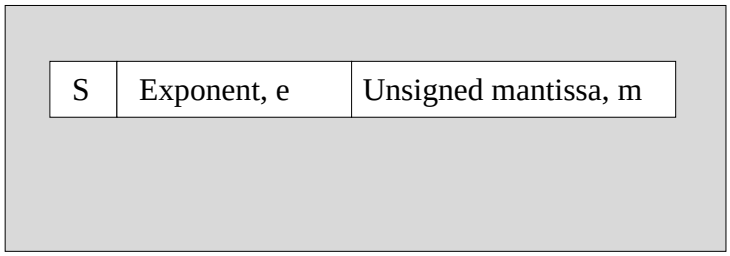


Figure 3.7: Floating Point Structure

Example 3.2.2.2. *Consider the 754-1985 ANSI/IEEE standard defining floating point numbers. A number can be represented by three different forms: single, double and extended. The only difference among them is the number of bits used to represent the number. Single precision contains 32 bits, double 64 and extended 80 bits. As a matter of an example, we will calculate the range for single precision numbers. The number of bits containing the mantissa, sign and exponent are illustrated in Figure 3.8.*

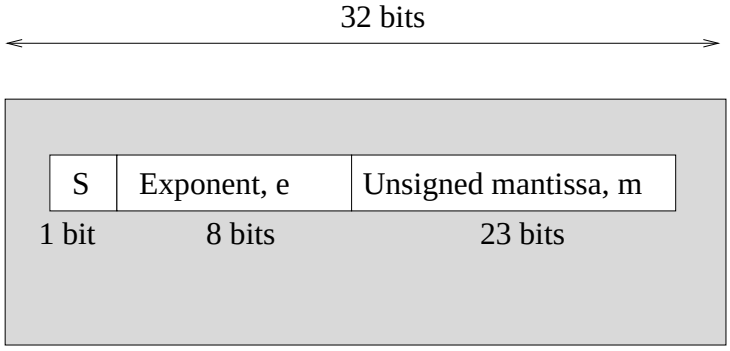


Figure 3.8: IEEE single precision structure

The bias is calculated using Equation (3.4).

$$bias = 2^{8-1} - 1 = 127 \tag{3.6}$$

Therefore the exponent is ranging between

$$-126 \leq e \leq 128 \tag{3.7}$$

We can see from the Equation (3.7) the possibility to represent a rank of values coming from very small numbers to very big ones thanks to the exponential representation. Thus, the range is calculated using Equation (3.5).

$$1,175494351 \cdot 10^{-38} \leq y \leq 3,402823466 \cdot 10^{38} \tag{3.8}$$

The disavantage of this representation is the need of the duplication of all the arithmetic to perform basic operations in signal processing: addition and multiplication. For example, every time we perform a multiplication, we need to have an arithmetic circuit performing the multiplication between both mantissas and the addition of both exponents. This situation discourages to use this number representation.

3.2.2.3 Conclusion

Fixed-point number are chosen to system to represent rational numbers, in order to make our design less complicated in terms of architecture and less size in logic gates.

Another crucial advantage of using fixed-point representation is the use of less power to perform arithmetic calculations [14]. This is a very important characteristic when the power supply being controlled by our FPGA is integrated inside of an embedded system.

For the rest of the report, we will validate our generic methodology by using the two controllers that were suggested to be implemented for VRM applications. The first transfer function will be the case for the serial PID because of the easiness to implement a transfer function having only one pole. After that, we will move to the programming of the 3^{rd}-order controller calculated by means of the pole-zero matching technique, [3].

3.2.3 Coefficient wordlength

3.2.3.1 Mathematical calculation

The first step before starting the design of a linear controller consists of the bit-width definition for the controller coefficients. Considering that this controller is an LTI system defined by :

$$u[n] = \sum_{i=0}^{k} b_i e[n-i] - \sum_{j=1}^{l} au[n-j] \tag{3.9}$$

The number of bits for the coefficients is imposed by :

$$N_{coefficients} = N_{integer} + N_{fractional} \tag{3.10}$$

where $N_{integer}$ is defined by:

$$N_{integer} = \max_i \lceil \log_2 |c_i| \rceil + 1 \tag{3.11}$$

where $c_i \in \{a_i, b_i\}$ are the coefficients referring Equation (3.9).

The number of decimal digits needed for the rational part of the coefficient is closely related to the shift of poles and zeros of the controller. The number of bits is also related on the type of structure to be programmed such as *direct form , canonical, coupled, parallel*. To develop a connection between the number of decimals needed for every coefficient and the shift of roots in z-plane, it is assumed that the digital transfer function to be programmed is in direct form. This topology is the most critical architecture suffering from the effects of pole deviation due to the fact that the transfer function of the controller relies on the relationship between two polynomials of a high degree. Last conclusion is explained in the following definition.

Definition 3.2.3.1. . *Every polynomial can be written in two different forms: serial and coupled. Direct form consists of a sum of terms, whereas the coupled form consists of a product of terms [15]*

- *Direct form*

$$F_c(x) = x^n + c_{n-1}x^{n-1} + ... + c_0 \tag{3.12}$$

- *Coupled form*

$$F_c(x) = (x - p_n)(x - p_{n-1})...(x - p_1) \tag{3.13}$$

being the set of real numbers, $C = \{c_{n-1}, c_{n-2}, ..., c_0\}$ the set of coefficients for the direct form polynomial, whereas the set $P = \{p_n, p_{n-1}, ..., p_1\}$ is the set of roots of the polynomial in the coupled form.

We define the quantization effects in Equations (3.12) and (3.13), as it is expressed in Equations (3.14) and (3.15). Two intial assumptions should be considered for the coefficient wordlength problem defined in the following definition.

$$\widehat{F}_c(x) = x^n + \widehat{c}_{n-1}x^{n-1} + ... + \widehat{c}_0 \tag{3.14}$$

$$\widehat{F}_c(x) = (x - \widehat{p}_n)(x - \widehat{p}_{n-2})...(x - \widehat{p}_1) \tag{3.15}$$

Definition 3.2.3.2. *The set of quantized* ***coefficients*** *and* ***roots*** *is related to the ideal counterpart by using the next formula*

$$\widehat{c} = c + \Delta c \tag{3.16}$$

$$\widehat{p} = p + \Delta p \tag{3.17}$$

Therefore, the question arising, when looking at the polynomial of Equation (3.14), is how much the set of quantized roots $\widehat{p}_n, \widehat{p}_{n-2}, ..., \widehat{p}_1$ are shifted regarding to the roots $p_n, p_{n-1}, ..., p_1$ after the quantization.

The answer to the previous problem can be solved considering the polynomial as a function of two unknowns , x and a generic coefficient c_j.

$$F_c(c_j, x) = x^n + c_{n-1}x^{n-1} + ... + c_j x^j + ... + c_1 x + c_0 \tag{3.18}$$

The same definition can be applied to the set of quantized coefficients as

$$F_c(\widehat{c}_j, x) = x^n + \widehat{c}_{n-1}x^{n-1} + ... + \widehat{c}_j x^j + ... + \widehat{c}_1 x + \widehat{c}_0 \tag{3.19}$$

Using Equation (3.18) and substituing the unknown x by the root p_i, the value of the polynomial should be equal to zero for the using Equation (3.13).

$$F_c\left(c_j, x = p_i\right) = 0 \tag{3.20}$$

The value of the quantized polynomial when the coefficient is equal to $\widehat{c}_j$ and the value of the pole is equal to $\widehat{p}_i$ can be approximated by means of Taylor series:

$$\begin{aligned} F\left(\widehat{c}_j, x = \widehat{p}_i\right) &= P\left(c_j + \Delta c_j, x = p_i + \Delta p_i\right) \\ &= P\left(c_j, p_i\right) + \left.\frac{\partial P}{\partial c_j}\right|_{x=p_i} \partial c_j + \left.\frac{\partial P}{\partial p_i}\right|_{x=p_i} \partial p_i + \\ &+ \underbrace{\quad ... \quad}_{\text{high-order differential terms}} \approx 0 \end{aligned} \tag{3.21}$$

Substituing Equation (3.20) into (3.21) and rearranging terms we can see how much the root p_i is shifted when the coefficient is suffering from having a quantization ∂c_j. As a result, the relationship is

$$\partial p_i = -\frac{\left.\frac{\partial F}{\partial c_j}\right|_{x=p_i} \partial c_k}{\left.\frac{\partial F}{\partial x}\right|_{x=p_i}} \tag{3.22}$$

We must, therefore, calculate the partial derivatives appearing in Equation (3.22) to have a closed expression to calculate the pole shift

$$\begin{aligned} \frac{\partial F}{\partial c_j} &= x^j \\ \left.\frac{\partial F}{\partial c_j}\right|_{x=p_i} &= p_i^j \end{aligned} \tag{3.23}$$

The second partial derivative is calculated using Equation (3.13) and the derivative of two multiplicative functions, $(ab)' = a'b + ab'$.

$$\begin{aligned} \frac{\partial F}{\partial x} &= \underbrace{(x-p_{n-1})(x-p_{n-2})...(x-p_j)...(x-p_1)}_{\text{derivative of term } (x-p_n)} + \\ &\quad + \underbrace{(x-p_n)(x-p_{n-2})...(x-p_j)...(x-p_1)}_{\text{derivative of term } (x-p_{n-1})} + ... + \\ &\quad + \underbrace{(x-p_n)(x-p_{n-1})...(x-p_1)}_{\text{derivative of term } (x-p_j)} + ... + \\ &\quad + \underbrace{(x-p_n)(x-p_{n-1})...(x-p_j)...(x-p_2)}_{\text{derivative of term } (x-p_1)} = \end{aligned} \tag{3.24}$$

$$= \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} (x - p_{ij}) \tag{3.25}$$

$$\left.\frac{\partial F}{\partial x}\right|_{x=p_i} = \prod_{j=0, j\neq i}^{n} (p_i - p_j) \tag{3.26}$$

Using Equations (3.22), (3.23) and (3.26) the shift of the root p_i due to the coefficient c_j.

$$\partial p_i = -\frac{p_i^j \partial c_j}{\prod_{i=1, i\neq j}^{n} (p_i - p_j)} \tag{3.27}$$

To know the influence of the set of n coefficients in the set of n poles, the best way is to use matrix notation. Considering the vector of coefficients, C and the vector of roots P as

$$C = (c_0 c_1 ... c_{n-1})^T \tag{3.28}$$

$$P = (p_0 p_1 ... p_{n-1})^T \tag{3.29}$$

The Jacobian matrix of sensitivities is now expressed in the form of

$$\frac{\partial P}{\partial C} = -A \cdot B = \begin{pmatrix} \frac{\partial p_1}{\partial c_0} & \frac{\partial p_1}{\partial c_1} & \cdots & \frac{\partial p_1}{\partial c_{n-1}} \\ \frac{\partial p_2}{\partial c_0} & \frac{\partial p_2}{\partial c_1} & \cdots & \frac{\partial p_n}{\partial c_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial p_n}{\partial c_0} & \frac{\partial p_n}{\partial c_1} & \cdots & \frac{\partial p_n}{\partial c_{n-1}} \end{pmatrix} \tag{3.30}$$

being the matrix A and B. The Jacobian matrix is the matrix of first-order partial derivatives of n-dimensional applications, $F : \Re^n \rightarrow \Re^n$. In our case, the application corresponds to the Equation (3.19) being n for this case the number of roots and coefficients of the polynomial.

$$A = \begin{pmatrix} \prod_{k=1k\neq 1}^{n} (p_1 - p_k)^{-1} & 0 & \cdots & 0 \\ 0 & \prod_{k=1k\neq 2}^{n} (p_2 - p_k)^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \prod_{k=1k\neq n}^{n} (p_n - p_k)^{-1} \end{pmatrix} \tag{3.31}$$

$$B = V^T = V\left(p_0 p_1 ... p_{n-1}\right) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ p_0 & p_1 & \cdots & p_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_0^{n-1} & p_1^{n-1} & \cdots & p_{n-1}^{n-1} \end{bmatrix}^T \tag{3.32}$$

The matrix appearing in Equation (3.32) ,V, is known as the Vandermonde matrix [16]. The Vandermonde matrix has its application in the field of polynomial interpolation.

Finally, in order to obtain the number of rational bits, it is necessary to calculate the most sensitive pole of the vector as it is shown in next demonstration. Firstly, it is worth to define the norm of a vector and a matrix [17].

Definition 3.2.3.3. *The ∞-vector norm of a vector $D = (d_0, d_1, ..., d_n) = (d_i)$ is*

$$\| D \|_\infty = \max_{1 \leq i \leq n} |d_i| \tag{3.33}$$

The ∞-matrix norm of a matrix $C = (c_{ij})$ is

$$\| C \|_\infty = \max_{1 \leq i \leq n} \sum_{j=0}^{n-1} (c_{ij}) \quad \text{"max row sum"} \tag{3.34}$$

Matrix norms are useful to concentrate the information of all numbers appearing in a matrix into one number.

For the case of the Jacobian matrix defined in Equation (3.30) and knowing that each element of the jacobian matrix have the form of Equation (3.27), we can use the definition of the ∞-matrix norm to assure a lower bound for the number of rational bits.

$$\left\|\frac{\partial P}{\partial C}\right\|_{\infty} = \max_{1\leq i\leq n} \frac{\sum_{k=0}^{n-1}\left|p_i^k\right|}{\left|\prod_{k=0,k\neq i}^{n}(p_i - p_k)\right|} \tag{3.35}$$

Considering that ∞-norm of the vector P is $\|P\|_{\infty} = \varepsilon$ and the ∞-norm of the vector $\frac{1}{\partial C}$ is equal to $2^{N_{rational}}$.The number of bits rational bits needed is finally

$$N_{fractional} = \left\lfloor \log_2\left(\left\|\frac{\partial P}{\partial C}\right\|_{\infty} \cdot \frac{1}{|\varepsilon|}\right)\right\rfloor + 1 \tag{3.36}$$

It is important to highlight few characteristics related to Equation (3.36). The denominator of this equation tends to zero when the roots of the polynomial are close. Therefore, for cases when poles or zeros of a given transfer function are concentrated in a cloud, more bits are needed to decrease the shift of thee roots. Last conclusion is validated for our study case in the next numerical example.

Example 3.2.3.1. *Consider the 3rd order transfer function given in [3]. The purpose of this example is to calculate the number of bits required to achieve a maximum shift of a 10 % of the magnitude of the roots of the denominator.*

$$G_c(z) = \frac{5z^2 - 9,7895529827277z + 4,7906487334929}{z^3 + 1,01066421884141z^2 - 0,244667890579294z + 0,255332109420706} \tag{3.37}$$

The vector of the coefficients and roots are :

$$\begin{aligned} C &= (c_2, c_1, c_0) \\ &= (1,01066421884141,\ -0,2446678905792940,255332109420706) \end{aligned} \tag{3.38}$$

$$P = (p_1, p_2, p_3) = (1, 0, 51066421884152, -0, 5)$$

The number of integer coefficients for the denominator was given in Equation (3.11).

$$N_{integer} = \lceil \log_2 \|C\|_\infty \rceil + 1 = 2 \tag{3.39}$$

$$\|C\|_\infty = 1,01066421884141 \tag{3.40}$$

To calculate the number of rational bits, we need to calculate the Jacobian matrix of Equation (3.30). The Jacobian matrix we are willing to form for this example is having the following form

$$\frac{\partial P}{\partial C} = \begin{pmatrix} \frac{\partial p_1}{\partial c_0} & \frac{\partial p_1}{\partial c_1} & \frac{\partial p_1}{\partial c_2} \\ \frac{\partial p_2}{\partial c_0} & \frac{\partial p_2}{\partial c_1} & \frac{\partial p_2}{\partial c_2} \\ \frac{\partial p_3}{\partial c_0} & \frac{\partial p_3}{\partial c_1} & \frac{\partial p_3}{\partial c_2} \end{pmatrix} \tag{3.41}$$

The matrix A *and* B *appearing in Equation (3.30) for this particular example are*

$$\begin{aligned} A &= \begin{pmatrix} \prod_{i=1, i\neq 1}^{3} (p_1 - p_i) & 0 & 0 \\ 0 & \prod_{i=1, i\neq 2}^{3} (p_2 - p_i) & 0 \\ 0 & 0 & \prod_{i=1, i\neq 3}^{3} (p_0 - p_i) \end{pmatrix} \\ &= \begin{pmatrix} 1,362391 & 0 & 0 \\ 0 & 2,022023 & 0 \\ 0 & 0 & 0,659632 \end{pmatrix} \end{aligned} \tag{3.42}$$

$$B = V^T (p_1, p_2, p_3) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0,51066 & 0,26077 \\ 1 & -0,5 & 0,25 \end{pmatrix} \tag{3.43}$$

Substituing the results of Equations (3.42) and (3.43) into equation (3.30), the result of the Jacobian matrix is

$$\frac{\partial P}{\partial C} = \begin{pmatrix} 1,362391 & 1,362391 & 1,362391 \\ 2,022023 & 1,0325749 & 0,527299 \\ 0,6596322 & -0,329816 & 0,164908 \end{pmatrix} \tag{3.44}$$

The ∞*-matrix norm of the Jacobian matrix is*

$$\left\| \frac{\partial P}{\partial C} \right\|_\infty = 4,087173015 \tag{3.45}$$

The number of rational bits is calculated using Equation (3.36) and the maximum shift of poles is $\varepsilon = 0,1$

$$N_{rational} \geq \left\lfloor \log_2 \left(4,087173015 \cdot \frac{1}{0,1} \right) \right\rfloor + 1 = 6 \tag{3.46}$$

Therefore, the codification of coefficients in fixed-point is following the patter 8Q6 as it has seen in Subsection 3.2.2.1. That is, the coefficients for the denominator of the transfer function need a total of eight bits whose six last bits are rational numbers.

The set of quantized coefficients and the associated quantized roots for the polynomial of the denominator in Equation (3.37) are :

$$\widehat{C} = (-1,015625, -0,25, 0,25) \tag{3.47}$$

$$\widehat{P} = (1,02056, 0,49247, -0,4974) \tag{3.48}$$

We see that the controller is unstable because one pole is moved outside of the unit circle.

Following the same procedure as in Example 3.2.3.1 for the numerator of Equation (3.37), we calculate the number of necessary bits, to assure a maximum shift among roots and quantized roots. Imposing an $\varepsilon = 0,1$, the required number of bits is 10, because the zeros of the transfer function are very close to the border of the unity circle. Therefore, the number of bits in the controller for our application will be imposed by the polynomial of numerator. The new roots for the denominator using the new constraint imposed by the numerator are

$$\widehat{P} \quad = \quad (1,001326780, 50925428, -0,4998388) \tag{3.49}$$

3.2.3.2 Real coefficient implementation

We can see that even if the number of bits was increased, the controller is still unstable. The arising question is how the controller can be made stable. The answer comes at looking Equation (3.36). When we calculate the number of bits, we only impose that the root is moved inside of a small disc of radius ∂p, but the main drawback of this method is that we are not able to parametrize the direction of this shift. Examples of quantized roots appearing in Equations (3.47) and (3.49) were coming using the rounding approach for the coefficients of the quantized polynomial. Nevertheless, in case, the coefficients are truncated instead of rounded, the new roots are

$$\widehat{P} = (0,99866866895454491, 0,509254282, -0,4998388745) \tag{3.50}$$

Figure 3.10 shows how the pole located at $z = -1$ is located whether if we are using rounding or truncation methods. Truncation method limits the

number of digits right after the last required real number, by discarding the rest of the word. On the other hand, rounding method imposes a number of real numbers approximating to the nearest value.

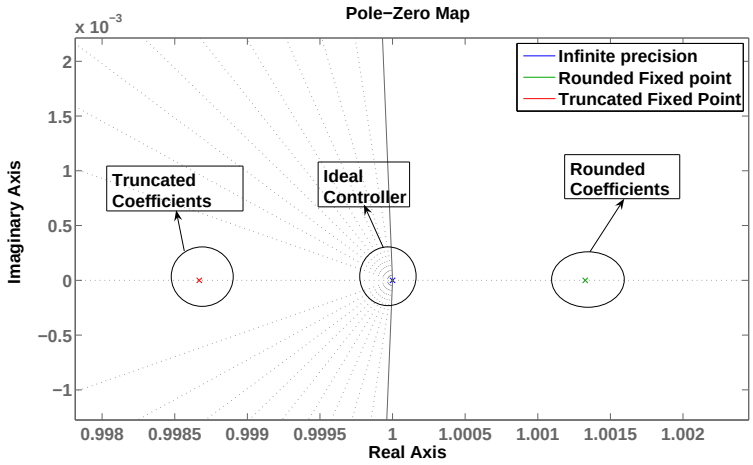


Figure 3.9: Pole-Zero map comparing the effect of rounding/truncating coefficients

The Bode chart can also show that the method of rounding/truncation can also differ, like in the previous case. The Figure 3.10 shows that due to the shift of the integrator, the Bode plot at frequencies close to zero differ because of the shift in the integrative pole. It is required, then, to increase the accuracy of the system, just increasing the number of rational bits. The new upper bound for our controller will be then $\varepsilon = 0,01$ applying truncation method for the quantized coefficients. The Bode and pole-zero map for the new accuracy are shown in Figures 3.12 and 3.11.

In Figure 3.11, we can see that for a maximum shift of $\varepsilon = 0,01$, the integrative pole for the case of truncated coefficients is exactly located at $z = 1$. On the other hand, for the case of rounded coefficients, the pole is located inside of the unit circle. Moreover, the problem comes when we observe the frequential response for both cases. In Figure 3.12, it can be observed the case of truncated coefficients. The plot matches with the response of the original controller. On the other hand, the shift of the integrative pole for the case of rounded coefficients is clearly appreciated in the Bode. The negative slope of $-20\frac{db}{dec}$ starts at the frequency of $20Hz$ and not in DC frequency.

3.2.3.3 Integrative part

As it has been shown in the previous point, the integrative part of our digital controller may be an important source of unstabilities. However, the integrative part is required in our study case in order to cancel the steady-state

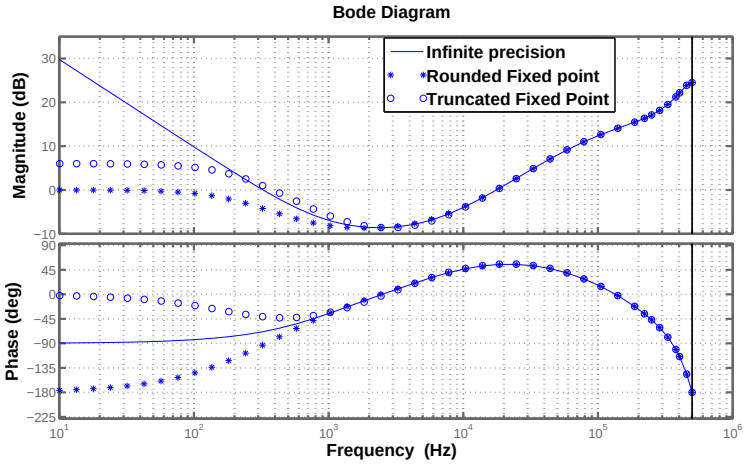


Figure 3.10: Bode map comparing the effect of rounding/truncating coefficients

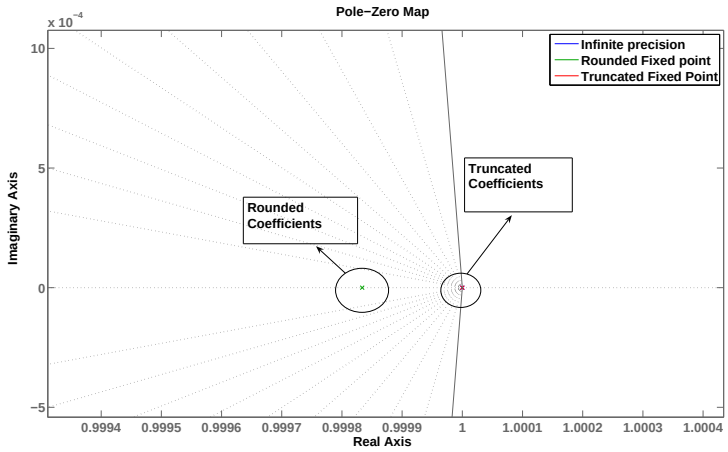


Figure 3.11: Bode diagram for $\varepsilon = 0.01$

error. Thus, the easiest way to solve problems related to controllers requiring an integrative action is the construction of the controller using parallel structures. As it can be observerd in Equation (3.51), the transfer function of a controller $G_c(z)$ can be written in two forms

$$G_c(z) = \underbrace{\frac{\sum_{i=0}^{m} b_i z^i}{1 + \sum_{j=1}^{n} a_j z^j}}_{\text{serial form}} = \underbrace{\sum_{j=0}^{n} \frac{r_j}{z - p_j}}_{\text{parallel form}} \quad \text{when} \quad m \leq n \tag{3.51}$$

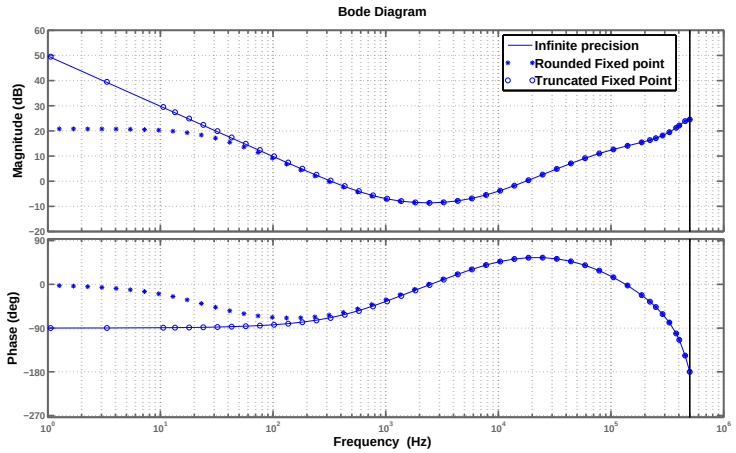


Figure 3.12: Pole-zero diagram for $\varepsilon = 0.01$

Equation (3.51) shows clearly that the transfer function of the controller written in parallel form correspond of a sum of first-order transfer functions. The coefficient p_j is clearly the root of every term. For parallel structures the quantization of the coefficient p_j corresponds to a shift of the root of the same magnitude (see Equation (3.52)). Therefore, when the controller requires an integrative action $p_j = 1$ and we don't need any shift in that root, the best option is to use parallel structures.

$$z - \widehat{p_j} = z - (p_j + \Delta p_j) \tag{3.52}$$

In the next Table 3.2 we summarize the number the bits required to obtain a given error. As we can see, the number of bits increases with the polynomial degree. For the case of PID, the polynomial of the denomintor is monic and it has only one pole. Therefore, the coefficient to be quantized will depend itself of the shift over the z-plane.

Table 3.2: Number of bits depending on the error

Error	PID	Pole-Zero Matching Controller
40%	3	8
10%	5	10
0.1%	8	13

3.2.4 Per-unit terminology

Common problems when arithmetics are performed in FPGA are in the lack of architecture and the non-existence of data types like in C compilers for DSP or computers. That's why it is important to calculate maximum values in output response in order to avoid common problems in quantized arithmetic: overflow and wrap. The effects are shown in Figure 3.13:

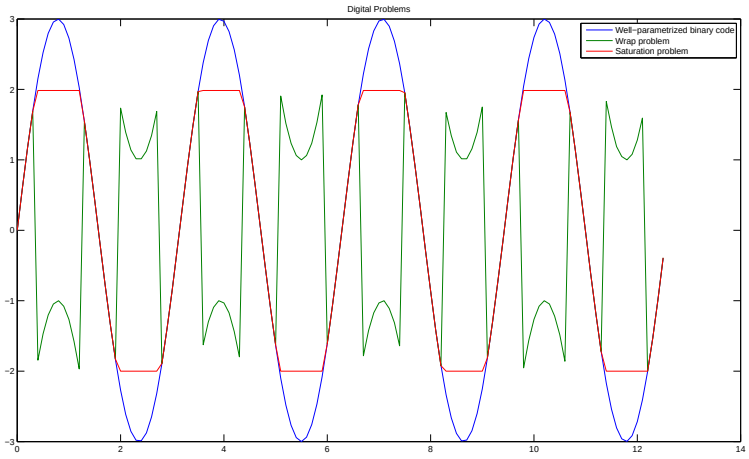


Figure 3.13: Problems in arithmetic digital circuits

Saturation problem happens when a result of an arithmetic operation exceeds the maximum allowed value for our binary code. On the other hand, wrap consists of using a sign bit of in the binary word when arithmetic operation result is by far too large. As it can be seen in the Figure, 3.13 wrap problem is seen to be a change in the sign of the quantized response.

To avoid both problems it is recommended to normalize filter output response by the following Equation:

$$u_{pu}(t) = \frac{u(t)}{\|u(t)\|_\infty} \tag{3.53}$$

$$e_{pu}(t) = \frac{e(t)}{\|e(t)\|_\infty} \tag{3.54}$$

Having LTI system defined by a difference equation

$$u[n] = \sum_{i=0}^{k} b_i e[n-i] - \sum_{j=1}^{1} a_j u[n-j] \tag{3.55}$$

the *per-unit* LTI system is

$$u_{pu}[n] = \sum_{i=0}^{k} b_i e_{pu}[n-i] - \sum_{j=1}^{l} a_j u_{pu}[n-j] \tag{3.56}$$

where u_{pu} and e_{pu} are defined by Equations (3.53) and (3.54)
In our case, the equation for a serial PID is as follows:

$$u_{pu}[n] = u_{pu}[n-1] + k_0 e_{pu}[n] + k_1 e_{pu}[n-1] + k_2 e_{pu}[n-2] \tag{3.57}$$

In this document, the maximum value is derived using the maximum value coming from the impulse response of the controller. The reason why the impulse response is taken as a pattern is related to the analysis of the transient behaviour of the step response in a control system. Considering a generic control system, as given in Figure 3.14, where the plant and the controller are LTI systems, the step response of the system can be described by a set of exponentials. If we consider that the output response of the plant is over-damped, the step response, $v_{out}(t)$ of the whole control system in closed-loop is

$$v_{out}(t) = V_{ref}\left(1 - \sum_{k=1}^{n} r_k e^{-\frac{t}{p_k}}\right) \quad \text{n being the number of poles and } p_k \in \Re \tag{3.58}$$

Considering that the set of exponential tends quickly to zero, the only exponential that shapes the output is the exponential that has the least constant time, that is, the exponential having the smallest pole magnitude (closest pole to the imaginary axis).

$$v_{out}(t) \approx V_{ref} \cdot \left(1 - e^{-\tau t}\right) \quad \text{where } \tau = \min_{k=1}^{n} p_k \tag{3.59}$$

Using the approximation appearing in Equation (3.59), the error signal (input signal to the controller) is

$$e[n] = V_{ref} - y[n] = V_{ref} \cdot e^{-\tau n} \tag{3.60}$$

Finally, the last assumption is to consider that τ is big enough to be tending to ∞. The limit of Equation (3.60) is tending to a discrete delta because at the first moment the error is maximum and tends quickly to zero. This is the reason why the maximum value of the impulse response of the controller was chosen as a parameter to normalize the signals of the loop.

$$e[n] = \lim_{\tau \to \infty} V_{ref} \cdot e^{-\tau n} u[n] \approx V_{ref} \cdot \delta[n] \tag{3.61}$$

Using same coefficients for the PID calculated in (2.77), *MATLAB* is able to calculate maximum value by means of the functions *dimpulse* and *max*.

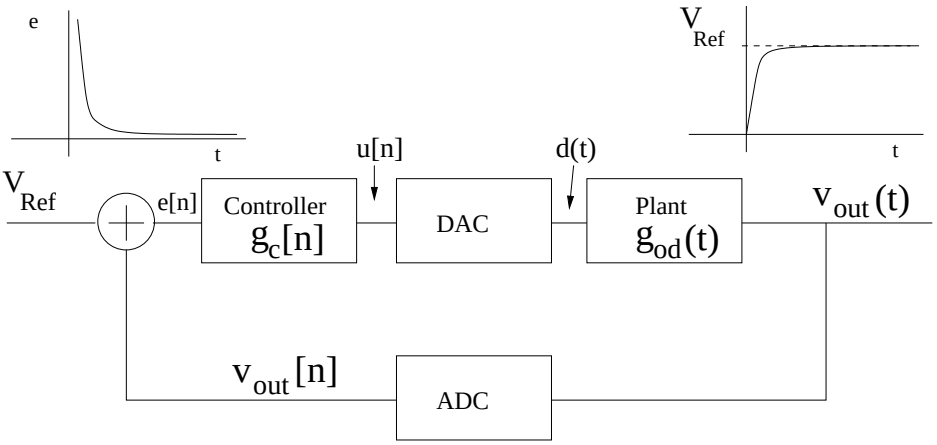


Figure 3.14: Step Response of the control system

$$\|g_c[n]\|_\infty = 1,5127 \tag{3.62}$$

For the case of the 3^{rd} order controller, the ∞-norm is

$$\|g_c[n]\|_\infty = 9,736232 \tag{3.63}$$

In [18- 19], a restrictive upper bound to calculate the maximum value of the controller is

$$\|u[n]\|_\infty \leq \|g_c[n]\| \cdot \|e[n]\|_\infty \tag{3.64}$$

where $\|\|_\infty$ and $\|\|_1$ denote respectively the l_∞ and l_1 norms defined previously. $\|u\|_\infty = \max_k^n |e[n]|$, and $\|g_c[n]\|_1 = \sum_{k=0}^{\infty} |g_c[n]|$.

The deduction of the upper bound for Equation (3.64) can be guessed interpreted using convolution of two signals because we are willing to know the time-domain output of the controller when the applied input is $e[n]$.

$$u[n] = \sum_{k=0}^{\infty} e[k] * g_c[k-n] \tag{3.65}$$

Applying ∞-norm to inputs and output of Equation (3.66) and applying finally Cauchy-Schwarz unequality, we have

$$\|u[n]\|_\infty = \sum_{k=0}^{\infty} g_c[k-n] * e[k] \leq \|g_c[n]\|_1 \cdot \|e[n]\|_\infty \tag{3.66}$$

Example 3.2.4.1. *Consider that we are willing to implement an output voltage regulated system using the 3^{rd}-order filter given in section 2.4.2.3. The whole control system is shown in Figure 3.14. The maximum value at the output of the controller preventing overflow was given in Equation (3.66). The 1-norm, calculated in* Matlab*, was*

$$\|g_c[n]\|_1 = 38,2511 \tag{3.67}$$

The maximum voltage coded by the ADC is $2,5V$, in case $V_{ref} = 1V$, then the maximum error is $\|e[n]\|_\infty = |1 - 2,5| = 1,5$

$$\|e[n]\|_\infty = 1,5V \tag{3.68}$$

The maximum value at the output of the controller is

$$\|u[n]\|_\infty < \|g_c[n]\|_1 \cdot \|e[n]\|_\infty = 38,2511 \cdot 2,5 = 57,37665 \tag{3.69}$$

The number of bits required to implement the result of Equation (3.69) is

$$N_{integer} = \lceil \log_2 (57,37665) \rceil + 1 = 7 \quad bits \tag{3.70}$$

Finally, to prevent overflow, we need to codify the signal using 7 bits. Obviously, this rule should not make us feeling free from implementing an overflow/underflow block after the controller, because the 3^{rd}-order filter has an integrator.

3.3 Effects of rounding multiplier output

After the analysis of the error due to the shift of the coefficients, the second source of error in the implementation of a digital control system comes when it is compulsory to shorten the number of bits at the multiplier output. A digital multiplier is an arithmetic circuit whose inputs are W-bit width and the output is 2W-width. For some reasons explained later in this chapter, the output must be W-bit width. There are many ways to modelize and parametrize the effects of having a limited number of bits at the output of a multiplier. The direct form to express the problem of rounding consists of a shift in the poles of the system to be implemented. Let's focus in the multiplier of Figure 3.15.

The output of the rounded multiplier can be seen as the addition of a source of noise Δy to the real output.

$$y + \Delta y = b \cdot x \tag{3.71}$$

If the rounded output is considered to be the same output as the output of a non-rounded multiplier consisting of a factor b', there is a shift of the

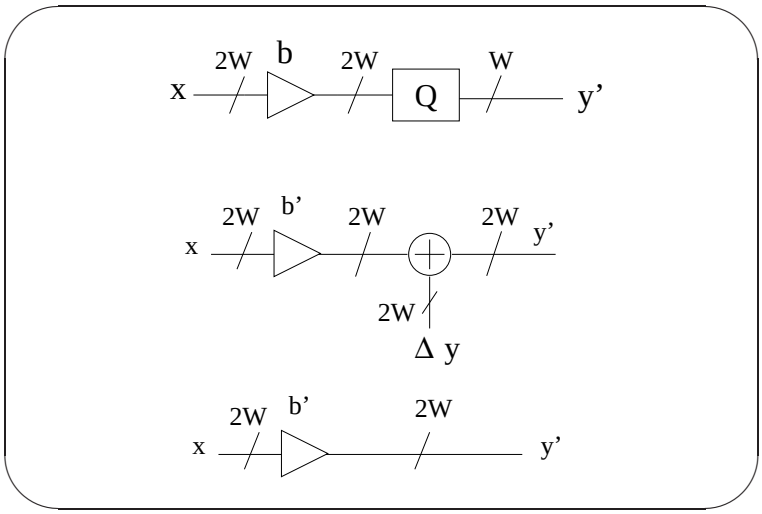


Figure 3.15: Model of rounding the output of a multiplier

coefficient and, therefore, of the pole of the system directly proportional to the rounding performed in the real multiplier (Eq. (3.73)).

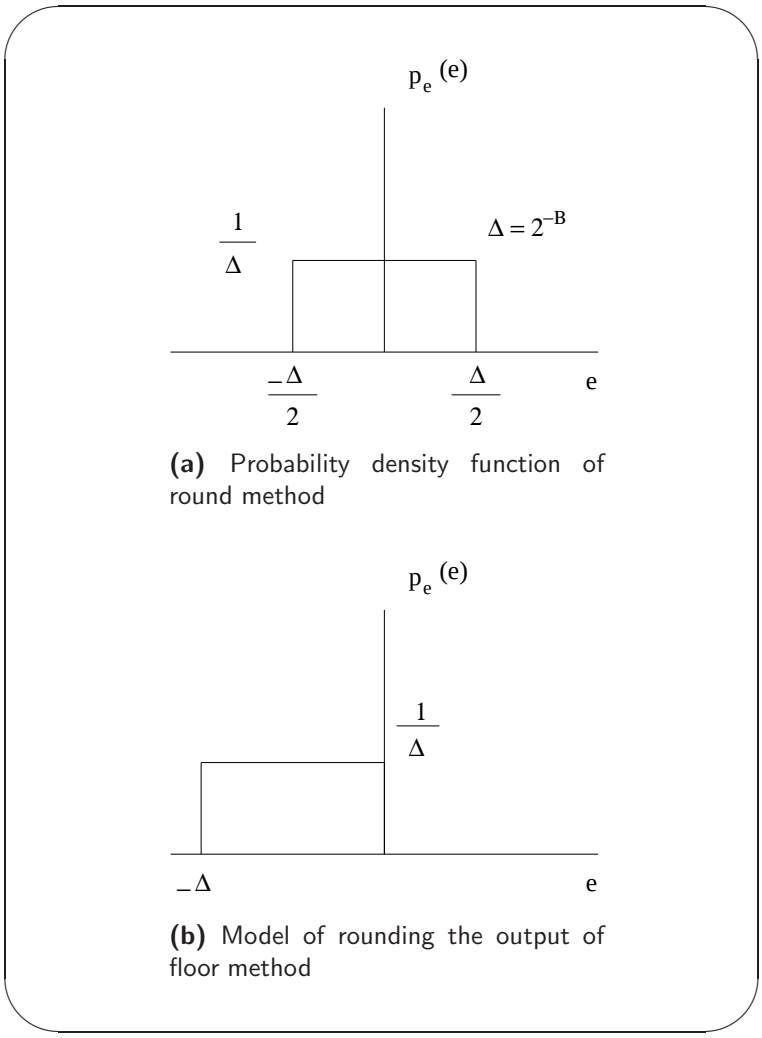
$$y + \Delta y = b'x = (b + \Delta b) \cdot x \tag{3.72}$$

$$\frac{\Delta y}{y} = \frac{\Delta b}{b} \tag{3.73}$$

The aforementioned answer is not as easy as the reader might think. First of all, the diference between the problem of rounding and the rounding of coefficients is that the shift is not static. That is, for the case of rounding coefficients the problem is defined and solved in design times. On the other hand, the rounding effect due to multiplier and the associated shift depends sample by sample whilst the system is operative. It is depending strongly also on the type of rounding operation performed on the system. This is why the error in Figure 3.15, Δy, can be described in a better way using the *pdf* (probability density function) of the rounding method that the quantizer block, Q, is applied at the output of the multiplier.

In this report, two types of rounding methods are going to be analyzed. First method is *floor*, which is equivalent to two's complement truncation, rounds to the closest representable number in the direction of negative infinity. On the other hand, the second method, called *round*, rounds to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity and it rounds negative numbers to the closest representable integer in the direction of negative infinite.

The statistical properties of both rounding methods can be derived by means of Figures 3.16a and 3.16b.



(a) Probability density function of round method

(b) Model of rounding the output of floor method

Figure 3.16: Probability density function of rounding/floor truncation methods

The mean and and variance for round method is [18]:

$$E(p_e) = 0 \tag{3.74}$$

$$\sigma_e^2 = \frac{2^{-2W}}{12} \tag{3.75}$$

And for floor method is [18]:

$$E(p_e) = -\frac{2^{-W}}{2} \tag{3.76}$$

$$\sigma_e^2 = \frac{2^{-2W}}{12} \tag{3.77}$$

3.4 Structures for controllers

The way that our transfer function is programmed plays an important role to succeed in our purpose of implementing a controller. There are several structures that are going to be analyzed using the fixed-point toolbox of *Matlab*. Even if the result of the all the structures must be the same, every network has different characteristics ranging from the number of memory elements, multipliers, computational complexity, coefficient wordlength requirements and roundoff noise. Structures analyzed are : direct form I, canonical, cascade and parallel.

3.4.1 Direct form I

The direct form I consists of the transcription of the difference equation of the controller as the Equation (3.55). This structure needs $N+M$ registers, multipliers and adders, where N and M are the degree of the numerator and denominator polynomials in Equation (3.55). It is obvious that the structure is not efficient in terms of used hardware resources. The high order of polynomials makes the structure to be very dependent on the rounding coefficient errors and the rounding multiplier errors.

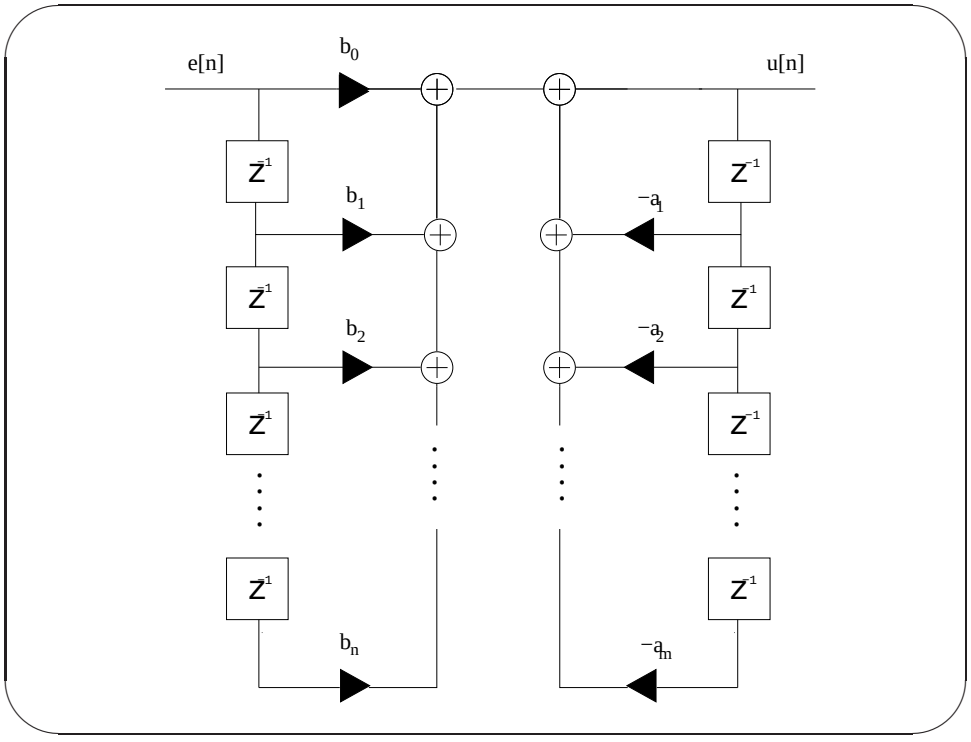


Figure 3.17: Direct form I structure

3.4.2 Canonical

The canonical structure is derived from the direct form structure. The derivation can be explained by means of the z-transform of the LTI system. The number of memories is reduced to the maximum number of poles/zeros of the controller. The effects of rounding coefficients are the same as before due to the fact that the degree of polynomials included in numerator and denominator, as it can be seen comparing Equations (3.55) and (3.78), is not decreased. On the other hand, the rounding noise even if it is not bearable for many aplications has a different nature than in previous case as it will be appreciated in the following section. Equation (3.78) represents the mathematical expresion why the Figure 3.17 turns into 3.18.

$$\begin{aligned} G_c(z) &= \frac{\sum_{k=1}^{N} b_k z^{-k}}{1 + \sum_{l=1}^{M} a_l z^{-} l} \\ &= \sum_{k=1}^{N} b_k z^{-k} \cdot \frac{1}{1 + \sum_{l=1}^{M} a_l z^{-} l} \end{aligned} \tag{3.78}$$

3.4.3 Cascade

The cascade structure consists of the rationalization of the polynomials in the numerator and denominator of the transfert function. Therefore, the output of the system is the multiplication of the response of a set of transfer function of first and second order systems, being called *biquad filter* when all the sections are made of second-order polynomials. Every individual structure can be configured. The error due to the quantization of the coefficients decreases as the degree of every section decreases to the number of two. It is also considered the fact that every section can be programmed either using direct form or canonical form.

Equation (3.79) represents mathematically that the transfer function of the controller can be represented by a product of second-order transfer functions.

$$G_c(z) = \prod_{i=0}^{n} \frac{b_{i0} + b_{i1} z^{-1} + b_{i2} z^{-2}}{a_{i0} + a_{i1} z^{-1} + a_{i2} z^{-2}} \tag{3.79}$$

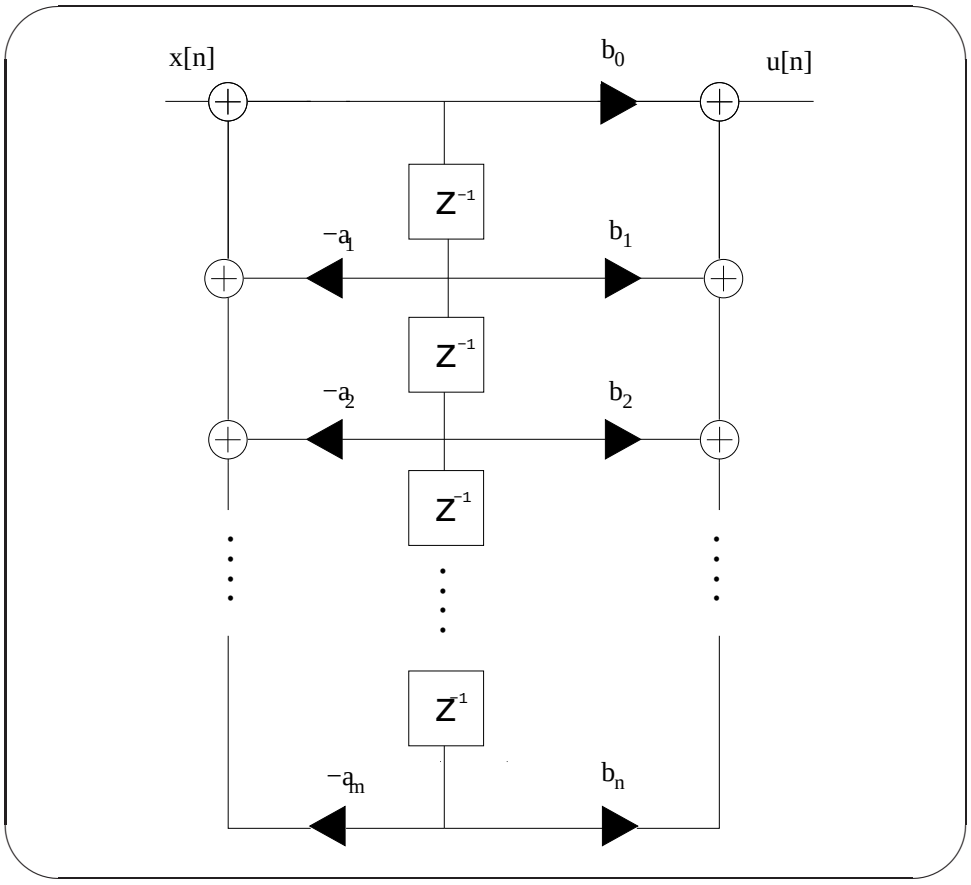


Figure 3.18: Canonical structure

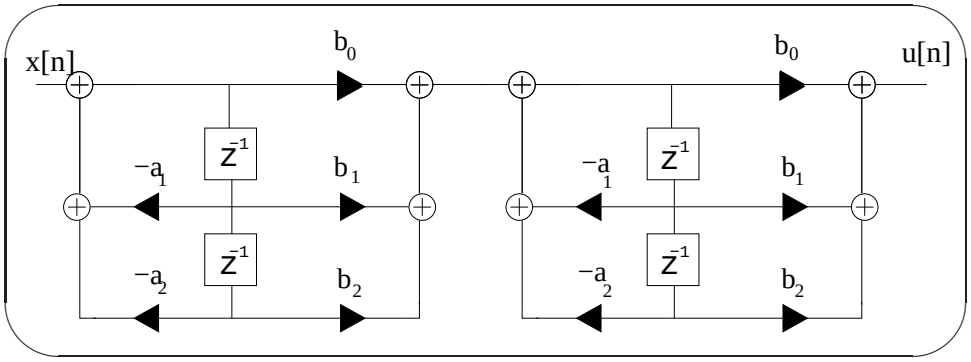


Figure 3.19: Cascade structure

3.4.4 Parallel

Parallel structure is derived from direct form by calculating the residues of the transfer function. The transfer function of the controller is as follows:

$$G_c(z) = \sum_{i=0} \frac{r_i}{z - p_i} \tag{3.80}$$

The error due to the quantization of the coefficient is the lowest of all the structures because of the low order of the polynomials involved in every term of Equation (3.80).

Finally, Table 3.3 summarizes all the concepts related to the characteristics of every structure.

Structure	Quantization Coefficient Sensitivity	Rounding Sensitivity
Direct I	High	High
Canonical	High	High
Cascade	Medium	Medium
Parallel	Low	Low

Table 3.3: Summary of structure characteristics

3.4.5 Simulation of the structures and output noise characterization

Using the *fixed-point* toolbox of *Matlab* we can perform arithmetic operations in fixed point arithmetic. The toolbox allows us to define the length of the word and also the length of the decimal part. Finally, it allows us to define the length and rounding method of the arithmetic operation. Therefore, this

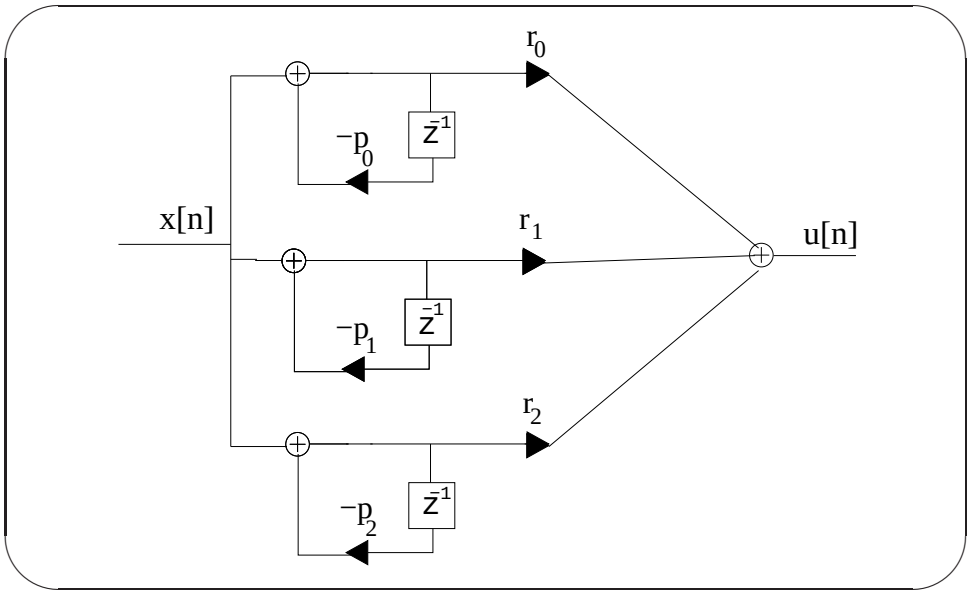


Figure 3.20: Parallel structure

toolbox allows us to compare, as a first step of design, the output error produced by the structure due to the limited fixed-point data length to represent the signals in our control system. The output voltage error due to the error produced by the length in the controller can be described using Figure 3.21.

The output of a noisy controller can be described by the sum of the response for an ideal controller plus a noise.

$$\widehat{u}[n] = u[n] + e_u[n] \tag{3.81}$$

The noise generated by the controller might modifiy the behaviour of the plant if the magnitude of the noise is big enough to be detected by the DAC converter. Therefore, the output of the DAC converter and the output response of the plant can be described by

$$e_{vout}(t) = \begin{cases} g_{od}(t) * e_d(t) & \text{if } e_u[n] > V_{DAC} \cdot 2^{-n} \\ 0 & \text{if } e_u[n] < V_{DAC} \cdot 2^{-n} \end{cases} \tag{3.82}$$

being n_{ADC} the number of bits of the DAC converter and V_{DAC} the reference voltage of the DAC.
The noise generated by the controller could be also an important contribution to the origin of limit cycles if $e_{vout}(t)$ is large enough to be detected by the ADC range.

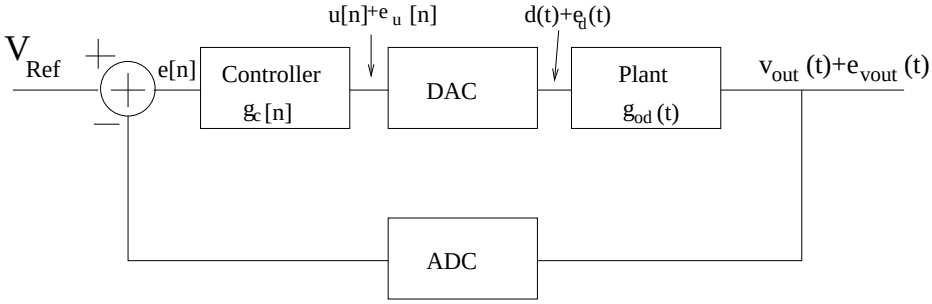


Figure 3.21: Control system considering controller as a noisy element

3.4.5.1 Results for PID controller

Due to the low order of the transfer function of a PID, cascade and parallel structures are not possible to be implemented. For this reason, the structure to be analyzed is going to be only Direct Form I. Figure 3.22 shows the impulse response of floating point and fixed-point accuracy and Figure 3.23 shows the error between both precisions.

As we can see in Figure 3.22, PID impulse response has almost the same impulse response in both cases. The output error, as can be seen in Figure 3.23, among them is constant over the time, therefore the error is not divergent. This means that we are still having an stable impulse response. The

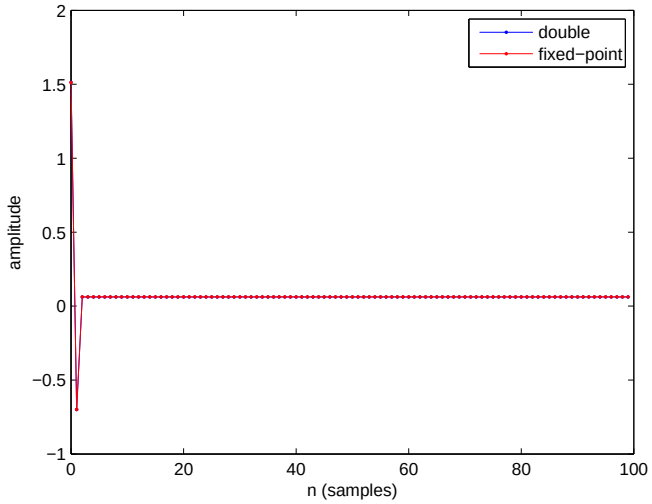


Figure 3.22: Impulse response of the simulated PID structure. Red plot is the impulse response of the fixed-point structure. Blue graph is the output of the PID structure using *MATLAB* double precission. floating point versus fixed point for the PID controller

reason why the error is not divergent can be explained seeing that the output error is less than $\frac{1}{2}LSB$ and the error is not fedback in the network.

3.4.5.2 Results for 3rd-order filter

The impulse response of the third-order filter is compared with two different rounding methods: *truncation* and *rounding*. First of all, simulations of every structure applying truncation method is presented and, finally, in the second subsection results for rounding method are presented.

Truncation This is the type of rounding method that is going to be implemented in our controller because of the lack of requiring extra hardware to be performed. The goals to be achieved in this analysis is to try to have the minimum error as it is posible. It must be thought that the main purpose of a control loop system is to have an stable output tending to the value of reference. When the output of the system equals the value of reference, the error should ideally be zero. Therefore, the most important task is to achieve a controller that having a zero input the output will not be divergent. This is another good reason why the impulse response was chosen to try the stability of the controller.

Direct Form The simulated structure is shown in Figure 3.25. The data width consists of w bits and the only truncation is applied in the block called Q. It is necessary to truncate the data length in order to adapt the data

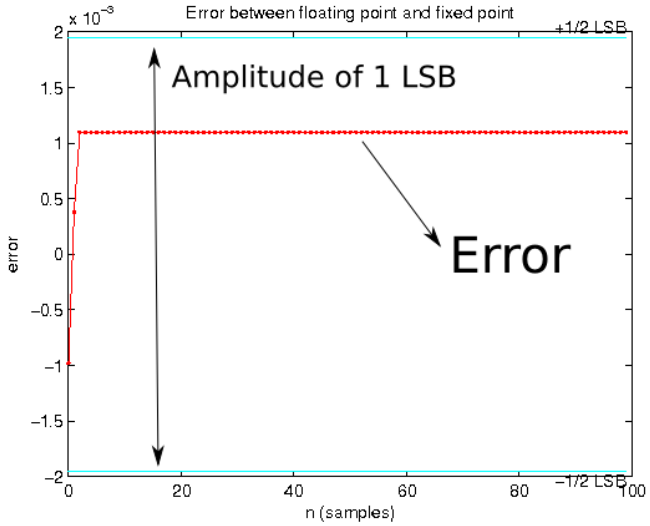


Figure 3.23: Error between floating impulse response and fixed point response for the PID controller

length of the register, z^{-1}. The truncation block can be replaced by an error source, e_n (see Figure 3.25), characterized statistically in Equations (3.76) and (3.77). Simulation of the error between floating point and fixed-point structures shows that the impulse response is in Figure 3.23. The fixed-point response is not suitable because when the input error is zero the response tends to $-\infty$. The explanation of this phenomena is linked with the given explanations in Sections 3.2.3 and 3.3. The error introduced in the system is only produced at the output stage.The input branch is not suffering from any truncation since the input to the b_n multiplier is W and the output is $2W$. Therefore the situation can be modeled using the difference equation of the transfer function

$$\widehat{u}[n] = \sum_{i=0}^{n} b_i e[n-i] - \sum_{j=1}^{m} a_j \widehat{u}[n-j] \tag{3.83}$$

being $\widehat{u}[n-j] = u[n-j] + e_n[n]$. We can assume that the results of the output branch due to the truncation can be given assuming that there is an ideal controller, who is not suffering from truncation, having coefficients $\widehat{a}_j$.

$$\widehat{u}[n] = \sum_{i=0}^{n} b_k e[n-i] - \sum_{j=1}^{m} \widehat{a}_j y[n-j] \tag{3.84}$$

being $\widehat{a}_j = a_j + \delta a_j$. The new coefficients are considered to suffer from a shift δa which is equivalent to $e_n[n]$. The z-transform of equation 3.84 is

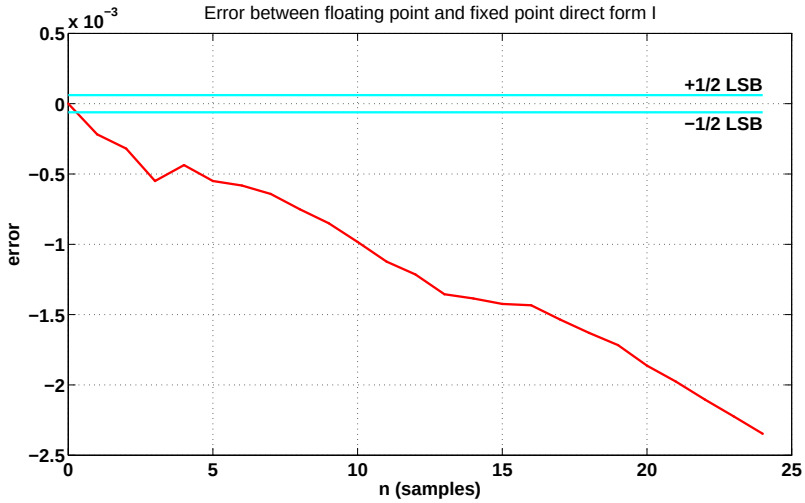


Figure 3.24: Simulated structure considering truncation as a source of error

$$G_c(z) = \frac{\sum_{i=0}^{n} b_k z^{-i}}{1 + \sum_{j=1}^{m} \widehat{a}_j z^{-j}} \tag{3.85}$$

The particularity of the new shift in the coefficients is that in this case, the shift is dynamic. It is depending on the value to be truncated sample by sample. Since the generated noise can be modelled as an ergodic process [20], the easiest way to model this shift is by means of statistics. That is we can calculate the mean and variance of this shift. For the case of truncation method, we have that

$$E\left(\delta a_j\right) = -\frac{2^{-w}}{2} = 2^{-w-1} \tag{3.86}$$
$$\sigma^2\left(\delta a_j\right) = -\frac{2^{-2w}}{12}$$

Finally, we can see that the truncation δp can be calculated using the results in Equation (3.27)

$$E\left(\partial p_i\right) = -\frac{p_i^j E\left(\partial a_j\right)}{\prod_{i=1, i\neq j}^{n}\left(p_i - p_j\right)} = \frac{p_i^j 2^{-w-1}}{\prod_{i=1, i\neq j}^{m}\left(p_i - p_j\right)} \tag{3.87}$$

The linear controller losses its stability because the integrator, located at $z = 1$ is pulled outside of the unit circle because of the effect of the truncation. The new pole is located at

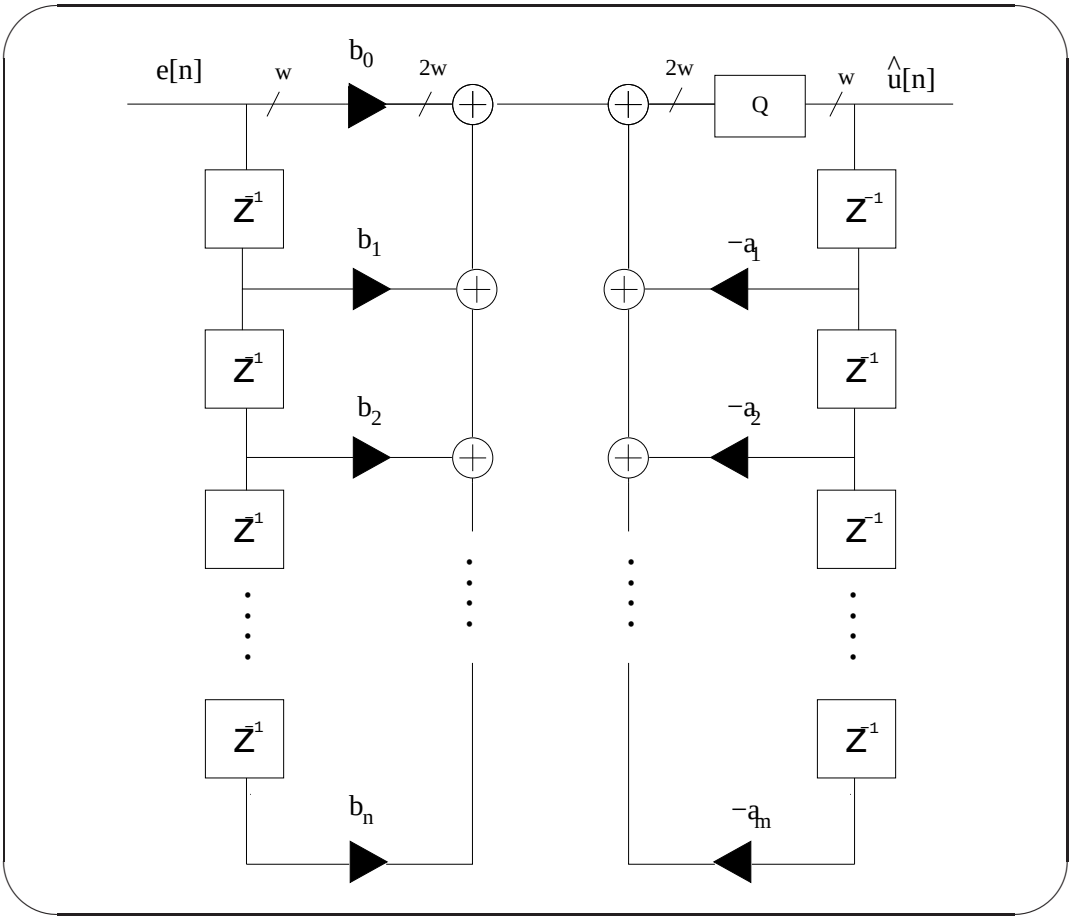


Figure 3.25: Direct form simulated structure

$$E\left(\widehat{p}_{integrator}\right) = p_{integrator} + \sum_{j=0}^{m} \frac{\partial p_{integrator}}{\partial c_j} = 1 + m \cdot \frac{2^{-w-1} \sum_{j=0}^{m} p^j}{\prod_{i=1, i\neq j}^{m} (p_i - p_j)} \quad (3.88)$$

$$p_{integrator} = 1$$

m being a positive number close to zero.

Parallel The programmed parallel structure is shown in Figure 3.27. The number of branches performing a 3^{rd} order filter is obviously three. The upper branch in Figure 3.27 is not having a multiplier because it is the branch performing the integrative action. It is important to notice that the integrative branch is not suffering from the truncation effect (we did not

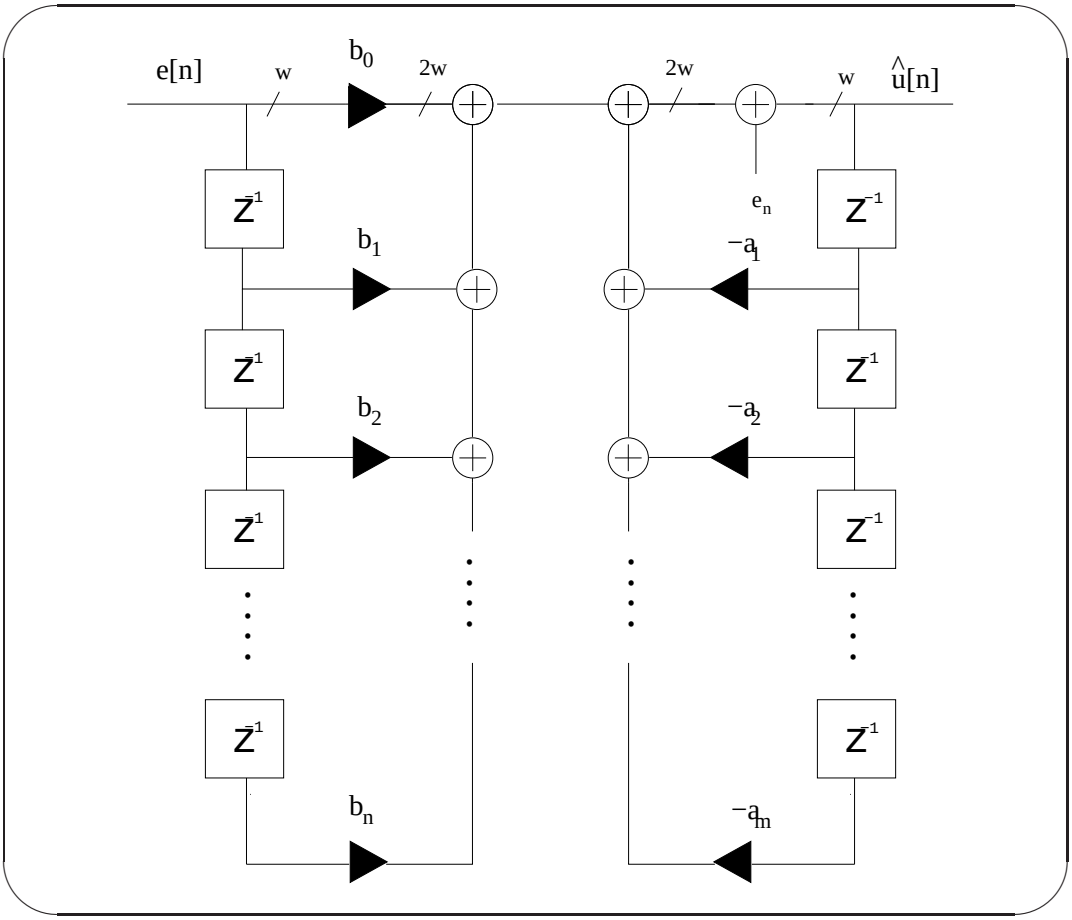


Figure 3.26: Simulated structure considering truncation as a source of error, e_n

include any quantization block). Thus, $p_{integrator}$ is not moved outside of the unit circle and the programmed filter is not having a divergent error like in the direct form case (see Figure 3.24). The source of error is coming from the two lower branches.

The model of one of the branches can be modelled analyzing Figure 3.28. The truncation block is placed after the arithmetic sum of the input and the previous state. The effect of the truncation is seen to be modelled as an error characterized by the mean and variance, see Equations (3.76) and (3.77). The perturbed output to the residue can be written by

$$\widehat{u}[n] = b[n] \cdot \widehat{w}[n] \tag{3.89}$$

The value after the addition, $\widehat{w}[n]$, is calculated in the following equation

$$\widehat{w}[n] = x[n] - p_i \widehat{w}[n-1] + e_n \tag{3.90}$$

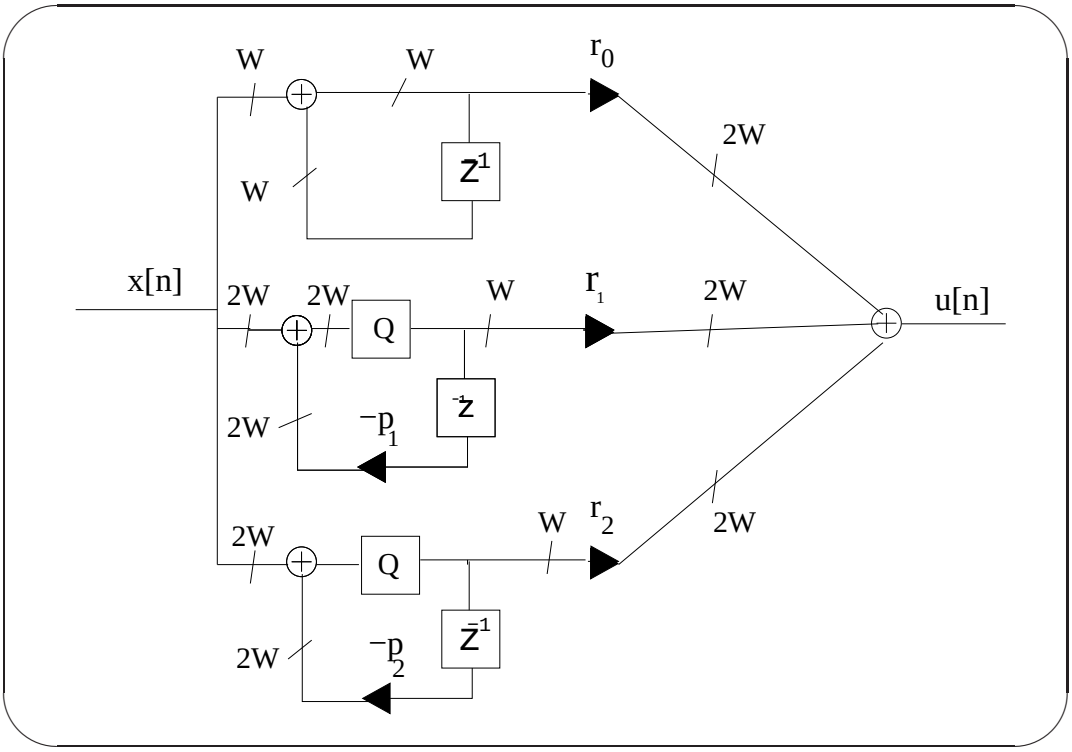


Figure 3.27: Parallel programmed structure

The same disturbed value can be calculated considering that there exists a shift in the pole p_i

$$\widehat{w}[n-1] = x[n] - \widehat{p}w[n-1] \tag{3.91}$$

Equalling Equations (3.90) and (3.91), we have that the pole shift by the truncation is

$$\Delta a = a - \widehat{a} = e_i \tag{3.92}$$

The disturbance at the output of the residue element, $\Delta u[n]$ is calculated as follows

$$\Delta u[n] = \sum_{i=0}^{n} r_i e_i[n] \tag{3.93}$$

n being the number of residues of the structure.

One important question about the disturbance at the output of the parallel structure is the number of bits at its output. We need to calculate the number of bits of the output signal in order to avoid the arithmetic error detected by the DAC or in our case by the DPWM. Therefore, the maximum acceptable error is

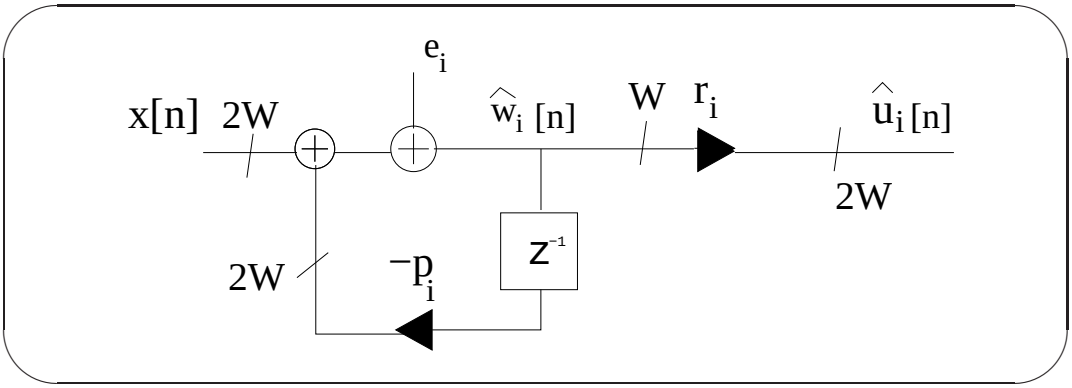


Figure 3.28: Error Analysis of a residue

$$\|\Delta u[n]\|_\infty \leq 2^{-n_{DAC}} \tag{3.94}$$

Using Equation (3.93) into Equation (3.94), we obtain the following unequality

$$\|\Delta u[n]\|_\infty = \|\sum_{i=0}^{n} r_i e_i[n]\|_\infty \leq \sum_{i=0}^{n} r_i \|e_i[n]\|_\infty \leq 2^{-n_{DAC}} \tag{3.95}$$

As we know, the maximum error produced by every residue using truncation is

$$\|e_i[n]\|_\infty = \frac{1}{\Delta} = 2^{-n_{controller}} \tag{3.96}$$

$n_{controller}$ being the number of fractional bits for quantizing the signal.

Replacing Equation (3.96) into Equation (3.95), we can derive the number of required bits for the parallel converter

$$2^{-n_{controller}} \sum_{i=0}^{n} r_i \leq 2^{-n_{DAC}} \tag{3.97}$$

Taking $\log_2$ to both sides of previous equations, and using the propierties of logarithms $\log_2(a \cdot b) = \log_2 a + \log_2 b$

$$\log_2 \left(2^{-n_{controller}} \sum_{i=0}^{n} r_i \right) \leq \log_2 \left(2^{-n_{DAC}} \right) \tag{3.98}$$

$$\log_2 2^{-n_{controller}} + \log_2 \left(\sum_{i=0}^{n} r_i \right) \leq -n_{DAC} \tag{3.99}$$

$$-n_{controller} \leq -n_{DAC} - \log_2 \sum_{i=0}^{n} r_i \tag{3.100}$$

The final unequality validating that the output error is lower than the dynamic range of the DAC converter is

$$n_{controller} \geq n_{DAC} + \log_2 \left(\underbrace{\sum_{i=0}^{n} r_i}_{\text{sum of residues}} \right) \tag{3.101}$$

Last equation gives us the number of necessary bits to obtain an acceptable error produced by arithmetic truncation. This number of bits must be necessarily bigger than the number of bits for the DAC converter plus a second term composed by the sum of the residues of the function. It is important to see that the magnitude of the residues plays an important role because they are multiplying the error introduced by the truncation block. Thus, the bigger the magnitudes of the residues are, the more bits are needed to compensate the multiplicative effect.

Example 3.4.5.1. *Consider the control system composed by our FPGA and the Buck converter. The maximum number of bits for the DPWM (DAC Converter) that our FPGA can implement is 10 bits. The number of necessary bits to not propagate the arithmetic errors to the DPWM can be calculated using two of the three residues in our 3^{rd}-order filter (r_0 and r_1). Remember that our integrative branch is not having any truncation block (see Figure 3.27)*

- $r_1 = 10,3048957$
- $r_2 = -3,8074951171850$

The number of bits required bits is calculated using Equation (3.101)

$$n_{controller} > 10 + \log_2 (10,3048957 - 3,807495117185) \geq 13 \tag{3.102}$$

Some simulations using Matlab fixed-point toolbox *were performed to validate Equation (3.101). The Figure (3.29a) shows a simulation having 12 bits as a rational part to code the impulse response signal. We can appreciate that during some samples, the arithmetic error is propagated to the DPWM, exceeding the LSB of the DPWM.*

The minimum input value detected by the DPWM in this example is

$$2^{-10} \approx 1 \cdot 10^{-3} \tag{3.103}$$

On the other hand, in Figure 3.29b we can appreciate that just increasing one bit, the propagation effect disappears. Thefore, the upper bound given in Equation (3.101) is validated

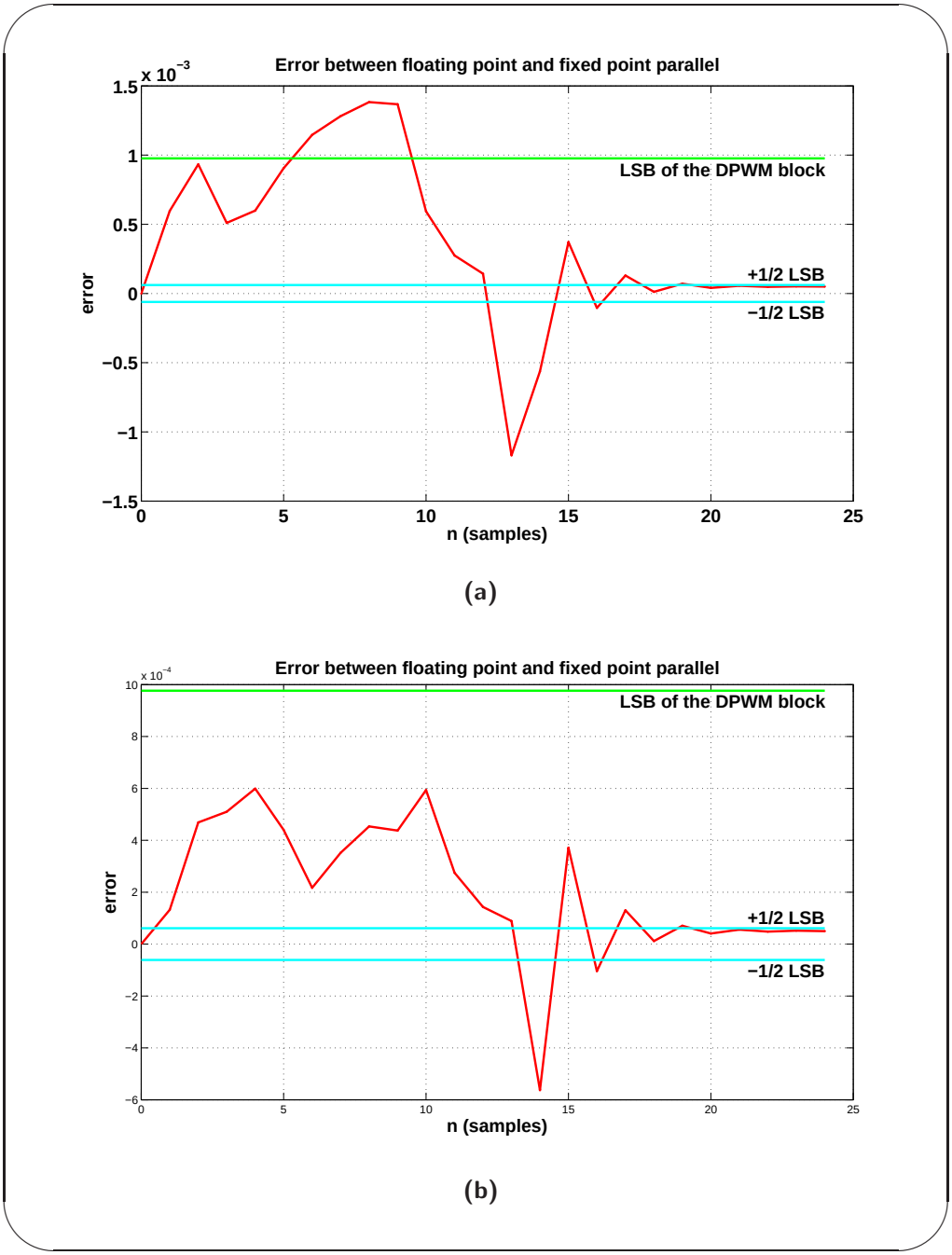


Figure 3.29: Error in the parallel structure implementing the 3^{rd}-order filter. (a) shows the error when the quantized signal is using 12 bits. (b) shows the error when the quantized signal is 13 bits.

3.5 Conclusions

The end of this chapter tries to summarize all the explained in this section. The purpose of it was to try to develop a theorical methodology to implement linear controllers in ASICs or FPGAs. First of all, the representation of the set of real numbers in computation was represented in a digital system. The choice is fixed-point representation although this representation has a range and accuracy much smaller than floating-point representation with the same data width. On the other hand, fixed point representations have the main advantage of requiring a less complicated hardware synthesis. This fact, allows to save size of sillicon in the FPGA/ASIC and also to save energy. After defining the type of numerical representation for our linear controller, it was necessary to identify many problems before the construction of a digital controller inside of an FPGA. The lack of a rigid arithmetic hardware such as DSPs cores and the specification of a data width as in C compilers, suggested the idea of the following two questions. Our starting point was the implementation of a linear controller using the difference equation. The difference equation of a linear controller can be written as

$$u[n] = \sum_{k=0}^{n} \underbrace{b_k}_{coefficient} e[n-k] - \sum_{i=1}^{m} a_i \underbrace{u[n-i]}_{signal} \tag{3.104}$$

The difference equation is formed by a set of two different informations (see Equation (3.104)) : coefficients and input/output signals ($e[n-k]$ and $u[n-i]$). The effect of quantization coefficients in the fixed-point representation was analyzed in Section 3.2.3. The data width of the coefficient representation is a deterministic problem that can be solved previously with a good design. The quantization effect of the coefficients has a direct connection in the frequency response to the linear controller. The codification of the coefficients was shown in Example 3.2.3.1.

Later on, the chapter discusses about the number of bits to quantify the input and output signals for the linear controller. First of all, it is important to know how many bits are needed to quantify the integer part of the signals to prevent overflow/underflow. It is stated in literature that overflow/underflow problems can cause unstability problems. Arithmetic errors produced in the implementation can not be modelled in advance as a deterministic process. Some literature [21- 23] defines the arithmetic error as an ergodic process. The generation of an arithmetic error is generated sample by sample due to the truncation of the output signal. Some authors related to communications field characterized this problem in terms of SNR of the digital filter. Our contribution was to define an upper bound to avoid the propagation of arithmetic error in a control system for parallel structures.

Finally, a table to summarize the data width for coefficients and signal is included in Table 3.4.

Table 3.4: Summary of data width

Number of bits	Coefficients	Signal
Integer	5	10
Fractional	13	13

Chapter 4

HDL architecture of digital driven converters

This section takes all the theoretical results from the previous chapter to start with the implementation of the whole digital system. The previous chapter finished providing us the data width for signals and coefficients. The data width used for the design of the digital system will be the most restrictive comparing both metrics, because our purpose is to unify the data width. The advantage of unifying both parameters is a more compact design of the controller in the *VHDL* file. Nevertheless, this solution is not going to be the most optimal in terms of size of sillicon. The data width used for the digital controller will be codified using 23Q13 fixed-point codification as it was stated in Section 3.5.

On the other hand, this chapter is not only focussed in the HDL programming of the controller. The output voltage regulation needs other blocks to be programmed in order to make the Buck converter work. The control law will be mostly divided into the blocks that have been mentioned in Section 2.4.1. The list of elements, shown in Figure 4.1, are listed as follows

- **Adquisition module :** This block is intended to communicate the ADC that samples the output voltage of the Buck converter.
- **Controller :** This block includes the linear controller studied in the last chapter.
- **Memory :** It saves during one switching period the duty cycle calculated in the controller.
- **DPWM :** It generates the duty cycle for the Buck converter.
- **Clock Manager :** It synchronizes all the modules to be working every T_{sw}

The reason why *VHDL* is chosen instead of *Verilog* was because, as it was explained in Section 3.1.2, *VHDL* has the possibility to program digital

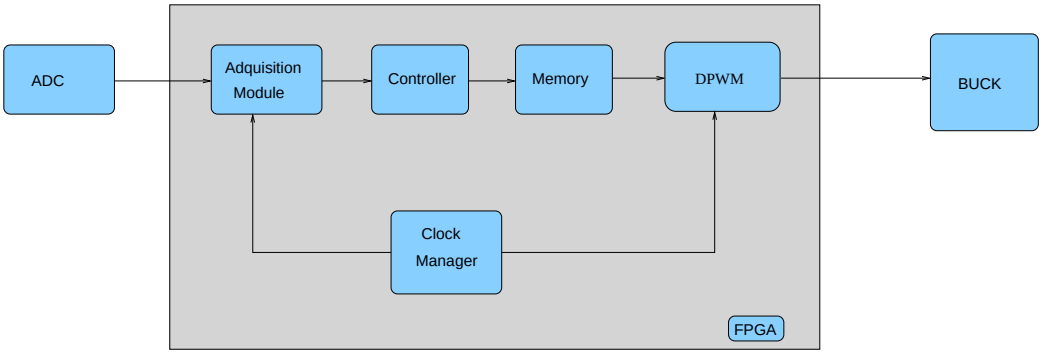


Figure 4.1: HDL blocks in our system

systems in a generic way. This feature has a tremendous advantage to allow reuse of *VHDL* code. For example, we have the possibility to implement linear controllers considering the problem of the data width of the coefficients and signal to be specified later. That is, in case the linear controller must be changed, we can reuse the same piece of code just changing the data width.

4.1 Clock manager

It is in charge of activating at the right moment all important events to make our architecture synchronous. For instance, it activates the DPWM module and it is in charge to generate the pulses driving D latches of the acquisition board. Since the FPGA is driven by an internal clock of $512MHz$ (see [24]), the clock manager must be an 9-bit counter, to activate the controller every $T_{sw} = 1MHz$. The clock manager is written as a behavioural statement that thanks to some "if" statements can activate every module at the right time.

4.2 Acquisition module

This module is associated to the external acquisition board (containing ADCs). It is in charge of driving the ADC and of reading a sample to be processed by the control algorithm. The design of this module consists (see Figure 4.2a) of a finite state machine. The states of the FSM can be seen in Figure 4.2a. E0 corresponds to the *"wait"*. The acquisition module is waiting until start signal is activated. E1 is the state in charge to start and stop the sequencer. E2 is the state that reads the data value of the ADC. Finally, E3 is the *"Done"* state, where the FSM sets the ouput signal *"Done"*. This machine drives a counter to perform the right number of pulses and as soon as the sample is ready, the counter is stopped and the the sample is stored. The resulting waveform is shown in Figure 4.2b.

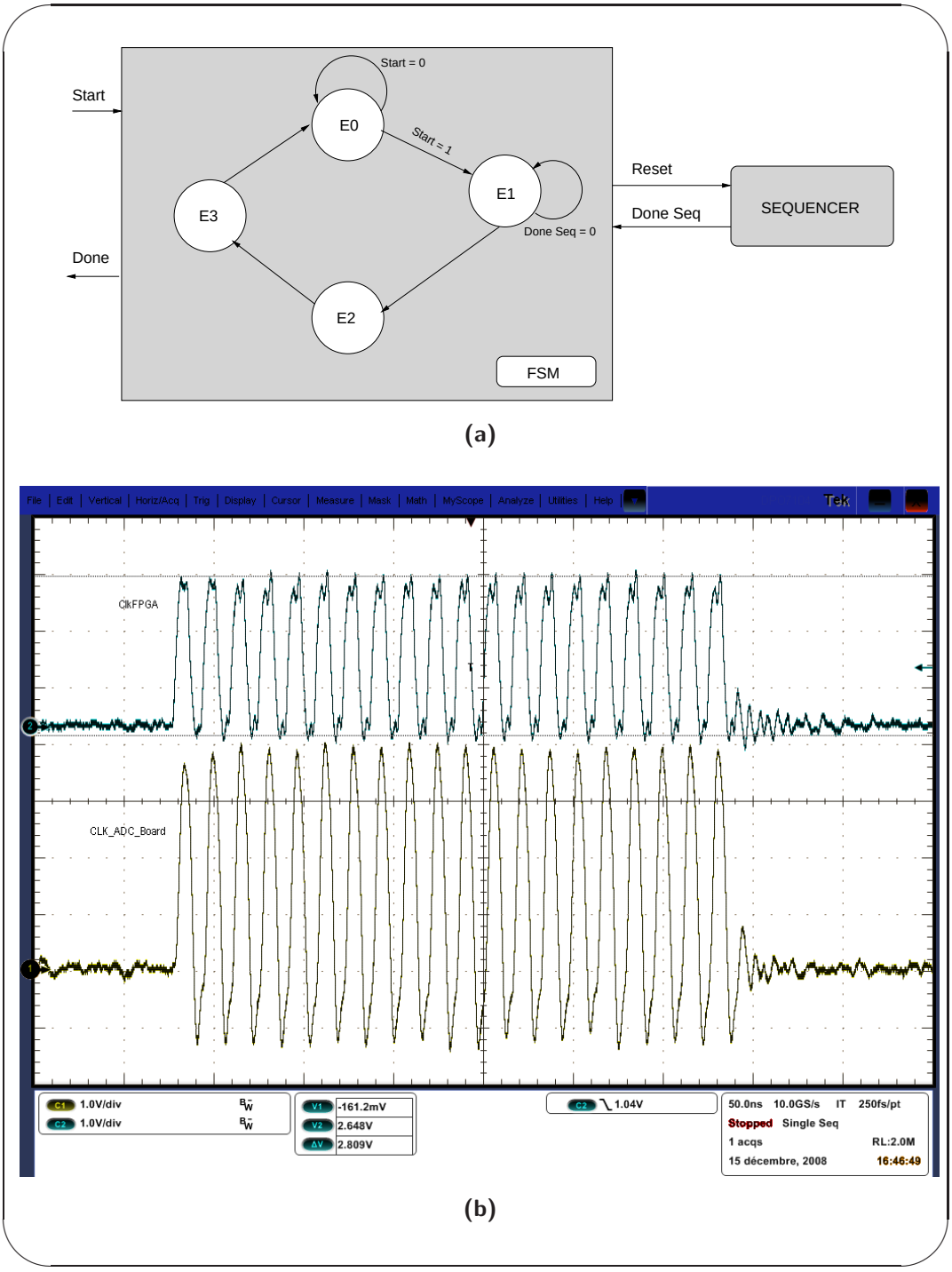


Figure 4.2: HDL design and experimental result of the acquisition module. (a) shows the design of the acquisition module. (b) shows a screen capture of the driving pulses for the ADC converter. The ADC [5] requires a minimum number of 6 clock pulses to sample the signal. We can see that for this case the frequency the FPGA drives the ADC converter was 60 MHz (time base $50\frac{ns}{div}$). For this case the number of impulses is 20. Thus, 14 samples were obtained in order to test our application.

4.3 Controller

4.3.1 Introduction

The controller block is in charge to calculate the command signal for the DPWM, depending on the sampled output voltage. Due to the complexity of this block and the high number of arithmetic circuits involved the latency associated to the calculation is difficult to be estimated. This is the reason why the best solution is to design a synchronous system that assures the output of a system by means of a clock. Therefore, each of the involved sub blocks are divided into two submodules: *Finite State Machine* and *Combinational Circuit*. The design of each sub block is as shown in Figure 4.3.

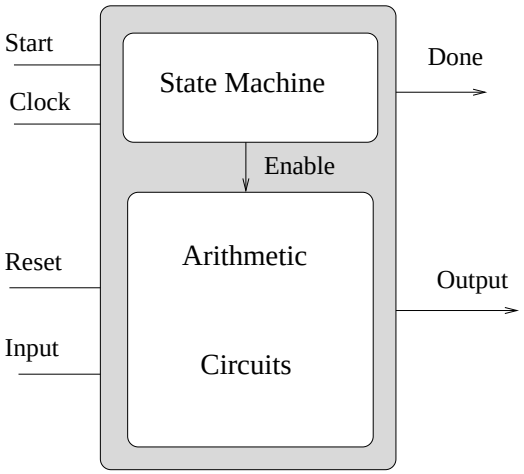


Figure 4.3: Structure of every sub-block

4.3.2 Finite state machine

This block has the task of coordinating which operations are calculated depending on the computability matrix and the degree of parallelism implemented in the controller [25]. The number of operations determines the number of states that the State Machine is required to have. Normally, FSM (Finite State Machines) have two structures: *Moore* and *Mealy* Structures. It has been proved using the Xilinx ISE software tool that substitutying the FSM by a ring counter increases the clock frequency, decreases the algorithm complexity and allows us easily to implement the function by *generics*. The ring counter consists of an array of D flip flops where the output of every single D flip flop represents the enable moment for every arithmetic circuit involved in the controller equation. The strucutre of the ring counter is shown in Figure 4.4a:

The ring counter has the following characteristics :

- In order to make the counter work, it is needed a pulse of the following characteristics: $T_{pulse} > T_{CLK}$
- A n number of D flip flops generates a sequence $modn$, allowing us to generate n instructions inside of each of the involved blocks.
- The maximum frequency that we are able to introduce is up to 1.5 GHz, according to the results obtained by XST, *Xilinx Synthetisable Tool*.

A simulation of the output of the ring counter is in Figure 4.4b

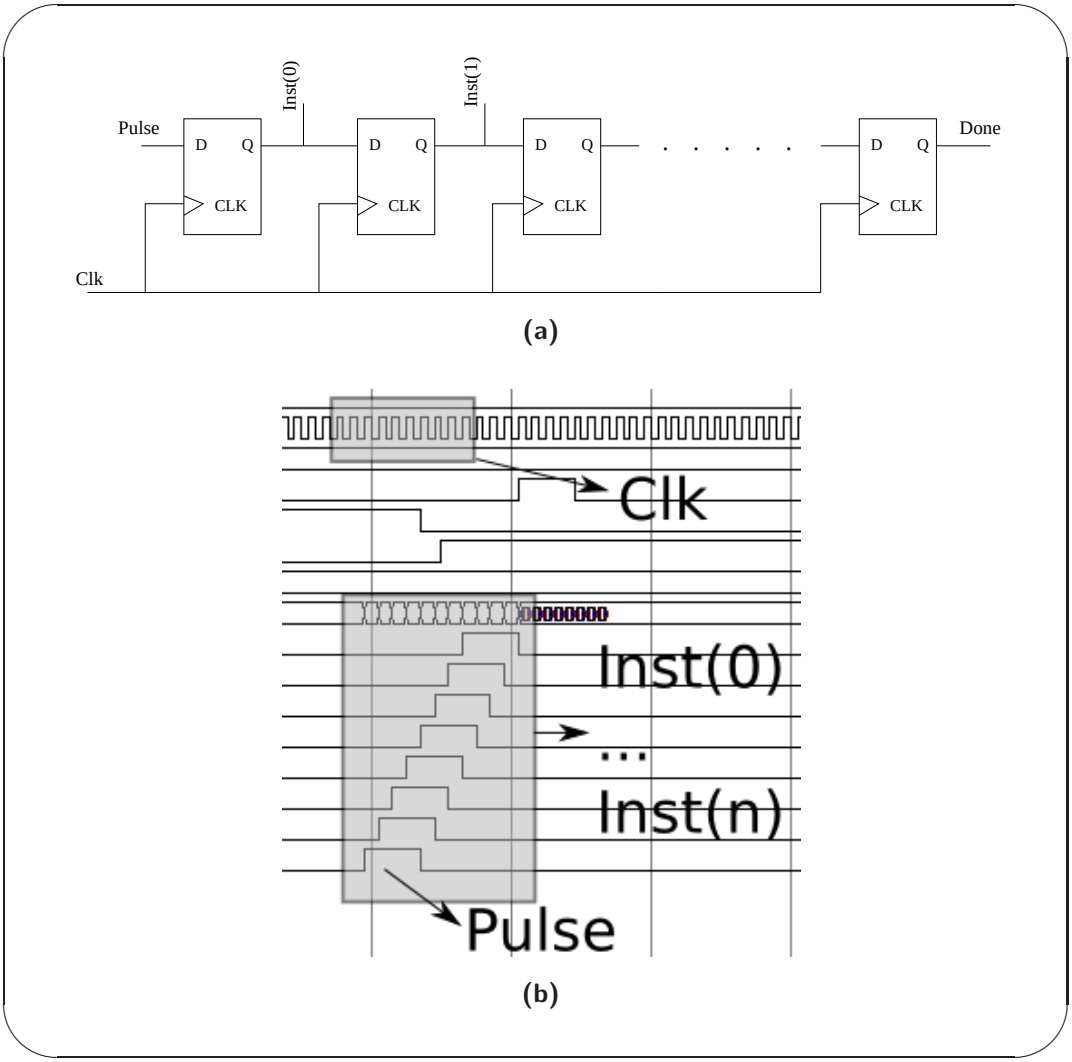


Figure 4.4: Ring counter. (a) shows the hardware structure. (b) shows a Modelsim capture of a ring counter.

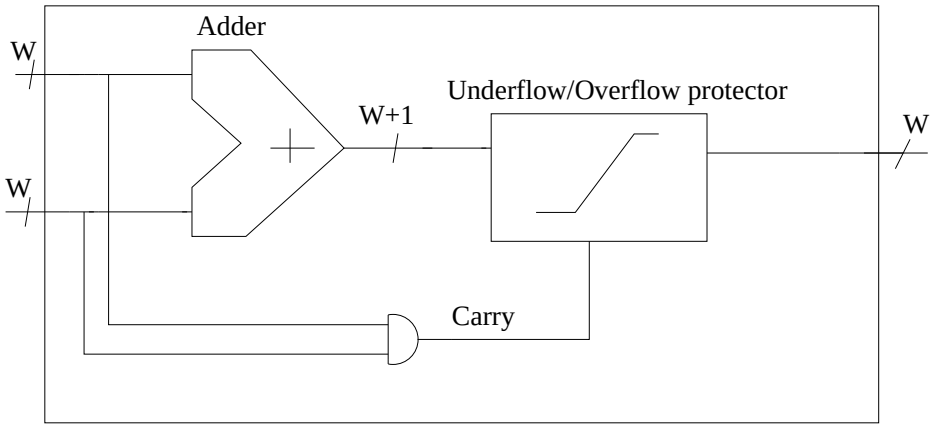


Figure 4.5: Adder block

4.3.3 Arithmetic circuits

4.3.3.1 Introduction

The sub module of arithmetic circuit tries to implemement the structure described in the previous section. For the case of the PID controller, the structure to be implemented will be direct form I. On the other hand, the 3-rd order filter is going to be implemented using parallel form due to the low rounding noise. The list of generic components that have been developped are : adder, multiplier, quantizer, saturation element, PID controller, 3^{rd}-order filter, integrative residue and residue function.

4.3.3.2 Adder

The adder has a registered output to be interfaced with the ring counter. The special feature of the adder is that internally it has an underflow/overflow detector. The detector is essential to prevent from non-linearities produced by wrap problems discussed in Section 3.2.4. Thus, the implementation of this arithmetic circuit with an underflow/overflow detector is essential for the integrative effect.

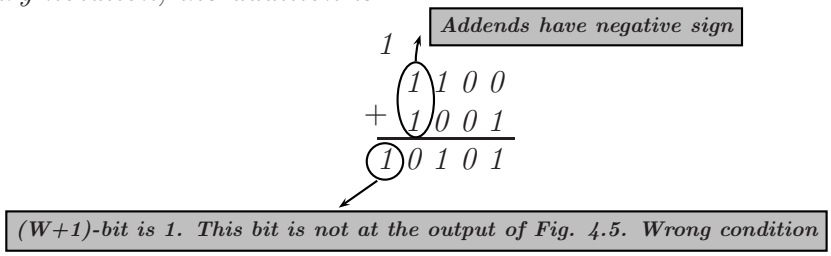
The *W-bit* adder is an arithmetic circuit whose inputs and outputs are real numbers using two's complement representation. To detect the underflow and overflow it is necessary to implement inside of the adder an *W+1* internal register that represents the addition of the two inputs. The detection of underflow and overflow conditions are detected using the *N+1* and *N* bits of the internal register together with the sign of the inputs. The structure of the adder can be seen in Figure 4.5. The previous stage before the output is the underflow/overflow protection. To the input of such block we have the addition of the addends and the carry/borrow of the addition (**AND** gate).

Example 4.3.3.1. *Considering we have in our circuit a 4-bit adder. The output of such adder, according to Figure 4.5, must have a data width of 5 bits. If the binary code used in our system is the 2-complement representation, the input/output range of the signal must not exceed the interval* $[-8, 7]$. *This example discusses the logical conditions that the underflow/overflow block in Figure 4.5 must meet to prevent from having wrap problems.*

The first analysed situation is the underflow detection. For example, when the addends are $-4_{10} = (1100)_2$ *and* $-7_{10} = (1001)_2$ *the addition using a decimal notation is*

$$-4 + (-7) = -11$$

In binary notation, the addition is



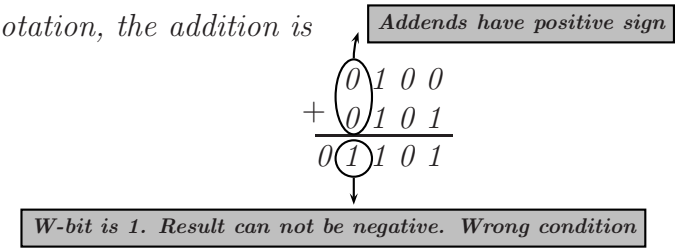
In this situation the output of the adder must be $-8_{10} = (1000)_2$. *According to the previous example, the conditions showing that the sum of two negative numbers must provoke an underflow is whenever the following two conditions are met.*

1. *The MSB of both addens is equal to one.*
2. *The resulting* W+1 *bit at the output of the adder is equal to one.*

On the other hand, when the addends are positive and the addition exceeds the positive range of the binary word, the underflow/overflow block should detect an overflow condition. For example, when the inputs to the adder are now $4_{10} = (0100)_2$ *and* $5_{10} = (0101)_2$, *the addition using decimal notation is*

$$4 + 5 = 9$$

In binary notation, the addition is



In this situation the output of the adder must be $7_{10} = (0111)_2$. *The conditions showing that the sum of two positive numbers are now provoking overflow are*

1. *The MSB of both addens is equal to zero.*
2. *The resulting* W *bit at the output of the adder is equal to one.*

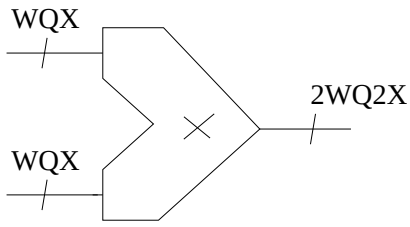


Figure 4.6: Multiplier block

4.3.3.3 Multiplier

As the adder, the multiplier has a registered output. The input of this block is coded into *WQX* but the output is coded into *2WQ2X*. Because the output of the multiplier has a double width than the inputs , we do not have any overflow problem as in the case of the adder. See Figure 4.6

4.3.3.4 Quantizer

This is the block less straightforward and more abstract of all defined in this Section. This element tries to apply a truncation effect to the output of the multiplier. This arithmetic circuit is crucial for the design of a linear controller. The function is to delimit the data width of the multiplier output. The data width of the multiplier output is having the same data width as the inputs when the quantizer is placed after the arithmetic circuit. The quantizer function can be described applying the operator $q\{\}$ to a number c_i. The numerical representation of a fixed-point number coded by a WQX format is:

$$A = \sum_{i=-X}^{W-X} a_i \cdot 2^i \tag{4.1}$$

The FPGA representation corresponds to:

$$B = \sum_{i=0}^{W} a_i \cdot 2^i \tag{4.2}$$

Equations (4.1) and (4.2) are related by a factor 2^W. After multiplying, A and B having WQX representation, the result C is :

$$C = \sum_{i=-2X}^{2(W-X)} c_i \cdot 2^i \tag{4.3}$$

$$C = \sum_{i=0}^{2W} c_i \cdot 2^i \tag{4.4}$$

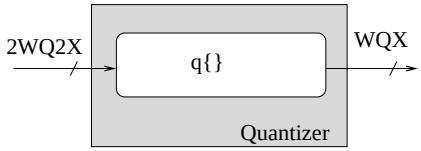


Figure 4.7: Quantizer block diagram

The implementation of the quantizer function is the answer to the following question. If the result of multiplication between two numbers, coded in WQX, is coded into $2WQ2X$, how can we return to the original code WQX?. The suggested Equation (4.5) shows a possible implementation to develop an arithmetic circuit using a dataflow description. First of all, and due to the numerical propierties of the 2-complement representation, the MSB is carrying the sign. Thus, the MSB for the input and output must be the same. The integer part of the output word will correspond to the input because the number of bits for the input signal was calculated to never have an overflow situation (see Section 3.2.4). The decimal part will be the X bits corresponding to output due that the rounding method is truncation. In mathematical terms the description is

$$q = \begin{cases} c_{2W-1} \cdot 2^{2W-1} & i = W-1 \\ \sum_{i=2X}^{W+X-1} c_i \cdot 2^i & X \leq i \leq W-2 \\ \sum_{i=0}^{X-1} c_i \cdot 2^i & 0 \leq i \leq W-2 \end{cases} \tag{4.5}$$

4.3.3.5 Saturation element

The block is in charge to compare the output value of the controller and to delimit the maximum values in order to prevent overflow/underflow effects. The equation for this block using WQX codification is :

$$y[n] = \begin{cases} -2^{W-X} & \text{if } x[n] \leq -2^{W-X} \\ 2^X - 1 & \text{if } x[n] \geq 2^X - 1 \end{cases} \tag{4.6}$$

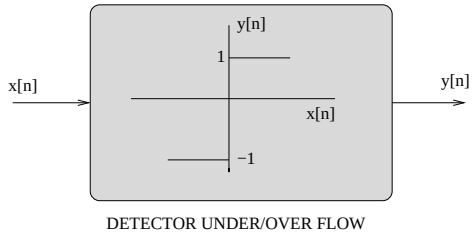


Figure 4.8: Schema of the under/over flow detector

4.3.3.6 PID controller

The architecture of the PID controller together with the data width for every path is shown in Figure 4.9. All the arithmetic circuits composing the PID were comented in the previous section. The block called as T is the saturation element analysed previously. Let's remember that the outputs and inputs, for the PID study case, were normalized using the simplifications made in Equations (3.53)-(3.62).

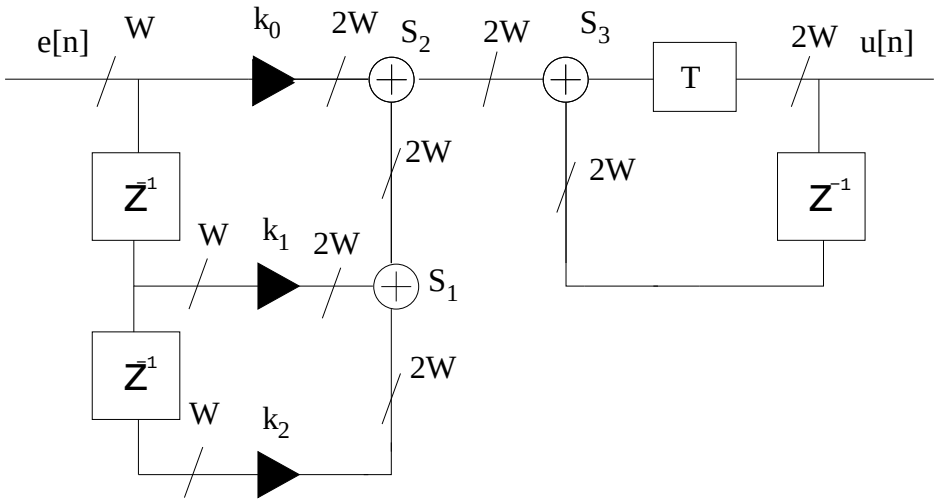


Figure 4.9: VHDL structural model of the PID controller

The operations performed in the PID controller are

1. Shift Registers $u[n-1] = u[n]$, $e[n-2] = e[n-1]$
2. Shift registers $e[n-1] = e[n]$
3. Register $e[n] = V_{ref} - V_{out}[n]$
4. Register $k_0e[n]$, $k_1e[n-1]$, $k_2e[n-2]$
5. Register S1
6. Register S2
7. Register S3
8. Indicate operation is $Done =' 1'$

The number of operations listed previously were eight. The main advantage of using FPGAs, compared with DSP architectures, is that the number of instructions were decreased. For instance, operation number 4 is calculating three results at the same time. The reason of increasing the degree of

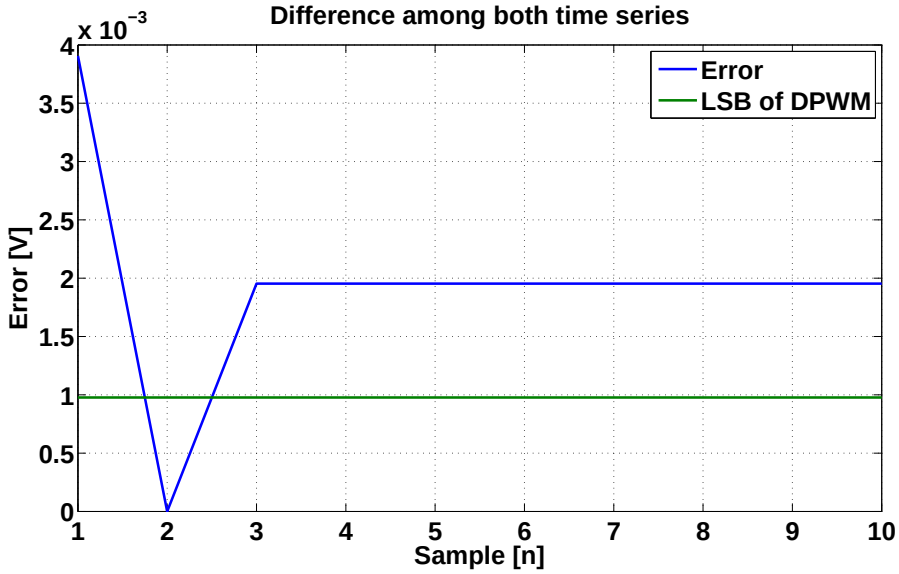


Figure 4.10: Numerical Error between PID impulse response using Matlab floating-point accuracy and the architecture designated in Figure 4.9. The data width of the coefficient/signals is summarized in Table 3.4. The error between both impulse response is expressed in voltage.

pipelining in the algorithm was because in the FPGA we can have physically all the adders, memory and multipliers.

The resulting error between the PID impulse response of our simulated VHDL architecture and the Matlab floating-point accuracy is shown in Figure 4.10. The script that process the data coming from *ModelSim* is found in Appendix F.

We can appreciate in Figure 4.10 that the numerical error for the PID signal is larger than the accuracy of the DPWM. The source of error is not coming from any truncated signal because in the architecture of the PID any quantizer block was needed (see Figure 4.9). The origin of this error is found in the normalization used in Equation (3.53). This normalization divides all the signals during the design time of our architecture. Thus, the signals flowing inside of our FPGA have a magnitude comprised in the range $[-1, 1]$. As the signal was codified in this case using the same codification than the coefficients, 11Q8, the signal is only using 8 bits to be represented. One solution should be the increase of the fractional length in order to enclose the error into the upper bound of the DPWM accuracy. Another solution should be the codification of the signal using Equations (3.64)-(3.66).

4.3.3.7 3^{rd} order filter

The structure of the third order filter is shown in Figure 4.11. As it can be seen, every *n-order* filter can be programmed generically decomposing the filter as the sum of its residues. The particularity of our controller is that it requires an integrative branch in order to cancel the steady-state error (error between the reference voltage and the output voltage). Therefore, the best programming method suited in a n^{th}-order controller, implemented using parallel structures, is using structural definitions.

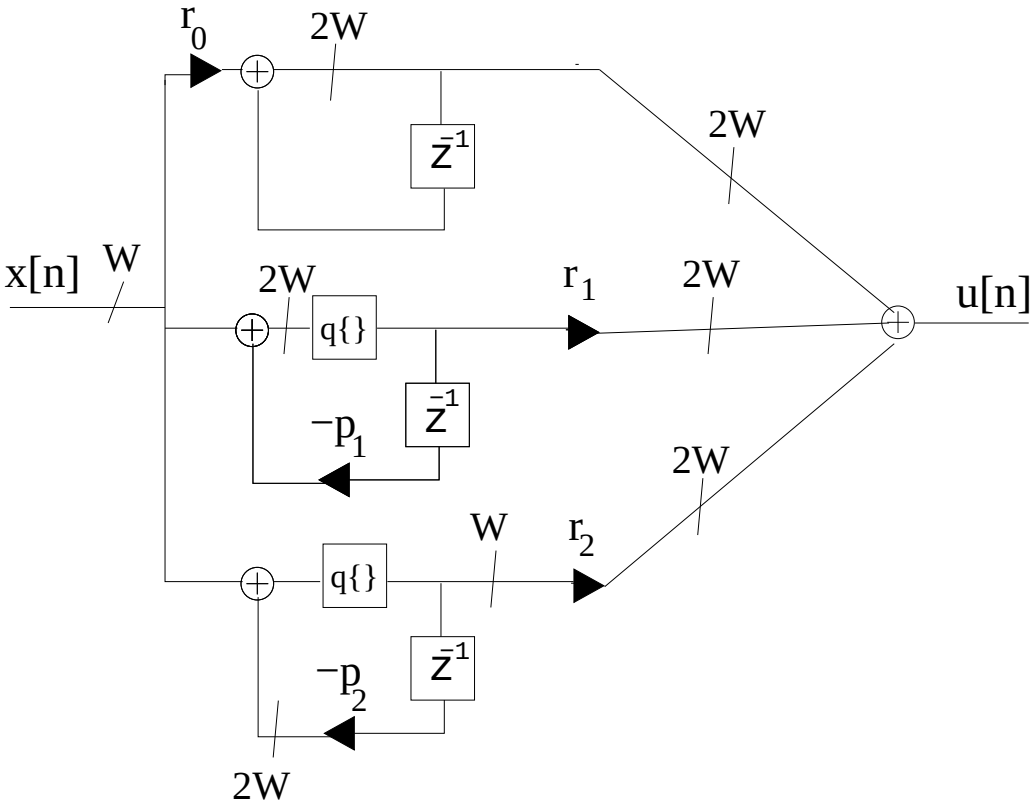


Figure 4.11: VHDL implementation of a 3^{rd}-order filter

Integrative residue The integrative residue is in charge to perform sequentially the accumulation of the input $e[n]$. The reason why in this structure the multiplier r_0 is placed before the adder is to delay the effect of saturation in the system. The parameter r_0 in this case was 0.001464843. Thus, the magnitude of the input is decreased and consenquently the effect of accumulation is delayed. Another reason why direct form structure saturates later than canonical structure is because the adder with saturation protection explained earlier (adder S1) has a data width of *2W*. Figure 4.12b compares the response of the 3^{rd}-order filter using both integrative branches. For the case when the integrative branch is implemented using canonical form, the

system saturates when the sample is ≥ 500. On the other hand, we see that direct form I is still not saturated in all the performed simulations.

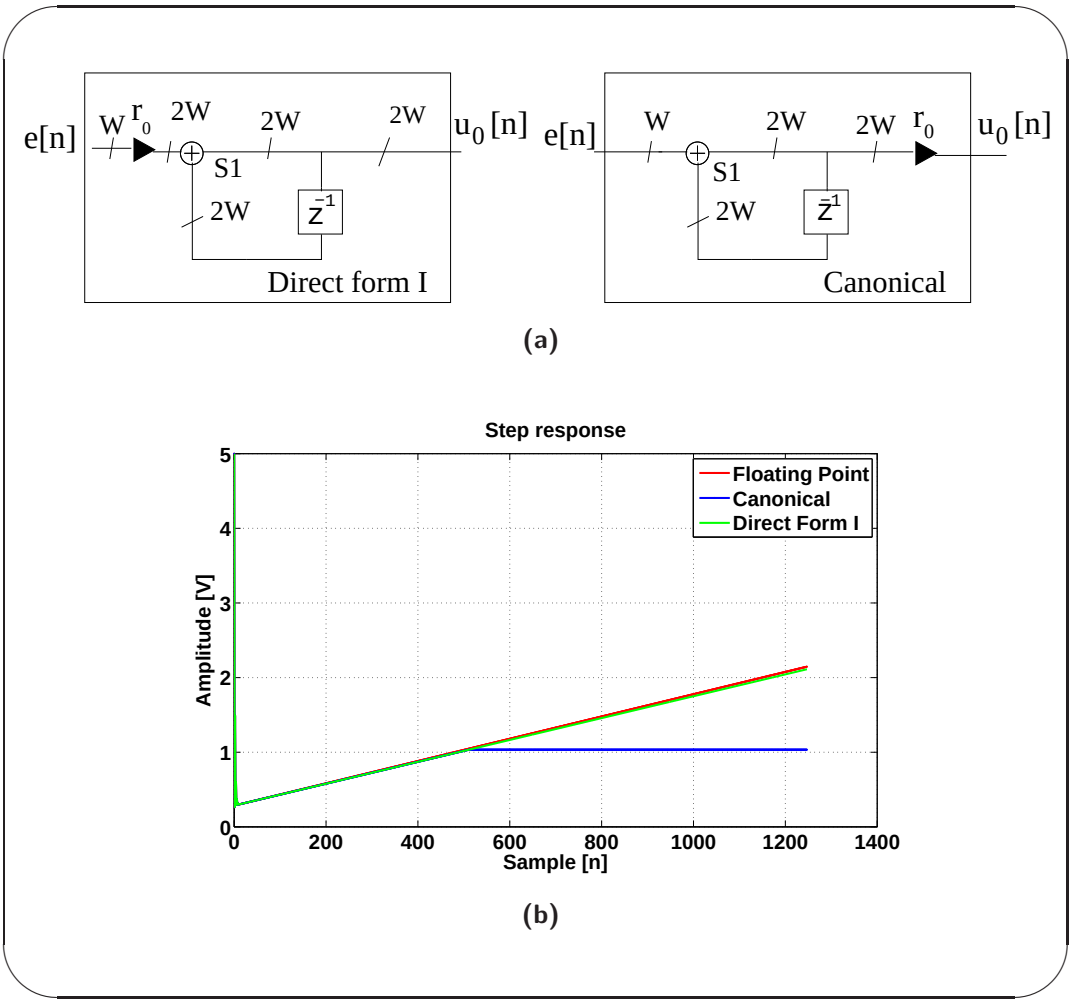


Figure 4.12: Integrative residue implementation and analysis. (a) shows 2-ways to implement integrative effect. (b) shows ModelSim results of the 3^{rd}-order filter using both integrative strategies.

The sequence of operations in the integrative branch in both forms is

1. Register $u_0[n-1] = u_0[n]$
2. Register result of multiplier $r_0 \cdot e[n]$
3. Register the result of the adder S1
4. $Done =' 1'$

The total number of operations is equal to 4.

Residue function This sub-block is included only for transfer functions that can be divided into the sum of its residues. Therefore, this block is only called for the case of a 3rd-order filter. The form of this function, including the length in bits of every branch is shwown in Figure 4.13 and it corresponds to Equation (3.80).

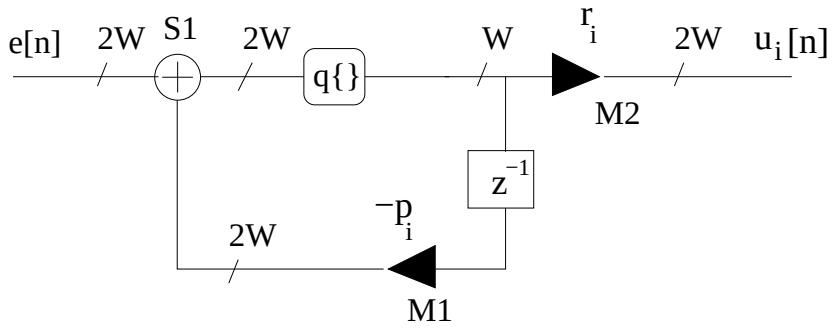


Figure 4.13: Residue architecture

The sequence of operations for one of the residues is

1. Register $u_i[n-1] = u_i[n]$
2. Register the result of the multiplier M1
3. Register the result of the adder S1
4. Register the result of the multiplier M2
5. $Done =' 1'$

The number of operations to calculate the output of the 3^{rd}-order filter is the maximum number of operations comparing the number of operations having both the integrative branch and the residue branch. In this case, the maximum number of operations corresponds to the residue function, being the result equal to 5. The main advantage of using FPGA in comparison with the use of DSPs is the possibility to compute in parallel the operations of the integrative branch. Oppositely to the operation of FPGAs, DSPs would calculate the operations of every branch in the 3^{rd}-order filter sequantially. Finally, the conclusion is that the computational complexity to calculate linear controllers in DSPs architectures increases in the same way as the order of the filter.

As a result, we can see in Figure 4.14 the comparison between the VHDL 3^{rd}-order structure and Matlab accuracy. It can be appreciated that in this case, the upper bound is respected because it was possible to calculate the number of necessary bits to quantize the signal (see Section 3.4.5.2). The figure also shows that the performed simulations in Section 3.4.5.2 using the *Fixed-Point* toolbox in *Matlab* and the *VHDL* architecture have the same results.

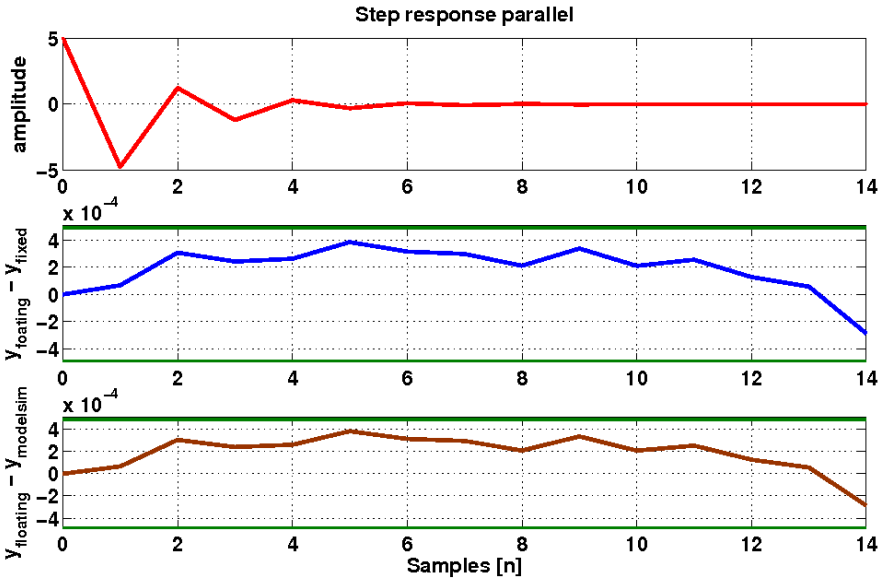


Figure 4.14: Comparison between ModelSim results and Matlab accuracy in a 3^{rd}-order filter. Top graph shows the impulse response of 3^{rd}-order filter. Middle graph shows the error between *Matlab* floating-point accuracy and *Matlab* fixed-point toolbox accuracy. Bottom graph shows the error between *Matlab* floating-point accuracy and the VHDL 3^{rd}-order filter architecture appearing in Figure 4.11. The data width for both coefficients and signals are summarized in Table 3.4.

4.4 Controller to DPWM transition

It is one of the most important sub blocks to be implemented. It is in charge to adapt the result of the output of the controller to the dynamic range of the driver in the MOSFET circuits. In [26], the minimum and maximum duty cycles are comprised between $0.07 < d < 0.85$. There is also another issue before modeling the transfer function of this block. First of all, it is important to model the PWM for the analog case and try to apply per-unit method. During simulations performed in [3], the analogical PWM is unipolar. The equation that generates the transfer function was already described in Equation (2.82), highlighting also that the PWM in our system was unipolar. Therefore, since the output values coming from the controller were normalized the transfer function of this block is :

$$d[n] = \begin{cases} 0,07 & \text{if } u[n] < 0 \\ 0,78 \cdot u[n] + 0,07 & \text{if } 0 \leq u[n] \leq 1 \end{cases} \tag{4.7}$$

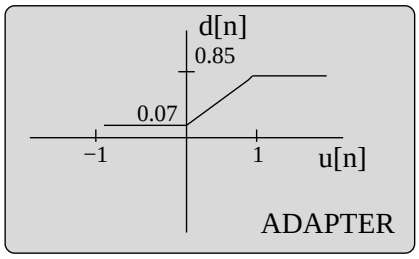


Figure 4.15: Working principles

4.5 Memory

It is in charge to create an initial duty cycle when both the Buck converter and the FPGA are switched on. It is also used to store the result coming from the controller to DPWM block and introduce the sample in the DPWM one switching period later. The resulting *VHDL* behavioral model synthetizes a n-array of D latches. Thus, the inclusion of a memory in the digital system prevents from creating PWM waveforms having duty cycles equal to 1.
The origin of this effect is the associated delay with the ADC and the FPGA circuitry. The aforementioned problem is illustrated in Figure 4.16. In the upper part of the figure, T_{cycle} is the needed time to calculate the current duty cycle, $d[n]$. In case T_{cycle} is bigger than T_{on}, the SR-latch of the DPWM (see next section) will not reset the control signal until the next cycle.

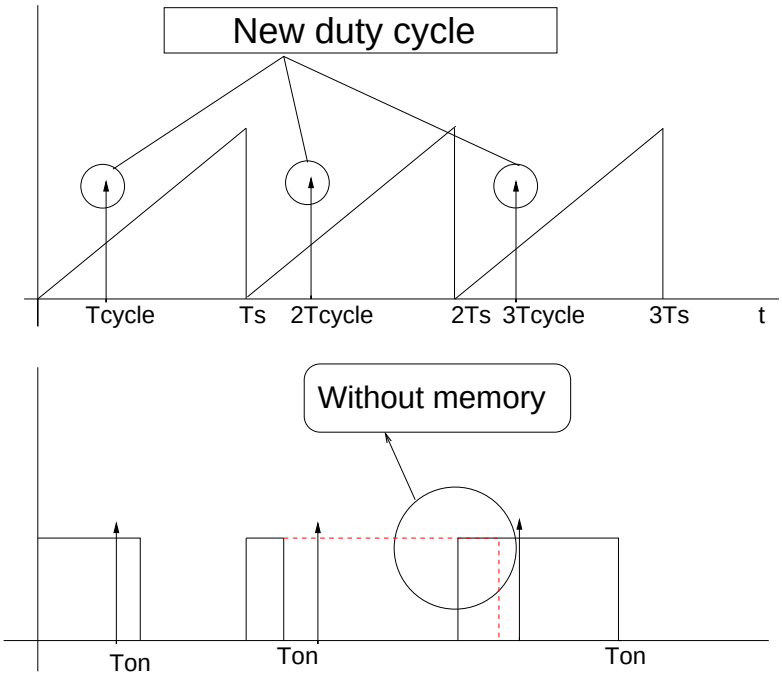


Figure 4.16: Skip problem

4.6 DPWM

DPWM is in charge to produce the duty cycle signal. The architecture of the block, implemented in our FPGA, resembles the structure of the fast clock counter mentioned in Section 2.5.1. Nevertheless, this time it is slightly modiffied in order to obtain the maximum accuracy. Equation (2.84) relates the number of bits and the clock frequency of a fast clock counter. Our goal is to maximize the data width of the DPWM input without exceeding the maximal inner PLL frequency of the Virtex V [24]. As the maximum inner frequency is 512 MHz the maximum number of bits is 9. Figure 4.18a shows the architecture of the DPWM block using the *Fast Clock Counter* strategy. The suggested modification is based on the design of two generic counters in *VHDL* instead of one. The only difference among both counters is that one is intended for counting even numbers contrary to the last counter designed for counting odd numbers. The final count is multiplexed by means of a multiplexer. Using this strategy we can artificially double $f_{counter}$ since we are now using both rising and falling edges to count. Thanks to the aforementioned modification, we obtain a gain of 1 bit in the DPWM block. The Register level scheme of the modified *Fast Clock Counter* is showed in Figure 4.18b. The last question is to know if the output multiplexer is able to switch the data bus at a speed of 512 MHz. The synthesized VHDL architecture of Figure 4.18b was tested in a real FPGA and the experimental results were dumped in Figure 4.17. As the figure shows the output of the multiplexer is a staircase function simulating the behaviour of an unipolar PWM. The logic analyzer shows that the hardware is able to generate counting sequences of 10 bits.

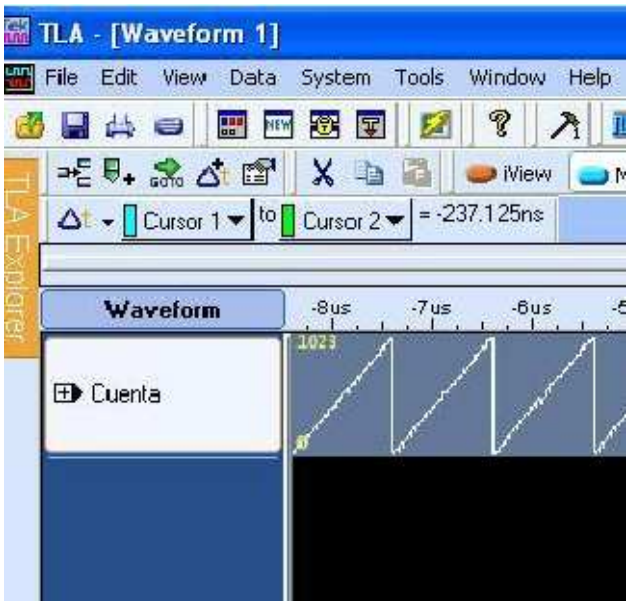


Figure 4.17: Logic analyzer capture of a 10-bit ramp

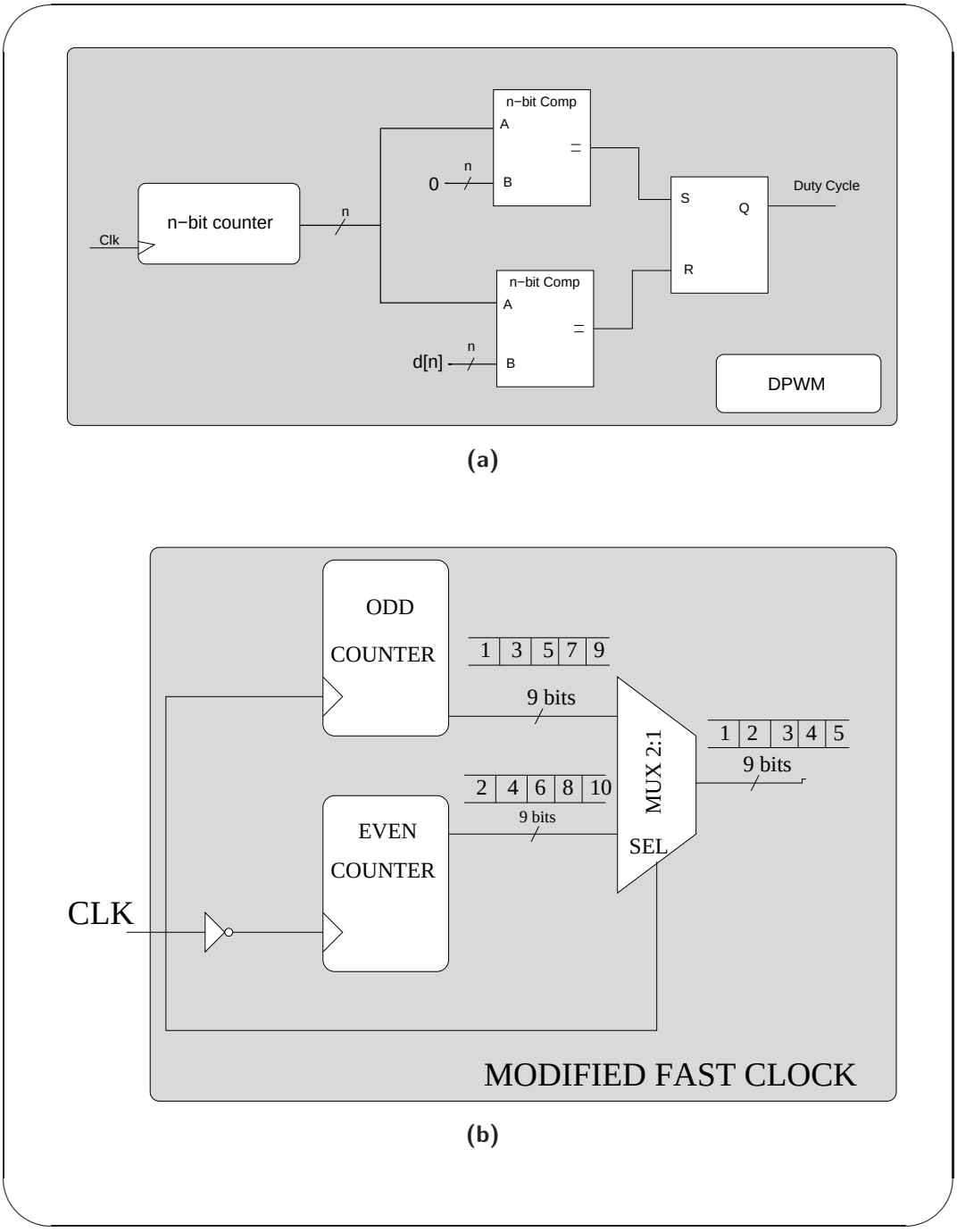


Figure 4.18: Fast clock counter and modified FPGA implementation. a) shows fast clock counter architecture. b) shows the FPGA synthetised architecture.

4.7 Simulations of the VHDL architecture

The last section of this chapter shows graphically, by means of the simulation tool *ModelSim*, the operation of all the blocks appearing in Figure 4.9. Every subsection of this chapter shows the operation of the controllers analyzed along this thesis. The first section shows the response for the PID controller. The second and last subsection shows graphically the simulations for the third-order controller. Thus, these simulations highlight the modularity of our digital controller because by replacing the PID block for another more suitable control strategy, the system should be ready to work.

4.7.1 Digital System with PID controller

Figure 4.19 shows the impulse response of the PID controller inside the digital system. The operation of the *Controller to DPWM* block can be seen in Figure 4.19 at the bottom. This block is essential for the correct operation of the Buck converter. At the intitial time, $n = 0$, the PID output is maximum and the duty cycle is also maximum. During the rest of time, the output of the PID is negative. Therefore, the duty cycle of the PWM generated pattern is having a minimum duty cycle.

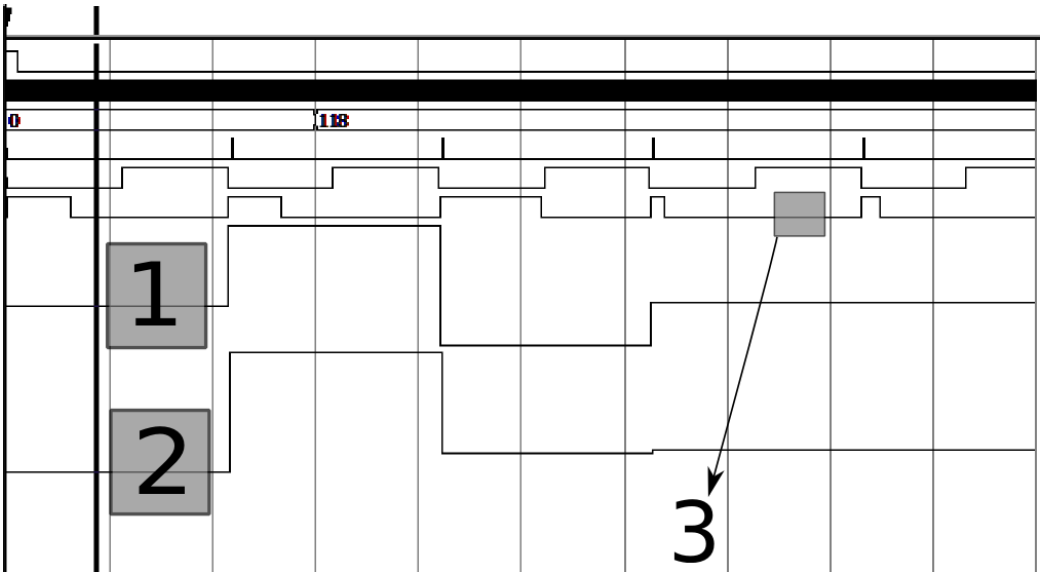


Figure 4.19: ModelSim simulation of the PID impulse response. 1) shows the output response of the controller. 2) corresponds to the output of the Controller to DPWM transition block. 3) shows the response of the DPWM block.

Figure 3.22 shows the response of the digital controller to the pulse. As in the previous simulation, the maximum value corresponds to the intial sample, $n = 0$. After this, the value of the PID controller increases sample after sample because of the integrative effect that it is inherently in this filter.

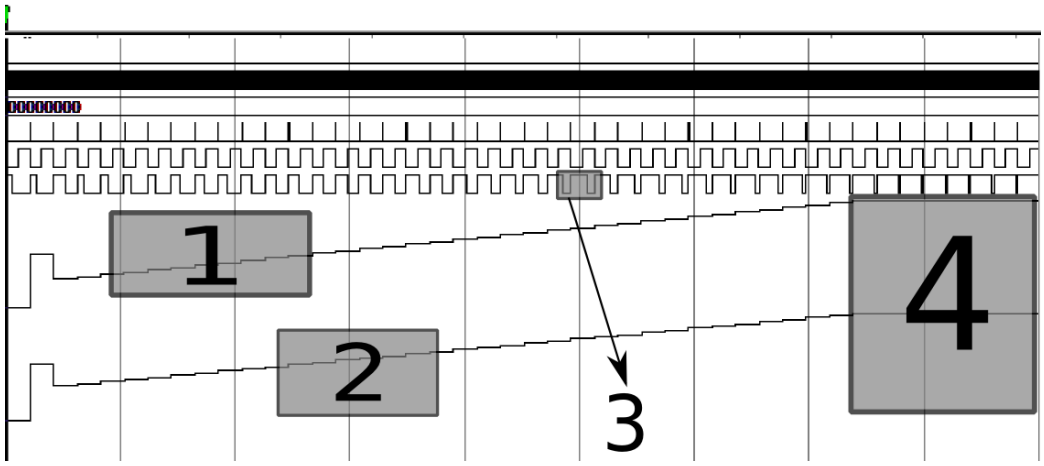


Figure 4.20: ModelSim simulation of the PID step response. 1) shows the output response of the controller. 2) corresponds to the output of the Controller to DPWM transition block. 3) shows the response of the DPWM block. 4) shows the moment the response is saturated.

The value of the PID is limited by the overflow protection block. The duty cycle at the PWM output of the digital system increases until the output of the controller saturates to its maximum value.

4.7.2 Digital System with 3^{rd}-order controller

This subsection was also divided into the same simulations as in the case for the PID controller. First of all, the system is going to be simulated using the impulse response. The most interesting thing to appreciate during this simulation is to see how the block in charge of adapting the output of the filter into the input of the DPWM is working. Last simulation was considering that the error voltage into the system is constant, for example this would be the case when the power supply of the converter is not still activated. For this case, the internal registers of the digital system should not have wrap problems. That is, the PWM patter should not vary from minimum to maximum.

Figure 4.21a shows how the digital system is working when an impulse is introduced into the system. The impulse response (signal labelled as 1) of the system matches with the impulse response of Figure 2.21b. The block *Controller to DPWM transition* is adapting the input to the DPWM block generator (signal 2). It can be seen that when the output of the filter is negative, the value introduced to the DPWM has the minimum duty cycle (signal 3) that the driver can accept, d_{min}.

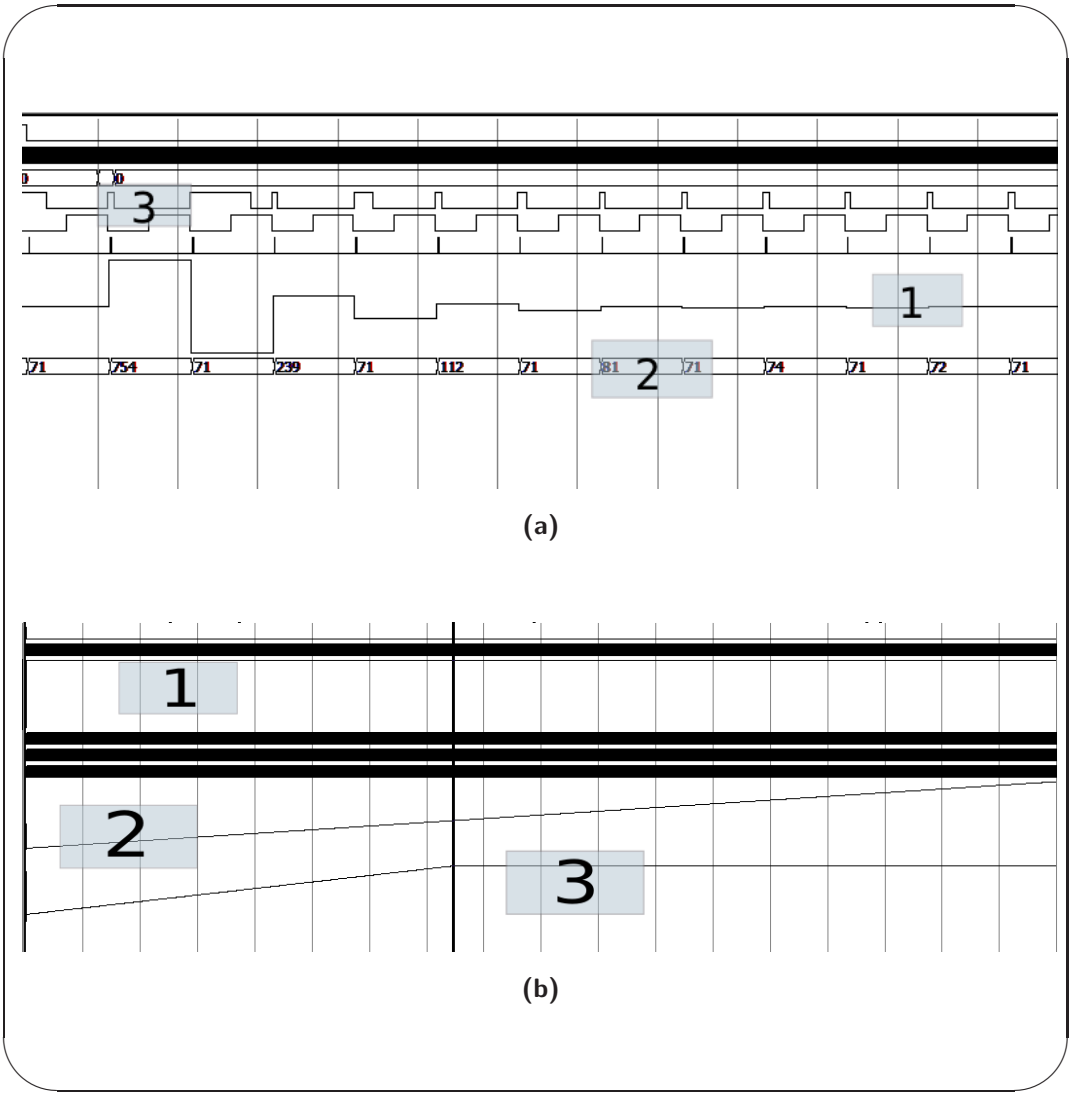
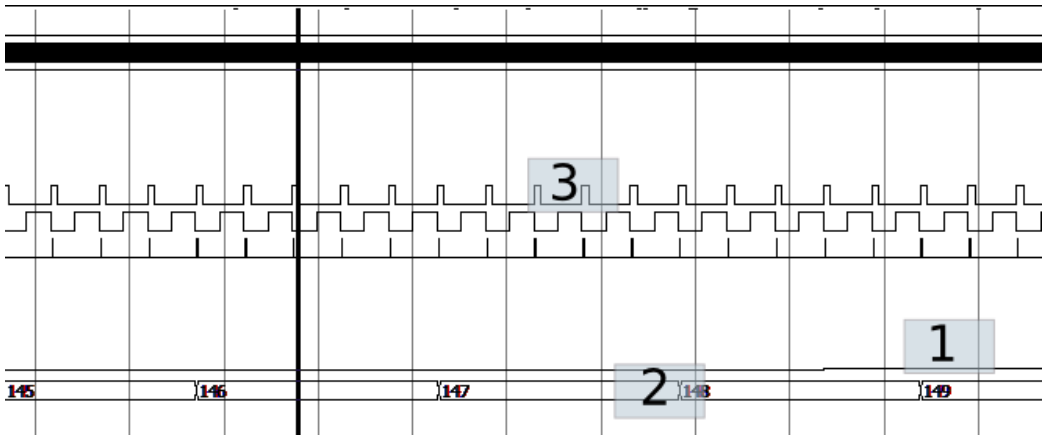
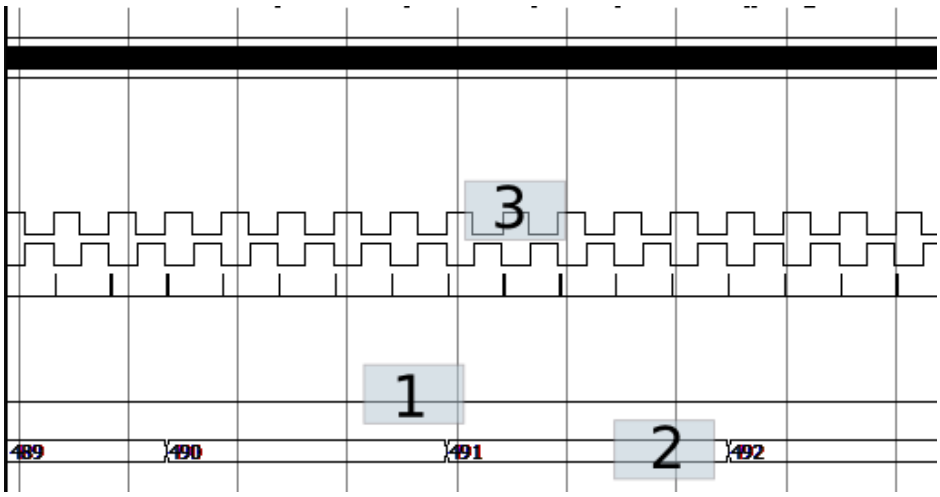


Figure 4.21: ModelSim simulation of the 3^{rd}-order controller. (a) shows the impulse response of the 3^{rd}-order controller. Figure (b) shows the pulse response (signal 1) to the system. The value of the filter (signal 2) increases indefinetely because the filter owns an integrative effect of the input. Nevertheless, the block *Controller to DPWM transition* (signal 3) can not increase indefinetely. As it can be seen in the Figure (b), the duty cycle increases until, d_{max}, which is the maximum duty cycle accepted by the driver of the synchronous buck.

Figures 4.22a and 4.22b represent different moments of the pulse response simulation. It can be seen in both figures how the value introduced in the DPWM evolves from the minimum, d_{min}, to the maximum duty cycle, d_{max}. The simulation also shows how the pulse width of the PWM signal evolves.



(a) Beggining of the simulation



(b) Middle instant of the simulation

Figure 4.22: Different instants of the pulse response simulation. 1) Pulse response 2) Input to DPWM 3) PWM pattern

4.8 Conclusion

This chapter was focussed on the design of a digital system that controls the output voltage of a Buck Converter. The target of this design was to program, using VHDL, a modular architecture. The advantages of using modular architectures is that in case there are future modifications in the

system, the architecture has minimum modifications. For example, if the need of replacing the ADC is arisen, our modifications will be only focussed on the redesign of the acquisition module in the system.
Another strategy, that was considered in the design of the digital architecture, was the design of the whole architecture system as a generic architecture. This advantage allow the designer to program, *a priori*, every block in the digital architecture without the need of knowing many specifications and contraints in the analog/digital system. What should be understood about specifications is the list of the following elements:

1. Number of bits of the ADC
2. Number of bits for the DPWM generation block
3. Degree of complexity of the linear controller (Number of poles)
4. Number of bits required for the coefficients
5. Number of bits required for the signal
6. Number of bits required for the rational part of signal/coefficients

The first two specifications corresponds to problems related to the selection of the hardware. The last constrainsts corresponds to problems analyzed in Chapter 3. The degree of complexity of the linear controller is not a problem either. The use of parallel structures and the construction of generic digital architecture of a residue branch allow the possibility to program the all type of IIR filters. Finally, when all specifications are known, the synthesis of the whole architecture only needs the assignement of concrete values to this generic values. This strategy allows to obtain different hardware, just making minimum modificiations in the source files.

An improvement of this chapter consisted of the possibility to increase the accuracy of the DPWM using a modified fast clock counter structure. The maximum inner frequency for the FPGA was $512MHz$, [24]. The modification of the counter was key to obtain 10-bit DPWM blocks inside of the FPGA. Figure 4.17 shows experimental results of this improvement in a logic analyzer.

Finally, simulations of the whole digital architecture, showed that the block *Controller to DPWM Architecture* was essential to adapt the and perform the comparison effect of a reference voltage with an unipolar sawtooth waveform (see Subsection 2.5.1).

Chapter 5

Experimental Results

5.1 Introduction

This section shows the behaviour of the output voltage for the Buck converter, comparing the implementation for the two controllers previously analyzed in Section 2.4.2. The behaviour of the complete control system was tested under several conditions. The electronic system, composed by the Buck converter (Table 2.1), ADC, controller and DPWM was analyzed in Section 2. First of all, the output voltage was seen under the influence of two static load currents. The static tests are intended to show that the output voltage ripple should not exceed the levels defined in [6]. Finally, the second type of tests are meant to show the performance of the regulation when the output current is varying. The purpose of the dynamical tests must show the effect of the overshoot and settling time in the output voltage.

The following table summarizes the characteristics of the tests performed in the laboratory.

Table 5.1: Summary of performed tests

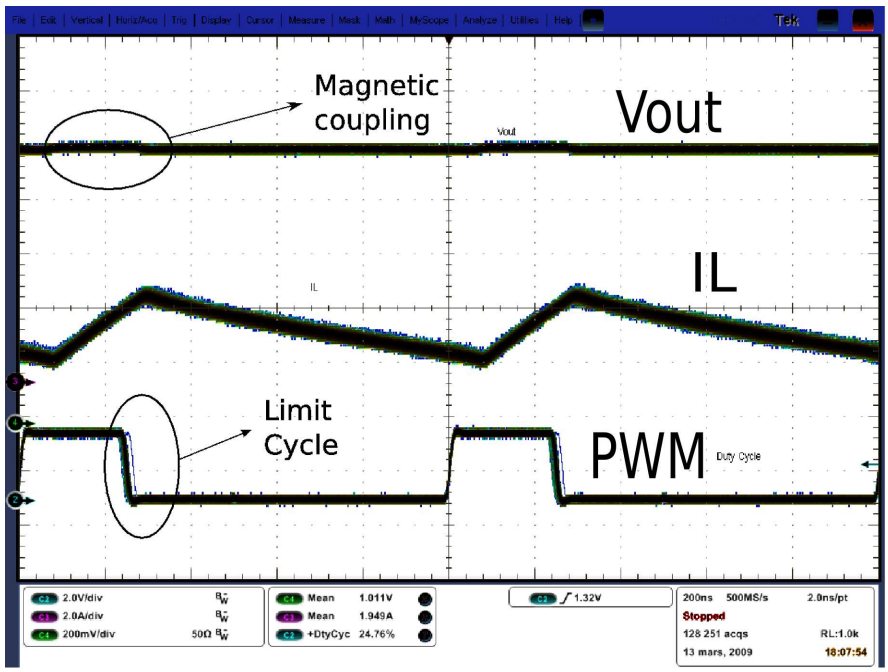
Type of test	Static 1	Static 2	Dynamic 1	Dynamic 2
Input Voltage [V]	5	5	5	5
Mean Static Current [A]	4	8	-	-
Mean Dynamic Current [A]	-	-	4-8	13-19
Output current period [ms]	-	-	100	100
Current Slew Rate $\left[\frac{mA}{\mu s}\right]$	-	-	20	20

The results of the test were displayed using the Fast Acquisition option of the oscilloscope. This mode is a method for accumulating and displaying digital waveform data, representing the behavior over time of an electrical signal. Therefore, using this option can help for the detection of limit-cycles or instabilities.

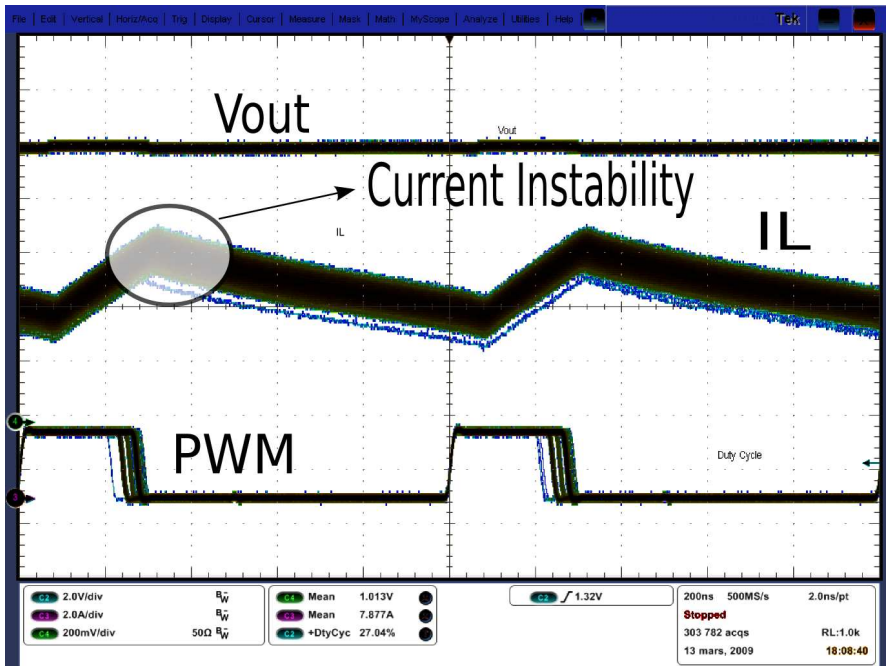
5.2 Results for the PID controller

Figures 5.1a and 5.1b show the results of the static tests using a PID controller. In Figure 5.1a, the test shows that the PID is completely stable. Nevertheless, it can be seen that the duty cycle presents a duty-cycle problem due to the fact that the duty cycle could only be generated using maximum 10 bits in a FPGA.
The magnetic coupling, highlighted in Figure 5.1a, also appeares in the figures of this chapter. The magnetic coupling is an EMC problem coming from the magnetic coupling between two magnetic circuits. The two coupled elements are the inductor of the Buck converter and the oscilloscope probe. The oscilloscope probe and the oscilloscope ground form a big magnetic antenna. The suggested solutions for this magnetic coupling are two. First of all, the isolation of the magnetic source, in this case the inductor, using a magnetic shield is the first solution. Magnetic shields for the case of magnetic circuits consists of the use of ferrite materials whose magnetic permeability is high in order to concentrate all the magnetic field. The second solution is to use a transformer in the oscilloscope to isolate the device and its own ground. From the point of view of electrical safety, the second solution should be avoided.
The origin of the unstability problems, that can be seen in some of the figures, is because of the design. The design of the controller was made for large currents. The transfer function of our plant is in Equation (2.35), whose parameter R models the output current. For high currents the resistor owns a low value. On the other hand, when the load is having low currents the resistor value is high. Thus, this parameter is not constant in our real application. Both controllers were designated to have a large stability margin for high currents. For the case of the 3^{rd}-order controller (Figure 5.4a), the output voltage has a low stability margin. Thus, the output voltage and the inductance current are oscillating. For the case of the PID controller, Figure 5.1b, the previous conclusion seems not to match. Another problem in our state-space averaging model is that parastic elements in the circuit were not considered. This inaccuracy between the real and the state-space model did not consider all the poles and zeros that the real system owns. In order to obtain the real frequency Bode plot of the Buck converter, we should measure the Buck converter using a frequency analyzer. Thus, the designed controllers were not optimal for the real Buck converter.

Figures 5.2a and 5.2b show the results of the dynamic tests using a PID controller. The output voltage in the Buck converter decreases during the load transient. This change is compensated by the regulator effect of the PID. We can see that the width of the overshoot is very short because the multiplicative and integrative branches in the PID are multiplied by coefficients having a large magnitude.

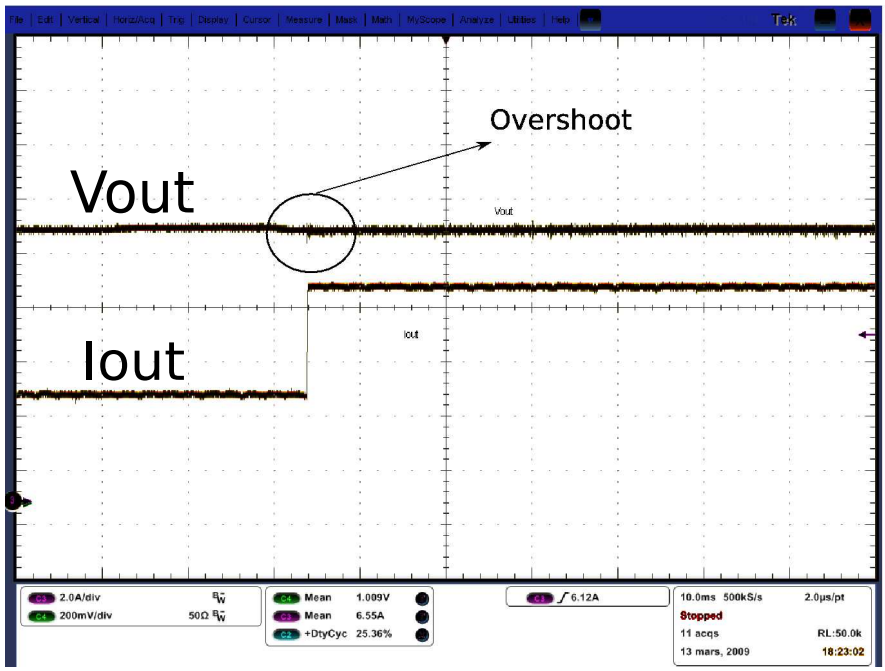


(a)

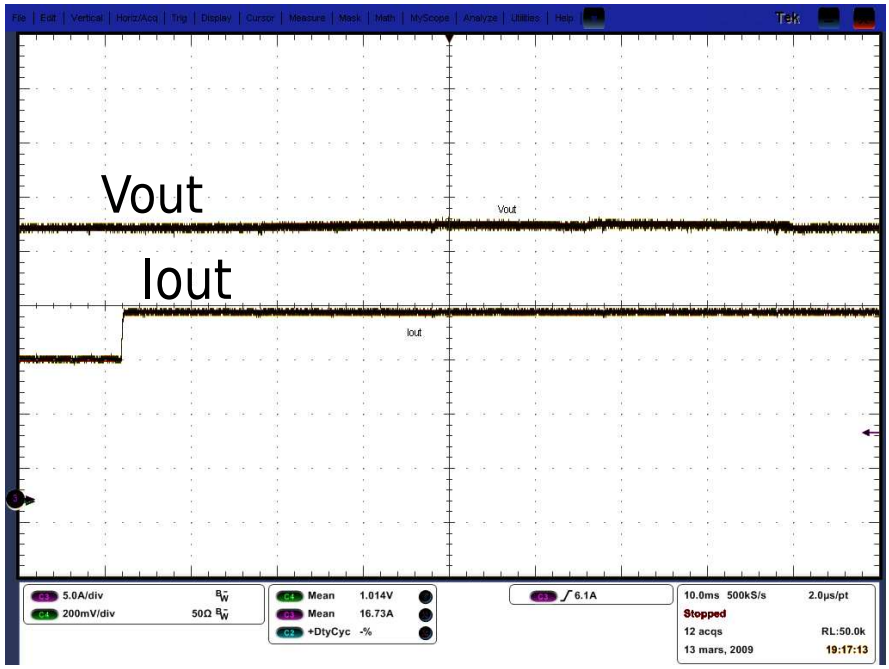


(b)

Figure 5.1: Static tests using PID controller. (a) shows the results for the static test 1. (b) shows the results for the static test 2. Voltage scale is $2\frac{V}{div}$. Time scale is $200\frac{ns}{div}$.



(a)



(b)

Figure 5.2: Dynamic tests using PID controller. (a) shows the results for the dynamic test 1. (b) shows the results for the dynamic test 2. Voltage scale is $2\frac{V}{div}$. Time scale is $200\frac{ns}{div}$.

5.3 Results for the 3^{rd}-order filter

Figures 5.4a and 5.4b show the results of the static tests using a third-order filter. It can be seen that the filter presents some unstable behaviour when the current is low. This unstability is explained because the third-order filter was calculated assuming high load currents in the system. On the other hand, when the output current has a big magnitude, the system is free of any unstable behaviour or limit-cycle. The reason of the lack of limit-cycles was explained in Section 3.4.5.

Figures 5.2a and 5.2b show the results of the dynamic tests using a third-order filter. The overshoot in the output voltage takes more time to be corrected than in the PID counterpart. The residue associated with the integrative fraction, see Equation (2.80), is a very small value ($a_0 = 0.001493$). Thus, whenever the error signal is not equal to zero, the linear controller should correct its ouput until the error signal is again equal to zero. Comparing the dynamic Figures 5.5a and 5.2a, we can see that the settling time is higher for the 3^{rd}-order controller. Comparing the integrative coefficients for both the PID and 3^{rd}-order controller, we can see that the magnitude is directly related to the settling time. For the PID case, the coefficient was $a_0 = 1.5127$ (see Equation (2.77)).
Finally, the reason why there is an undershoot or overshoot in the output voltage when the output current is changed will be discussed. The origin of this change is found to be in the output capacitor of the Buck converter. The current/voltage relationship of a capacitor shows that there is a differential relation. Thus, a change in the current also carries a change in the voltage. Since the output voltage of the Buck converter is related to the output capacitor (see Equation (2.12)), the output voltage changes proportionally to the capacitor voltage.

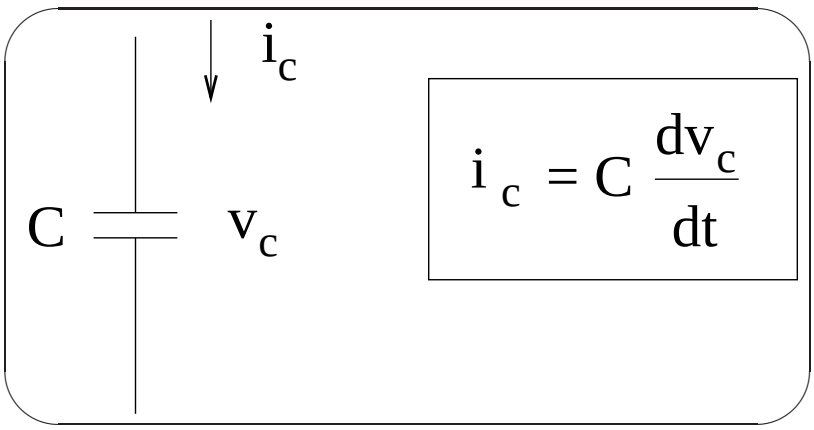
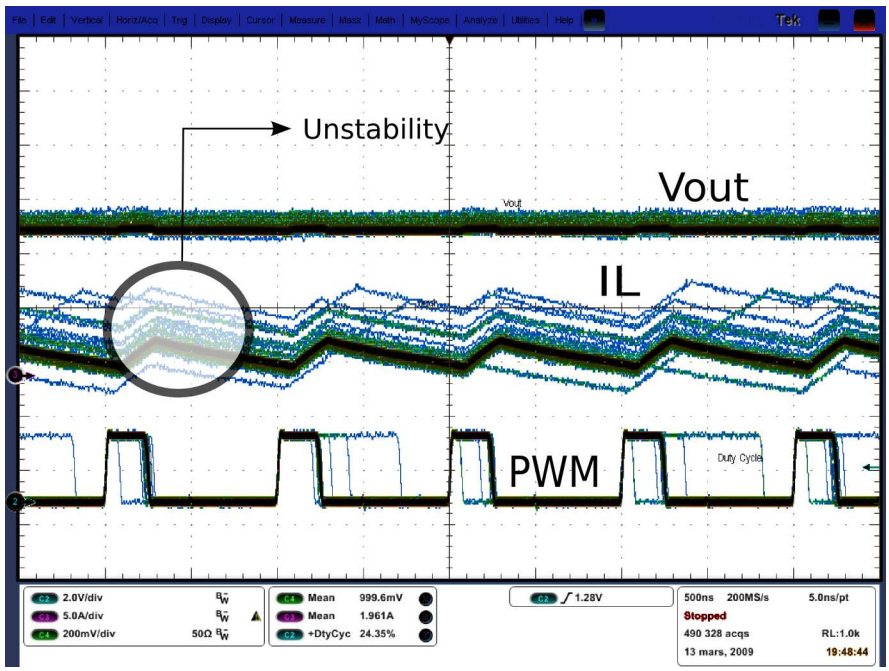
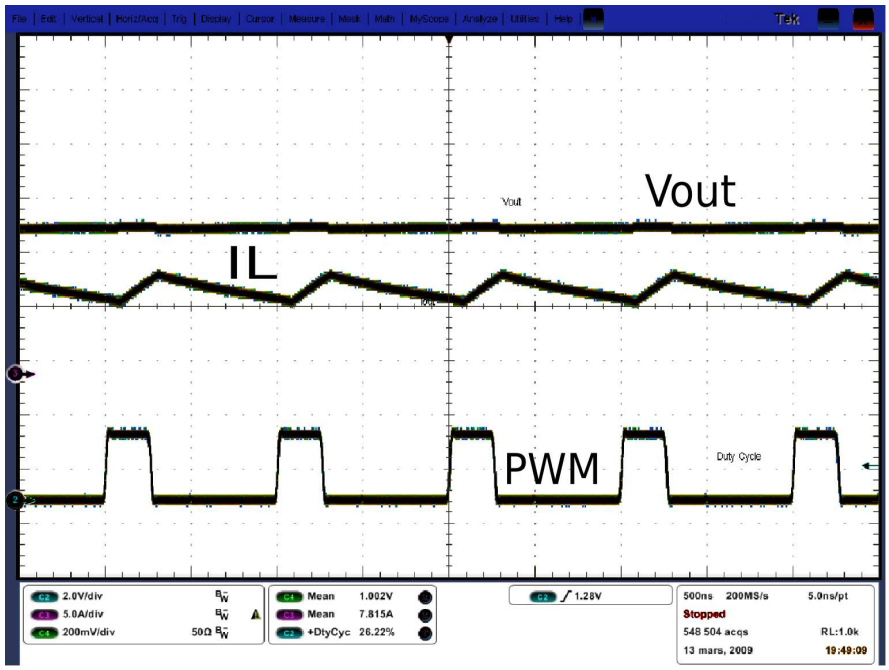


Figure 5.3: Voltage/Current relation in a capacitor

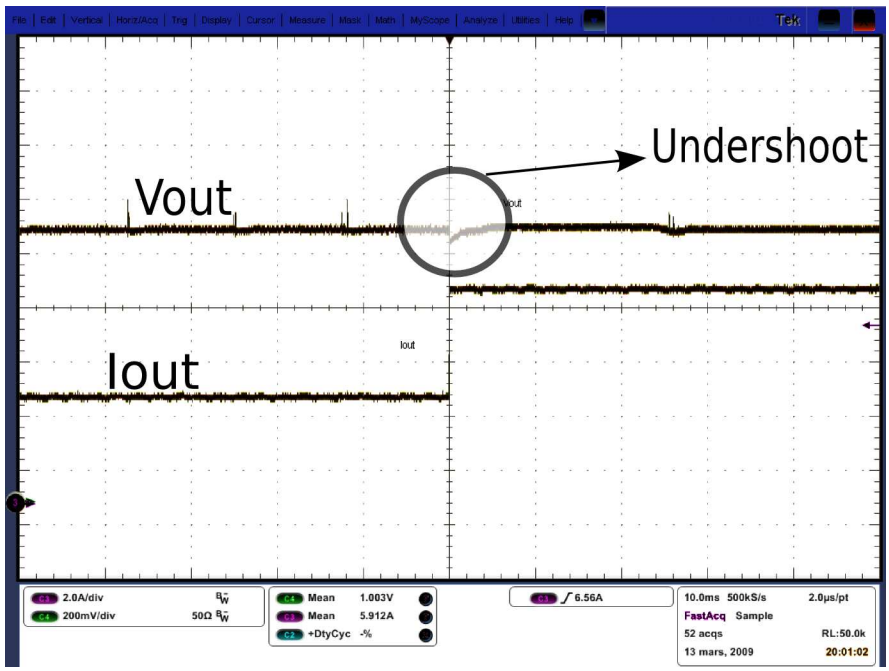


(a)

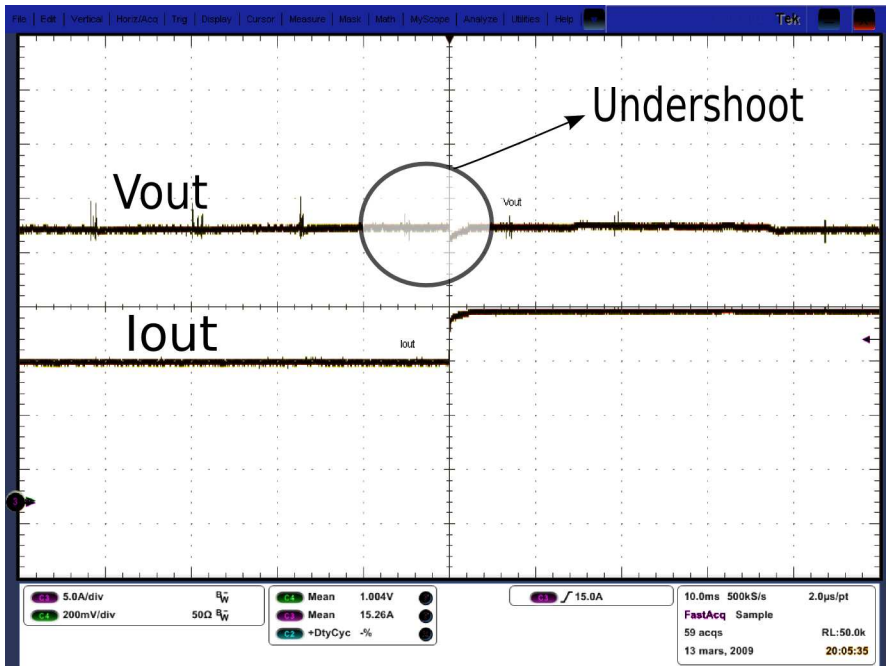


(b)

Figure 5.4: Static tests using 3^{rd}-order controller. (a) shows the results for the static test 1. (b) shows the results for the static test 2. Voltage scale is $2\frac{V}{div}$. Time scale is $200\frac{ns}{div}$.



(a)



(b)

Figure 5.5: Dynamic tests using 3^{rd}-order controller. (a) shows the results for the dynamic test 1. (b) shows the results for the dynamic test 2. Voltage scale is $2\frac{V}{div}$. Time scale is $200\frac{ns}{div}$.

Chapter 6

Conclusions and future perspective

The last chapter tries to summarize the conclusions given along this thesis and suggest a future work perspective.

Firstly, in the second chapter, all the models of a single-phase Buck converter were introduced in order to model the behaviour of our plant. The first conclusions that might be drawn after the simulations are the innacuracy of the output and state signals for frequencies higher than $\frac{f_{sw}}{2}$. Later on, the analytical expressions to calculate the spectrum of a single-phase and multiphase Buck converter were extracted. The expression in Equation (2.45), showed that the input current of a multiphase Buck converter is the same as the single-phase counterpart with the multiplication of the *Form-factor function*. The previous function makes the input current of a N-phase converter be equivalent to the a single-phase converter one, but switching at $N \cdot f_{sw}$ in terms of spectrum distribution. This characterization was useful to calculate the value of the input capacitance in a multiphase Buck converter. The second half of the chapter discusses all the concepts regarding to the implementation of digital linear control. The chapter ends with the analysis of a novel technique to generate a PWM waveform without the need of a DAC converter. Therefore, the electronic system has the advantage of being more compact.

A prospective work, related to the dynamic systems field, consists of the derivation and simulation of a non-linear models for switching converters. After the analysis of non-linear models, it might be also interesting to perform some simulations using the linear controllers presented in Section 2.4.2. The latter comparison with both Buck converter models should help to find a better control strategy. As a matter of fact, the so-called "better control strategy" should not constraint this time to the idea of the implementation

of linear controllers as it was in this Master's Thesis.

The third chapter was focussed on the numerical accuracy, stability analysis of linear controllers when fixed-point system are used to represent the set of rational numbers. Hardware using fixed-point arithmetic circuits have the advantages of being faster, smaller in terms of sillicon size and energetically more efficient. After the selection of the numerical representation, the chapter tries to solve the question of how many bits are necessary in order to respect the frequency properties of the filter. The next problem to be solved was the quantization of input/output signals of the controller. To calculate the number of bits for the signal is also an essential and very important task. The main question were the number of bits required to avoid overflow and the number of bits in order to avoid the effect of truncation are propagated in the DPWM block.

A future work in this chapter could be the analysis and desgin of linear controllers using the δ-operator instead of the shift operator (z^{-1}). The delta operator, as it is pointed out in [27- 28], uses the Euler approximation to simulate an analogue derivative. Thus, any digital linear system described by means of the δ-operator converges to a differential equation when the sampling time of the digital system tends to zero. One of the advantages of using delta operator is the direct translation of the analogue linear controllers to the FPGA since both behavioral equations are equivalent. Finally, the last analysis should be the calculation of number of bits for both coefficient and signal using the delta operator.

The fourth chapter uses all the metrics analyzed in the third chapter in order to program the linear controller. The chapter is also analyzing the best strategies to build the controller system given in Figure 2.1b. The main drawback of FPGAs is the impossibility of performing very difficult digital circuits due to the non-constant delay among its components. Therefore, the best solution was to implement a synchronous system driven by a general clock. The design of a synchronous system constrained the design of all blocks by dividing them into two parts. The first part is in charge to implement the functionality for the control system (*arithmetic circuit*). Oppositely, the second part corresponds to the implementation of a state machine activating and storing the results coming from the arithmetic circuit. Therefore, every block tries to copy the concept behind the harvard computer architecture where there are independent data and signal paths. Finally, another strategy to quickly program state machines using the generic approach that *VHDL* offers, is the synthesis of ring counters as a replacement of the classical Mealy/Moore Finite State Machines.

Some future work, that could be developped, is the behavioral description of the Buck converter in order to simulate the response of the complete control system. The main goal of this approach is to compare both the VHDL and *Simulink* output voltage simulations. The complete simulation of the control system using VHDL help us to verify the behaviour of our complete electronic system using the HDL synthetizable code for the FPGA. If a more relialable simulation is required, the AMS (*Analog Mixed Signal*) standard library can be used to program the Buck converter using continous differential equations. The VHDL-AMS is a derivative of the hardware description language VHDL (IEEE standard 1076-1993). A last implementation before testing the real application is the development of synthetizable descriptions of the Buck converter and to plot the results in a debugging tool. This approach could allow to verify inside of the FPGA the behaviour of our digital controller and see that there are no timing problems inside of the FPGA. Finally, the last future work proposal related to Chapter 4 is the construction of a generic library for the implementation of filter structures using the δ-operator. This block would replace the normal serial register intended to program filter structures using serial registers (z variable). This option should only be used in case the δ-operator is used for the linear controller.

Finally the fifth chapter shows that after modelling (Chapter 1) the control system, design the controller (Chapter 1), translating the algorithms into fixed-point number representation (Chapter 3) and simulating our digital system (Chapter 4) the Buck converter is able to control the output voltage. The comparison of the Simulink simulations given in [3] and our work are able to validate our methodology.

Bibliography

[1] B. Ross. IBIS and ICEM interaction. *Microelectronics Journal*, 35(6):497–500, 2004.

[2] Katshuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 4th edition edition, 2001.

[3] Adan Simon Muela. *Architectures of High-Current, Low-Voltage Digitally-Controlled DC/DC Power Converters*. PhD thesis, Universite Toulouse III - Paul Sabatier, November 2008.

[4] Hwei P. Hsu. *Analisis de Fourier*. Prentice Hall, 1984.

[5] National Semiconductor. *ADC08060*, February 8 2008.

[6] Intel Corporation. *Voltage Regulator-Down 11.0, Processor Power Deliveries Design Guidelines - For Desktop LGA775*, November 2006.

[7] Karl Johan Astrom and Tore Hägglund. *PID Controllers: [theory, Design, and Tuning]*. The International Society for Measurement and Control, 1995.

[8] Murray Spiegel. *Schaum's Outline of Laplace Transforms*. McGraw-Hill, June 1965.

[9] AP Dancy, R. Amirtharajah, AP Chandrakasan, and H. Synqor. High-efficiency multiple-output DC-DC conversion for low-voltagesystems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):252–263, 2000.

[10] E. O'Malley and K. Rinne. A programmable digital pulse width modulator providing versatile pulse patterns and supporting switching frequencies beyond 15 MHz. In *Applied Power Electronics Conference and Exposition, 2004. APEC'04. Nineteenth Annual IEEE*, volume 1, 2004.

[11] A. Prodic and D. Maksimovic. Digital PWM controller and current estimator for a low-powerswitching converter. In *Computers in Power Electronics, 2000. COMPEL 2000. The 7th Workshop on*, pages 123–128, 2000.

[12] A. Prodic, D. Maksimovic, and RW Erickson. Design and implementation of a digital PWM controller for ahigh-frequency switching DC-DC power converter. In *Industrial Electronics Society, 2001. IECON'01. The 27th Annual Conference of the IEEE*, volume 2, 2001.

[13] E. Fossas and G. Olivar. Study of chaos in the buck converter. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 43(1):13–25, 1996.

[14] L. Harnefors. Implementation of resonant controllers and filters in fixed-point arithmetic. *IEEE Transactions on*, pages 1–1, 2003. Accepted for future publication Industrial Electronics.

[15] M.B. Naumovic and M.R. Stojic. Comparative study of finite word length effects in digital filter design via the shift and delta transforms. In *Proc. 4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services*, volume 1, pages 141–144 vol.1, 1999.

[16] Richard Wesley Hamming. *Numerical Methods for Scientists and Engineers*. McGraw-Hill, 1962.

[17] Nicholas J. Higham. *Accuracy and Stability of NUmerical Algorithms*. SIAM, 2nd edition edition, August 2002.

[18] Alan V. Oppenheim and Ronald W. Schafer. *Tratamiento de Señales en Tiempo Discreto*. Prentice Hall, 2nd edition edition, 1999.

[19] Zhengwei Fang, J.E. Carletta, and R.J. Veillette. A methodology for fpga-based control implementation. 13(6):977–987, 2005.

[20] Michael Brin and Garrett Stuck. *Introduction to Dynamical Systems*. Cambridge University Press, 2002.

[21] W.R. Bennett. Spectra of quantized signals. *Nonlinear Systems: Processing of Random Signals, Classical Analysis*, page 217, 1975.

[22] B. Widrow. Statistical analysis of amplitude quantized sampled-data systems. *Trans. AIEE, Part. II: Applications and Industry*, 79:555–568, 1960.

[23] B. Widrow. A study of rough amplitude quantization by means of Nyquist sampling theory. *Circuit Theory, IRE Transactions on*, 3(4):266–276, 1956.

[24] Xilinx. *Virtex-5 FPGA Data Sheet: DC and Switching Characteristics*, 5.1 edition, January 2009.

[25] R.E. Crochiere and A.V. Oppenheim. Analysis of linear digital networks. 63(4):581–595, 1975.

[26] IRF. *iP2003*, November 2004.

[27] Richard H. Middleton and Graham Clifford Goodwin. *Digital control and estimation*. Prentice Hall, 1990.

[28] R. Middleton and G. Goodwin. Improved finite word length characteristics in digital control using delta operators. 31(11):1015–1021, 1986.

Appendix A

Residue theorem

Definition A.0.0.1. *If* $f(z) = \sum_{\infty}^{\infty} c_k (z - z_0)^k$ *in a neighbourhood of* z_0, C_{-1} *is called the* residue *of* f *at* z_0. *We use the notation* $c_{-1} = Res(f; z_0)$

If f has a simple pole at z_0*; i.e., if*

$$f(z) = \frac{A(z)}{B(z)} \tag{A.1}$$

where A and B are analytic at z_0 *and B has a simple zero at* z_0*, then*

$$C_{-1} = \lim_{z \to z_0} (z - z_0) f(z) = \frac{A(z_0)}{B'(z_0)} \tag{A.2}$$

$$f(z) = \frac{A(z)}{B(z)} \tag{A.3}$$

Since

$$\begin{aligned} f(z) =& \frac{c_{-1}}{z - z_0} + c_0 + c_1 (z - z_0) + ... \\ (z - z_0) =& c_{-1} + c_0 (z - z_0) + c_1 (z - z_0)^2 + ... \end{aligned} \tag{A.4}$$

and

$$\lim_{z \to z_0} f(z) = C_{-1} \tag{A.5}$$

The second equality in A.2 follows since

$$\lim_{z \to z_0} (z - z_0) f(z) = \lim_{z \to z_0} (z - z_0) \frac{A(z)}{B(z)} \tag{A.6}$$

$$= \lim_{z \to z_0} \frac{A(z)}{\frac{B(z) - B(z_0)}{z - z_0}} = \frac{A(z_0)}{B'(z_0)} \tag{A.7}$$

The same principle can be applied for functions having more than a simple pole. Applying A.2 to every pole of a rational function, we can decompose the function calculating the residue for every pole in the function.

Appendix B

Demonstration of the state-space representation of a Buck converter including parastic resistances

The state-space averaging model of a Buck converter including all the parastic resistances can be found using the same methodology as in Section 2.2.1.1. First of all, the state-space equations for both states, *on* and *off* must be found.

B.1 On-state

According to Figure B.1a, a set of two equations must be written using KVL and KCL. The state variables are as in the previous case, $x = (i_L, v_c)^T$

$$\begin{aligned} V_g &= L \cdot \frac{di_L}{dt} + R_{inductance} \cdot i_L - v_{out} \\ C \cdot \frac{dv_c}{dt} &= i_L - i_{out} \end{aligned} \tag{B.1}$$

The main difference among this model and the lossless model is that the output voltage is not equal to the capacitance voltage. The Equation (B.1) is having three unknowns and a couple equations. Thus, it is necessary to find a third equation in order to solve the system of equations. The last equation can be found using the superposition principle to see the effect that every state variable is having in the output voltage.

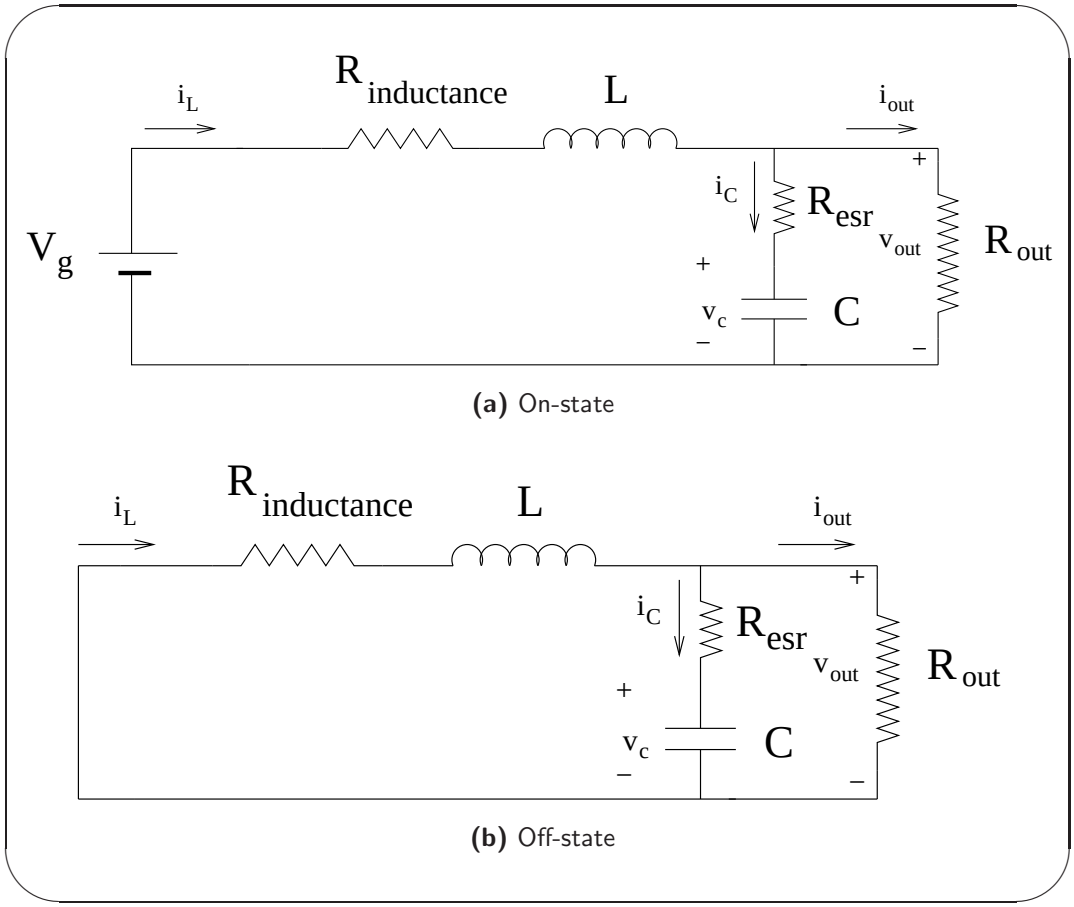


Figure B.1: Buck Converter including parastic resistances

$$v_{out} = \underbrace{i_L \cdot \underbrace{\frac{R_{out}R_{esr}}{R_{out}+R_{esr}}}_{Current\ divider}}_{Short-circuiting\ capacitance} + \underbrace{v_c \cdot \underbrace{\frac{R_{out}}{R_{out}+R_{esr}}}_{voltagedivider}}_{open-circuitinductance} \tag{B.2}$$

The output current can also be found using the relationship

$$i_{out} = \frac{v_{out}}{R_{out}} = \frac{R_{esr}}{R_{out}+R_{esr}} + \frac{1}{R_{out}+R_{esr}} \tag{B.3}$$

Using Equations (B.2) and (B.3), the output vector, $\mathbf{y} = (v_{out}, i_{out})^T$ can be written

$$\mathbf{y} = \mathbf{C} \cdot \mathbf{x} = \begin{pmatrix} \frac{R_{out}R_{esr}}{R_{out}+R_{esr}} & \frac{R_{out}}{R_{out}+R_{esr}} \\ \frac{R_{esr}}{R_{out}+R_{esr}} & \frac{1}{R_{out}+R_{esr}} \end{pmatrix} \cdot \mathbf{x} \tag{B.4}$$

Finally substituing Equation (B.2) into (B.1), we obtain the last equation to represent the state space equation of the system in Figure B.1

$$\begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \mathbf{x}' = \begin{pmatrix} -\left(R_{inductance} + \frac{R_{out}R_{esr}}{R_{out}+R_{esr}}\right) & -\frac{R_{out}}{R_{out}+R_{esr}} \\ -\frac{R_{out}}{R_{out}+R_{esr}} & \frac{1}{R_{out}+R_{esr}} \end{pmatrix} \mathbf{x} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \mathbf{u} \tag{B.5}$$

being the $\mathbf{u} = (V_g, I_g)^T$

B.2 Off-state

For the off-state, the derivation of the state-space model is exactly the same as in the *On-State*. Nevertheless, the main difference in the equations is on the input vector, $\mathbf{u}$. In Figure B.2, it is seen that any voltage source is connected to the Buck converter during this period. Thus, the input voltage vector must be equal to zero. The state-space equations are now

$$\begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \mathbf{x}' = \begin{pmatrix} -\left(R_{inductance} + \frac{R_{out}R_{esr}}{R_{out}+R_{esr}}\right) & -\frac{R_{out}}{R_{out}+R_{esr}} \\ -\frac{R_{out}}{R_{out}+R_{esr}} & \frac{1}{R_{out}+R_{esr}} \end{pmatrix} \mathbf{x} \tag{B.6}$$

$$\mathbf{y} = \mathbf{C} \cdot \mathbf{x} = \begin{pmatrix} \frac{R_{out}R_{esr}}{R_{out}+R_{esr}} & \frac{R_{out}}{R_{out}+R_{esr}} \\ \frac{R_{esr}}{R_{out}+R_{esr}} & \frac{1}{R_{out}+R_{esr}} \end{pmatrix} \cdot \mathbf{x} \tag{B.7}$$

B.3 State-space averaging

Finally, the application of the state-space averaging technique, used in Equation (2.11), is useful to linearize the dynamic system. The final equation corresponds to the same state-space representation as in Equation (2.18).

$$\begin{cases} \begin{pmatrix} L & 0 \\ 0 & C \end{pmatrix} \mathbf{x}' = \begin{pmatrix} -\left(R_{inductance} + \frac{R_{out}R_{esr}}{R_{out}+R_{esr}}\right) & -\frac{R_{out}}{R_{out}+R_{esr}} \\ -\frac{R_{out}}{R_{out}+R_{esr}} & \frac{1}{R_{out}+R_{esr}} \end{pmatrix} \mathbf{x} + \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} \mathbf{u} \\ \mathbf{y} = \begin{pmatrix} \frac{R_{out}R_{esr}}{R_{out}+R_{esr}} & \frac{R_{out}}{R_{out}+R_{esr}} \\ \frac{R_{esr}}{R_{out}+R_{esr}} & \frac{1}{R_{out}+R_{esr}} \end{pmatrix} \mathbf{x} \end{cases} \tag{B.8}$$

Appendix C

Input current modelling script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Serie de Fourier discreto mediante la formula
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear ;
clc;

Imin=10;
Imax=20;
n=[1:25];
D=[0.3 0.25 0.2 0.10 0.08];
coeficiente=(Imax-Imin);

dc=(coeficiente/2)*D;

for a=1:length(D),
    argumento(a,:)=2*pi*n*D(a)*i;
end

for a=1:length(D),
    for b=0:length(n),
        if b==0,
            seriefourier(a, b+1)=dc(a);
        else
            seriefourier(a,b+1)=(coeficiente/D(a))*((1+argumento(b))*
                exp(argumento(b))-1)./(2*pi*n(b))^2;
        end
    end
end

for a=1:length(D),
    for b=1:length(n)+1
        fouriermagnitud(a,b)=abs(seriefourier(a,b));
    end
end

```

```
37
38 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39 % Representacion de las funciones
40 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41
42 n=[0:25];
43 figure(1)
44 hold on;
45 h1=stem(n, fouriermagnitud(1,:), 'g')
46 h2=stem(n, fouriermagnitud(3,:), 'r')
47 h3=stem(n, fouriermagnitud(4,:), 'b')
48 h4=stem(n, fouriermagnitud(5,:), 'c')
49 set(h1, 'LineWidth', 3)
50 set(h2, 'LineWidth', 3)
51 set(h3, 'LineWidth', 3)
52 set(h4, 'LineWidth', 3)
53 title('Input Current Spectrum of several duty cycles');
54 legend('D=0.3', 'D=0.2', 'D=0.1', 'D=0.08');
55 xlabel('Discrete Frequency [n \cdot f_s]');
56 grid on;
57 ylabel('Magnitude of the fourier coefficient [A]')
58
59 figure(2);
60
61 h5=stem3(n, D, fouriermagnitud);
62 title('Input Current Spectrum of several duty cycles');
63 ylabel('DutyCycle');
64 xlabel('Discrete Frequency [n \cdot f_s]');
65 zlabel('Magnitude of the fourier coefficient [A]')
66
67
68 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
69 % Definicion sinc digital
70 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
71
72
73 sinc=sin(n*pi)./sin(n*pi/4);
74 figure(3);
75 h6=stem(n,abs(sinc));
76 set(h6, 'LineWidth', 3)
77 title('4-phase filter form factor function');
78 xlabel('Discrete frequency [n \cdot f_s]');
79 ylabel('Amplitude')
80
81 for b = 1:length(D),
82     for a=1:length(n),
83         if a==1,
84             multifase(b, a)=4*fouriermagnitud(b,a);
85         else
86             multifase(b,a)=abs(sinc(a)*fouriermagnitud(b,a));
87         end
88     end
89 end
```

```
90
91  figure(4);
92  hold on;
93  h10=stem(n, multifase(1,:), 'g')
94  h11=stem(n, multifase(3,:), 'r')
95  h12=stem(n, multifase(4,:), 'b')
96  h13=stem(n, multifase(5,:), 'c')
97  set(h10, 'LineWidth', 3)
98  set(h11, 'LineWidth', 3)
99  set(h12, 'LineWidth', 3)
100 set(h13, 'LineWidth', 3)
101 title('Input current 4 phase system');
102 grid on;
103 legend('D=0.3', 'D=0.2', 'D=0.1', 'D=0.08');
104 xlabel('Discrete Frequency [n \cdot f_s]');
105 ylabel('Magnitude of the fourier coefficient [A]')
106
107 figure(5);
108 h20=stem3(n, D, multifase);
109 set(h20, 'LineWidth' , 3)
110 title('Input Current Spectrum of several duty cycles');
111 ylabel('DutyCycle');
112 xlabel('Discrete Frequency [n \cdot f_s]');
113 zlabel('Magnitude of the fourier coefficient')
```

Appendix D

Coefficient wordlength script

```
%% Calculo del numero de bits de la parte decimal

ceros = [0.99391288222701 0.96399771431853];
polos = [1 0.51066421884152 -0.5];

num_bits_ceros = calculobits2(ceros, 0.01);
num_bits_polos = calculobits2(polos, 0.01);

if (max(num_bits_ceros) > max(num_bits_polos)),
    num_bits_decimal = floor(max(num_bits_ceros))+1;
else
    num_bits_decimal = floor(max(num_bits_polos))+1;
end

%% Calculo del numero de bits de la parte entera

max_ceros = max(abs(ceros));
max_polos = max(abs(polos));

if (max_ceros > max_polos),
    max_valor = max_ceros;
else
    max_valor = max_polos;
end

num_bits_entero = floor(log2(max_valor))+1;


%% Valores discretizados

Controlador = zpk(ceros, polos, 6.5, 1e-6);
Controlador = tf(Controlador);
[numerador, denominador] = tfdata(Controlador);
numerador = cell2mat(numerador);
denominador = cell2mat(denominador);

```

```
for i=1:length(numerador) ,
    temp=dec2bin(round(abs(numerador(i))*2^num_bits_decimal));
    num_cuantizados(i)=sign(numerador(i))*bin2dec(temp)
            /(2^num_bits_decimal);
end

for i=1:length(denominador) ,
    temp=dec2bin(round(abs(denominador(i))*2^num_bits_decimal));
    den_cuantizados(i)=sign(denominador(i))*bin2dec(temp)
                /(2^num_bits_decimal);
end


for i=1:length(numerador) ,
    temp=dec2bin(floor(abs(numerador(i))*2^num_bits_decimal));
    num_cuantizados2(i)=sign(numerador(i))*bin2dec(temp)
                        /(2^num_bits_decimal);
end

for i=1:length(denominador) ,
    temp=dec2bin(floor(abs(denominador(i))*2^num_bits_decimal));
    den_cuantizados2(i)=sign(denominador(i))*bin2dec(temp)
                        /(2^num_bits_decimal);
end

ceros_cuantizados = roots(num_cuantizados);
polos_cuantizados = roots(den_cuantizados);

ceros_cuantizados2 = roots(num_cuantizados2);
polos_cuantizados2 = roots(den_cuantizados2);

%% Definir sistemas lineales y graficar su diagrama de polos/ceros

Controlador_cuantizado = tf(num_cuantizados, den_cuantizados, 1e-6);
Controlador_cuantizado2 = tf(num_cuantizados2, den_cuantizados2, 1e-6);

uno = figure;
bode(Controlador, '-', Controlador_cuantizado, '*',
                  Controlador_cuantizado2, 'o');
grid on;
legend('Infinite precision', 'Rounded Fixed point',
                        'Truncated Fixed Point');
dos = figure;
pzmap(Controlador, Controlador_cuantizado, Controlador_cuantizado2);
grid on;
legend('Infinite precision', 'Rounded Fixed point',
                        'Truncated Fixed Point');

ratio_ceros = max(abs((ceros-ceros_cuantizados')./ceros));
ratio_polos = max(abs((polos-polos_cuantizados')./polos));
```

```
function [bits] = calculobits2 (x, e)
% Calcula la formula indicada en mi cuadernos
% x es el polinomio
% e es el error en porcentaje

matriz_identidad=eye(length(x));


for a=1:length(x),
    for b=1:length(x),
        matriz_vander(a,b)=x(b)^(a-1);
    end
end

for j=1:length(x),
    producto=1;
    for i=1:length(x),
        if(i == j)
            continue;
        else
            producto=producto*(x(i)-x(j));
        end;
    end;
    vector_raices(j)=1/producto;
end

for a=1:length(x),
    for b=1:length(x),
        if(a==b),
            matriz_raices(a,b)=abs(vector_raices(b));
        else
            matriz_raices(a,b)=0;
        end
    end
end

vector_error=(1/e)*ones(length(x),1);
matriz_sensible=matriz_raices*matriz_vander';
matriz_sensible=abs(matriz_sensible);
matriz_sensible=matriz_sensible*vector_error;
bits=log2(matriz_sensible);
```

Appendix E

Script to analyze IIR structures

```
%% Borrar parametros

clear;
clc;

%% Definicion filtro e impulso

indice=[1:25];
w = 18;
d = 13;

b = [5 -9.7895529827277 4.7906487334929];
a = [1 -1.01066421884141 -0.244667890579294 0.255332109420706];

q = length(b);
p = length(a);

for i=1:length(indice) ,
    if (i == 1)
        x(i)=1;
    else
        x(i)=0;
    end
end

for i=1:length(indice)
   vector_dac(i)=2^-10;
end

%% Forma directa I

x_n = zeros(q, 1);
y_n = zeros(p-1, 1);

for i= 1:length(x),
    x_n(2:q) =x_n(1:q-1);
```

```
37      x_n(1)=x(i);
38
39      y_directo(i)=b*x_n-a(2:p)*y_n;
40
41      y_n(2:p-1)=y_n(1:p-2);
42      y_n(1)=y_directo(i);
43  end
44
45  y_directo_float = y_directo;
46  %
47  % % %% Forma directa II
48  %
49  % x_n = zeros(p,1);
50  %
51  % for i=1:length(x),
52  %         x_n(2:q)=x_n(1:q-1);
53  %         x_n(1)= x(i) - a(2:q)*x_n(2:q);
54  %
55  %         y_canonico(i) = b*x_n;
56  % end
57  %
58  % y_canonico_float = y_canonico;
59  % %
60  % % %% Forma en cascada
61  %
62  % b_1 = [1 -1.9464 0.94675724];
63  % a_1 = [1 -0.5 -0.5];
64  %
65  % b_2 = [6.5 0];
66  % a_2 = [1 -0.4125];
67  %
68  % y_intermedio = iircanonico(b_1, a_1, x);
69  %
70  % y_cascada = iircanonico(b_2, a_2, y_intermedio);
71  %
72  % y_intermedio_float=y_intermedio;
73  % y_cascada_float = y_cascada;
74
75  % %% Forma paralelo
76
77  num = [6.5 -12.6516 6.15392206];
78  den = [1 -0.9125 -0.29375 0.20625];
79
80  [raiz,polo,k]=residue(num,den);
81
82  a_11 =[raiz(1) 0];
83  b_11 =[1 -polo(1)];
84
85  a_22 = [raiz(2) 0];
86  b_22 = [1 -polo(2)];
87
88  a_33 = [raiz(3) 0];
89  b_33 = [1 -polo(3)];
```

```
90
91
92  y_paralelo1 = iircanonico(a_11, b_11, x);
93  y_paralelo2 = iircanonico(a_22, b_22, x);
94  y_paralelo3 = iircanonico(a_33, b_33, x);
95
96  y_paralelo = y_paralelo1+y_paralelo2+y_paralelo3;
97
98  y_paralelo1_float = y_paralelo1;
99  y_paralelo2_float = y_paralelo2;
100 y_paralelo3_float = y_paralelo3;
101
102 y_paralelo_float = y_paralelo;
103
104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
105 % Analisis de cada estructura%%%%%%%%%
106 % Mediante la toolbox fixed point%%%%%
107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
108 %% Forma directa I
109
110 f = fimath;
111 f.ProductWordlength = 2*w;
112 f.ProductFractionLength = 2*d;
113 f.ProductMode = 'SpecifyPrecision';
114 f.SumMode = 'SpecifyPrecision';
115 f.SumWordLength = 2*w;
116 f.SumFractionLength = 2*d;
117 f.RoundMode = 'floor';
118
119 b = fi(b, 1, w, d);
120 a = fi(a, 1, w, d);
121
122 x = fi(x, 1, w, d);
123 y_directo = fi(zeros(size(x)), 1, w, d);
124 x_n = fi(zeros(q, 1), 1, w, d);
125 y_n = fi(zeros(p-1, 1), 1, w, d);
126
127 a.fimath = f;
128 b.fimath = f;
129 x.fimath = f;
130 y_directo.fimath = f;
131 x_n.fimath = f;
132 y_n.fimath = f;
133
134 for i= 1:length(x),
135     x_n(2:q) =x_n(1:q-1);
136     x_n(1)=x(i);
137
138     y_directo(i)=b*x_n-a(2:p)*y_n;
139
140     y_n(2:p-1)=y_n(1:p-2);
141     y_n(1)=y_directo(i);
142 end
```

```
143
144 n=[0:length(y_directo)-1];
145 uno = figure
146 uno = plot(n,y_directo_float,'.-b',n,y_directo,'.-r')
147 legend('double','fixed-point')
148 xlabel('n (samples)'); ylabel('amplitude')
149 title('Step response direct I form')
150
151  e = double(lsb(y_directo));
152 dos = figure
153 dos = plot(n,double(y_directo)-y_directo_float,'.-r', ...
154      [n(1) n(end)],[e/2 e/2],'c', ...
155      [n(1) n(end)],[-e/2 -e/2],'c')
156 text(n(end),e/2,'+1/2 LSB','HorizontalAlignment','right',
157         'VerticalAlignment','bottom')
158 text(n(end),-e/2,'-1/2 LSB','HorizontalAlignment','right',
159         'VerticalAlignment','top')
160 xlabel('n (samples)'); ylabel('error')
161 title('Error between floating point and fixed point direct form I');
162 %
163 % % %% Forma directa II
164 %
165 % y_canonico = fi(zeros(size(x)), 1, 2*w, 2*d);
166 % x_n = fi(zeros(p, 1), 1, w, d);
167 %
168 % y_canonico.fimath = f;
169 % x_n.fimath = f;
170 %
171 % for i=1:length(x),
172 %         x_n(2:q)=x_n(1:q-1);
173 %         x_n(1)= x(i) - a(2:q)*x_n(2:q);
174 %
175 %         y_canonico(i) = b*x_n;
176 % end
177 %
178 % n=[0:length(y_canonico)-1];
179 % tres = figure
180 % tres = plot(n,y_canonico_float,'.-b',n,y_canonico,'.-r')
181 % legend('double','fixed-point')
182 % xlabel('n (samples)'); ylabel('amplitude')
183 % title('Step response direct II form')
184 %
185 % cuatro = figure
186 % cuatro = plot(n,double(y_canonico)-y_canonico_float,'.-r', ...
187 %      [n(1) n(end)],[e/2 e/2],'c', ...
188 %      [n(1) n(end)],[-e/2 -e/2],'c')
189 % text(n(end),e/2,'+1/2 LSB','HorizontalAlignment','right',
190 %           'VerticalAlignment','bottom')
191 % text(n(end),-e/2,'-1/2 LSB','HorizontalAlignment','right',
192 %           'VerticalAlignment','top')
193 % xlabel('n (samples)'); ylabel('error')
194 % title('Error between floating point and fixed point direct form II');
195 %
```

```
196 %
197 % % %% Forma cascada
198 %
199 % b_1 = [1 -1.9464 0.94675724];
200 % a_1 = [1 -0.5 -0.5];
201 %
202 % b_2 = [6.5 0];
203 % a_2 = [1 -0.4125];
204 %
205 % y_intermedio = iircanonicofixed(b_1, a_1, x, w, d);
206 %
207 % y_cascada = iircanonicofixed(b_2, a_2, y_intermedio, w, d);
208 %
209 % n=[0:length(y_canonico)-1];
210 % cinco = figure
211 % cinco = plot(n,y_cascada_float,'.-b',n,y_cascada,'.-r')
212 % legend('double','fixed-point')
213 % xlabel('n (samples)'); ylabel('amplitude')
214 % title('Step response cascade')
215 %
216 % seis = figure
217 % seis = plot(n,double(y_cascada)-y_cascada_float,'.-r', ...
218 %      [n(1) n(end)],[e/2 e/2],'c', ...
219 %      [n(1) n(end)],[-e/2 -e/2],'c')
220 % text(n(end),e/2,'+1/2 LSB','HorizontalAlignment','right',
221 %       'VerticalAlignment','bottom')
222 % text(n(end),-e/2,'-1/2 LSB','HorizontalAlignment','right',
223 %       'VerticalAlignment','top')
224 % xlabel('n (samples)'); ylabel('error')
225 % title('Error between floating point and fixed point cascade');
226
227
228
229 % %% Forma paralelo
230
231 w2=18;
232 d2=13;
233
234 f2 = fimath;
235 f2.ProductWordlength = w2+w;
236 f2.ProductFractionLength = d2+d;
237 f2.ProductMode = 'SpecifyPrecision';
238 f2.SumMode = 'SpecifyPrecision';
239 f2.SumWordLength = w2+w;
240 f2.SumFractionLength = d2+d;
241 f2.RoundMode = 'floor';
242
243 % x = fi(x, 1, w2, d2);
244 % k = fi(k, 1, w2, d2);
245
246 k.fimath = f2;
247 x.fimath = f2;
248
```

```
x2 = fi(x, 1, 18, 13);

b_11 = fi(b_11, 1, w, d);
a_11 = fi(a_11, 1, w, d);
b_22 = fi(b_22, 1, w, d);
a_22 = fi(a_22, 1, w, d);
b_33 = fi(b_33, 1, w, d);
a_33 = fi(a_33, 1, w, d);

y_paralelo1 = iircanonicofixed(a_11, b_11, x2, w, d, w2, d2);
y_paralelo2 = iircanonicofixed(a_22, b_22, x2, w, d, w2, d2);
y_paralelo3 = iircanonicofixed(a_33, b_33, x2, w, d, w2, d2);

y_paralelo = y_paralelo1+y_paralelo2+y_paralelo3;

n=[0:length(y_paralelo_float)-1];
siete = figure;
siete = plot(n,y_paralelo_float,'.-b',n,y_paralelo,'.-r')
legend('double','fixed-point')
xlabel('n (samples)'); ylabel('amplitude')
title('Step response parallel')

ocho = figure
ocho = plot(n,double(y_paralelo)-y_paralelo_float,'.-r', ...
     [n(1) n(end)],[e/2 e/2],'c', ...
     [n(1) n(end)],[-e/2 -e/2],'c', n, vector_dac, 'g')
text(n(end),e/2,'+1/2 LSB','HorizontalAlignment',
'right','VerticalAlignment','bottom')
text(n(end),-e/2,'-1/2 LSB',
'HorizontalAlignment','right','VerticalAlignment','top')
text(n(end),vector_dac(1),'LSB of the DPWM block' ,
'HorizontalAlignment','right','VerticalAlignment','top')
xlabel('n (samples)'); ylabel('error')
title('Error between floating point and fixed point parallel');
```

```
function y = iircanonico(b,a,x)

q = length(b);
p = length(a);

x_n = zeros(p,1);

for i=1:length(x),
        x_n(2:q)=x_n(1:q-1);
        x_n(1)= x(i) - a(2:q)*x_n(2:q);

        y(i) = b*x_n;
end
```

```
function y = iircanonicofixed (b,a,x,w1,d1, w2, d2)
```

```
2
3  f = fimath;
4  f.ProductWordlength = w1+w2;
5  f.ProductFractionLength = d1+d2;
6  f.ProductMode = 'SpecifyPrecision';
7  f.SumMode = 'SpecifyPrecision';
8  f.SumWordLength = w1+w2;
9  f.SumFractionLength = d1+d2;
10 f.RoundMode = 'floor';
11
12 q = length(b);
13 p = length(a);
14
15 b = fi(b, 1, w1, d1);
16 a = fi(a, 1, w1, d1);
17
18 x_n = fi(zeros(p,1),1 , w2 , d2);
19 x = fi(x, 1, w2, d2);
20
21 a.fimath = f;
22 b.fimath = f;
23 x.fimath = f;
24 x_n.fimath = f;
25
26 for i=1:length(x),
27         x_n(2:q)=x_n(1:q-1);
28         x_n(1)= x(i) - a(2:q)*x_n(2:q);
29
30         y(i) = b*x_n;
31 end
```

Appendix F

Script processing ModelSim results

```
%% Limpia memorias

clc;
clear all;

%% Abrir archivo de resultados de MODELSIM

load respuestaescalontercer.lst

resultados = respuestaescalontercer(: , 3);
resultados = resultados(2:length(resultados));
resultados = resultados*6.7204*2^-26;

%% Desarrolla la respuesta impulsional real

b = [6.5 -12.6516 6.15392206];
a = [1 -0.9125 -0.29375 0.20625];
Filtro = tf(b, a, 1e-6);

for i=1:length(resultados) ,
    if (i == 1)
        x(i)=1;
    else
        x(i)=1;
    end
end

z = [0;0;0;0];
y = zeros(size(x));

for i=1:length(x) ,
    y(i) =  b(1)*x(i)+z(1);
    z(1) = (b(2)*x(i)+z(2))-a(2)*y(i);
```

```
34      z(2) = (b(3)*x(i)+z(3))-a(3)*y(i);
35      z(3) = -a(4)*y(i);
36  end
37
38  yfloat = y;
39
40  %% Procesado con la toolbox fixed point
41
42  f = fimath;
43  f.ProductWordlength = 36;
44  f.ProductFractionLength = 26;
45  f.ProductMode = 'SpecifyPrecision';
46  f.SumMode = 'SpecifyPrecision';
47  f.SumWordLength = 36;
48  f.SumFractionLength = 26;
49  f.RoundMode = 'floor';
50
51
52  % El numero uno da signo
53
54  b = fi(b, 1, 18, 13);
55  a = fi(a, 1, 18, 13);
56
57  x = fi(x, 1, 18, 13);
58  y = fi(zeros(size(resultados)), 1, 36, 26);
59  z = fi([0;0;0], 1, 18, 13);
60
61  a.fimath = f;
62  b.fimath = f;
63  x.fimath = f;
64  y.fimath = f;
65  z.fimath = f;
66
67  for i=1:length(x) ,
68       y(i) =  b(1)*x(i)+z(1);
69       z(1) = (b(2)*x(i)+z(2))-a(2)*y(i);
70       z(2) = (b(3)*x(i)+z(3))-a(3)*y(i);
71       z(3) = -a(4)*y(i);
72   end
73
74  %% Grafica las dos respuestas
75
76  n=[0:length(y)-1];
77  uno=figure;
78  plot(n, yfloat, '.-r', n, resultados, '.-b');
79  title('Step response');
80  legend('Floating Point','ModelSim Results');
81
82  diferencia= yfloat-resultados';
83  dos=figure;
84  plot(n, diferencia)
85  title('Error between two time series');
```